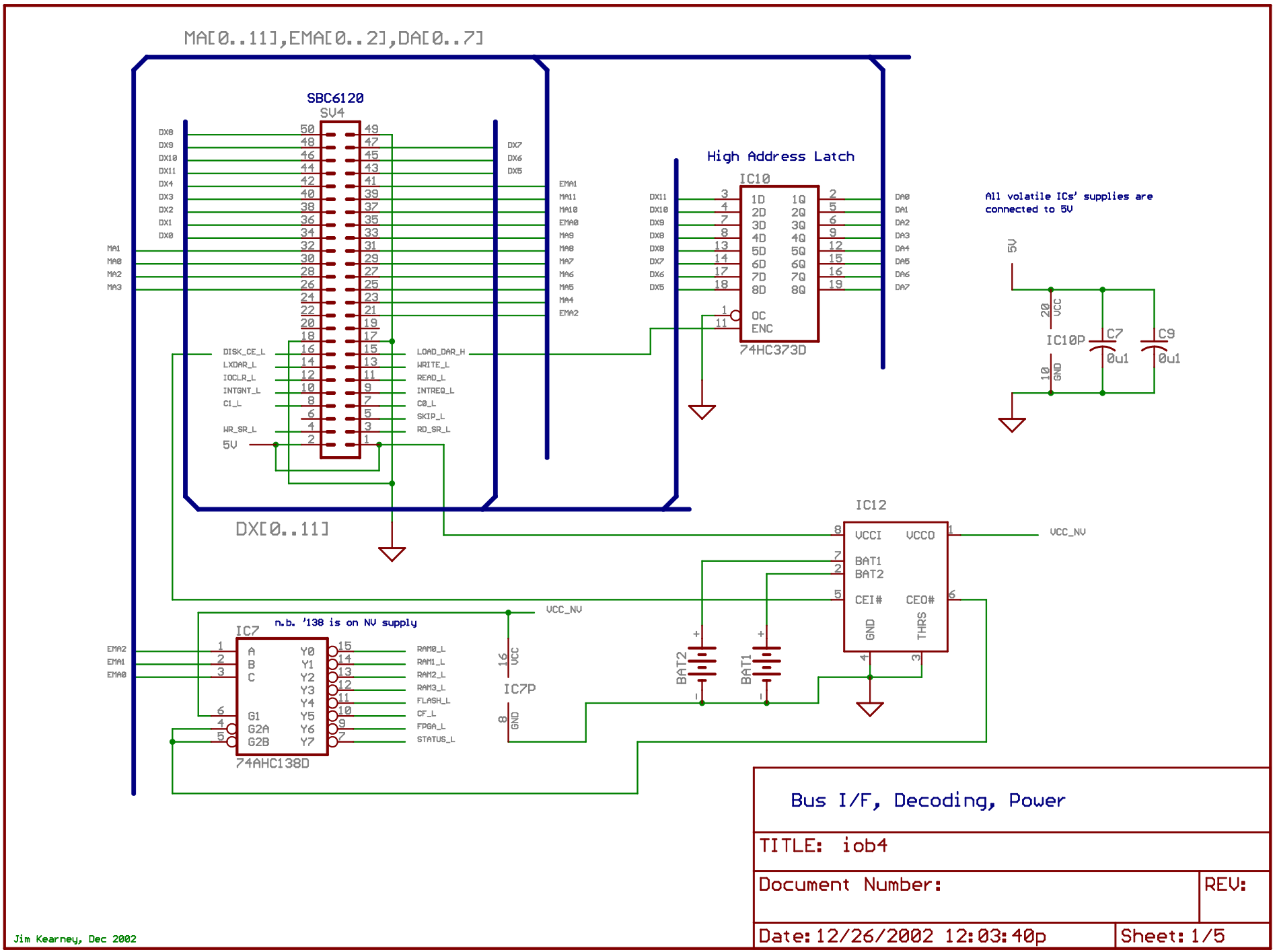
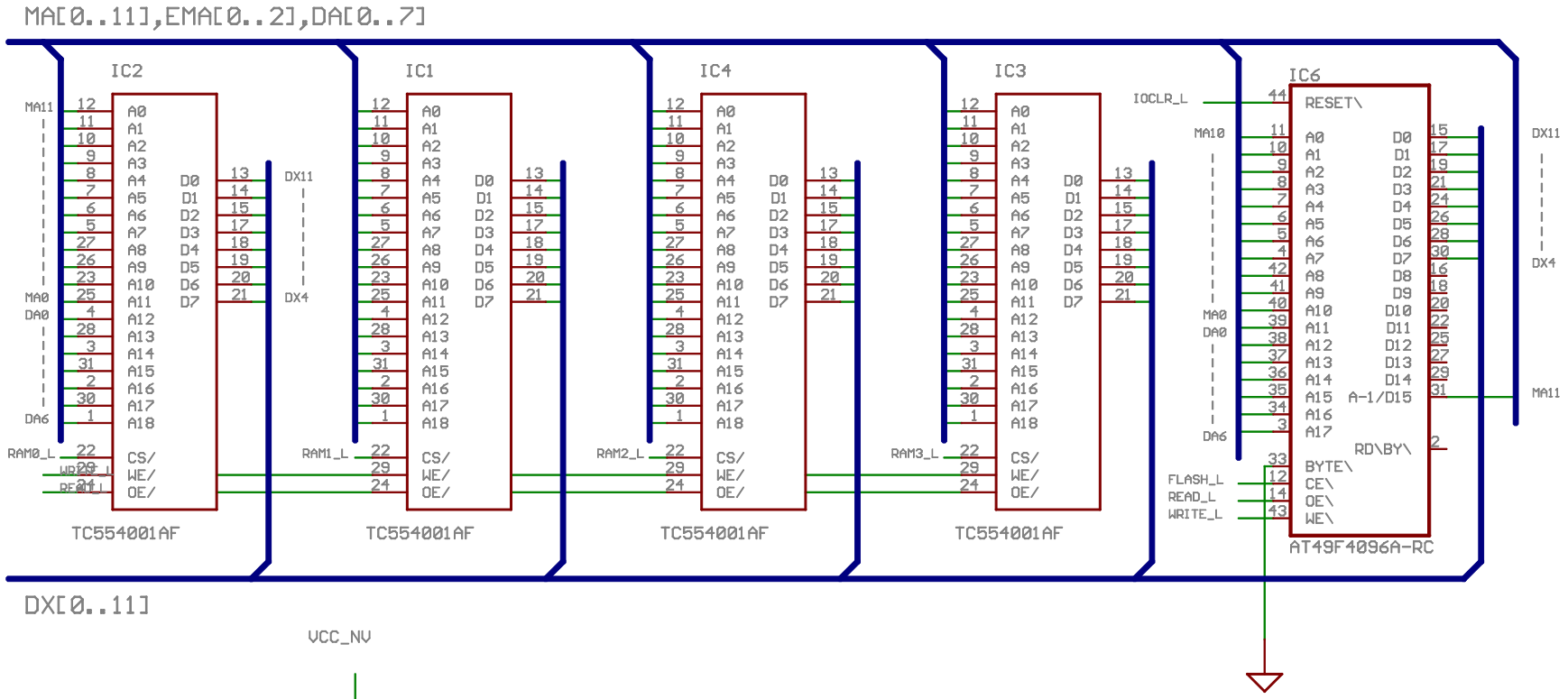
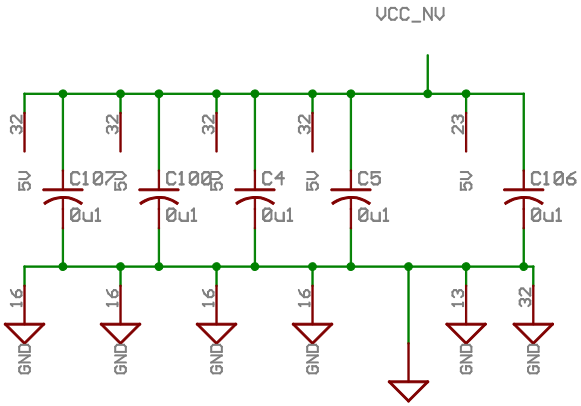
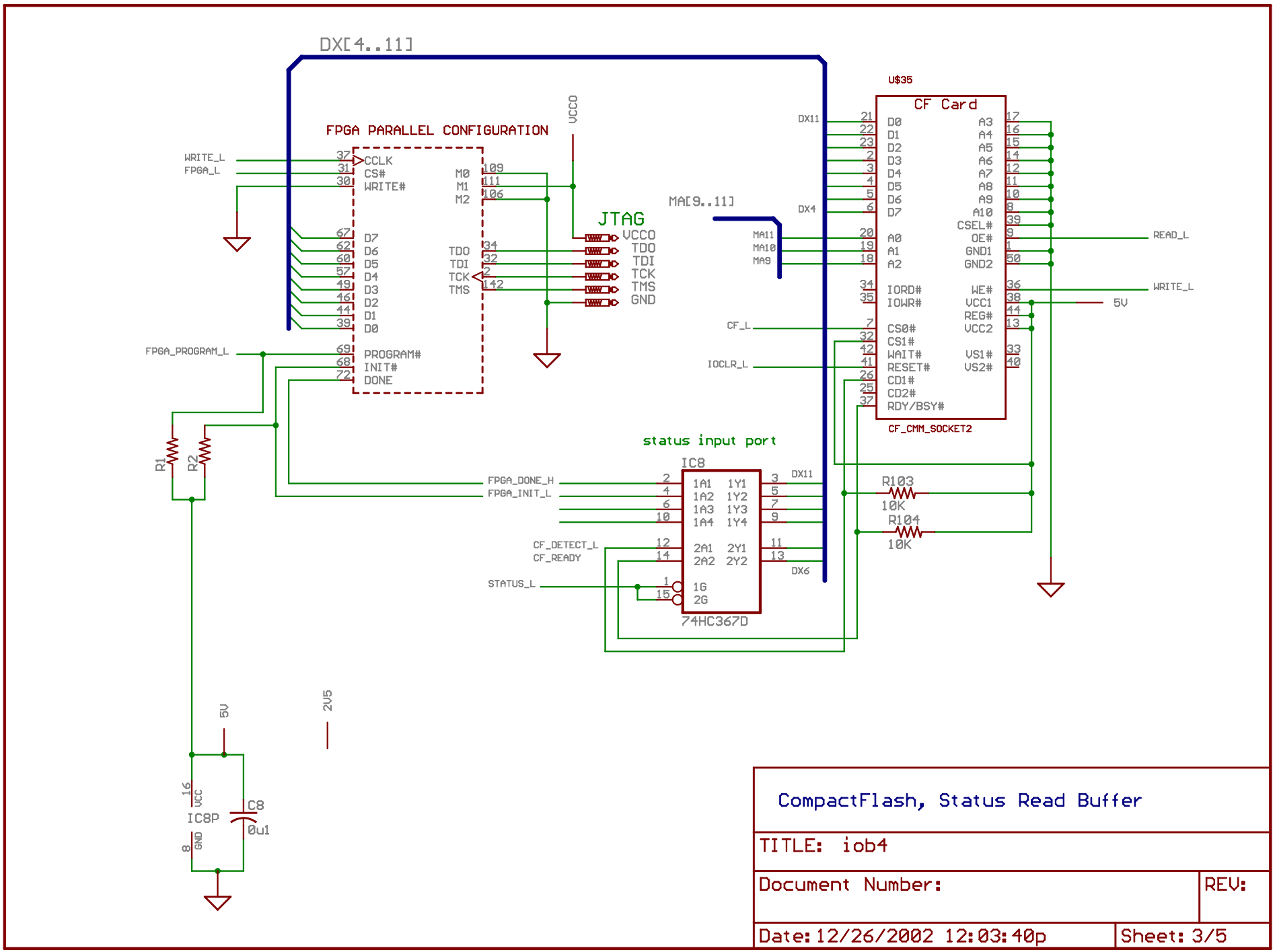

MA[0..11],EMA[0..2],DA[0..7]
SBC6120
SV4
High Address Latch
IC10
74HC373D
All volatile ICs' supplies are connected to 5V
IC10P
C7
C9
0u1
0u1
DX[0..11]
IC12
VCCI
VCCO
BAT1
BAT2
CEI#
CEO#
GND
THRS
VCC_NV
n.b. '138 is on NV supply
IC7
74AHC138D
IC7P
BAT2
BAT1
DISK_CE_L
LXDAR_L
IOCLR_L
INTGNT_L
C1_L
WR_SR_L
5V
LOAD_DAR_H
WRITE_L
READ_L
INTREQ_L
C0_L
SKIP_L
RD_SR_L
RAM0_L
RAM1_L
RAM2_L
RAM3_L
FLASH_L
CF_L
FPGA_L
STATUS_L
Bus I/F, Decoding, Power
TITLE: iob4
Document Number:
REV:
Date: 12/26/2002 12:03:40p
Sheet: 1/5
Jim Kearney, Dec 2002





Memory	
TITLE: iob4	
Document Number:	REV:
Date: 12/26/2002 12:03:40p	Sheet: 2/5



CompactFlash, Status Read Buffer

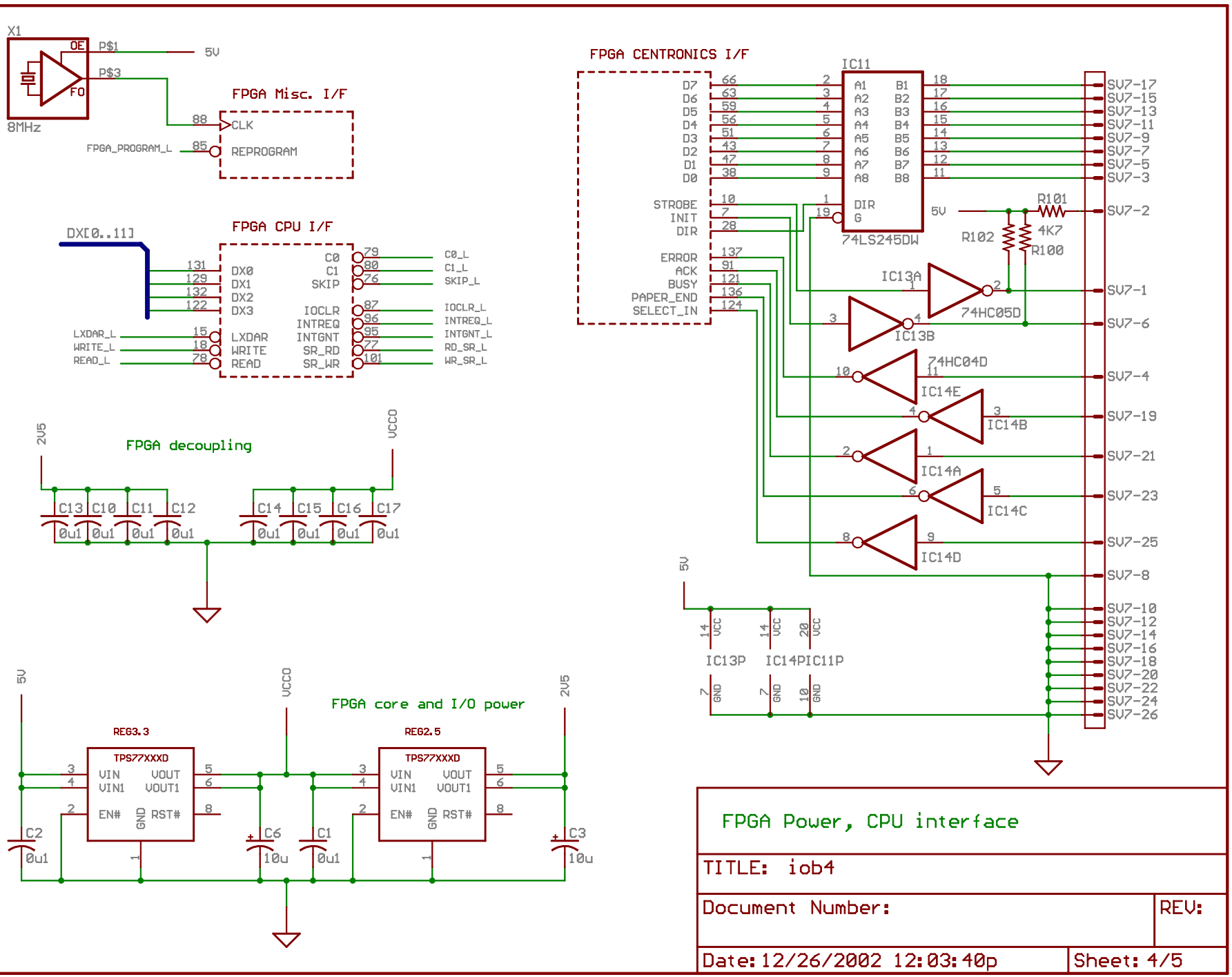
TITLE: iob4

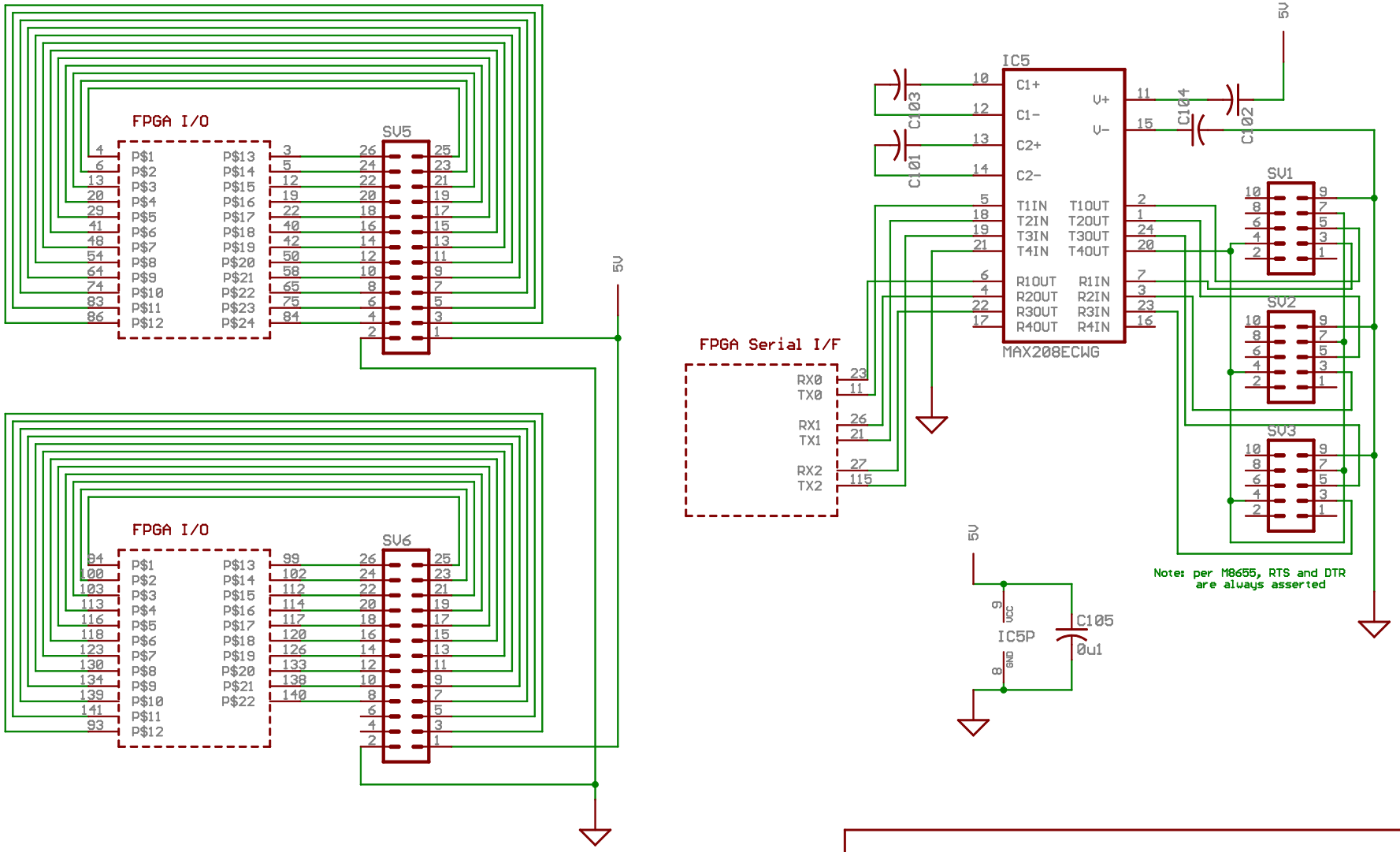
Document Number:

REV:

Date: 12/26/2002 12:03:40p

Sheet: 3/5


X1
OE
P$1
5V
P$3
FO
8MHz
FPGA Misc. I/F
88
CLK
FPGA_PROGRAM_L
85
REPROGRAM
DX[0..11]
FPGA CPU I/F
131
129
132
122
DX0
DX1
DX2
DX3
C0
C1
SKIP
IOCLR
INTREQ
INTGNT
SR_RD
SR_WR
79
80
76
87
96
95
77
101
C0_L
C1_L
SKIP_L
IOCLR_L
INTREQ_L
INTGNT_L
RD_SR_L
WR_SR_L
LXDAR_L
WRITE_L
READ_L
15
18
78
LXDAR
WRITE
READ
2V5
FPGA decoupling
VCCO
C13
C10
C11
C12
C14
C15
C16
C17
0u1
5V
VCCO
2V5
FPGA core and I/O power
REG3.3
REG2.5
TPS77XXXD
VIN
VIN1
VOUT
VOUT1
EN#
GND
RST#
C2
C6
C1
C3
10u
FPGA CENTRONICS I/F
IC11
D7
D6
D5
D4
D3
D2
D1
D0
66
63
59
56
51
43
47
38
A1
A2
A3
A4
A5
A6
A7
A8
B1
B2
B3
B4
B5
B6
B7
B8
18
17
16
15
14
13
12
11
SV7-17
SV7-15
SV7-13
SV7-11
SV7-9
SV7-7
SV7-5
SV7-3
STROBE
INIT
DIR
10
7
28
DIR
G
74LS245DW
ERROR
ACK
BUSY
PAPER_END
SELECT_IN
137
91
121
136
124
5V
R101
R102
4K7
R100
SV7-2
IC13A
74HC05D
SV7-1
IC13B
SV7-6
74HC04D
IC14E
SV7-4
IC14B
SV7-19
IC14A
SV7-21
IC14C
SV7-23
IC14D
SV7-25
SV7-8
SV7-10
SV7-12
SV7-14
SV7-16
SV7-18
SV7-20
SV7-22
SV7-24
SV7-26
IC13P
IC14P
IC11P
VCC
GND
FPGA Power, CPU interface
TITLE: iob4
Document Number:
REV:
Date: 12/26/2002 12:03:40p
Sheet: 4/5



Serial ports, programmable I/O

TITLE: iob4

Document Number: REV:

Date: 12/26/2002 12:03:40p Sheet: 5/5

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

package IOB_Config is

-- generic types
subtype DevID is unsigned(0 to 5);
subtype DevCmd is unsigned(0 to 2);

-- serial ports
constant NUMUARTS: integer := 3;

type UARTIDArray is array (0 to NUMUARTS-1) of DevID;

constant UARTIBASE: UARTIDArray := (O"40", O"30", O"32");
constant UARTOBASE: UARTIDArray := (O"41", O"31", O"33");

-- parallel printer port
constant PARPTIBASE: DevID := O"65";
constant PARPTOBASE: DevID := O"66";

-- IOTs for PIO
constant PIOBASE:  DevID := O"36";  -- set to "00" to remove PIO code
constant PIOSETBANK: DevCmd := O"0";
constant PIOSETDIR: DevCmd := O"1";
constant PIOSETBITS: DevCmd := O"2";

end IOB_Config;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

use work.IOB_Config.ALL;

entity iob is
   port
   ( -- clocks
   clk : in std_logic;

   -- CPU bus interface
   cpu_ioclr_n : in std_logic;
   dx : inout std_logic_vector(0 to 11);
   clk_write_n : in std_logic;
   cpu_read_n : in std_logic;
   cpu_sr_read_n : in std_logic;
   cpu_sr_write_n : in std_logic;
   clk_lxdar_n : in std_logic;
   cpu_c0_n : out std_logic;
   cpu_c1_n : out std_logic;
   cpu_skip_n : out std_logic;
   cpu_intreq_n : out std_logic;
   cpu_intgrnt_n : in std_logic;

   -- serial ports
   txd : out std_logic_vector(0 to NUMUARTS-1);
   rxd : in std_logic_vector(0 to NUMUARTS-1);

   -- LPT port
   lpt_ack: in std_logic;
   lpt_busy_n: in std_logic;
   lpt_paper_end_n: in std_logic;
   lpt_select_in_n: in std_logic;
   lpt_error: in std_logic;
   lpt_strobe: out std_logic;
   lpt_ddir: out std_logic;
   lpt_data: inout std_logic_vector(7 downto 0);
   lpt_init: out std_logic;

   -- bits for custom applications
   iobits: inout std_logic_vector(0 to 45);

   -- special purpose
   reprogram: out std_logic   -- drive low to restart FPGA configuration
   );
end iob;

architecture RTL of iob is

   -- PosEdge: in this implementation, most processes are
   -- synchronous to clk, so this component is used to sample
   -- slower signals, and generates a 1-clock pulse on their rising edge
   component PosEdge
   port
     (reset: in boolean;
      clk: in std_logic;
      inp: in std_logic;
      outp: out std_logic);
   end component;

-- standard IOT input instructions
constant KCF: DevCmd := O"0";
constant KSF: DevCmd := O"1";
constant KCC: DevCmd := O"2";
constant KRS: DevCmd := O"4";
constant KIE: DevCmd := O"5";
```

```
constant KRB: DevCmd := O"6";

-- standard IOT output instructions
constant TFL: DevCmd := O"0";
constant TSF: DevCmd := O"1";
constant TCF: DevCmd := O"2";
constant TPC: DevCmd := O"4";
constant TSK: DevCmd := O"5";
constant TLS: DevCmd := O"6";

-- registers & signals shared between all devices

signal reset: boolean;                    -- true when reset (IOCLR)
signal IOTact: boolean;                   -- true when in IOT (LXDAR)
signal IOTdev: DevID;                     -- latched IOT device code
signal IOTcmd: DevCmd;                    -- latched IOT command
signal IOTread: boolean;                  -- true during IOT read cycle

signal brg: unsigned(9 downto 0);         -- baud rate generator

signal uIRQ: unsigned(0 to NUMUARTS-1);   -- masked flags from the serial ports
signal pIRQ: std_logic;                   -- masked flag from the printer port

-- dummy switch register registers & signals

signal switchreg: std_logic_vector(0 to 11);
signal sr_write: std_logic;

begin

-------------------------------------------------------------------------------
-- utility and shared logic
-------------------------------------------------------------------------------

reprogram <= '1';

-- common IOT decoding

reset <= cpu_ioclr_n = '0';

-- latch IOT device and command at start of LXDAR
process (clk_lxdar_n)
begin
   if falling_edge(clk_lxdar_n) then
      IOTdev <= unsigned(dx(3 to 8));
      IOTcmd <= unsigned(dx(9 to 11));
   end if;
end process;

IOTact <= (clk_lxdar_n = '0');

-- here we stretch the read signal from when it is active
-- until the end of LXDAR; IM6120 wants IOT read data
-- at end of LXDAR, not READ
process (reset, clk, IOTact, cpu_read_n)
begin
   if reset or not IOTact then
      IOTread <= false;
   elsif rising_edge(clk) then
      if cpu_read_n = '0' then
         IOTread <= true;
      end if;
   end if;
end process;

---- bring CPU IRQ low when printer or serial ports request it
cpu_intreq_n <= '0' when pIRQ = '1' or (uIRQ /= 0) else 'Z';
```

```
-- baud rate generator for uarts
process (reset, clk)
begin
   if reset then
      brg <= (others => '0');
   elsif rising_edge(clk) then
      brg <= brg + 1;
   end if;
end process;

-----------------------------------------------------------------------------------------
-- switch register dummy implementation - just a latch
-----------------------------------------------------------------------------------------

process (cpu_sr_read_n, dx, switchreg)
begin
   if cpu_sr_read_n = '0' then
      dx <= switchreg;
   else
      dx <= (others => 'Z');
   end if;
end process;

sr_w_detect: PosEdge port map (reset, clk, cpu_sr_write_n, sr_write);

process (clk, sr_write)
begin
   if rising_edge(clk) then
      if sr_write = '1' then
         switchreg <= dx;
      end if;
   end if;
end process;

-----------------------------------------------------------------------------------------
-- KL8JA serial ports
-----------------------------------------------------------------------------------------

KL8JA: for uidx in 0 to NUMUARTS-1 generate

   -- common
   signal uiflag,                                       -- "keyboard flag"
         uoflag,                                        -- "printer flag"
         IE,                                            -- Interrupt Enable
         SE: std_logic;                                 -- Status Enable

   signal baud_sel: unsigned(0 to 2);                   -- baud config register
            -- "000".."111" -> 38400, 19200, 9600, 4800, 1200, 600, 300, 150

      -- these next 3 registers are stored but not used yet
   signal bits_sel: unsigned(0 to 1);                   -- # of bits config register
   signal parity_sel: unsigned(0 to 1);                 -- parity config register
   signal stops_sel: unsigned(0 to 0);                  -- # of stop bits config register

   signal clk16_raw, clk16, clk1: std_logic;            -- clocking
   signal div16: unsigned(3 downto 0);

   -- receive part
   signal rsel, rclrflag: boolean;
   signal RBR,                                          -- receive buffer register
         RR: std_logic_vector(7 downto 0);              -- receive data register
   signal framing_error, overrun_error: std_logic;
   signal rbit: integer range 0 to 11;
      -- 0 = waiting
      -- 1 = start bit
      -- 2-10 = data bits
```

```
      -- 11 = stop bit
   signal inpv: std_logic_vector(0 to 2);
   signal startdet: boolean;
   signal sample: unsigned(0 to 3);
   signal RBRF_1, RBRF_2,                                  -- Receive Buffer Register Full
          RBRF, RBRF_posedge: std_logic;

   -- transmit part
   signal tsel, write: boolean;
   signal tsetflag, tclrflag: boolean;
   signal TRE: boolean;                                    -- Transmit Register Empty
   signal TR: std_logic_vector(0 to 8);                    -- Transmit Register
   signal tbit: integer range 0 to 9;
      -- 0 = start bit
      -- 1-8 = data bits
      -- 9 = stop bit
   signal THR: std_logic_vector(0 to 7);                   -- Transmit Holding Register
   signal THRE_1, THRE_2, THRE, THRE_posedge: std_logic; -- Transmit Holding Register Empty

begin

   -- clock select
   clk16_raw <= brg(to_integer(baud_sel) + 2);
   baud: PosEdge port map (reset, clk, clk16_raw, clk16);

   -- generate /16 clock
   process (reset, clk, clk16, div16)
   begin
      if reset then
         div16 <= (others => '0');
         clk1 <= '0' after 10 ns;
      elsif rising_edge(clk) then
         if clk16 = '1' then
            if div16 = 15 then
               div16 <= "0000";
               clk1 <= '1' after 10 ns;
            else
               div16 <= div16 + 1;
               clk1 <= '0' after 10 ns;
            end if;
         else
            clk1 <= '0' after 10 ns;
         end if;
      end if;
   end process;

   rsel <= IOTact and (IOTdev = UARTIBASE(uidx));
   uIRQ(uidx) <= IE and (uoflag or uiflag);

   tsel <= IOTact and (IOTdev = UARTOBASE(uidx));
   write <= tsel and (clk_write_n = '0');

   -- manage the registers set by KIE
   -- the IOT is extended here for serial format and speed setting
   -- format:
   --     1 1 b b b n n p p s S I   set baud, number of bits, parity, stop bits, SE, IE
   --     X X X X X X X X X S I   set SE and IE
   process (reset, clk_write_n)
   begin
      if reset then
         baud_sel <= "010";   -- 9600 baud
         IE <= '1';
         SE <= '0';
      elsif rising_edge(clk_write_n) then
         if rsel and (IOTcmd = KIE) then
            SE <= dx(10);
            IE <= dx(11);
```

```
            if (dx(0) = '1') and (dx(1) = '1') then
               baud_sel <= unsigned(dx(2 to 4));
               -- future use --
               bits_sel <= unsigned(dx(5 to 6));
               parity_sel <= unsigned(dx(7 to 8));
               stops_sel <= unsigned(dx(9 to 9));
            end if;
         end if;
      end if;
   end process;

   -- c0/1/skip feedback
   process (rsel, iotcmd, uiflag)
   begin
      if rsel then
         -- c0/c1
         case IOTcmd is

            when KCC | KRB =>
               cpu_c0_n <= '0';

            when others =>
               cpu_c0_n <= 'Z';

         end case;

         case IOTcmd is

            when KRS | KRB =>
               cpu_c1_n <= '0';

            when others =>
               cpu_c1_n <= 'Z';

         end case;

         -- skip
         case IOTcmd is

            when KSF =>
               if uiflag = '1' then
                  cpu_skip_n <= '0';
               else
                  cpu_skip_n <= 'Z';
               end if;

            when others =>
               cpu_skip_n <= 'Z';

         end case;
      else
         cpu_c0_n <= 'Z';
         cpu_c1_n <= 'Z';
         cpu_skip_n <= 'Z';
      end if;
   end process;

   -- put data out to the CPU when requested
   process (rsel, IOTread, RBR, se, framing_error, overrun_error)
   begin
      if rsel and IOTread then
         dx(4 to 11) <= RBR;
         if SE = '1' then
            dx(0 to 3) <= (framing_error or overrun_error) &
                          '0' & framing_error & overrun_error;
         else
            dx(0 to 3) <= (others => '0');
```

```
         end if;
      else
         dx <= (others => 'Z');
      end if;
   end process;

   -- manage "keyboard flag"
   RBRF <= '1' when RBRF_1 /= RBRF_2 else '0';
   RBRFdet: PosEdge port map (reset, clk, RBRF, RBRF_posedge);

   rclrflag <= rsel and (clk_write_n = '0') and
             ((IOTcmd = KCF) or (IOTcmd = KCC) or (IOTcmd = KRB));

   process (reset, rclrflag, clk, RBRF)
   begin
      if reset or rclrflag then
         uiflag <= '0';
         RBRF_2 <= '0';
      elsif rising_edge(clk) then
         if RBRF_posedge = '1' then
            uiflag <= '1';
            RBRF_2 <= RBRF_1;
         end if;
      end if;
   end process;

   -- receive data
   ---- 'vote' last three samples for start bit detection
   startdet <= (inpv = "000");

   process (reset, clk, clk16, RBRF_2)
   begin
      if reset then
         rbit <= 0;
         overrun_error <= '0';
         framing_error <= '0';
         RBRF_1 <= '0';
         inpv <= (others => '1');
         sample <= (others => '0');
      elsif rising_edge(clk) then
         if clk16 = '1' then
            inpv <= rxd(uidx) & inpv(0 to 1);   -- keep last 3 samples
            if rbit = 0 then
               if startdet then
                  rbit <= 1;                  -- start receiving
                  sample <= div16 + "0111";  -- should be the middle of the baud period
               end if;
            elsif div16 = sample then
               if rbit = 10 then              -- end of receive?

                  -- set overrun previous char hasn't been cleared
                  -- !!! this might be too strict; may need to latch
                  -- !!! whether RBR has been read
                  if uiflag = '1' then
                     overrun_error <= '1';
                  end if;

                  -- transfer received byte into RBR and set flag
                  RBRF_1 <= not RBRF_2;
                  RBR <= RR;

                  -- this should be a stop bit
                  if inpv(0) /= '1' then
                     framing_error <= '1';
                  end if;

                  rbit <= 0;                    -- receiver ready for another
```

```
               else
                  -- shift in serial data bit
                  RR <= inpv(0) & RR(7 downto 1);
                  rbit <= rbit + 1;
               end if;
            end if;
         end if;
      end if;
   end process;

   tsetflag <= write and (IOTcmd = TFL);
   tclrflag <= write and ((IOTcmd = TCF) or (IOTcmd = TLS));

   -- manage transmit buffer register
   process (reset, clk_write_n, tsel, THRE_2)
   begin
      if reset then
         THRE_1 <= '0';
      elsif rising_edge(clk_write_n) then
         if tsel and ((IOTcmd = TPC) or (IOTcmd = TLS)) then
            THR <= dx(4 to 11);
            THRE_1 <= not THRE_2;
         end if;
      end if;
   end process;

   -- "printer flag" management
   THRE <= '1' when THRE_1 = THRE_2 else '0';
   THREdet: PosEdge port map (reset, clk, THRE, THRE_posedge);

   process (reset, clk, tsetflag, tclrflag, THRE_posedge)
   begin
      if reset or tclrflag then
         uoflag <= '0';
      elsif tsetflag then
         uoflag <= '1';
      elsif rising_edge(clk) then
         if THRE_posedge = '1' then
            uoflag <= '1';
         end if;
      end if;
   end process;

   -- c0/1/skip feedback
   process (tsel, iotcmd, uoflag, uiflag)
   begin
      if tsel then
         -- c0/c1
         cpu_c0_n <= '1';
         cpu_c1_n <= '1';

         -- skip
         case IOTcmd is

            when TSF =>
               if uoflag = '1' then
                  cpu_skip_n <= '0';
               else
                  cpu_skip_n <= 'Z';
               end if;

            when TSK =>
               if (uoflag or uiflag) = '1' then
                  cpu_skip_n <= '0';
               else
                  cpu_skip_n <= 'Z';
               end if;
```

```
            when others =>
               cpu_skip_n <= 'Z';

         end case;
      else
         cpu_c0_n <= 'Z';
         cpu_c1_n <= 'Z';
         cpu_skip_n <= 'Z';
      end if;
   end process;

   -- serial output
   txd(uidx) <= TR(8);
   TRE <= (tbit = 9);

   process (reset, clk, clk1, THRE_1, THRE)
   begin
      if reset then
         TR(8) <= '1';
         tbit <= 9;
         THRE_2 <= '0';
      elsif rising_edge(clk) then
         if clk1 = '1' then
            if TRE then
               -- if buffered data, start it
               if THRE = '0' then
                  tbit <= 0;
                  TR <= THR & '0';
                  THRE_2 <= THRE_1;
               end if;
            else
               tbit <= tbit + 1;
               TR <= '1' & TR(0 to 7);
            end if;
         end if;
      end if;
   end process;

end generate;

-------------------------------------------------------------------------------------------
-- LC8E printer interface with Centronics output
-------------------------------------------------------------------------------------------

-- LC8E registers & signals

LC8E_gen: if true generate

   signal lpt_sel, lpt_wr: boolean;
   signal lpt_ready, lpt_strobe_int,
        lpt_flag, lpt_IE: std_logic;

begin

   lpt_ready <= '1' when (lpt_busy_n = '1') and (lpt_error = '0') and
                (lpt_paper_end_n = '1') and (lpt_select_in_n = '0')
                else '0';

   lpt_init <= not cpu_ioclr_n;
   lpt_strobe <= lpt_strobe_int;
   lpt_ddir <= '1';                          -- always output for now
   lpt_data(7 downto 0) <= dx(4 to 11);

   lpt_sel <= (clk_lxdar_n = '0') and (IOTdev = PARPTOBASE);
   lpt_wr <= lpt_sel and (clk_write_n = '0');
   lpt_strobe_int <= '1' when lpt_sel and (clk_write_n = '0') and
```

```
               ((IOTcmd = TPC) or (IOTcmd = TLS)) else '0';

   pIRQ <= lpt_IE and lpt_flag;

   --maintain printer flag

   process (reset, lpt_wr, IOTcmd, lpt_ack)
   begin
      if reset or (lpt_wr and (IOTcmd = TCF)) then
         lpt_flag <= '0';
      elsif lpt_wr and ((IOTcmd = TFL) or (IOTcmd = TLS)) then
         lpt_flag <= '1';
      elsif rising_edge(lpt_ack) then
         lpt_flag <= '1';
      end if;
   end process;

   -- handle c0/c1/skip

   process (lpt_sel, IOTcmd, lpt_flag, lpt_ready)
   begin
      if lpt_sel then
         -- skip
         case IOTcmd is

            when TSF | TSK =>
               if (lpt_flag and lpt_ready) = '1' then
                  cpu_skip_n <= '0';
                else
                  cpu_skip_n <= 'Z';
               end if;

            when others =>
               cpu_skip_n <= 'Z';

         end case;
      else
         cpu_skip_n <= 'Z';
      end if;
   end process;

   -- handle IOTs
   -- in this case, the print strobe is passed straight through to the printer,
   -- so the only command we have to decode here is KIE to maintain the
   -- Interrupt Enable flag
   process (reset, clk_write_n, dx)
   begin
      if reset then
         lpt_IE <= '0';
      elsif rising_edge(clk_write_n) then
         if (clk_lxdar_n = '0') and
            (IOTdev = PARPTIBASE) and
            (IOTcmd = KIE) then
            lpt_IE <= dx(11);
         end if;
      end if;
   end process;

end generate; -- LC8E

-------------------------------------------------------------------------------
-- Port bits IOT
-- for now, just implement programmable I/O ports, 3x 12 bit and 1x 10 bit

iobits_if: if PIOBASE /= O"00" generate

   signal bits_bank: unsigned(0 to 1);
```

```
   signal bits_sel: boolean;
   signal bits_dir: std_logic_vector(iobits'range);
   signal bits_out: std_logic_vector(iobits'range);

begin

   process (IOTact, IOTdev)
   begin
      bits_sel <= IOTact and (IOTdev = PIOBASE);
   end process;

   process (bits_out, bits_dir)
   begin
      for i in iobits'range loop
         if bits_dir(i) = '0' then
            iobits(i) <= 'Z';
         else
            iobits(i) <= bits_out(i);
         end if;
      end loop;
   end process;

   -- reading
   process (IOTread, bits_sel, bits_bank, iobits)
   begin
      if bits_sel and IOTread then
         case bits_bank is

            when "00" => dx <= iobits(0 to 11);

            when "01" => dx <= iobits(12 to 23);

            when "10" => dx <= iobits(24 to 35);

            when "11" => dx <= iobits(36 to 45) & "00";

            when others => null;

         end case;
      else
         dx <= (others => 'Z');
      end if;
   end process;

   -- writing
   process (reset, bits_sel, clk_write_n, dx)
   begin
      if reset then
         bits_dir <= (others => '0');  -- inputs on reset
      elsif rising_edge(clk_write_n) then
         if bits_sel then
            if IOTcmd = PIOSETBITS then -- write bits
               case bits_bank is

                  when "00" => bits_out(0 to 11) <= dx;

                  when "01" => bits_out(12 to 23) <= dx;

                  when "10" => bits_out(24 to 35) <= dx;

                  when "11" => bits_out(36 to 45) <= dx(0 to 9);

                  when others => null;

               end case;
            elsif IOTcmd = PIOSETBANK then   -- select bank
               bits_bank <= unsigned(dx(10 to 11));
```

```
            elsif IOTcmd = PIOSETDIR then -- set direction
               case bits_bank is

                  when "00" => bits_dir(0 to 11) <= dx;

                  when "01" => bits_dir(12 to 23) <= dx;

                  when "10" => bits_dir(24 to 35) <= dx;

                  when "11" => bits_dir(36 to 45) <= dx(0 to 9);

                  when others => null;

               end case;
            end if;
         end if;
      end if;
   end process;

end generate; -- if PIOBASE /= O"00"

end RTL;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity PosEdge is
    port ( reset: in boolean;
           clk, inp: in std_logic;
           outp: out std_logic);
end PosEdge;

architecture RTL of PosEdge is

signal inp_prev: std_logic;

begin

   process (reset, clk, inp)
   begin
      if reset then
         outp <= '0';
      elsif rising_edge(clk) then
         inp_prev <= inp;
         if (inp = '1') and (inp_prev = '0') then
            outp <= '1' after 10 ns;
         else
            outp <= '0' after 10 ns;
         end if;
      end if;
   end process;

end RTL;
```

```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 8Mhz HIGH 50 %;

CONFIG PROHIBIT = "P68";
CONFIG PROHIBIT = "P30";
CONFIG PROHIBIT = "P31";

# map data bus to parallel config pins
#NET "dx<11>" LOC = "P39";
#NET "dx<10>" LOC = "P44";
#NET "dx<9>" LOC = "P46";
#NET "dx<8>" LOC = "P49";
#NET "dx<7>" LOC = "P57";
#NET "dx<6>" LOC = "P60";
#NET "dx<5>" LOC = "P62";
#NET "dx<4>" LOC = "P67";
#NET "dx<3>" LOC = "L";
#NET "dx<2>" LOC = "L";
#NET "dx<1>" LOC = "L";
#NET "dx<0>" LOC = "L";

# locate clocks explicitly
#NET "clk_write_n" LOC = "GCLKPAD2";
#NET "clk_lxdar_n" LOC = "GCLKPAD3";

# constrain other groups to be on side near their
# associated circuitry
#NET "cpu_*" LOC = "B";
#NET "txd<*>" LOC = "T","L";
#NET "rxd<*>" LOC = "T","L";
#NET "lpt_data<*>" LOC = "L","T";
#NET "lpt_ack_n" LOC = "GCLKPAD1";
#NET "lpt_busy" LOC = "T","L";
#NET "lpt_paper_end" LOC = "T","L";
#NET "lpt_select_in" LOC = "T","L";
#NET "lpt_error_n" LOC = "T","L";
#NET "lpt_strobe_n" LOC = "T","L";
#NET "lpt_strobe_n" LOC = "T","L";
#NET "lpt_ddir" LOC = "T","L";
#NET "lpt_init_n" LOC = "T","L";

# locked pins
NET "clk" LOC = "P88";
NET "clk_lxdar_n" LOC = "P15";
NET "clk_write_n" LOC = "P18";
NET "cpu_c0_n" LOC = "P79";
NET "cpu_c1_n" LOC = "P80";
#NET "cpu_intgrnt_n"  LOC =  "P95";
NET "cpu_intreq_n" LOC = "P96";
NET "cpu_ioclr_n" LOC = "P87";
NET "cpu_read_n" LOC = "P78";
NET "cpu_skip_n" LOC = "P76";
NET "cpu_sr_read_n" LOC = "P77";
NET "cpu_sr_write_n" LOC = "P101";
NET "dx<0>" LOC = "P131";
NET "dx<1>" LOC = "P129";
NET "dx<2>" LOC = "P132";
NET "dx<3>" LOC = "P122";
NET "dx<4>" LOC = "P67";
NET "dx<5>" LOC = "P62";
NET "dx<6>" LOC = "P60";
NET "dx<7>" LOC = "P57";
NET "dx<8>" LOC = "P49";
NET "dx<9>" LOC = "P46";
NET "dx<10>" LOC = "P44";
NET "dx<11>" LOC = "P39";
NET "lpt_ack" LOC = "P91";
```

```
NET "lpt_busy_n" LOC = "P121";
NET "lpt_data<0>" LOC = "P38";
NET "lpt_data<1>" LOC = "P47";
NET "lpt_data<2>" LOC = "P43";
NET "lpt_data<3>" LOC = "P51";
NET "lpt_data<4>" LOC = "P56";
NET "lpt_data<5>" LOC = "P59";
NET "lpt_data<6>" LOC = "P63";
NET "lpt_data<7>" LOC = "P66";
NET "lpt_ddir" LOC = "P28";
NET "lpt_error" LOC = "P137";
NET "lpt_init" LOC = "P7";
NET "lpt_paper_end_n" LOC = "P136";
NET "lpt_select_in_n" LOC = "P124";
NET "lpt_strobe" LOC = "P10";
NET "reprogram" LOC = "P85";
NET "rxd<0>" LOC = "P23";
NET "rxd<1>" LOC = "P26";
NET "rxd<2>" LOC = "P27";
NET "txd<0>" LOC = "P11";
NET "txd<1>" LOC = "P21";
NET "txd<2>" LOC = "P115";
NET "iobits<0>" LOC = "P4";
NET "iobits<1>" LOC = "P6";
NET "iobits<2>" LOC = "P13";
NET "iobits<3>" LOC = "P20";
NET "iobits<4>" LOC = "P29";
NET "iobits<5>" LOC = "P41";
NET "iobits<6>" LOC = "P48";
NET "iobits<7>" LOC = "P54";
NET "iobits<8>" LOC = "P64";
NET "iobits<9>" LOC = "P74";
NET "iobits<10>" LOC = "P83";
NET "iobits<11>" LOC = "P86";
NET "iobits<12>" LOC = "P3";
NET "iobits<13>" LOC = "P5";
NET "iobits<14>" LOC = "P12";
NET "iobits<15>" LOC = "P19";
NET "iobits<16>" LOC = "P22";
NET "iobits<17>" LOC = "P40";
NET "iobits<18>" LOC = "P42";
NET "iobits<19>" LOC = "P50";
NET "iobits<20>" LOC = "P58";
NET "iobits<21>" LOC = "P65";
NET "iobits<22>" LOC = "P75";
NET "iobits<23>" LOC = "P84";
NET "iobits<24>" LOC = "P94";
NET "iobits<25>" LOC = "P100";
NET "iobits<26>" LOC = "P103";
NET "iobits<27>" LOC = "P113";
NET "iobits<28>" LOC = "P116";
NET "iobits<29>" LOC = "P118";
NET "iobits<30>" LOC = "P123";
NET "iobits<31>" LOC = "P130";
NET "iobits<32>" LOC = "P134";
NET "iobits<33>" LOC = "P139";
NET "iobits<34>" LOC = "P141";
NET "iobits<35>" LOC = "P93";
NET "iobits<36>" LOC = "P99";
NET "iobits<37>" LOC = "P102";
NET "iobits<38>" LOC = "P112";
NET "iobits<39>" LOC = "p114";
NET "iobits<40>" LOC = "P117";
NET "iobits<41>" LOC = "P120";
NET "iobits<42>" LOC = "P126";
NET "iobits<43>" LOC = "P133";
NET "iobits<44>" LOC = "P138";
```

```
NET "iobits<45>" LOC = "P140";
```

```
-- VHDL Test Bench Created from source file iob.vhd -- 18:26:00 12/23/2002
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

   constant SERBIT: time := 104.167 us;

   COMPONENT iob
   PORT(
      clk : IN std_logic;
      cpu_ioclr_n : IN std_logic;
      clk_write_n : IN std_logic;
      cpu_read_n : IN std_logic;
      cpu_sr_read_n : IN std_logic;
      cpu_sr_write_n : IN std_logic;
      clk_lxdar_n : IN std_logic;
      cpu_intgrnt_n : IN std_logic;
      rxd : IN std_logic_vector(0 to 2);
      lpt_ack : IN std_logic;
      lpt_busy_n : IN std_logic;
      lpt_paper_end_n : IN std_logic;
      lpt_select_in_n : IN std_logic;
      lpt_error : IN std_logic;
      dx : INOUT std_logic_vector(0 to 11);
      lpt_data : INOUT std_logic_vector(7 downto 0);
      iobits : INOUT std_logic_vector(0 to 45);
      cpu_c0_n : OUT std_logic;
      cpu_c1_n : OUT std_logic;
      cpu_skip_n : OUT std_logic;
      cpu_intreq_n : OUT std_logic;
      txd : OUT std_logic_vector(0 to 2);
      lpt_strobe : OUT std_logic;
      lpt_ddir : OUT std_logic;
      lpt_init : OUT std_logic;
      reprogram : OUT std_logic
      );
   END COMPONENT;

   SIGNAL clk :  std_logic;
   SIGNAL cpu_ioclr_n :  std_logic;
   SIGNAL dx :  std_logic_vector(0 to 11);
   SIGNAL clk_write_n :  std_logic;
   SIGNAL cpu_read_n :  std_logic;
   SIGNAL cpu_sr_read_n :  std_logic;
   SIGNAL cpu_sr_write_n :  std_logic;
   SIGNAL clk_lxdar_n :  std_logic;
   SIGNAL cpu_c0_n :  std_logic;
   SIGNAL cpu_c1_n :  std_logic;
   SIGNAL cpu_skip_n :  std_logic;
   SIGNAL cpu_intreq_n :  std_logic;
   SIGNAL cpu_intgrnt_n :  std_logic;
   SIGNAL txd :  std_logic_vector(0 to 2);
   SIGNAL rxd :  std_logic_vector(0 to 2);
```

```
   SIGNAL lpt_ack :  std_logic;
   SIGNAL lpt_busy_n :  std_logic;
   SIGNAL lpt_paper_end_n :  std_logic;
   SIGNAL lpt_select_in_n :  std_logic;
   SIGNAL lpt_error :  std_logic;
   SIGNAL lpt_strobe :  std_logic;
   SIGNAL lpt_ddir :  std_logic;
   SIGNAL lpt_data :  std_logic_vector(7 downto 0);
   SIGNAL lpt_init :  std_logic;
   SIGNAL iobits :  std_logic_vector(0 to 45);
   SIGNAL reprogram :  std_logic;

   signal receive_done: boolean;

BEGIN

   uut: iob PORT MAP(
      clk => clk,
      cpu_ioclr_n => cpu_ioclr_n,
      dx => dx,
      clk_write_n => clk_write_n,
      cpu_read_n => cpu_read_n,
      cpu_sr_read_n => cpu_sr_read_n,
      cpu_sr_write_n => cpu_sr_write_n,
      clk_lxdar_n => clk_lxdar_n,
      cpu_c0_n => cpu_c0_n,
      cpu_c1_n => cpu_c1_n,
      cpu_skip_n => cpu_skip_n,
      cpu_intreq_n => cpu_intreq_n,
      cpu_intgrnt_n => cpu_intgrnt_n,
      txd => txd,
      rxd => rxd,
      lpt_ack => lpt_ack,
      lpt_busy_n => lpt_busy_n,
      lpt_paper_end_n => lpt_paper_end_n,
      lpt_select_in_n => lpt_select_in_n,
      lpt_error => lpt_error,
      lpt_strobe => lpt_strobe,
      lpt_ddir => lpt_ddir,
      lpt_data => lpt_data,
      lpt_init => lpt_init,
      iobits => iobits,
      reprogram => reprogram
   );

   clock : process
   begin
      clk <= '1';
      wait for 101.725 ns;
      clk <= '0';
      wait for 101.725 ns;
   end process;

   receive: process
   begin
      receive_done <= false;

      rxd <= (others => '1');
      wait for 2 us;
      rxd(0) <= '0';
      wait for SERBIT;
      rxd(0) <= '1';
      wait for 3*SERBIT;
      rxd(0) <= '0';
      wait for 5*SERBIT;
      rxd(0) <= '1';
      wait for SERBIT;
```

```
      wait for SERBIT;
      receive_done <= true;
      wait;
   end process;

-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
      -- initial state of all inputs
      dx <= (others => 'Z');
      clk_write_n <= '1';
      cpu_read_n <= '1';
      cpu_sr_read_n <= '1';
      cpu_sr_write_n <= '1';
      clk_lxdar_n <= '1';
      lpt_ack <= '0';
      lpt_busy_n <= '1';
      lpt_paper_end_n <= '0';
      lpt_select_in_n <= '0';
      lpt_error <= '0';
      cpu_ioclr_n <= '1';


      -- reset
      wait for 1 us;
      cpu_ioclr_n <= '0';
      wait for 500 ns;
      cpu_ioclr_n <= '1';
      wait for 500 ns;

      -- write a character to serial port
      dx <= O"6416";
      wait for 125 ns;
      clk_lxdar_n <= '0';
      wait for 180 ns;
      dx <= (others => 'Z');
      wait for 20 ns;
      clk_write_n <= '0';
      wait for 175 ns;
      dx <= O"0255";
      wait for 200 ns;
      clk_write_n <= '1';
      wait for 127 ns;
      dx <= (others => 'Z');
      wait for 400 ns;
      clk_lxdar_n <= '1';
      wait for 392 ns;

      -- write another character to serial port, testing double-buffering
      wait for 600 us;
      dx <= O"6416";
      wait for 125 ns;
      clk_lxdar_n <= '0';
      wait for 180 ns;
      dx <= (others => 'Z');
      wait for 20 ns;
      clk_write_n <= '0';
      wait for 175 ns;
      dx <= O"0255";
      wait for 200 ns;
      clk_write_n <= '1';
      wait for 127 ns;
      dx <= (others => 'Z');
      wait for 400 ns;
      clk_lxdar_n <= '1';
      wait for 392 ns;
```

```
      -- receive a character
      -- ask for the received character
      wait until receive_done;

      dx <= O"6406";
      wait for 125 ns;
      clk_lxdar_n <= '0';
      wait for 180 ns;
      dx <= (others => 'Z');
      wait for 20 ns;
      clk_write_n <= '0';
      wait for 175 ns;
      dx <= O"0000";
      wait for 200 ns;
      clk_write_n <= '1';
      wait for 127 ns;
      dx <= (others => 'Z');
      wait for 200 ns;
      cpu_read_n <= '0';
      wait for 425 ns;
      cpu_read_n <= '1';
      wait for 220 ns;
      clk_lxdar_n <= '1';
      wait for 392 ns;

      -- receive a character with a framing error

      wait; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```