"Switches and Networks in VHDL - A Class Example"

Monday, Tuesday - Sept 16,17, 2013

Prof. Ted Szymanski
Dept. of ECE
McMaster University

Switches and Networks

- Switches and Interconnection networks are used in many computing systems:
- Single-chip multiprocessors use a 'Network-on-Chip' (NoC)
- Cloud data-centers use networks of 10Gbit Ethernet switches spanning tens to hundreds of meters
- Internet routers use basic switches with 100s of Gbit/sec bandwidth, with a control processor to run Internet protocols
- Let take a look at basic a basic switch

Cloud Datacenter (see class textbook)

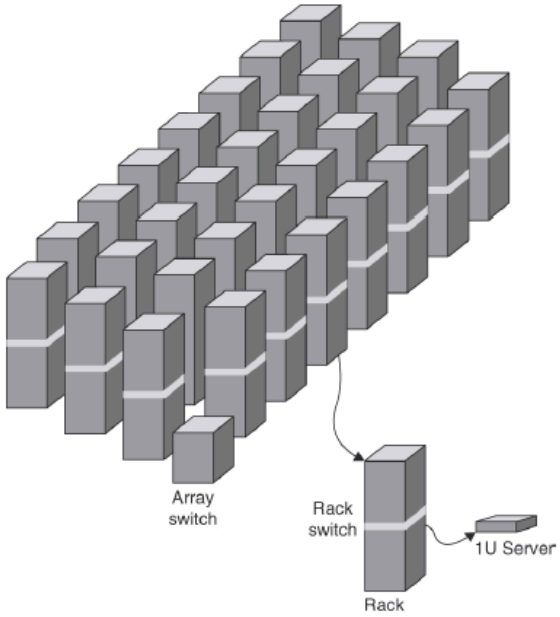
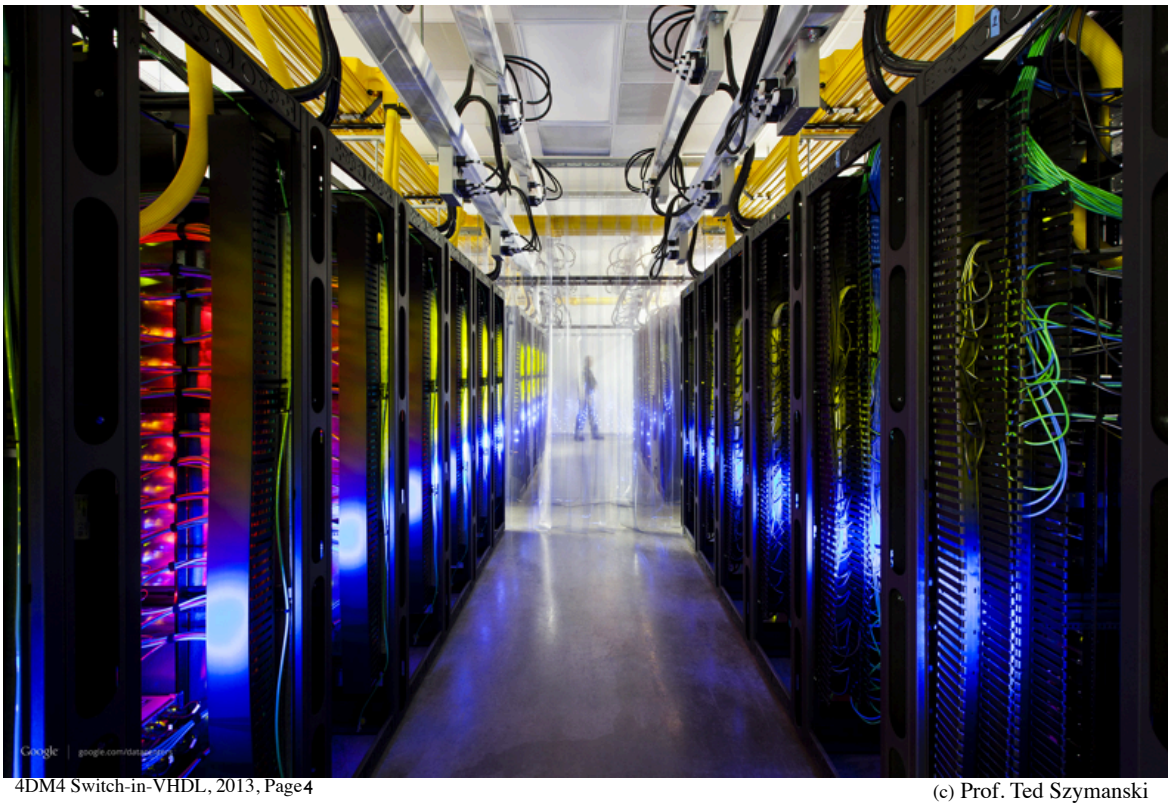


Figure 6.5 Hierarchy of switches in a WSC. (Based on Figure 1.2 of Barroso and Hölzle [2009].)

Pictures: A Google DataCenter



Network-on-Chip for Multiprocessors

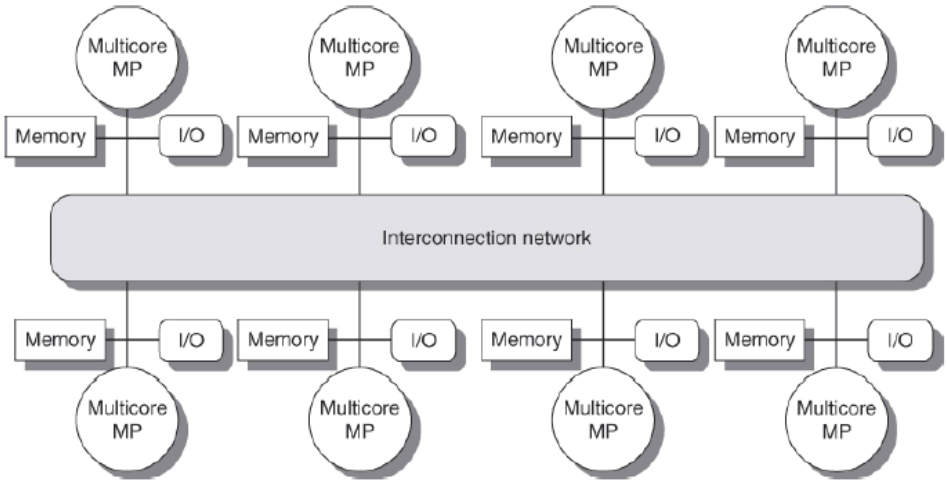


Figure 5.2 The basic architecture of a distributed-memory multiprocessor in 2011 typically consists of a multicore multiprocessor chip with memory and possibly I/O attached and an interface to an interconnection network that connects all the nodes. Each processor core shares the entire memory, although the access time to the lock memory attached to the core's chip will be much faster than the access time to remote memories.

Some Supercomputer Network Topologies

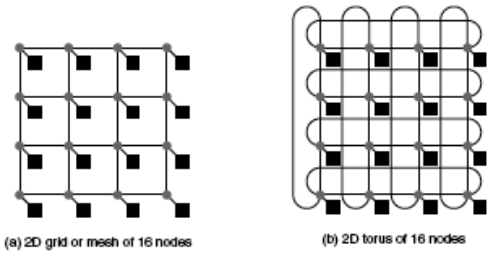
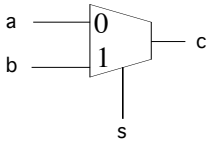


Figure F.14 Direct network topologies that have appeared in commercial systems, mostly supercomputers. The shaded circles represent switches, and the black squares represent end node devices. Switches have many bidirectional network links, but at least one link goes to the end node device. These basic topologies can be supplemented with extra links to improve performance and reliability. For example, connecting the switches on the periphery of the 2D mesh, shown in (a), using the unused ports on each switch forms a 2D torus, shown in (b). The hypercube topology, shown in (c) is an n-dimensional interconnect for 2^n nodes, requiring $n + 1$ ports per switch: one for the n nearest neighbor nodes and one for the end node device.

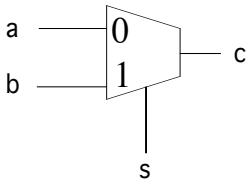
Recall: 2-to-1 Multiplexer - Behavioural Entity

```
Entity  mux21 is
      Port ( a, b , s :   in   STD_LOGIC;
                  c     :  out      STD_LOGIC ) ;
End  entity  mux21;

-- an architecture definition without using a process statement
Architecture   mux21_arch1  of  mux21   is
begin
      c <= a  when  s = '0'  else   b;
end  mux21_arch;

-- using a process statement to generate combinational logic
Architecture   mux21_arch2  of   mux21  is
begin
      MUX : Process (a, b,  s)
      begin
            if  (s  = '1')  then
                        c <= b;
            else
                        c <= a;
            end
      end process  MUX;
end  architecture  mux21_arch2;
```



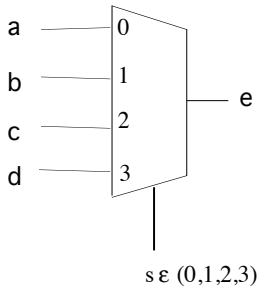
VHDL rule: 'If-then-else' statement must be in a process.


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.all;

entity mux21 is
  generic(width : integer := 16);
  port(
    a       : in std_logic_vector(width-1 downto 0);
    b       : in std_logic_vector(width-1 downto 0);
    sel     : in std_logic;
    q       : out std_logic_vector(width-1 downto 0)
                          );
end mux;

architecture mux21_arch1 of mux21 is
begin
  q <= a when (sel = '0') else b;
end architecture rtl;
```



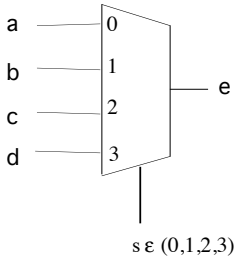
A 4-to-1 MUX - Behavioural Entity

```
entity  mux4  is
      port ( a,c,b,d :    IN   BIT;
             s  :         IN INTEGER RANGE 0 to 3;
             e  :         OUT BIT);
end mux4 ;

architecture  mux4_a  of  mux4  is
begin

      -- the 'with-select' construct usually synthesizes to a multiplexer

      with s select
            e <=  a when 0 ,
                  b when 1,
                  c when 2,
                  d when 3;

end mux4_a ;
```



Use the 'With-Select' statement

A 4-to-1 MUX - Behavioural Entity

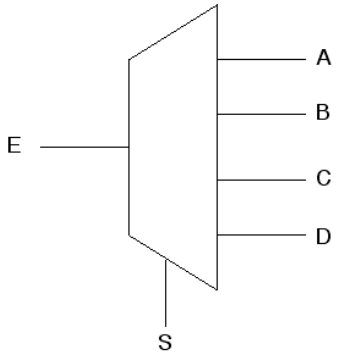



```
architecture mux4_arch1 of mux4 is
begin
      MUX: process(sel,d0,d1,d2,d3) is
      begin
            case sel is
                  when 0 =>
                        z <= d0;
                  when 1 =>
                        z <= d1;
                  when 2 =>
                        z <= d2;
                  when 3 =>
                        z <= d3;
            end case;
      end process MUX;
end architecture mux4_arch1;
```

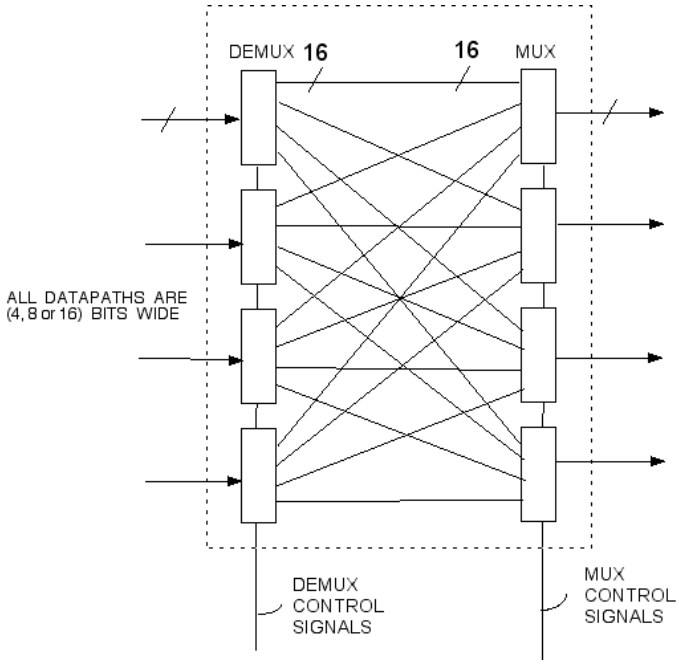
-- introduce propagation delay, page 111 ashenden

```
architecture mux4_arch1 of mux4 is
begin
      MUX: process(sel,d0,d1,d2,d3) is
      begin
            case sel is
                  when 0 =>
                        z <= d0 after prop_delay;
                  when 1 =>
                        z <= d1 after prop_delay;
                  when 2 =>
                        z <= d2 after prop_delay;
                  when 3 =>
                        z <= d3 after prop_delay;
            end case;
      end process MUX;
end architecture mux4_arch1;
```

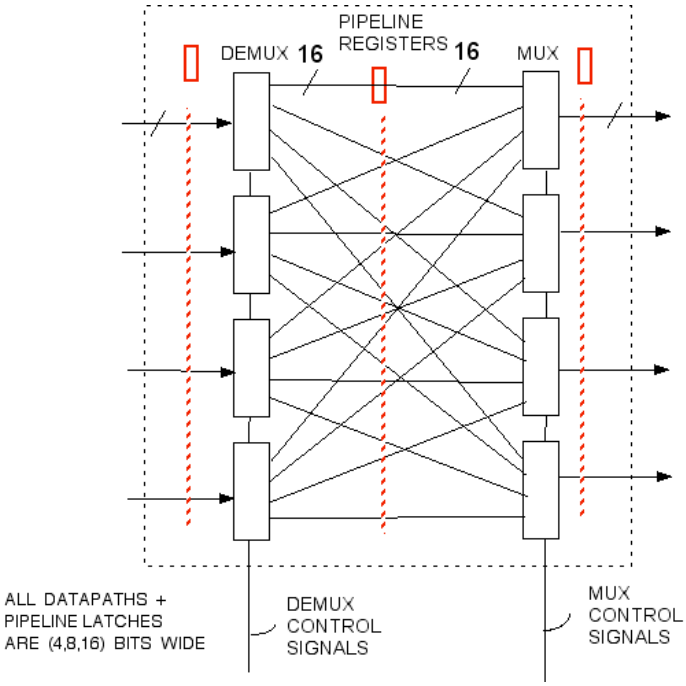
A 1-to-4 De-Multiplexer - Behavioural Entity



A 4-by-4 Crossbar Switch - Unpipelined



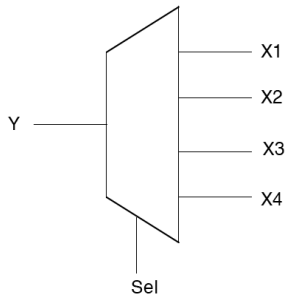
A 4-by-4 Crossbar Switch - Pipelined



A 4-to-1 MUX in VHDL

```
use work.SWITCH_PKG.all;

ENTITY MX4 IS
--GENERIC ( DATA_WIDTH : INTEGER := 16;
--          SEL_SIZE  : INTEGER := 2); --logN in N-1 MUX
PORT (
       sel              : IN STD_LOGIC_VECTOR(SEL_SIZE-1 DOWNTO 0);
       x1, x2, x3, x4   : IN STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0);
       y                : OUT STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0));
END MX4;

ARCHITECTURE MX4_Behavior OF MX4 IS
BEGIN
  process(sel, x1, x2, x3, x4)
  begin
   CASE sel IS
         WHEN "00" =>
                y <= x1;
         WHEN "01" =>
                y <= x2;
         WHEN "10" =>
                y <= x3;
         WHEN OTHERS =>
                y <= x4;
   END CASE;
  end process;
END MX4_Behavior;
```

A 4-to-1 DEMUX in VHDL




```
ENTITY DMX4 IS
--GENERIC ( DATA_WIDTH : INTEGER := 16;
--          SEL_SIZE   : INTEGER := 2); --or logN in an N-port DMX
PORT (
       sel              : IN STD_LOGIC_VECTOR(SEL_SIZE-1 DOWNTO 0);
       x1, x2, x3, x4   : OUT STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0);
       y                : IN STD_LOGIC_VECTOR(DATA_WIDTH-1 DOWNTO 0));
END DMX4;

ARCHITECTURE DMX4_Behavior OF DMX4 IS
BEGIN
  process(sel, y)
  begin
   CASE sel IS
         WHEN "00" =>
        x1 <= y;
        x2 <= (others=>'0');
        x3 <= (others=>'0');
        x4 <= (others=>'0');
         WHEN "01" =>
        x1 <= (others=>'0');
        x2 <= y;
        x3 <= (others=>'0');
        x4 <= (others=>'0');
         WHEN "10" =>
        x1 <= (others=>'0');
        x2 <= (others=>'0');
        x3 <= y;
        x4 <= (others=>'0');
         WHEN OTHERS =>
            x1 <= (others=>'0');
        x2 <= (others=>'0');
        x3 <= (others=>'0');
        x4 <=  y;
    END CASE;
   end process;
END DMX4_Behavior;
```