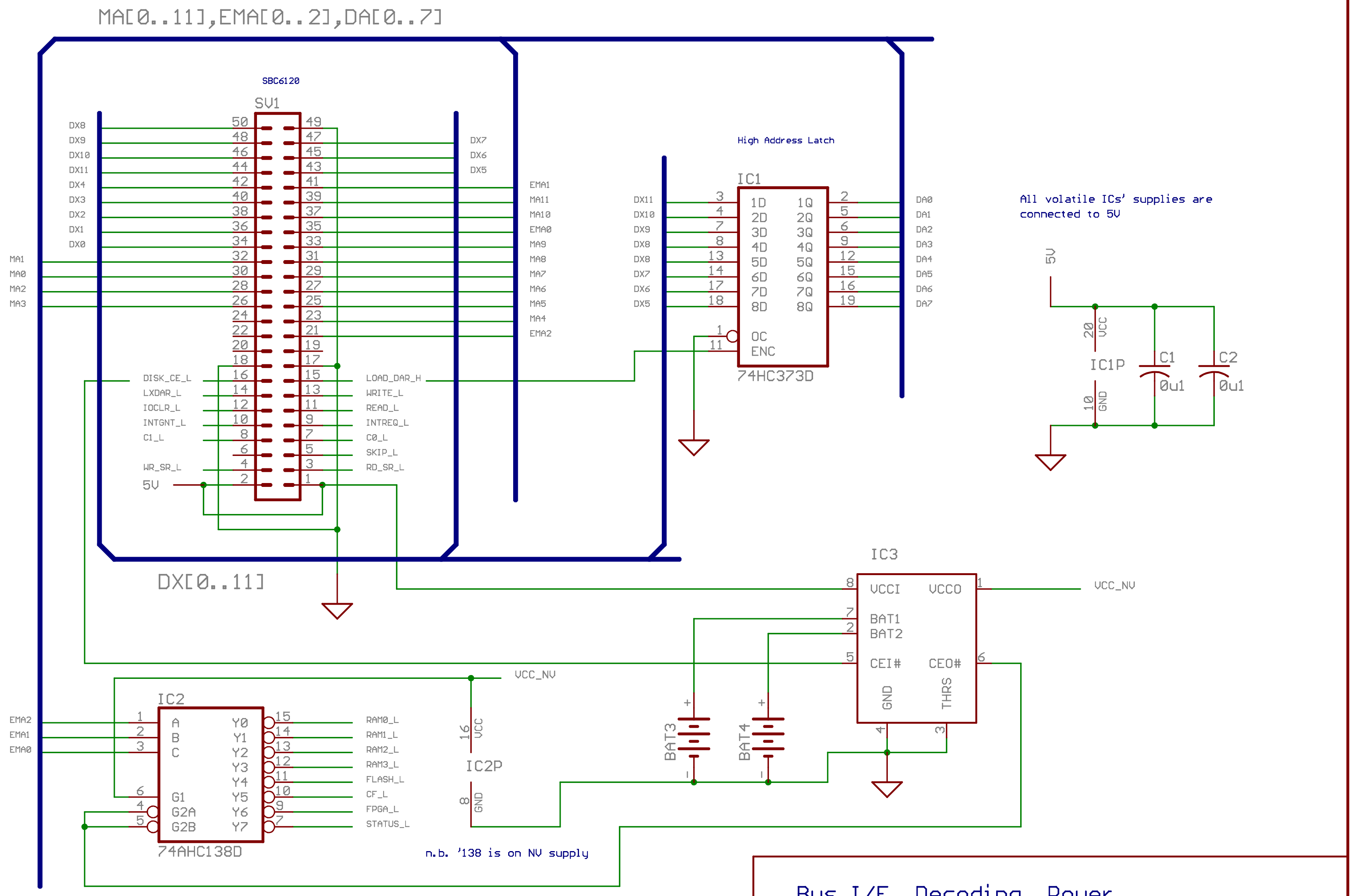


Bus I/F, Decoding, Power

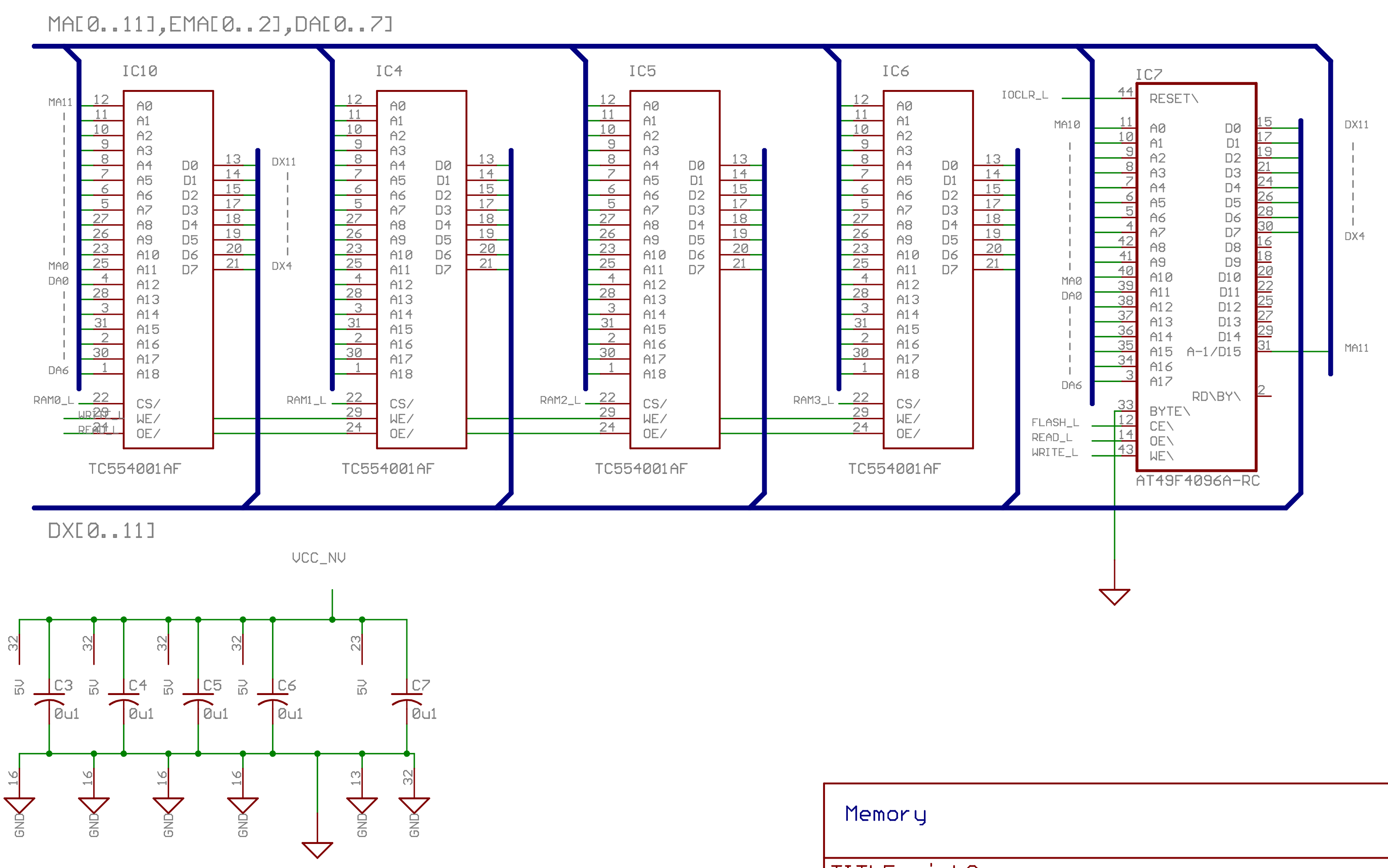
TITLE: iob3

Document Number:

REV:

Date: 12/21/2002 11:38:58a

Sheet: 1/5



Memory

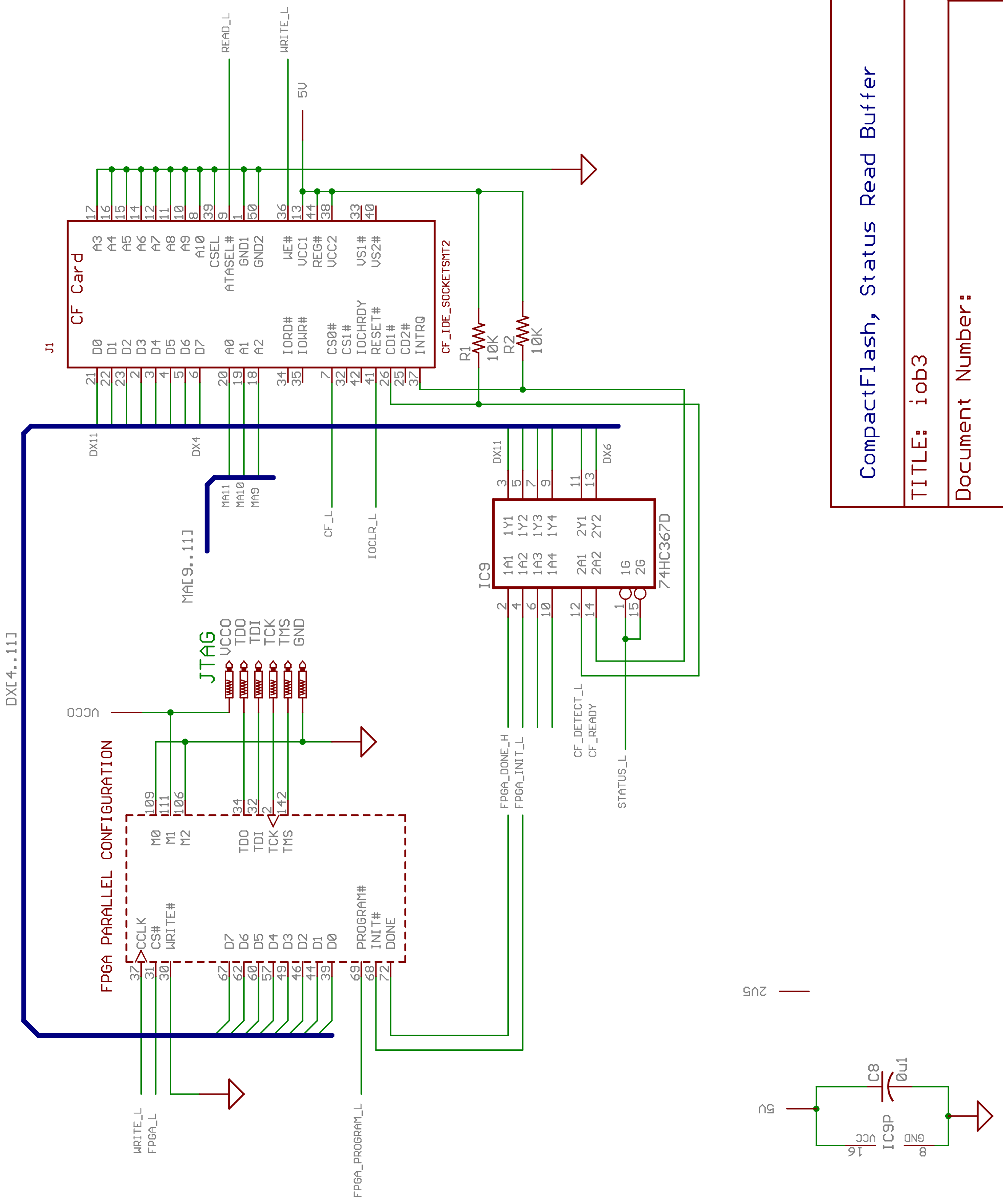
TITLE: iob3

Document Number:

REV:

Date: 12/21/2002 11:38:58a

Sheet: 2/5



CompactFlash, Status Read Buffer

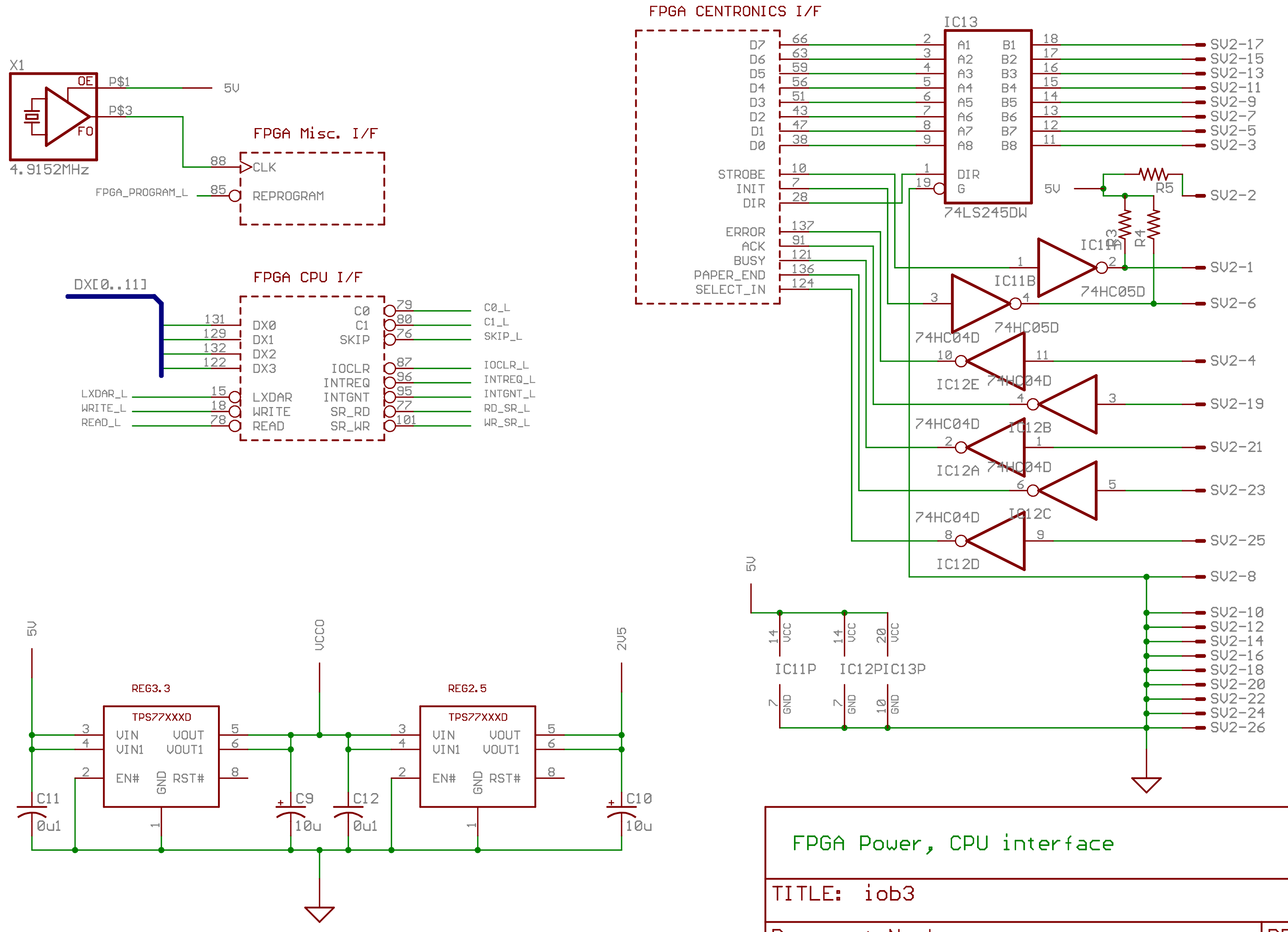
TITLE: iob3

Document Number:

REV:

Date: 12/21/2002 11:38:58a

Sheet: 3/5



FPGA Power, CPU interface

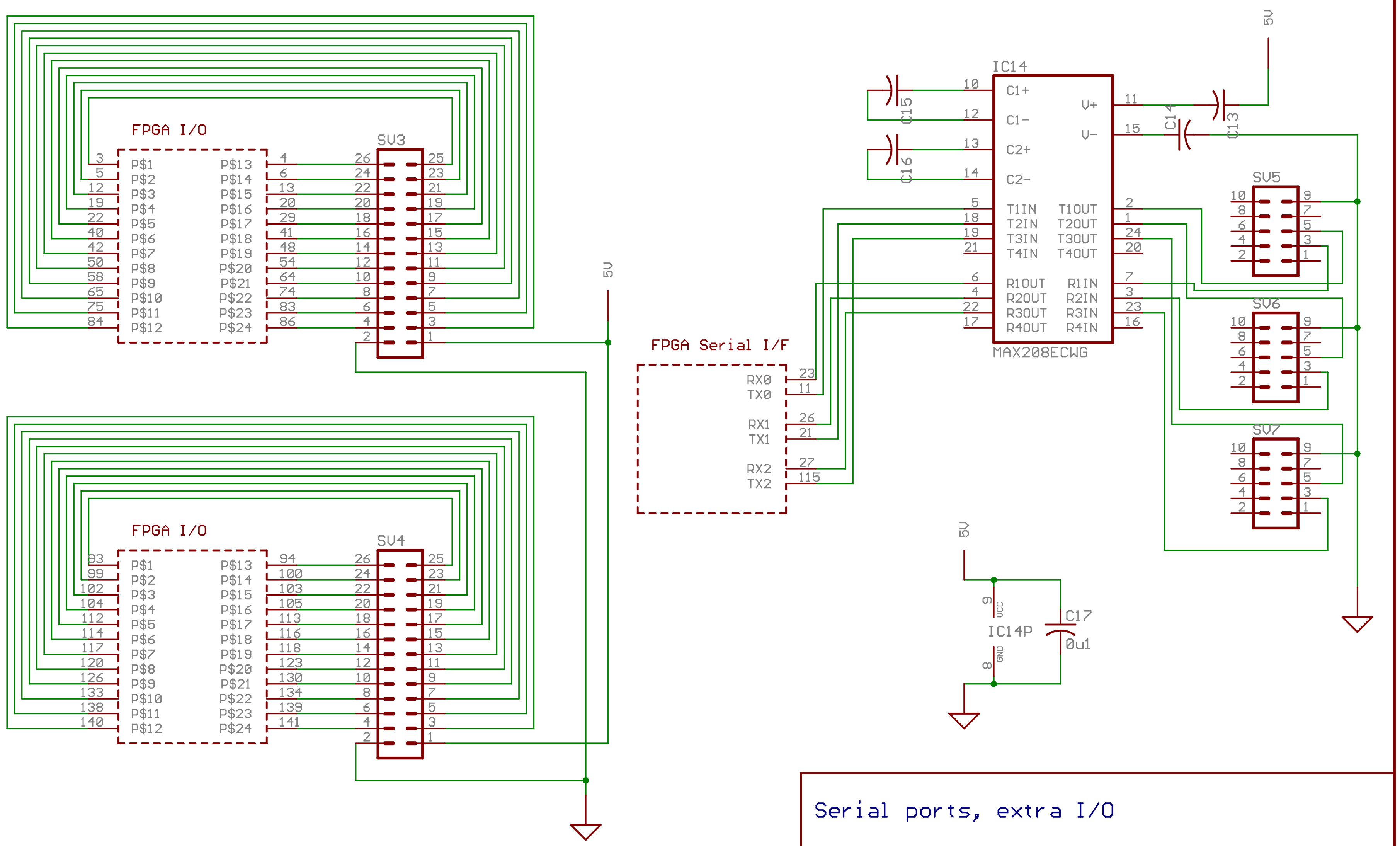
TITLE: iob3

Document Number:

REV:

Date: 12/21/2002 11:38:58a

Sheet: 4/5



Serial ports, extra I/O

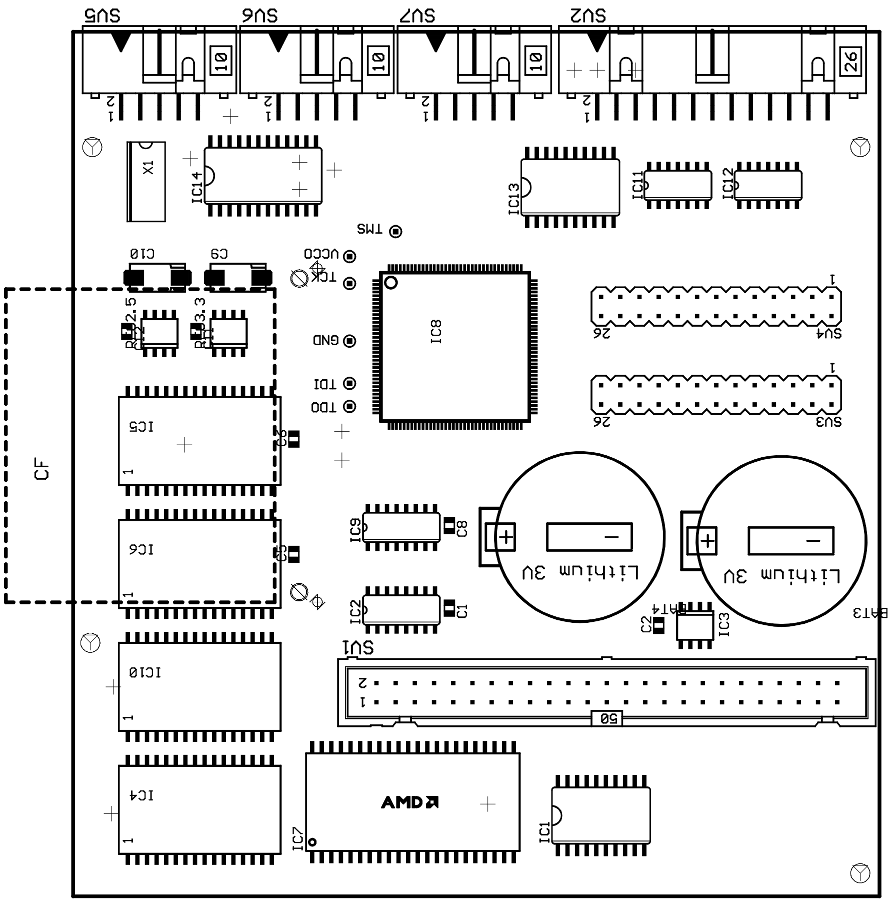
TITLE: iob3

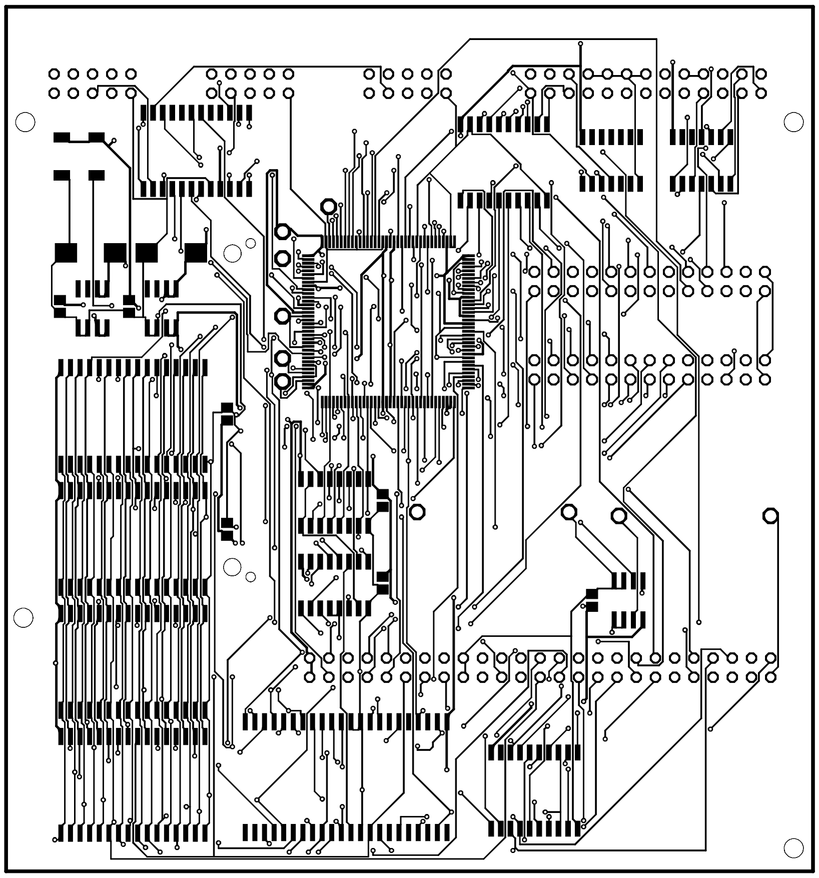
Document Number:

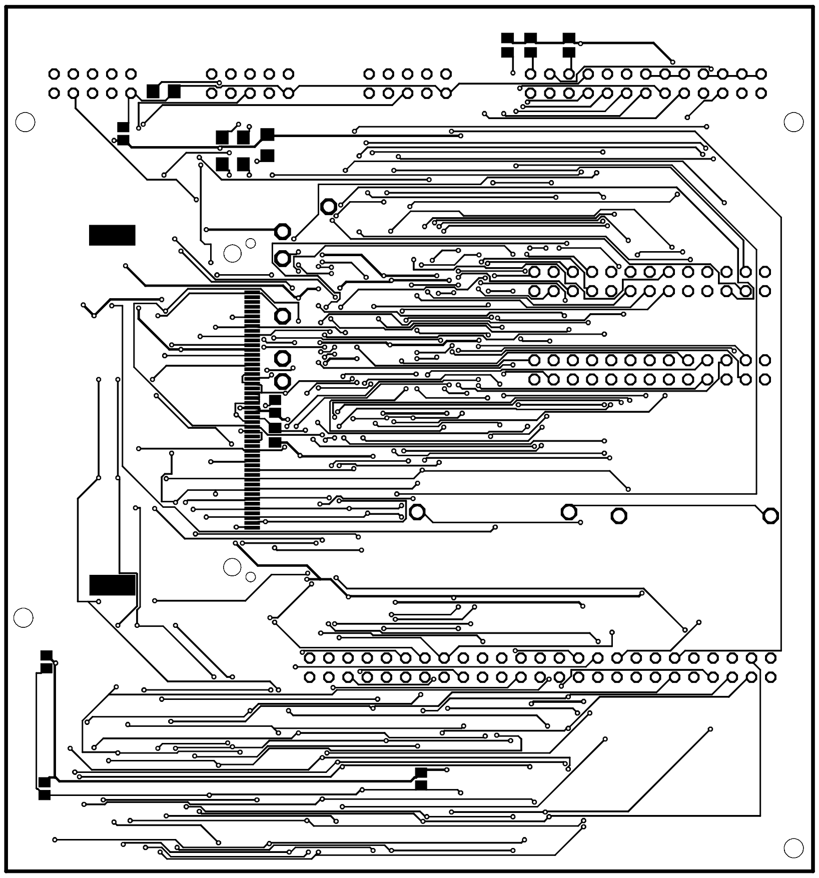
REV:

Date: 12/21/2002 11:38:58a

Sheet: 5/5


SU5
SU6
SU7
SU2
10
10
10
26
X1
IC14
IC13
IC11
IC12
TMS
VCCO
TCK
GND
TDI
TDO
IC8
SU4
SU3
1
26
C10
C9
2.5
3.3
CF
IC5
IC6
IC10
IC4
IC7
AMD
IC1
IC9
IC2
C8
C1
C2
IC3
BAT4
BAT3
Lithium 3V
Lithium 3V
SU1
50





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

package IOB_Config is

constant OSCFREQ: integer := 50000000;
constant NUMUARTS: integer := 3;

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

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");
constant PARPTBASE: DevID := O"66";

end IOB_Config;
```

```vhdl
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);
             --cpu_ma : in std_logic_vector(0 to 11);
             --cpu_ema : in std_logic_vector(0 to 2);
           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_autofd_n: out std_logic; -- hold high
           lpt_init: out std_logic;
               --lpt_select_out: out std_logic; -- hold low

           -- bits for custom applications
           iobits: inout std_logic_vector(0 to 47);

           -- special purpose
           reprogram: out std_logic
           );
end iob;

architecture RTL of iob is

constant OSCDIV: integer := OSCFREQ / (38400*16);

-- serial 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";

-- serial 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";

-- shared between all devices

signal reset: std_logic;
signal IOTact: boolean;
signal IOTdev: DevID;
signal IOTcmd: DevCmd;

signal clkdiv: integer range 0 to OSCDIV-1;
signal clkout: std_logic;

-- common between KL8 input and output

subtype UARTBitArray is unsigned(0 to NUMUARTS-1);
signal uiflag, uoflag, IE: UARTBitArray;

subtype BaudReg is unsigned(0 to 2);
type BaudRegs is array (0 to NUMUARTS-1) of BaudReg;
signal baud_sel: BaudRegs;

signal uartclk: unsigned(0 to 6);
signal clk16, clk1: UARTBitArray;

subtype Divider16 is unsigned(3 downto 0);
type Div16Array is array (0 to NUMUARTS-1) of Divider16;
signal div16: Div16Array;

signal raiseUIRQ, IRQ: std_logic;

-- LC8E signals

signal lpt_sel: boolean;
signal lpt_ready, lpt_acked,
       lpt_strobe_int,
       lpt_flag_1, lpt_flag_2: std_logic;

begin

-- dummy code to use all the declared inputs and outputs so they're mapped

iobits(0) <= cpu_sr_read_n;
iobits(1) <= cpu_sr_write_n;
iobits(2 to 47) <= (others => '0');
reprogram <= '1';

-- common IOT decoding

reset <= not cpu_ioclr_n;

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');

-- base clock divider for uarts

process (reset, clk)
begin
     if reset = '1' then
          clkdiv <= 0;
     elsif rising_edge(clk) then
          if clkdiv = OSCDIV-1 then
               clkdiv <= 0;
               clkout <= '1';
          else
               clkdiv <= clkdiv + 1;
               clkout <= '0';
          end if;
     end if;
end process;

-- common between the transmit and receive parts
---- interrupt management

raiseUIRQ <= '0' when ((IE and (uoflag or uiflag)) = 0) else '1';
cpu_intreq_n <= '0' when IRQ = '1' else 'Z';

process (reset, raiseUIRQ, cpu_intgrnt_n)
begin
     if (reset = '1') or (cpu_intgrnt_n = '0') then
          IRQ <= '0';
     elsif rising_edge(raiseUIRQ) then
          IRQ <= '1';
     end if;
end process;

---- baud rate generator

process (clkout)
begin
     if rising_edge(clkout) then
          uartclk <= uartclk + 1;
     end if;
end process;

KL8JA_common: for uidx in 0 to NUMUARTS-1 generate

     -- baud rate clock
     process (baud_sel, clkout, baud_sel)
     begin
          case to_integer(baud_sel(uidx)) is
               when 0 =>
                    clk16(uidx) <= std_logic(clkout);       -- 38400
               when others =>
                    clk16(uidx) <= uartclk(to_integer(baud_sel(uidx))-1); -- 19200..
          end case;
     end process;

     clk1(uidx) <= div16(uidx)(3);

     process (clk16(uidx))
     begin
          if falling_edge(clk16(uidx)) then
               div16(uidx) <= div16(uidx) + 1;
          end if;
     end process;

     process (reset, clk_write_n)
     begin
          if reset = '1' then
               baud_sel(uidx) <= "010"; -- 9600 baud
          elsif rising_edge(clk_write_n) then
               if IOTact and (IOTdev = (UARTIBASE(uidx))) and (IOTcmd = KIE) and
                  (dx(0 to 2) = not(dx(3 to 5))) then
                    baud_sel(uidx) <= unsigned(dx(0 to 2));
               end if;
          end if;
     end process;

end generate;

-- K8JA input section
KL8JA_receiver: for uidx in 0 to NUMUARTS-1 generate

     signal sel: boolean;
     signal data: std_logic_vector(7 downto 0);
     signal SE: std_logic;
     signal framing_error, overrun_error: std_logic;
     signal bit: integer range 0 to 11;
     signal inpv: std_logic_vector(0 to 2);
     signal inp: std_logic;
     signal sample: unsigned(0 to 3);
     signal rbf_1, rbf_2: std_logic;

begin

     sel <= IOTact and (IOTdev = (UARTIBASE(uidx)));

     -- manage the SE/IE flags
     process (reset, clk_write_n)
     begin
          if reset = '1' then
               SE <= '0';
               IE(uidx) <= '0';
          elsif rising_edge(clk_write_n) then
               if sel then
                    SE <= dx(10);
                    IE(uidx) <= dx(11);
               end if;
          end if;
     end process;

     -- c0/1/skip feedback
     process (sel, iotcmd, uiflag)
     begin
          if sel then
               -- c0/c1
               case IOTcmd is
                    when KCC | KRB =>
                         cpu_c0_n <= '0';
                    when others =>
                         cpu_c0_n <= '1';
               end case;

               case IOTcmd is
                    when KRS | KRB =>
                         cpu_c1_n <= '0';
                    when others =>
                         cpu_c1_n <= '1';
               end case;

               -- skip
               case IOTcmd is
                    when KSF =>
                         cpu_skip_n <= not uiflag(uidx);
                    when others =>
                         cpu_skip_n <= '1';
               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 (sel, cpu_read_n, data, se, framing_error, overrun_error)
     begin
          if sel and (cpu_read_n = '0') then
               dx(4 to 11) <= data;
               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 flag
     process (reset, clk)
     begin
          if reset = '1' then
               uiflag(uidx) <= '0';
               rbf_2 <= '0';
          elsif rising_edge(clk) then
               if rbf_1 /= rbf_2 then
                    uiflag(uidx) <= '1';
                    rbf_2 <= rbf_1;
               elsif clk_write_n = '0' then
                    if sel then
                         case IOTcmd is
```

```vhdl
                              when KCF | KCC | KRB =>
                                   uiflag(uidx) <= '0';
                              when others =>
                                   null;
                         end case;
                    end if;
               end if;
          end if;
     end process;

     -- receive data
     ---- 'vote' last three samples
     inp <= inpv(0) and inpv(1) and inpv(2);

     process (reset, clk16(uidx))
     begin
          if reset = '1' then
               bit <= 0;
               overrun_error <= '0';
               framing_error <= '0';
               rbf_1 <= '0';
          elsif rising_edge(clk16(uidx)) then
               inpv <= inpv(1 to 2) & rxd(uidx);
               if bit = 0 then
                    if inp = '0' then
                         bit <= 1;
                         sample <= div16(uidx) + "0111";
                         if uiflag(uidx) = '1' then
                              overrun_error <= '1';
                         end if;
                    end if;
               elsif bit = 10 then
                    bit <= 0;
                    rbf_1 <= not rbf_1;
                    if inp = '0' then
                         framing_error <= '1';
                    end if;
               else
                    if div16(uidx) = sample then
                         data <= data(7 downto 1) & inp;
                         bit <= bit + 1;
                    end if;
               end if;
          end if;
     end process;

end generate;

-- K8JA output section
KL8JA_transmitter: for uidx in 0 to NUMUARTS-1 generate

     signal sel, write: boolean;
     signal print: std_logic;
     signal setflag, clrflag: std_logic;
     signal tbre_1, tbre_2: std_logic;
     signal sr: std_logic_vector(0 to 9);
     signal bit: integer range 0 to 10;

begin
```

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

     print <= '1' when write and ((IOTcmd = TPC) or (IOTcmd = TLS)) else '0';
     setflag <= '1' when write and (IOTcmd = TFL) else '0';
     clrflag <= '1' when write and ((IOTcmd = TCF) or (IOTcmd = TLS)) else '0';

     -- flag management
     process (reset, clk, setflag, clrflag)
     begin
          if (reset = '1') or (clrflag = '1') then
               uoflag(uidx) <= '0';
               tbre_2 <= '0';
          elsif setflag = '1' then
               uoflag(uidx) <= '1';
          elsif rising_edge(clk) then
               if (tbre_1 /= tbre_2) then
                    if tbre_1 = '1' then
                         uoflag(uidx) <= '1';
                    end if;
                    tbre_2 <= tbre_1;
               end if;
          end if;
     end process;

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

               -- skip
               case IOTcmd is
                    when TSF =>
                         cpu_skip_n <= not uoflag(uidx);
                    when TSK =>
                         cpu_skip_n <= not (uoflag(uidx) or uiflag(uidx));
                    when others =>
                         cpu_skip_n <= '1';
               end case;
          else
               cpu_c0_n <= 'Z';
               cpu_c1_n <= 'Z';
               cpu_skip_n <= 'Z';
          end if;
     end process;

     -- serial output
     txd(uidx) <= sr(0);
     tbre_1 <= '1' when (bit = 9) else '0';

     process (reset, clk1(uidx))
     begin
          if reset = '1' then
               sr(0) <= '1';
               bit <= 9;
          elsif rising_edge(clk1(uidx)) then
               if bit = 9 then
```

```
                    if print = '1' then
                         sr <= '0' & dx(4 to 11) & '1';
                         bit <= 0;
                    end if;
               else
                    bit <= bit + 1;
                    sr <= sr(1 to 9) & '1';
               end if;
          end if;
     end process;

end generate;

-- LC8E printer interface with Centronics output

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

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

lpt_sel <= (clk_lxdar_n = '0') and (IOTdev = PARPTBASE);
lpt_strobe_int <= '1' when lpt_sel and (clk_write_n = '0') and
                    ((IOTcmd = TPC) or (IOTcmd = TLS)) else '0';

-- maintain 'ack' flag

process (reset, lpt_ack, lpt_strobe_int)
begin
     if lpt_strobe_int = '1' then
          lpt_acked <= '0';
     elsif reset = '1' then
          lpt_acked <= '1';
     elsif rising_edge(lpt_ack) then
          lpt_acked <= '1';
     end if;
end process;

--maintain printer flag

process (reset, lpt_ready, lpt_flag_2)
begin
     if reset = '1' then
          lpt_flag_1 <= '0';
     elsif rising_edge(lpt_ready) then
          lpt_flag_1 <= not lpt_flag_2;
     end if;
end process;

-- handle c0/c1/skip

process (lpt_sel, IOTcmd, lpt_flag_1, lpt_flag_2)
begin
     if lpt_sel then
          -- c0/c1
          cpu_c0_n <= '1';
          cpu_c1_n <= '1';
          -- skip
          case IOTcmd is
               when TSF | TSK =>
                    cpu_skip_n <= not (lpt_flag_1 xor lpt_flag_2);
               when others =>
                    cpu_skip_n <= '1';
          end case;
     else
          cpu_c0_n <= 'Z';
          cpu_c1_n <= 'Z';
          cpu_skip_n <= 'Z';
     end if;
end process;

-- handle IOTs

process (reset, clk_write_n, lpt_flag_1)
begin
     if reset = '1' then
          lpt_flag_2 <= '0';
     elsif falling_edge(clk_write_n) then
          if lpt_sel then
               case IOTcmd is

                    when TCF =>
                         lpt_flag_2 <= lpt_flag_1;

                    when TFL | TLS =>
                         lpt_flag_2 <= not lpt_flag_1;

                    when others =>
                         null;

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

end RTL;
```

```
#CONFIG PROHIBIT = "P38";
CONFIG PROHIBIT = "P68";

# 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";
```

1. make exact measurements of SBC6120, make sure IOB dimensions,
connectors and holes are in the right spots.

2. add decoupling around FPGA

3. rework VHDL for KL8J - double buffering, bit formats

4. does KS8E do interrupts?

5. double-check layout/position of SBC connector