XSA Flash Programming and SpartanII Configuration

March 17, 2004 (Version 1.1)

Application Note by D. Vanden Bout

Summary

This application note describes the circuits that let the XC9572XL CPLD program the Flash on the XSA Board and then configure the SpartanII FPGA with the data stored in the Flash.

Using Flash with the XSA Board

The 2 Mb Flash chip on the XSA Board can be used to store configurations for the SpartanII FPGA. When used in this way, there are three steps used to set up the Flash:

1. Configure the XC9572XL CPLD with a programming circuit that connects the Flash to the parallel port.

2. Program the Flash by passing the SpartanII configuration bitstream through the parallel port.

3. Load the CPLD with a configuration circuit that will, upon power-up of the XSA Board, load the SpartanII with the bitstream stored in the Flash.

The Flash Programming Interface

Listing 1 and Listing 2 show the VHDL code and pin assignments for the CPLD circuit that connects the Flash to the parallel port. This circuit is simply an interface that allows the PC to read and write the Flash using only four data bits and two control signals. The PC uses this simple interface to control the higher-level Flash programming functions such as erasing the Flash sectors before they are programmed with new data.

The Flash programming circuit performs the following functions:

- It collects six successive nybbles from the parallel port and concatenates these into a 24-bit Flash address.

- It collects two successive nybbles from the parallel port and concatenates these into a byte of Flash data.

- It writes the data into the Flash at the given address and then loops back to await the arrival of another set of address and data nybbles.

- While gathering the nybbles for an address, it also reads the byte of Flash data from the previous loop and passes it to the parallel port in segments of 3, 3 and 2 bits each.

How the VHDL implements these functions is described below.

Lines 10-32 of Listing 1 define the interface for the circuit. It uses six data pins of the parallel port: four for passing data and address nybbles, one for a synchronizing clock to drive the state machine in the CPLD, and one as a reset for the state machine. Three status pins of the parallel port are used to send Flash data back to the PC and to report the current state of the CPLD state machine. The CPLD also interfaces to the address, data, and control pins of the Flash chip and the PROGRAM pin of the SpartanII FPGA.

The eight states of the Flash programming state machine are defined on lines 43-53. Six states are used to gather the 24-bit Flash address in nybble chunks, and then two more states are used to collect the data byte that will be written to the Flash.

Line 63 makes the CPLD pull the SpartanII PROGRAM pin low so it stays in its unconfigured state. This tristates the pins of the SpartanII so it can't interfere with the programming of the Flash chip. Line 64 makes sure the reset of the Flash chip is released so it can be programmed.

Lines 65-67 just rename the parallel port data pins with more understandable names that reflect their underlying functions.

The main process for the Flash programming state machine begins on line 71. Lines 74-81 just set the default values for the outputs from the state machine.

Lines 87-132 implement the six states that concatenate nybbles from the parallel port into a 24-bit address. During each of these states, the nybble from the parallel port is placed into the appropriate slot in the address register. The current state is also reported back to the PC through the parallel port status lines in states load_a20, load_a4, and load_a0. The Flash programming code in the PC uses this information to make sure it is in sync with the state machine.

However, during states load_a16, load_a12 and load_a8 the status lines are used to carry the segments of a data byte from the Flash back to the PC over the parallel port status lines. The location of this data in the Flash is stored in the next_addr register in state load_a20 (line 92). This address appears on the Flash address lines at the start of state load_a16 and persists until the next_addr register is written to again. During states load_a16, load_a12 and load_a8, the Flash chip-enable and output-enable lines are forced low and the upper three bits, middle three bits and lowest two bits of the Flash data byte at the given address are passed through the parallel port (lines 103, 113 and 124, respectively). The Flash programming code in the PC gathers these nybbles and assembles the byte of Flash data.

Lines 133-152 implement the two states that concatenate two data nybbles into a byte of data that is written into the Flash at the address loaded during the previous six states. The actual write occurs in the second half of state load_d0 when the clock is low. this gives the address time to settle from the previous cycle before the write occurs. When the clock goes high to end the write pulse, the state machine transfers to state load_a20. Note that when state load_a20 is first entered, line 90 ensures that the Flash data lines are still carrying the same value as they were in state load_d0. This ensures the data hold time for the Flash.

The process on lines 168-182 updates the state, address, and data registers on the rising clock edge. A reset from the parallel port will clear the data register and send the state machine to the load_a20 state to start another Flash address cycle. Note that the reset will not clear the address register. This allows the PC to read the Flash without writing it by forcing a reset after state load_a0. When the state machine returns to the load_a16, load_a12 and load_a8 states, the PC can read the Flash data at the address that was loaded during the previous loop. This would not be possible if the address register was cleared by a reset.

The process on lines 186-193 updates the register that drives the Flash address lines. (The connection of this register to the Flash address lines is done on line 195.) The address lines change on the falling clock edge. This ensures the address lines are stable before any potential write operation is initiated on the next rising clock edge.

The SpartanII-Flash Configuration Circuit

Listing 3 and Listing 4 show the VHDL code and pin assignments for the CPLD circuit that configures the SpartanII FPGA with the bitstream programmed into the Flash. This circuit is simply increments an address counter which reads out the next byte of Flash data and strobes it into the SpartanII. When the SpartanII signals that it is completely configured, then the CPLD ceases operations. How the VHDL implements these functions is described below.

Lines 10-37 of Listing 3 define the interface for the circuit. It uses the programmable oscillator on the XSA Board as the main clock. The CPLD also interfaces to the address and control pins of the Flash chip so it can fetch the bytes of the SpartanII configuration bitstream. (It doesn't need to access the Flash data pins since these are already directly connected to the configuration data inputs of the SpartanII chip on the XSA Board.) The CPLD stuffs the bitstream into the SpartanII using the configuration control pins.

Line 55 merely renames the S2_dout pin of the SpartanII to S2_busy since the SpartanII will use this signal to indicate when it is busy storing a byte of configuration data. Line 58 causes the CPLD to output the code onto the mode pins of the SpartanII that place it in the Slave Parallel configuration mode. In this mode, the SpartanII chip accepts bytes of configuration data on the rising edge of the configuration clock as long as its chip-select and write-enable are active.

Lines 62-65 set the Flash control pins so it can output the data bytes of the SpartanII bitstream. The CPLD releases its control of these pins when the SpartanII signals that the configuration process is done (S2_done=HI).

The Flash chip has an access time of 90 ns while the XSA Board oscillator can run as fast as 100 MHz.

Lines 70-77 implement a counter that divides the oscillator frequency by 16 and uses the slower clock to drive the configuration of the SpartanII.

After power is applied to the XSA Board, the SpartanII FPGA needs some time to settle before configuration starts. Lines 81-91 create a power-on timeout counter and a reset signal that is active until the counter reaches zero. Then the reset is removed and the configuration starts. Line 7 of Listing 4 ensures that the timeout counter in the CPLD is initialized to the 11…1 state upon power-up of the XSA Board.

Lines 96-98 use the power-on reset to lower the PROGRAM pin of the SpartanII when the board powers up. The PROGRAM signal is pulled high after the power-on timeout expires and the SpartanII configuration starts.

Lines 102-109 select the SpartanII chip for configuration when the PROGRAM pin is high and the SpartanII is not indicating a configuration error by pulling its INIT pin low. The internal chip-select signal is inverted and drives the SpartanII chip-select and write-enable pins on lines 114-115. The CPLD releases control of these pins when the configuration process is done.

The process on lines 122-131 controls the fetching of configuration data from the Flash. The Flash address register is set to zero while the SpartanII is held in its reset state with the PROGRAM pin pulled low. After the PROGRAM pin goes high and configuration starts, the Flash address is incremented on every clock cycle as long as the SpartanII chip is selected and the SpartanII is not signaling a configuration error (INIT=HI) or that it is busy with a previous byte of configuration data (BUSY=LO). The value in the address counter is passed to the Flash chip address pins on line 135.

After the SpartanII is configured, line 138 passes the clock from the programmable oscillator to the SpartanII for use as a clock waveform by whatever logic is now active in the FPGA.

Listing 1: VHDL code for the Flash programming interface.

```
--
-- XC9500 CPLD design which controls the loading of the XSA Flash
-- with data from the PC parallel port.
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity updnload is
   generic
   (
      ADDR_LEN: positive := 18      -- number of address bits for XSA FLASH
   );
   port
   (
      -- parallel port data and status pins
      ppd: in std_logic_vector(5 downto 0);   -- data nybble, clk, reset from par. port
      pps: out std_logic_vector(5 downto 3);  -- status nybble to parallel port

      -- Flash data, address, and control pins
      fd: inout std_logic_vector(7 downto 0); -- data bus to XSA FLASH
      fa: out std_logic_vector(ADDR_LEN-1 downto 0); -- address bus to XSA FLASH
      fceb: out std_logic;                    -- chip-enable for XSA FLASH
      foeb: out std_logic;                    -- output-enable for XSA FLASH
      fweb: out std_logic;                    -- write-enable for XSA FLASH
      frstb: out std_logic;                   -- reset for XSA FLASH

      -- spartan2 FPGA pins
      S2_progb: out std_logic                 -- spartan2 PROGRAM pin
   );
end updnload;


architecture updnload_arch of updnload is

   constant LO : std_logic := '0';
   constant HI : std_logic := '1';
   constant NO : std_logic := '0';
   constant YES: std_logic := '1';

   -- states for the state machine that programs the Flash
   type flash_state_type is
   (
      load_a20,     -- load address nybble A23-A20
      load_a16,     -- load address nybble A19-A16, read data nybble D7-D5
      load_a12,     -- load address nybble A12-A15, read data nybble D4-D2
      load_a8,      -- load address nybble A8-A11,  read data nybble D1-D0
      load_a4,      -- load address nybble A4-A7
      load_a0,      -- load address nybble A0-A4
      load_d4,      -- load data nybble D4-D7
      load_d0       -- load data nybble D0-D3
   );

   signal flash_state, next_flash_state: flash_state_type;
   signal clk, reset: std_logic;
   signal nybble: std_logic_vector(3 downto 0);
   signal addr, next_addr: std_logic_vector(ADDR_LEN-1 downto 0);
   signal addr_reg, next_addr_reg: std_logic_vector(23 downto 0);
   signal data_reg, next_data_reg: std_logic_vector(3 downto 0);

begin
   S2_progb<= LO;             -- keep spartan2 in reset state so it doesn't interfere
```

```
    frstb  <= HI;              -- remove Flash reset so the chip is enabled
    reset  <= not ppd(0);      -- Flash prog. state machine reset from D0 of parallel port data
    clk<= not ppd(1);          -- state machine clock from D1 of parallel port data
    nybble <= ppd(5 downto 2); -- Flash data nybble from parallel port data

    -- this process directs the state transitions of the Flash programming
    -- state machine and sets the control outputs for each state
    process(addr,addr_reg,fd,data_reg,nybble,ppd,flash_state)
    begin
       -- the following statements set the default values for the outputs
       foeb <= HI;             -- Flash chip data pin drivers disabled
       fceb <= HI;             -- Flash chip disabled
       fweb <= HI;             -- no write operations to Flash chip
       fd   <= (others=>'Z');  -- no data driven into the Flash chip
       pps <= "111";           -- illegal state reported on status pins
       next_addr <= addr;      -- Flash address does not change
       next_addr_reg <= addr_reg;
       next_data_reg <= data_reg; -- Flash data does not change

       -- now use the current state to determine the outputs and the
       -- next state for the Flash programming state machine
       case flash_state is

          when load_a20 =>
             -- load Flash address bits A23-A20 and output the
             -- last complete Flash address that was assembled previously
             fd <= data_reg & nybble;      -- complete data byte written to Flash
             next_addr_reg(23 downto 20) <= nybble;         -- store A23-A20
             next_addr <= addr_reg(ADDR_LEN-1 downto 0);    -- output last addr
             pps <= "000";                 -- report current state through parallel port
             next_flash_state <= load_a16; -- go to next state

          when load_a16 =>
             -- load Flash address bits A19-A16, read the contents
             -- from the previous Flash address, and send the upper
             -- 3 bits of the Flash data back through the parallel port
             next_addr_reg(19 downto 16) <= nybble;  -- store A19-A16
             fceb <= LO;                   -- enable Flash
             foeb <= LO;                   -- read Flash
             pps <= fd(7 downto 5);        -- send upper 3 data bits back to PC
             next_flash_state <= load_a12; -- go to next state

          when load_a12 =>
             -- load Flash address bits A15-A12, read the contents
             -- from the previous Flash address, and send the middle
             -- three bits of the Flash data back through the parallel port
             next_addr_reg(15 downto 12) <= nybble;  -- store A15-A12
             fceb <= LO;                   -- enable Flash
             foeb <= LO;                   -- read Flash
             pps <= fd(4 downto 2);        -- send middle 3 data bits back to PC
             next_flash_state <= load_a8;  -- go to next state

          when load_a8  =>
             -- load Flash address bits A11-A8
             -- load Flash address bits A11-A8, read the contents
             -- from the previous Flash address, and send the lowest
             -- two bits of the Flash data back through the parallel port
             next_addr_reg(11 downto 8)  <= nybble;  -- store A11-A8
             fceb <= LO;                   -- enable Flash
             foeb <= LO;                   -- read Flash
             pps <= "0" & fd(1 downto 0);  -- send lowest 2 data bits back to PC
             next_flash_state <= load_a4;  -- go to next state

          when load_a4  =>
             -- load Flash address bits A7-A4
```

```
            next_addr_reg(7 downto 4)   <= nybble;  -- store A7-A4
            pps <= "001";                   -- report current state through parallel port
            next_flash_state <= load_a0;  -- go to next state

         when load_a0  =>
            -- load Flash address bits A3-A0
            next_addr_reg(3 downto 0)   <= nybble;  -- store A3-A0
            pps <= "010";                   -- report current state through parallel port
            next_flash_state <= load_d4;  -- go to next state

         when load_d4  =>
            -- output the assembled address to the Flash and load the
            -- upper nybble of data that will be written to the Flash
            next_addr <= addr_reg(ADDR_LEN-1 downto 0); -- output complete addr
            fceb <= LO;                     -- enable the Flash
            next_data_reg <= nybble;        -- store upper data nybble from par port
            fd <= data_reg & nybble;        -- output data to the Flash
            pps <= "011";                   -- report current state through parallel port
            next_flash_state <= load_d0;  -- go to the next state

         when load_d0  =>
            -- now get the lower nybble of data from the parallel port
            -- and write the complete byte to the Flash during the
            -- second half of the clock phase
            fceb <= LO;                     -- keep the Flash enabled
            fweb <= clk;                    -- write goes low during second half of clock cycle
            fd <= data_reg & nybble;        -- complete data byte written to Flash
            pps <= "100";                   -- report current state through parallel port
            next_flash_state <= load_a20; -- go back to the start

         when others =>
            -- return the state machine to the initial state if it
            -- ever gets into an erroneous state
            next_flash_state <= load_a20;

      end case;
   end process;

   -- update the programming machine state and other registers
   process(reset,clk)
   begin
      if (reset=HI) then
         -- asynchronous reset sets state machine to initial state
         -- and clears data register
         flash_state <= load_a20;
         data_reg <= (others=>'0');

      elsif (clk'event and clk=HI) then
         -- update the machine state and other registers on rising clock edge
         flash_state <= next_flash_state;
         addr_reg <= next_addr_reg;
         data_reg <= next_data_reg;
      end if;
   end process;

   -- output Flash addresses one-half cycle early.  This gives the Flash
   -- address time to settle and activate the appropriate location for writing.
   process(clk)
   begin
      -- change Flash address during the second half of the clock cycle
      -- before the machine changes states
      if (clk'event and clk=LO) then
         addr <= next_addr;
      end if;
   end process;
```

```

    fa <= addr;  -- output address to the Flash chip

end updnload_arch;
```

Listing 2: Pin assignments for the Flash programming interface.

```
#
# pin assignments for the XC9572XL CPLD chip on the XSA Board
#

# Spartan2 FPGA connections to CPLD
# net S2_clk      loc=p42;
# net S2_tck      loc=p13;
# net S2_dout     loc=p18;
# net S2_din      loc=p2;
# net S2_wrb      loc=p19;
# net S2_csb      loc=p15;
# net S2_initb    loc=p38;
# net S2_done     loc=p40;
net S2_progb      loc=p39;
# net S2_cclk     loc=p16;
# net S2_m<0>     loc=p36;
# net S2_d<0>     loc=p2;
# net S2_d<1>     loc=p4;
# net S2_d<2>     loc=p5;
# net S2_d<3>     loc=p6;
# net S2_d<4>     loc=p7;
# net S2_d<5>     loc=p8;
# net S2_d<6>     loc=p9;
# net S2_d<7>     loc=p10;

# Flash RAM
net fd<0>         loc=p2;
net fd<1>         loc=p4;
net fd<2>         loc=p5;
net fd<3>         loc=p6;
net fd<4>         loc=p7;
net fd<5>         loc=p8;
net fd<6>         loc=p9;
net fd<7>         loc=p10;
net fa<0>         loc=p1;
net fa<1>         loc=p64;
net fa<2>         loc=p63;
net fa<3>         loc=p62;
net fa<4>         loc=p61;
net fa<5>         loc=p60;
net fa<6>         loc=p59;
net fa<7>         loc=p58;
net fa<8>         loc=p45;
net fa<9>         loc=p44;
net fa<10>        loc=p57;
net fa<11>        loc=p43;
net fa<12>        loc=p56;
net fa<13>        loc=p46;
net fa<14>        loc=p47;
net fa<15>        loc=p52;
net fa<16>        loc=p51;
net fa<17>        loc=p48;
net frstb         loc=p50; # Flash reset
net foeb          loc=p12; # Flash output-enable
net fweb          loc=p49; # Flash write-enable
net fceb    loc=p11; # Flash chip-enable
```

```
# DIP and pushbutton switches
# net dipsw<1>    loc=p47;
# net dipsw<2>    loc=p52;
# net dipsw<3>    loc=p51;
# net dipsw<4>    loc=p48;

# 7-segment LEDs
# net s<0>        loc=p10;
# net s<1>        loc=p2;
# net s<2>        loc=p9;
# net s<3>        loc=p8;
# net s<4>        loc=p5;
# net s<5>        loc=p7;
# net s<6>        loc=p6;
# net dp          loc=p4;

# programmable oscillator
# net clk         loc=p17;

# parallel port
net ppd<0>        loc=p33;
net ppd<1>        loc=p32;
net ppd<2>        loc=p31;
net ppd<3>        loc=p27;
net ppd<4>        loc=p25;
net ppd<5>        loc=p24;
# net ppd<6>      loc=p23;
# net ppd<7>      loc=p22;
net pps<3>        loc=p34;
net pps<4>        loc=p20;
net pps<5>        loc=p35;
```

Listing 3: VHDL code for the SpartanII-Flash configuration circuit.

```
--
-- XC9500 CPLD design which controls the configuration of the XSA Spartan2
-- with data from the Flash chip.
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity config is
   generic
   (
      ADDR_LEN: positive := 18         -- number of Flash address bits
   );
   port
   (
      clk    : in std_logic;            -- clock from DS1075 prog. osc.

      -- Flash address and control pins
      fa     : out std_logic_vector(ADDR_LEN-1 downto 0);   -- Flash address
      fceb   : out std_logic;           -- Flash chip-enable
      foeb   : out std_logic;           -- Flash output-enable
      fweb   : out std_logic;           -- Flash write-enable
      frstb  : out std_logic;           -- Flash reset

      -- Spartan2 configuration pins
      S2_clk    : out std_logic;        -- Spartan2 global clock input
      S2_progb  : out std_logic;        -- Spartan2 PROGRAM pin
      S2_cclk      : out std_logic;     -- Spartan2 config clock
      S2_csb    : out std_logic;        -- Spartan2 config chip-select
      S2_wrb    : out std_logic;        -- Spartan2 config write-enable
      S2_initb  : in std_logic;         -- Spartan2 config init status
      S2_dout      : in std_logic;      -- Spartan2 config busy status
      S2_done      : in std_logic;      -- Spartan2 config done status
      S2_m         : out std_logic_vector(0 downto 0)   -- Spartan2 config. mode pins
   );
end config;

architecture config_arch of config is
   constant LO      : std_logic := '0';
   constant HI      : std_logic := '1';
   constant FLOAT   : std_logic := 'Z';

   signal clk_cnt          : std_logic_vector(3 downto 0);
   signal cclk             : std_logic;
   signal programb, cs     : std_logic;
   signal addr, next_addr  : std_logic_vector(ADDR_LEN-1 downto 0);
   signal poweron_reset    : std_logic;
   signal poweron_cnt      : std_logic_vector(19 downto 0);
   signal S2_busy          : std_logic;
   signal button_progb     : std_logic;
   component pullup port(O: out std_logic); end component;
begin

   S2_busy <= S2_dout;     -- give this signal a better name

   -- set Spartan2 mode to Slave Parallel so it can be configured from Flash
   S2_m      <= "0";

   -- Flash is enabled for reading while Spartan2 is not yet configured
   -- and then the Flash pins float when configuration is done
   foeb      <= LO when (S2_done=LO) else FLOAT;
```

```
    fceb      <= LO when (S2_done=LO) else FLOAT;
    fweb      <= HI when (S2_done=LO) else FLOAT;  -- disable Flash writes
    frstb     <= HI;                               -- remove Flash reset

    -- generate configuration clock for Spartan2 from the XSA clock.
    -- The XSA clock could be as much as 100 MHz, so divide by 16
    -- to exceed the access time of the Flash.
    process(clk)
    begin
       if(clk'event and clk=HI) then
          clk_cnt <= clk_cnt + 1;
       end if;
    end process;
    cclk   <= clk_cnt(3);   -- internal configuration clock
    S2_cclk   <= cclk;      -- also send config. clock to Spartan2

    -- Apply reset when the power to the XSA Board is first applied.
    -- Remove the power-on reset after the counter reaches 0.
    process(cclk)
    begin
       if(cclk'event and cclk=HI) then
          if(poweron_cnt = 0) then
             poweron_reset <= LO;-- remove reset when timeout expires
          else
             poweron_cnt <= poweron_cnt - 1;
             poweron_reset <= HI;
          end if;
       end if;
    end process;

    -- initiate Spartan2 configuration by lowering the /PROGRAM pin
    -- during the initial power-on reset and then raising it when
    -- the power-on timeout expires and the manual program control is high
    programb <= not(poweron_reset);
    u0: pullup port map(O=>S2_progb);-- place a pullup on the Spartan2 PROGRAM pin
    S2_progb <= LO when programb=LO else 'Z';   -- programming pulse comes from parallel port

    -- Select the Spartan2 for configuration as long as the /PROGRAM pin
    -- is not held low and the INIT pin is not low.
    process(cclk,programb)
    begin
       if(programb = LO) then
          cs <= LO;
       elsif(cclk'event and cclk=HI) then
          cs <= S2_initb;
       end if;
    end process;

    -- Select the Spartan2 for configuration by lowering its chip-select
    -- and write inputs when the internal chip-select is high.  Then
    -- float these pins after the Spartan2 configuration is done.
    S2_csb <= not(cs)   when (S2_done=LO) else FLOAT;
    S2_wrb <= not(cs)   when (S2_done=LO) else FLOAT;

    -- increment the Flash address so the next byte of configuration
    -- data is presented to the Spartan2.  Stop incrementing if the
    -- Spartan2 is not selected, signals a config. error (INIT=0), or
    -- is busy.  Reset the address counter to zero whenever the
    -- /PROGRAM pin goes low and a new configuration sequence begins.
    process(cclk)
    begin
       if(cclk'event and cclk=HI) then
          if((cs=HI) and (S2_initb=HI) and (S2_busy=LO)) then
             addr <= addr + 1;
          elsif(programb = LO) then
```

```
            addr <= (others=>LO);
         end if;
      end if;
   end process;

   -- pass the Flash address out to the Flash chip.  Float the address
   -- lines once configuration is done.
   fa <= addr when (S2_done=LO) else (others=>FLOAT);

   -- pass the clock from the DS1075 to the Spartan2 after it is configured
   S2_clk <= clk when (S2_done=HI) else FLOAT;

end config_arch;
```

Listing 4: Pin assignments for the SpartanII-Flash configuration circuit.

```
#
# pin assignments for the XC9572XL CPLD chip on the XSA Board
#

# set all the bits in the initial state of the power-on
# counter so we get the maximum timeout interval
inst poweron_cnt_reg<*> INIT=S;

# Spartan2 FPGA connections to CPLD
net S2_clk       loc=p42;
# net S2_tck     loc=p13;
net S2_dout      loc=p18;
# net S2_din     loc=p2;
net S2_wrb       loc=p19;
net S2_csb       loc=p15;
net S2_initb     loc=p38;
net S2_done      loc=p40;
net S2_progb     loc=p39;
net S2_cclk      loc=p16;
net S2_m<0>      loc=p36;
# net S2_d<0>    loc=p2;
# net S2_d<1>    loc=p4;
# net S2_d<2>    loc=p5;
# net S2_d<3>    loc=p6;
# net S2_d<4>    loc=p7;
# net S2_d<5>    loc=p8;
# net S2_d<6>    loc=p9;
# net S2_d<7>    loc=p10;

# Flash RAM
# net fd<0>      loc=p2;
# net fd<1>      loc=p4;
# net fd<2>      loc=p5;
# net fd<3>      loc=p6;
# net fd<4>      loc=p7;
# net fd<5>      loc=p8;
# net fd<6>      loc=p9;
# net fd<7>      loc=p10;
net fa<0>        loc=p1;
net fa<1>        loc=p64;
net fa<2>        loc=p63;
net fa<3>        loc=p62;
net fa<4>        loc=p61;
net fa<5>        loc=p60;
net fa<6>        loc=p59;
net fa<7>        loc=p58;
net fa<8>        loc=p45;
net fa<9>        loc=p44;
net fa<10>       loc=p57;
net fa<11>       loc=p43;
net fa<12>       loc=p56;
net fa<13>       loc=p46;
net fa<14>       loc=p47;
net fa<15>       loc=p52;
net fa<16>       loc=p51;
net fa<17>       loc=p48;
net frstb        loc=p50;  # Flash reset
net foeb         loc=p12;  # Flash output-enable
net fweb         loc=p49;  # Flash write-enable
net fceb         loc=p11;  # Flash chip-enable

# DIP and pushbutton switches
# net dipsw<1>   loc=p47;
```

```
# net dipsw<2>   loc=p52;
# net dipsw<3>   loc=p51;
# net dipsw<4>   loc=p48;

# 7-segment LEDs
# net s<0>       loc=p10;
# net s<1>       loc=p2;
# net s<2>       loc=p9;
# net s<3>       loc=p8;
# net s<4>       loc=p5;
# net s<5>       loc=p7;
# net s<6>       loc=p6;
# net dp         loc=p4;

# programmable oscillator
net clk          loc=p17;

# parallel port
# net ppd<0>      loc=p33;
# net ppd<1>      loc=p32;
# net ppd<2>     loc=p31;
# net ppd<3>     loc=p27;
# net ppd<4>     loc=p25;
# net ppd<5>     loc=p24;
# net ppd<6>     loc=p23;
# net ppd<7>     loc=p22;
# net pps<3>     loc=p34;
# net pps<4>     loc=p20;
# net pps<5>     loc=p35;
```