PARAMETRIC CODING

- Techniques: generic input size, for-generate, if-generate, conv_integer.
- Custom-defined arrays, functions, and packages.
- Examples: vector mux, vector demux, vectordemux (2D input array), barrel shifter, 2D convolution kernel.
Techniques: Generic input size

- **Example: Tri-state buffer**

  Input data: N bits
  Output data: N bits
  Number of 3-state buffers required: N

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity buffer_tri is
  generic (N: INTEGER := 4);
  port ( OE: in std_logic;
         A: in std_logic_vector(N-1 downto 0);
         F: out std_logic_vector(N-1 downto 0));
end buffer_tri;

architecture bhv of buffer_tri is
begin
  with OE select
  F <= A when '1',
       (others => 'Z') when others;
end bhv;
```
**Techniques: for-generate**

- **Example: Binary to Gray Code converter**
  - Input data: N bits
  - Output data: N bits
  - Number of XOR gates required: N-1

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity bin2gray is
  generic (N: INTEGER:= 8);
  port ( B: in std_logic_vector(N-1 downto 0);
         G: out std_logic_vector(N-1 downto 0));
end bin2gray;

architecture bhv of bin2gray is
begin
  G(N-1) <= B(N-1);
  pg: for i in N-2 downto 0 generate
    G(i) <= B(i) xor B(i+1);
  end generate;
end bhv;
```

**Generic Testbench:**
Parameter N used to generate all possible combinations

- bin2gray.zip: bin2gray.vhd, tb_bin2gray.vhd
**Techniques: conv_integer**

- **Example: MUX N-to-1**

  Input data: N bits.  Output data: 1 bit
  Number of selector bits: calculated using real functions.
  The MUX output is selected via indexing.

  ```vhdl
  library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;
  use ieee.std_logic_arith.all;
  use ieee.std_logic_unsigned.all;
  use ieee.math_real.log2;
  use ieee.math_real.ceil;

  entity muxNto1 is
    generic (N: INTEGER:= 8);
    port (D: in std_logic_vector (N-1 downto 0);
          sel:in std_logic_vector(integer(ceil(log2(real(N))))-1 downto 0);
          y: out std_logic);
  end muxNto1;

  architecture structure of muxNto1 is
  begin
    y <= D(conv_integer(sel));
  end structure;
  ```

  *muxNto1.zip:*
  - muxNto1.vhd,
  - tb_muxNto1.vhd
**Techniques: if-generate**

- This is useful if we want to pick between different pieces of hardware. Only one hardware piece will be implemented.

**Example: Johnson/Ring Counter.** At Synthesis, we select which counter is to be implemented. It can’t be changed at running time.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity my_johnson_ring_counter is
  generic (COUNTER: STRING := "JOHNSON"; -- "RING"
    N: INTEGER := 4);
  port (clock, resetn, E: in std_logic;
    Q: out std_logic_vector (N-1 downto 0));
end my_johnson_ring_counter;

architecture Behavioral of my_johnson_ring_counter is
  component my_johnson_counter is
    generic (N: INTEGER := 4);
    port (clock, resetn, E: in std_logic;
      Q: out std_logic_vector (N-1 downto 0));
  end component;

  component my_ring_counter is
    generic (N: INTEGER := 4);
    port (clock, startn, E: in std_logic;
      Q: out std_logic_vector (N-1 downto 0));
  end component;

  begin
    jo: if COUNTER = "JOHNSON" generate
      jc: my_johnson_counter generic map (N => N)
        port map (clock=>clock,resetn=>resetn,E=>E,Q=>Q);
    end generate;

    ri: if COUNTER = "RING" generate
      rc: my_ring_counter generic map (N => N)
        port map (clock=>clock,startn=>resetn,E=>E,Q=>Q);
    end generate;
  end Behavioral;
```

- Here, the parameter `COUNTER=“JOHNSON”` → only the block `my_johnson_counter` will be implemented.
Custom-defined arrays

- VHDL allows for custom-defined data types, like custom arrays:
- They are defined in the architecture. Here, N, B, M are constants.
- Linear Array of vectors: It can also be seen as a 2D array of bits.

```
type chunk is array (N-1 downto 0) of std_logic_vector (B-1 downto 0);
signal data: chunk;
```

- Matrix of Vectors: It can also be seen as a 3D array of bits.

```
type chunk2D is array (N-1 downto 0, M-1 downto 0) of std_logic_vector (B-1 downto 0);
signal vf: chunk2D;
```

- Unconstrained array type: The signal definition will constrain the array
  type chunk is array (natural range <>) of std_logic_vector (B-1 downto 0);
  signal data: chunk(N-1 downto 0);
Custom-defined arrays

- **Example: Vector mux:** Input: N×B bits. For simplicity, we convert the input into an N-element array of B bits per element: $D_i$

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.math_real.log2;
use ieee.math_real.ceil;

entity vectormux is
  generic (B: INTEGER:= 8; -- bitwidth of each input
            N: INTEGER:= 16); -- number of inputs
  port (D: in std_logic_vector (B*N -1 downto 0);
        sel: in std_logic_vector (integer(ceil(log2(real(N))))-1 downto 0);
        F: out std_logic_vector (B-1 downto 0));
end vectormux;

architecture structure of vectormux is
  type chunk is array (N-1 downto 0) of std_logic_vector (B - 1 downto 0);
  signal Di: chunk;

begin
  -- Converting std_logic_vector to chunk:
  st: for i in 0 to N-1 generate
    Di(i) <= D(B*(N-i) -1 downto B*(N - (i+1)));
  end generate;

  F <= Di(conv_integer(unsigned(sel))); -- Di: array, --> MUX easily described
     -- either unsigned(sel) or use ieee.std_logic_unsigned.all
end structure;
```
Custom-defined arrays

- Example: Vector MUX → Generic testbench.
- Arrays can be displayed in the Simulation Window.

```
ENTITY tb_vectormux IS
  generic (B: INTEGER:= 8;
            N: INTEGER:= 16);
end tb_vectormux;

ARCHITECTURE behavior OF tb_vectormux IS
  component vectormux
    port ( D: in std_logic_vector (B*N -1 downto 0);
           sel: in std_logic_vector (integer (ceil(log2(real(N)))) -1 downto 0);
           F: out std_logic_vector (B-1 downto 0));
  end component;

  signal D: std_logic_vector(B*N-1 downto 0);
  signal sel: std_logic_vector ( NSEL -1 downto 0):= (others => '0');
  signal F: std_logic_vector (B-1 downto 0);

  type chunk is array (N-1 downto 0) of std_logic_vector(B-1 downto 0);
  signal Dv: chunk;

begin
  -- Converting std_logic_vector to chunk:
  st: for i in 0 to N-1 generate
    D(B*(N-i) -1 downto B*(N - (i+1)) ) <= Dv(i);
  end generate;

  uut: vectormux port map (D, sel, F);

  tb : PROCESS
  BEGIN
    bi: for i in 0 to N - 1 loop
      bj: for j in 0 to N-1 loop
        sel <= conv_std_logic_vector (j, NSEL);
        Dv(j) <= conv_std_logic_vector (i*N +j,B); wait for 20 ns;
      end loop;
    end loop;
    wait;
  END PROCESS tb;
END;
```

They must match those of vectormux.vhd
For easy data feeding, we define arrays inside the testbench
Each element Dv(i) of array Dv is assigned to vector D
Instead of input D, we provide data as Dv(i):

 vectormux.zip:
 vectormux.vhd, tb_vectormux.vhd
Custom arrays and functions

- **Example: Vector demux:** We use a custom function `myconv_integer(data)`.

- **Function:** Subprogram that computes a result to be used in expressions.

```vhdl
entity vectordemux is
  generic (B: INTEGER:= 8; -- bitwidth of each output
            N: INTEGER:= 16); -- number of outputs
  port (F: out std_logic_vector (B*N -1 downto 0);
        sel: in std_logic_vector (integer(ceil(log2(real(N))))-1 downto 0);
        D: in std_logic_vector (B-1 downto 0));
end vectordemux;
```

```vhdl
architecture structure of vectormux is
  function myconv_integer(data: in std_logic_vector) return integer is
    variable result: integer := 0;
    begin
      for i in data'range loop -- Loop thru all bits
        if data(i) = '1' then
          result:= result + 2**i;
        end if;
      end loop;
      return result;
    end function myconv_integer;
  type chunk is array (N-1 downto 0) of std_logic_vector (B-1 downto 0);
  signal Fi: chunk;
```
**Custom arrays and functions**

- **Example: Vector demux:** Input has B bits. For simplicity, we define the output data as an N-element array of B bits per element: Fi.
- **Array slicing:** We can assign \( \text{Fi}(1)(2), \text{Fi}(1)(3 \text{ downto } 0) \).

```vhdl
-- Converting chunk to std_logic_vector:
st: for i in 0 to N-1 generate
    F(B*(N-i) -1 downto B*(N - (i+1)) ) <= Fi(i);
end generate;
end structure;
```

➢ vectordemux.zip: vectordemux.vhd, tb_vectordemux.vhd
**Custom arrays and packages**

**Packages:** We can define custom data types, functions, and components.

- We can define arrays of vectors and use them in any design file. However, the size of the array and of the elements is fixed:

  ```vhdl
  type myar is array (7 downto 0) of std_logic_vector (3 downto 0);
  ```

- Generic data types are not allowed in VHDL-1993:

  ```vhdl
  type myar is array (N-1 downto 0) of std_logic_vector (B-1 downto 0);
  ```

  We cannot define this in a package file as N and B are unknown.

- We can define unconstrained arrays (1D, 2D) of a bit or a fixed bitwidth:

  ```vhdl
  type u1D is array (natural range <> ) of std_logic_vector (3 downto 0);
  type u2D is array (natural range <>, natural range <> ) of std_logic_vector (7 downto 0);
  ```

  When we use these datatypes in a design file, we define the array size:

  ```vhdl
  generic (N: integer:= 16);
  port ( data_in: u1D (N-1 downto 0)); → This is an array of N 4-bit vectors
  ```

  This is useful, but the data type u1D still has fixed-size (4-bit) elements.

- Special custom array (std_logic_2d) 2D array of one bit. This allows us to use this data type in any design file and to pass 2D generic data:

  ```vhdl
  type std_logic_2d is array (natural range <>, natural range <> ) of std_logic;
  ```
Custom arrays and packages

- **Example: Vector Mux 2D with std_logic_2d:**
- **Package file:** pack_xtras.vhd, here we define: the datatype std_logic_2d, the function ceil_log2(), and two hardware components.
  - With std_logic_2d, we can have a generic vector mux with a 2D input.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;

package pack_xtras is
  type std_logic_2d is array (natural RANGE <>, NATURAL RANGE <>) of std_logic;
  function ceil_log2(dato: in integer) return integer;
  component dffe
    port ( d, clk, ena : in STD_LOGIC;
           clrn,prn: in std_logic:= '1';
           q : out STD_LOGIC);
  end component;

  component my_rege
    generic (N: INTEGER:= 4);
    port (clock, resetn, E, sclr: in std_logic;
           D: in std_logic_vector (N-1 downto 0);
           Q: out std_logic_vector (N-1 downto 0));
  end component;
end package pack_xtras;

package body pack_xtras is
  function ceil_log2(dato: in integer) return integer is
    variable i, valor: integer;
  begin
    i:= 0; valor:= dato;
    while valor /= 1 loop
      valor := valor - (valor/2);
      i:= i + 1;
    end loop;
    return i;
  end function ceil_log2;
end package body pack_xtras;
```

Components 'dffe' and 'my_rege' declaration:
These files must exist in the project if we plan to use these components.

When using the package 'pack_xtras', there is no need to declare these components in the architecture portion.

Function description.
This function computes the result at synthesis time.
Custom arrays and packages

- **Example: Vector Mux 2D with std_logic_2d:**
- **Array Slicing** with std_logic_2d: not possible, we can only assign one bit at a time: \(D(1,\text{B}-1)\).

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library work;
use work.pack_xtras.all;

entity vectormux2d is
  generic (B: INTEGER:= 8; -- bitwidth of each input
            N: INTEGER:= 16); -- number of inputs
  port (D: in std_logic_2d (N-1 downto 0, B-1 downto 0);
        sel: in std_logic_vector (ceil_log2(N)-1 downto 0);
        F: out std_logic_vector (B-1 downto 0));
end vectormux2d;

architecture structure of vectormux2d is
  type chunk is array (N-1 downto 0) of std_logic_vector (B-1 downto 0);
signal Di: chunk;
begin
  -- Converting std_logic_2d to chunk:
  st: for i in 0 to N-1 generate
      sk: for j in 0 to B-1 generate
          Di(i)(j) <= D(i,j);
      end generate;
  end generate;

  F <= Di(conv_integer(unsigned(sel)));
end structure;
```

 vectormux2d.zip: vectormux2d.vhd, tb_vectormux2d.vhd pack_xtras.vhd
Custom arrays and packages

- **Example: Barrel shifter**: It uses: custom arrays $X_l$, $X_r$ ($N$ $N$-bit vectors), and the \texttt{ceil\_log2()} function defined in \texttt{pack\_xtras.vhd}.
  - Parameters: $N$, \texttt{SHIFTTYPE}: ARITHMETIC, LOGICAL, ROTATION.
  - Ports: \texttt{idata/odata} [$N$-$1$...0], \texttt{dir} (shift direction), \texttt{dist}: # of bits to shift.

```vhdl
entity mybarrelshift is
  generic (N: integer:= 8;
            SHIFTTYPE: string:= "ARITHMETIC"); -- "ARITHMETIC","LOGICAL","ROTATION"
  port ( idata: in std_logic_vector (N-1 downto 0);
         dist: in std_logic_vector (ceil_log2(N) - 1 downto 0); -- dist: 0 to N-1
         dir: in std_logic; -- dir=$1$ -> RIGHT, dir =$0$: LEFT
         odata: out std_logic_vector (N-1 downto 0));
end mybarrelshift;
```

<table>
<thead>
<tr>
<th>dir</th>
<th>dist[2..0]</th>
<th>data[7..0]</th>
<th>ARITHMETIC</th>
<th>LOGICAL</th>
<th>ROTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0</td>
<td>abcdefgh</td>
<td>abcdefgh</td>
<td>abcdefgh</td>
<td>abcdefgh</td>
</tr>
<tr>
<td>0</td>
<td>0 0 1</td>
<td>abcdefgh</td>
<td>bcddefgh0</td>
<td>bcddefgh</td>
<td>bcddefgh</td>
</tr>
<tr>
<td>0</td>
<td>0 1 0</td>
<td>abcdefgh</td>
<td>cdefgh00</td>
<td>cdefgh00</td>
<td>cdefghab</td>
</tr>
<tr>
<td>0</td>
<td>0 1 1</td>
<td>abcdefgh</td>
<td>defgh0000</td>
<td>defgh0000</td>
<td>defghabc</td>
</tr>
<tr>
<td>0</td>
<td>1 0 0</td>
<td>abcdefgh</td>
<td>efgh00000</td>
<td>efgh00000</td>
<td>efghabed</td>
</tr>
<tr>
<td>0</td>
<td>1 0 1</td>
<td>abcdefgh</td>
<td>fgh000000</td>
<td>fgh000000</td>
<td>fghabcde</td>
</tr>
<tr>
<td>0</td>
<td>1 1 0</td>
<td>abcdefgh</td>
<td>gh0000000</td>
<td>gh0000000</td>
<td>ghabcdef</td>
</tr>
<tr>
<td>0</td>
<td>1 1 1</td>
<td>abcdefgh</td>
<td>h00000000</td>
<td>h00000000</td>
<td>habcdefg</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0</td>
<td>abcdefgh</td>
<td>abcdefgh</td>
<td>abcdefgh</td>
<td>abcdefgh</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1</td>
<td>abcdefgh</td>
<td>aabcdefg</td>
<td>aabcdefg</td>
<td>habcdefg</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0</td>
<td>abcdefgh</td>
<td>aaabcdeff</td>
<td>aabcdeff</td>
<td>aabcdeff</td>
</tr>
<tr>
<td>1</td>
<td>0 1 1</td>
<td>abcdefgh</td>
<td>aaaaabcdeff</td>
<td>aaaaabcdeff</td>
<td>aaaaabcdef</td>
</tr>
<tr>
<td>1</td>
<td>1 0 0</td>
<td>abcdefgh</td>
<td>aaaaabcdeff</td>
<td>aaaaabcdeff</td>
<td>aaaaabcdef</td>
</tr>
<tr>
<td>1</td>
<td>1 0 1</td>
<td>abcdefgh</td>
<td>aaaaaabc</td>
<td>aaaaaabc</td>
<td>aaaaaabc</td>
</tr>
<tr>
<td>1</td>
<td>1 1 0</td>
<td>abcdefgh</td>
<td>aaaaaaab</td>
<td>aaaaaaab</td>
<td>aaaaaaab</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1</td>
<td>abcdefgh</td>
<td>aaaaaaaaa</td>
<td>aaaaaaaaa</td>
<td>aaaaaaaaa</td>
</tr>
</tbody>
</table>

\[ N = 8 \]

- **mybarrelshift.zip**: mybarrelshift.vhd, tb_mybarrelshift.vhd
  - \texttt{pack\_xtras.vhd}
**Example: Convolution Kernel**

- This example uses parameterization (if-generate, for-generate) custom arrays, 2D vector arrays.

Convolution Size: \( N \times N \)

**Parameters:** \( N, B, C, \text{REP} \) (unsigned/signed).

**Ports:** \( Di, Hi, F \). \( Di \): std_logic_2d of size \( N^2 \times B \), \( Hi \): std_logic_2d of size \( N^2 \times C \).

**I/O Delay:** \( \lfloor \log_2 N^2 \rfloor + 2 \) clock cycles

- Components: an input register array, an array of multipliers, and an adder tree.

- The figure shows a 3x3 (\( N=3 \)) case:

- For easier manipulation, \( Di, Hi \) are turned into 2D vector arrays:

**myconv2.zip:**

myconv2.vhd, my_rege.vhd, dffe.vhd, adder_tree.vhd, my_addsub.vhd, fulladd.vhd

tb_myconv2.vhd

pack_xtras.vhd