RECRLab
Reconfigurable Computing Research Laboratory

DIGITAL LOGIC DESIGN VHDL Coding for FPGAs Unit 9

✓**MISCELANEOUS TOPICS**

- I/O text files for simulation and synthesis. Example: serial multiplier, basic NI-to-NO LUT, VGA.
- Using Xilinx primitives: BRAMs, XADC, FIFOs, MMCMs, DSPs.

Daniel Llamocca

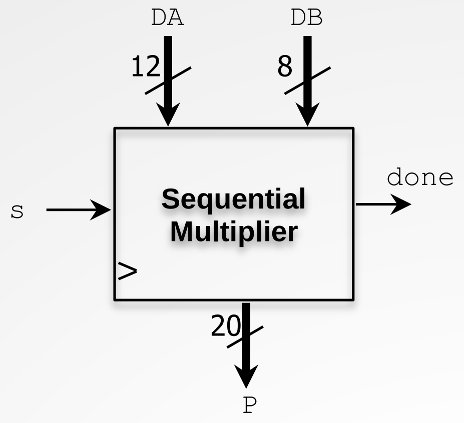
OAKLAND UNIVERSITY

✓I/O textfiles: Simulation

- **Reading/Writing text files for Simulation**:
    - UNSIGNED ITERATIVE MULTIPLIER: Digital Library→Arithmetic Cores: (code).
    - Testbench: `tb_mult_iter.vhd`. Input text file: `in_benchN12M8.txt`
    - *Stimulus process*: reads file line by line and place data on the inputs.

```
library ieee;
use ieee.std_logic_1164.all;
use std.textio.all;
use ieee.std_logic_textio.all;

entity tb_mult_iter is
 generic (N: INTEGER:= 12;
          M: INTEGER:= 8);
end tb_mult_iter;
...
  stim: process
       file IN_FILE: TEXT open READ_MODE is "in_benchN12M8.txt";
       variable BUFI: line;
       variable VAR: std_logic_vector (N+M-1 downto 0);
  begin
    wait for 100 ns; resetn <= '1'; -- required for primitives
    l_tb: loop
            exit l_tb when endfile(IN_FILE);
            readline (IN_FILE, BUFI);
            read (BUFI, var); -- read binary data. Use hread for hexadecimal data
            dA <= var (N+M-1 downto M) ; dB <= var (M-1 downto 0);
            s <= '1'; wait for T; s <= '0'; wait for T*(M+2); -- wait before the next start
          end loop;
         wait;
  end process;
...
```



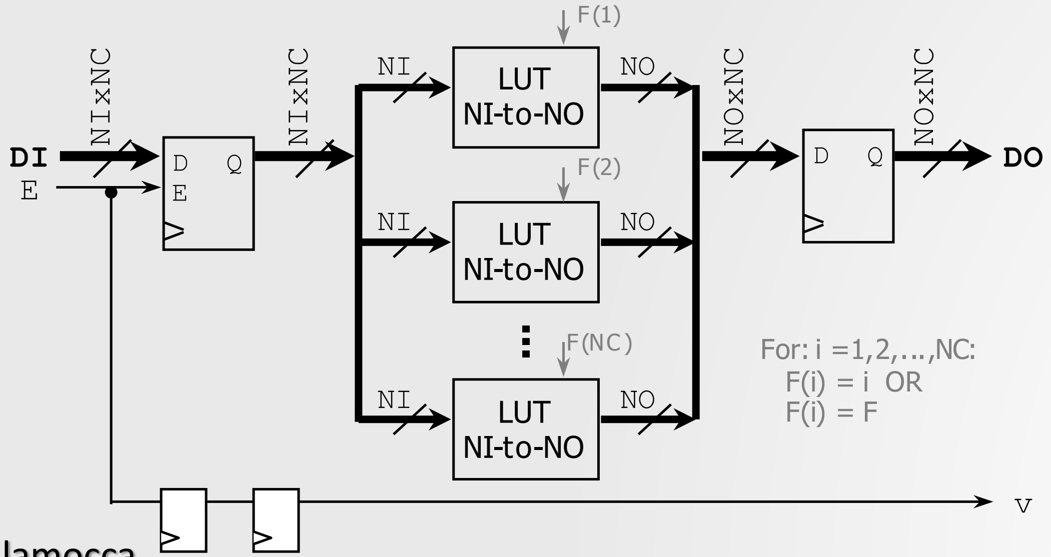
✓I/O textfiles: Simulation

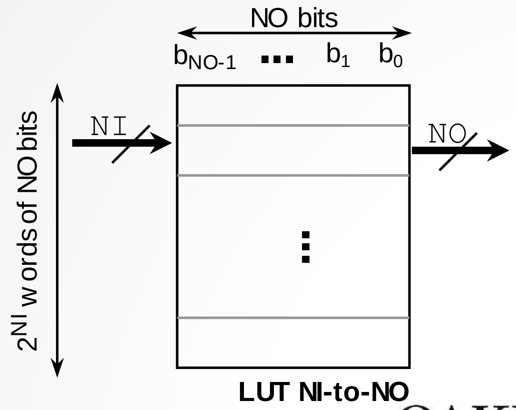
- **Reading/Writing text files for Simulation:**
  - Multiplier: Data input: A: 12 bits, B: 8 bits. Data output: P: 20 bits.
  - *Output capture process*: It captures output data and writes on file line by line. Most circuits include a 'done' signal: at the rising clock edge, data is retrieved.
  - Vivado:
    - Include the input text file as a Simulation Source in the Project.
    - The output text file will be written in `sim/sim_1/behav`.

```
...
  tb_o: process
         file OUT_FILE: TEXT open WRITE_MODE is "out_benchN12M8.txt";
         variable BUFO: line;
         variable OVAR: std_logic_vector (N+M -1 downto 0);
        begin
          lp: loop
                 wait until done = '1' and (clock'event and clock='1');
                 OVAR:= P;
                 write(BUFO, OVAR); -- you can use hwrite if data is multiple of 4.
                 writeline(OUT_FILE, BUFO);
                 wait for T;
              end loop;
          end process;
END;
```

✓I/O textfiles: Synthesis

- **Reading text files for Synthesis:** LUT WITH I/O REGISTERS
  - **`LUTsys.zip`**:
    `test.vhd, LUT_group.vhd, LUT_NItoNO.vhd, dffe.vhd, atb_test.vhd, LUT_values{NI}to{NO}.txt` (this text file has to be in the same folder as `LUT_NItoNO.vhd`).
  - `LUT_group`: NC NI-to-NO LUTs. We can load each LUT NI-to-NO with: a different function (1 to NC), or the same function (specified by F).
  - `LUT_NItoNO`: NI-to-NO LUT. It selects an *NO*-bit output word based on the NI-bit input. The LUT contents are read from an input text file.
- **Vivado:** Include the input text file(s) for testbenches as Simulation Sources. Output text file(s) will be written in `/sim/sim_1/behav`.





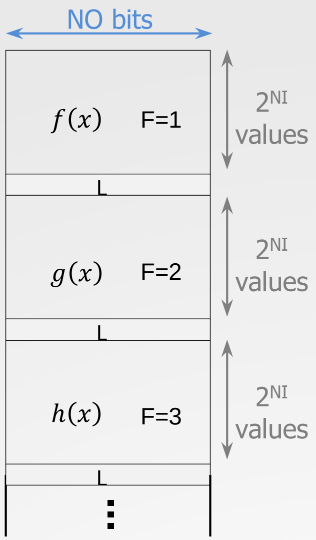
Daniel Llamocca

✓I/O textfiles: Synthesis

- **Reading text files for Synthesis:**
  - Text File: LUT_values{NI}to{NO}.txt.

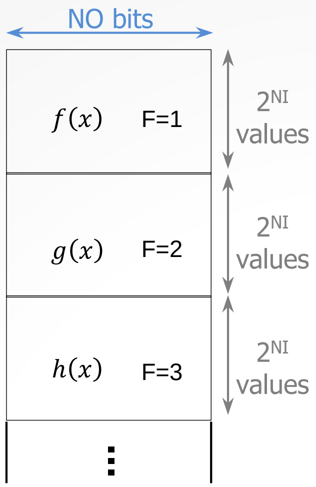
    This file holds 2^{NI} NO-bit words for each function, separated by a line. Do not leave this line blank (we use an 'L' character), otherwise *readline* will skip it and read the next line). This is the approach followed in this example for F=1,2,3,4,5.
  - For a given function (F: 1,2,…), we start at line:

    $$(F-1)\times(2^{NI}+1)+1.$$



- Alternative: You can create a file LUT_values{NI}to{NI}.txt with no line separators (it will be difficult to tell where function values are). In this case, the formula to know where to start reading (depending on F) will have to be modified to:

  $$(F-1)\times(2^{NI})+1.$$



✓I/O textfiles: Synthesis

- **Reading text files for Synthesis:** `LUT_values{NI}to{NO}.txt`
  - VHDL code:

```
...
architecture structure of LUT_NItoNO is

constant START_POINTER: INTEGER:= (F-1)*(2**NI + 1) + 1;
type chunk is array (2**NI -1 downto 0) of std_logic_vector (NO-1 downto 0);

  impure function ReadfromFile (FileName: in string; P: in integer) return chunk is
       FILE IN_FILE  : text open read_mode is FileName
       variable BUFF : line;
       variable val  : chunk;
  begin
   if P /= 1 then
       for j in 1 to P-1 loop
              readline (IN_FILE, BUFF);
       end loop;
   end if;
   for i in 0 to 2**NI - 1 loop
        readline (IN_FILE, BUFF);
        read (BUFF, val(i));
   end loop;
   return val;
  end function;

  constant LUT_val: chunk:= ReadFromFile(file_LUT, START_POINTER);

begin
    LUT_out <= LUT_val(conv_integer(LUT_in));
end structure;
```

Depending on the function, we start at a different place in the text file

To start at the proper place in the text file, we read P-1 lines without storing data.

Text File is read with this function. **impure** qualifier required when the function output changes even when passing the same parameters, e..g: when text file contents change

The NItoNO LUT behaves like a mux with constant inputs