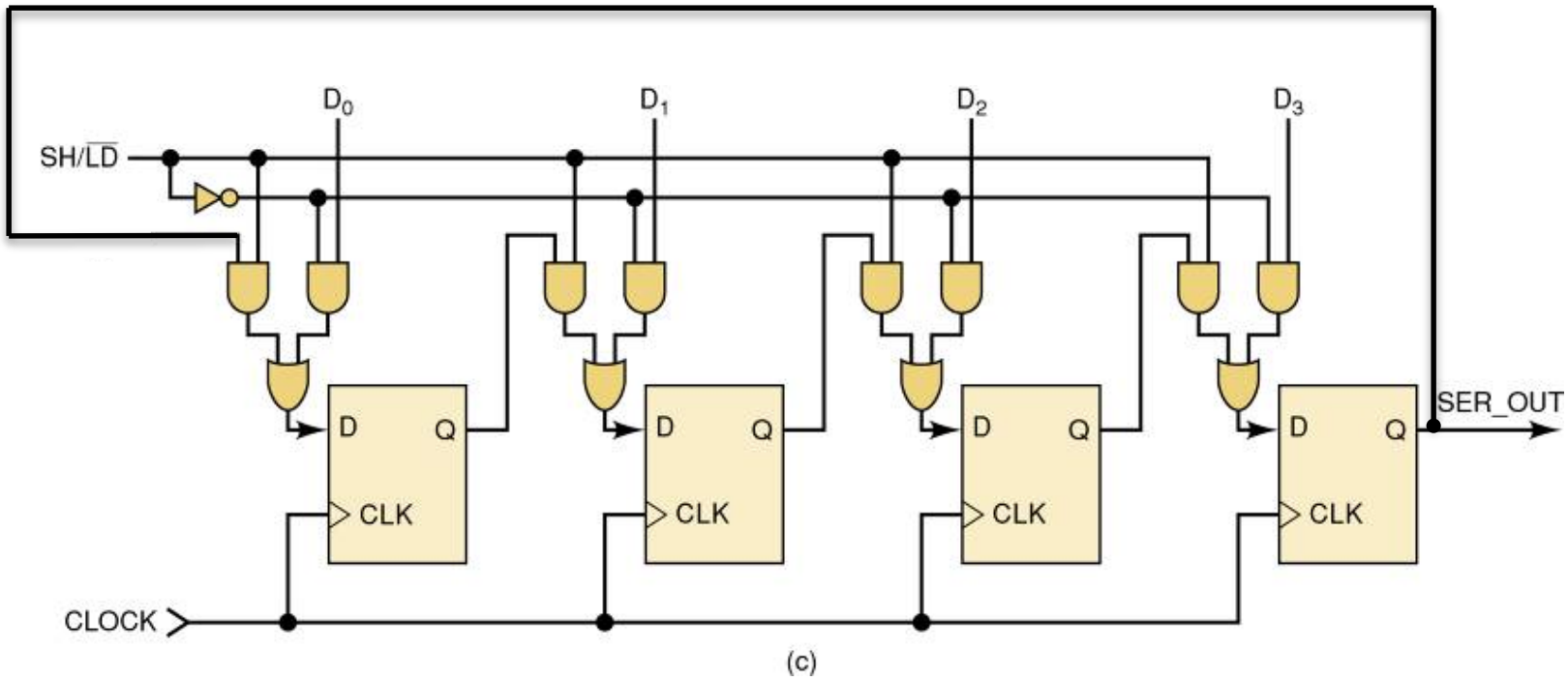
Arrays and Memory Inference

Announcements

- Homework #9 due Wednesday
- Quiz on Wednesday on the homework
- Reading Assignment:
  - Ch. 13, sections 1-2, 4-5

Shift Register Review



What is this? Look closely to where the input is coming from.

Circular Shift Register

```
ENTITY shift_circ IS
PORT(
      clk          : IN STD_LOGIC;
      reset_n    :  IN STD_LOGIC;
     sh_ld       :  IN STD_LOGIC;
      D  :  IN STD_LOGIC_VECTOR(3 downto
    0);
    ser_out     : OUT STD_LOGIC
     );
END shift_circ;
```

```
ARCHITECTURE rtl OF shift_reg IS
     signal shift : std_logic_vector(3 downto 0);
BEGIN
shifter: PROCESS(clk,reset_n)
BEGIN
  IF (reset_n = '0') THEN
     shift <= (others => '0');
  ELSIF (clk'event and clk = '1') THEN
       IF  (sh_ld = '0') THEN
             shift <= D;
       ELSE
       _____________________________________
       _____________________________________
       END IF;    --load
   END IF;       --clk
END PROCESS;
END rtl;
```

What goes here?

Arrays

- An array is an indexed collection of values all of the same type
- Represented as a new data type in VHDL
- Can be single or multi-dimensional
- Constrained – bounds for index are established when the type is defined
- Unconstrained – bounds are established after the type is defined
- Each position in the array has a scalar index value

Array Declaration

- Syntax

  array (discrete_range { ,… }) of element_subtype_indication;

- Example of unconstrained

  Type STD_LOGIC_VECTOR is array (natural range <>) of STD_LOGIC;

  - <> is called box and is used as a place holder for index range
  - Box is filled later when type is used

- Declaring a signal of array type
  - Unconstrained :
    - signal BYTE_BUS : STD_LOGIC_VECTOR(7 downto 0);

Array Declaration (con't)

- Example of constrained
  - Type MY_BYTE is array (7 downto 0) of STD_ULOGIC;

- Declaring a signal of array type
  - Constrained
    - Signal TYPE_BUS : MY_BYTE;

Array Declaration: **2 dimensional**

- Ex: 1

```
Type Large_word is array (63 downto 0) of std_logic;
Type Array_list is array (0 to 7) of large_word;
```

- Ex: 2

```
Type RAM_ARRAY is array(natural <>) of std_logic_vector(7 downto 0);
Signal MY_RAM : RAM_ARRAY (1023 downto 0);

MY_RAM(255) <= "11110000";
MY_RAM(255)(0) <= '0';
```

Array References

- Arrays can be equated rather than having to transfer element by element
- Refer to individual elements by:
  - Single index value - A(5), A(0)
  - Range – must be contiguous sequence in one-dimensional array – A(15 downto 0)

Concatenation

- Example

```
SIGNAL byte  : STD_LOGIC_VECTOR(7 downto 0);
SIGNAL nibble_A, nibble_B : STD_LOGIC_VECTOR(3 downto 0);
byte <= nibble_A & nibble_B;
                                --byte(7)  ← nibble_A(3)
                                --byte(6)  ← nibble_A(2)
                                --byte(5)  ← nibble_A(1)
                                --byte(4)  ← nibble_A(0)
                                --byte(3)  ← nibble_B(3)
                                --byte(2)  ← nibble_B(2)
                                --etc…
```

- Can also be done with single elements

```
SIGNAL Z_BUS  : STD_LOGIC_VECTOR(3 downto 0);
SIGNAL a, b, c, d : STD_LOGIC;
Z_Bus <= a & b & c & d;
        --Z_Bus(3) ← a,  Z_Bus(2) ← b,  Z_Bus(1) ← c, Z_Bus(0) ← d
```

Assignments

- Elements are assigned according to position, not their number
- Example

```
SIGNAL Z_BUS  : STD_LOGIC_VECTOR (3 downto 0);
SIGNAL C_BUS  : STD_LOGIC_VECTOR (0 to 3);
Z_BUS <= C_BUS;


--Z_BUS(3) ← C_BUS(0)
--Z_BUS(2) ← C_BUS(1)
--Z_BUS(1) ← C_BUS(2)
--Z_BUS(0) ← C_BUS(3)
```

- Be consistent to avoid issues

Aggregates

- Purpose is to bundle signals together
- May be used on both sides of an assignment
- A list of element values enclosed in parentheses
- Used to initialize elements of an array to literal values
- Keyword ‘others’ selects all remaining elements

Aggregates (con't)

```
ARCHITECTURE example OF aggregates IS
    SIGNAL byte     : STD_LOGIC_VECTOR(7 downto 0);
    SIGNAL nibble   : STD_LOGIC_VECTOR(3 downto 0);
    SIGNAL a_bit, b_bit, c_bit, d_bit  :  STD_LOGIC;
BEGIN
    --the following are positional references
    nibble <= (a_bit, b_bit, c_bit, d_bit);
    (a_bit, b_bit, c_bit, d_bit) <= STD_LOGIC_VECTOR("1011");
    (a_bit, b_bit, c_bit, d_bit) <= byte(3 downto 0);
    --the following is named association reference. Order doesn't
matter
    byte <= (7 => '1', 5 downto 1 => '1', 6 => B_BIT, others => '0');
End EXAMPLE;
```

Aggregate (con't)

- Consider the 3-to-8 decoder

Input			Output							
2^2	2^1	2^0	0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Note: The selected output goes HIGH.

```
ENTITY decode3to8 IS
    PORT(Sel    : IN  STD_LOGIC_VECTOR(2 downto 0);
         Y_out  : OUT STD_LOGIC_VECTOR(7 downto 0));
END decode3to8;

--*****************************************************
--this architecture uses an array

ARCHITECTURE cond_signal OF decode3to8 IS
    BEGIN
        PROCESS (Sel) IS
```

Can you complete the process in 2 lines?

Using Arrays to Infer Memory

- ROM – Read only memory
  - Does not change
  - Consider it a look-up-table

```
constant ZERO:  std_logic_vector(6 downto 0) := "1000000";
constant ONE:   std_logic_vector(6 downto 0) := "1111001";
constant TWO:   std_logic_vector(6 downto 0) := "0100100";
constant THREE: std_logic_vector(6 downto 0) := "0110000";
constant FOUR:  std_logic_vector(6 downto 0) := "0011001";
constant FIVE:  std_logic_vector(6 downto 0) := "0010010";
constant SIX:   std_logic_vector(6 downto 0) := "0000010";
constant SEVEN: std_logic_vector(6 downto 0) := "1111000";
constant EIGHT: std_logic_vector(6 downto 0) := "0000000";
constant NINE:  std_logic_vector(6 downto 0) := "0010000";

TYPE ssd_array_type IS ARRAY (0 TO 9) OF std_logic_vector(6 downto 0);
CONSTANT ssd_array_C : ssd_array_type := ( ZERO,  ONE, TWO, THREE,
        FOUR,  FIVE,  SIX,  SEVEN,  EIGHT,  NINE);


--HEX0 <= ssd_array_C(to_integer(ones_dig));  replace CASE statement
```

RAM

- Random Access Memory
  - Read and write
  - Synchronous

```
entity raminfr is
  port(
       clk, we_n : in std_logic;
       a : in std_logic_vector(11 downto 0);
       din : in std_logic_vector(31 downto 0);
       dout  : out std_logic_vector(31 downto 0)
    );
end raminfr;

architecture rtl of raminfr is

type ram_type is array (natural range <>) of std_logic_vector (31 downto 0);
signal RAM : ram_type(4095 downto 0);  --4K x 32 RAM
signal read_a : std_logic_vector(11 downto 0);

begin

process(clk)
begin
  if (clk'event and clk = '1') then
    if (we_n = '0') then
      RAM(to_integer(a)) <= din;
    end if;
    read_a <= a;
  end if;
end process;

dout <= RAM(to_integer(read_a));

end rtl;
```