```
  shift_out <= second_stage when (shift_control(2) = '0') else
               ("0000" & second_stage(0 to 4));
end dataflow;
```

```
  car(4*m+2) <= g(4*m+1) or (p(4*m+1) and g(4*m)) or
                (p(4*m+1) and p(4*m) and car(4*m));
  car(4*m+3) <= g(4*m+2) or (p(4*m+2) and g(4*m+1)) or
                (p(4*m+2) and p(4*m+1) and g(4*m)) or
                (p(4*m+2) and p(4*m+1) and p(4*m) and car(4*m));
end generate G2;
car(0) <= cin;
car(4) <= gg(0) or (pg(0) and car(0));
car(8) <= gg(1) or (pg(1) and gg(0)) or
          (pg(1) and pg(0) and car(0));
car(12) <= gg(2) or (pg(2) and gg(1)) or
           (pg(2) and pg(1) and gg(0)) or
           (pg(2) and pg(1) and pg(0) and car(0));
car(16) <= gg(3) or (pg(3) and gg(2)) or
           (pg(3) and pg(2) and gg(1)) or
           (pg(3) and pg(2) and pg(1) and gg(0)) or
           (pg(3) and pg(2) and pg(1) and pg(0) and car(0));
cout <= car(16);
end dataflow;
```

8-bit right shifter

```
library ieee;
use ieee.std_logic_1164.all;
entity eight_bit_shifter is
  port(shift_in: in std_ulogic_vector(0 to 7);
       shift_out: out std_ulogic_vector(0 to 7);
       shift_control: in std_ulogic_vector(0 to 2));
end eight_bit_shifter;
architecture dataflow of eight_bit_shifter is
signal first_stage: std_ulogic_vector(0 to 7);
signal second_stage: std_ulogic_vector(0 to 7);
begin
  first_stage <= shift_in when (shift_control(0) = '0') else
                 ("0" & shift_in(0 to 6));
  second_stage <= first_stage when (shift_control(0) = '0') else
                  ("00" & first_stage(0 to 5));
```

```
  memory: process(ra, rb, clk)
    type register_file_word is std_ulogic_vector(0 to 31);
    variable reg_file: array(0 to 31) of register_file_word;
    begin
      busa <= reg_file(to_integer(ra));
      busb <= reg_file(to_integer(rb));
      if (clk = '1' and not(clk'stable) and write_enable) then
        reg_file(to_integer(rw)) := busw;
      end if;
    end process;
end dataflow;
```

16-bit carry-lookahead adder

```
library ieee;
use ieee.std_logic_1164.all;
entity adder_16 is
  port(a, b: in std_ulogic_vector(0 to 15);
       sum: out std_ulogic_vector(0 to 15);
       cout: out std_ulogic);
end adder_16;
architecture dataflow of adder_16 is
signal car: std_ulogic_vector(0 to 16);
signal pg, gg: std_ulogic_vector(0 to 3);
begin
-- full adder blocks
G1: for m in 0 to 15 generate
  sum(m) <= p(m) xor g(m) xor car(m);
  g(m) <= a(m) and b(m);
  p(m) <= a(m) or b(m);
end generate G1;
G2: for m in 0 to 3 generate
  gg(m) <= g(4*m+3) or (p(4*m+3) and g(4*m+2)) or
           (p(4*m+3) and p(4*m+2) and g(4*m+1)) or
           (p(4*m+3) and p(4*m+2) and p(4*m+1) and g(4*m));
  pg(m) <= p(4*m+3) and p(4*m+2) and p(4*m+1) and p(4*m));
  car(4*m+1) <= g(4*m) or (p(4*m) and car(4*m));
```

Columbia University
Department of Electrical Engineering
EE E4340. Dataflow macros.

Positive edge-trigger 32-bit register

```
library ieee;
use ieee.std_logic_1164.all;
entity register_32 is
  port(clk, write_enable: in std_ulogic;
       data_in: in std_ulogic_vector(0 to 31);
       data_out: out std_ulogic_vector(0 to 31));
end register_32;
architecture dataflow of register_32 is
begin
   process(clk)
   begin
     if ((clk = '1') and not(clk'stable)) and write_enable) then
         data_out <= data_in;
     end if;
   end process;
end dataflow;
```

Register file (two read ports, one write port)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY register_file IS
    PORT( rw, ra, rb: out std_ulogic_vector(0 to 4);
          busa, busb: out std_ulogic_vector(0 to 31);
          write_enable: in std_ulogic;
          busw: in std_ulogic_vector(0 to 31);
          clk: in std_ulogic;
    );
END register_file;
architecture dataflow of register_file is
begin
```