

VHDL (III)

A Full Adder in VHDL

- In this course, we use VHDL to assist our circuit design.
- So, if we have a more complex circuit, we can leverage some VHDL to help with the validation.
- What is **VHDL**?
 - An acronym for **Very High Speed Integrated Circuit Hardware Description Language**.
 - A program language used to **describe a logic circuit** by function, data flow behavior, or structure.
- What is **GHDL**?
 - Like Java and C, VHDL programs need to be compiled before running.
 - GHDL is a **compiler for VHDL**.
 - GHDL is short for **G Hardware Design Language**. Currently, G has no meaning.
- How do you get VHDL and GHDL?
 - The dwarves in the robotics lab have VHDL and GHDL ready for you to use.
- Now, let's see how to use VHDL to describe the full adder we talked in last class.

```
-- Comments in VHDL start with --
-- Ying Li
-- A full adder

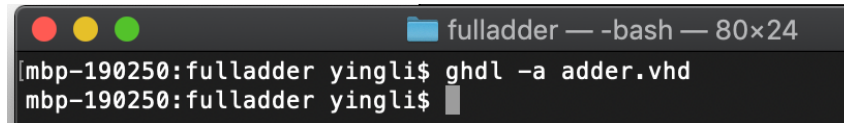
-- import useful packages
library ieee;
use ieee.std_logic_1164.all; -- provide enhanced signal types

-- filename must be the same as entity name
-- entity defines the inputs and outputs of the circuit
entity adder is
  -- A, B, Ci are inputs
  -- S and Co are the outputs
  port (
    A, B, Ci : in std_logic; -- std_logic represent one bit signal
    S, Co: out std_logic      -- that can take on the value 0 or 1
  );
end adder;

-- architecture define the circuit
architecture behavior of adder is
begin
  -- statements in the architecture enclosed by begin and end are executed concurrently
  -- compute the sum
  s <= A xor B xor ci; -- <= assign the result of A xor B xor ci to s
  -- compute the carry
  co <= (A and B) or ((a xor b) and ci); -- VHDL is case-insensitive
end behavior;

-- to analyze this design, using ghdl -a adder.vhd
```

- The above code is in a file named `adder.vhd`.
- It describes the full adder circuit with 3 inputs, 2 outputs, two XOR gates, two AND gates, and one OR gate.
- To compile the program, use the command `ghdl -a adder.vhd`
 - `-a` means analysis



```
fulladder — -bash — 80x24
[mbp-190250:fulladder yingli$ ghdl -a adder.vhd
mbp-190250:fulladder yingli$
```

- Now, we need a test program for the circuit so that we can tell whether the circuit works correctly.

```
-- Ying Li
-- A testbench for the full adder

library ieee;
use ieee.std_logic_1164.all;

-- A testbench has no ports
entity addertest is
end addertest;

architecture behavior of addertest is
    -- Declaration of the component that will be instantiated
    component adder
        port (
            A, B, Ci: in std_logic;
            S, Co: out std_logic
        );
    end component;

    -- the adder signals
    signal I0, I1, I2, O0, O1 : std_logic;

begin
    -- component instantiation
    -- connect the inputs and outputs of the entity to the local signals
    adder1 : adder port map (A=>I0, B=>I1, Ci=>I2, Co=>O0, S=>O1);

    I0 <= '0', '1' after 4 ns; -- send signals to port a, the signal is 0 in the first 4 ns
    and 1 afterward
    I1 <= '0', '1' after 2 ns, '0' after 4 ns, '1' after 6 ns;
    I2 <= '0', '1' after 1 ns, '0' after 2 ns, '1' after 3 ns, '0' after 4 ns, '1' after 5
    ns, '0' after 6 ns, '1' after 7 ns, '0' after 8 ns;

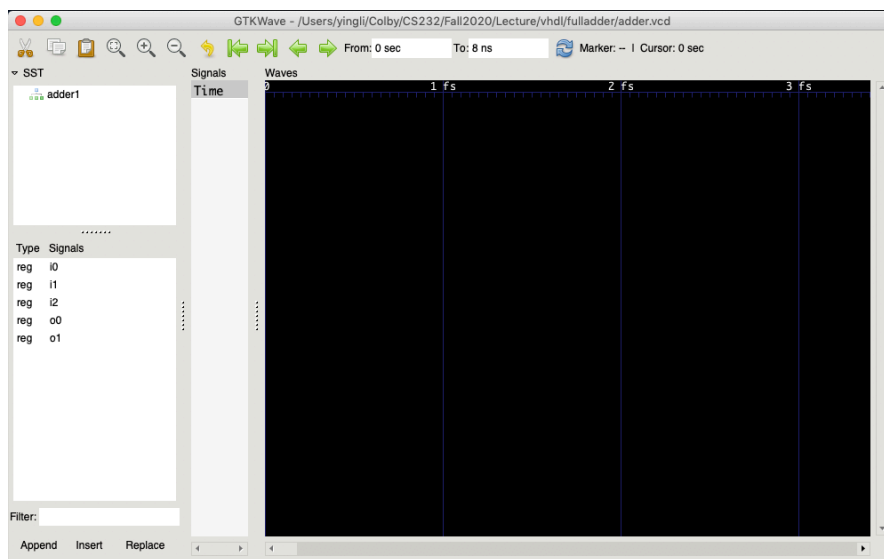
end behavior;
```

- The above code is in a file named `addertest.vhd`
- The testbed gives three inputs signals to the three inputs of the full adder. In the first 8 ns, the first input has value 0 during the first 4 ns and 1 afterward. The second input starts with 0 and varies the value every 2 ns. The third input starts with 0 and varies the value every ns.

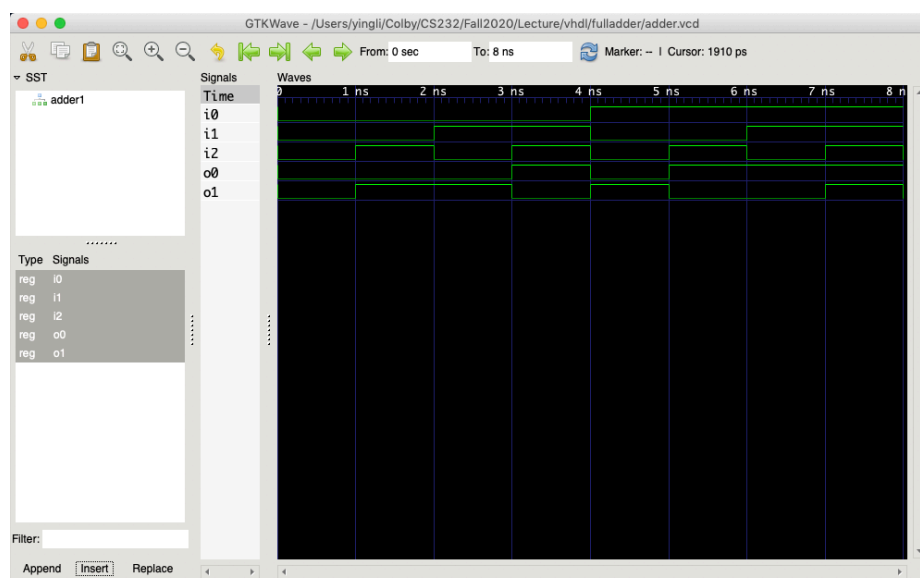
- We need to compile the program using `ghdl -a addertest.vhd`
- Then, elaborate the testbench: `ghdl -e addertest`
 - This step may not generate output files on some platforms.
- The last step is to run the test program using `ghdl -r addertest --vcd=adder.vcd`
- This step will generate an `adder.vcd` file. You can use [gtkwave](#) to open the vcd file and valid the correctness of the full adder behavior.

```
mbp-190250:fulladder yingli$ ghdl -a addertest.vhd
mbp-190250:fulladder yingli$ ghdl -e addertest
mbp-190250:fulladder yingli$ ghdl -r addertest --vcd=adder.vcd
mbp-190250:fulladder yingli$
```

- Open the `adder.vcd` file using `gtkwave`, you will get a window like this



- Select the all the signals and click “Insert”, then click the “Zoom Fit” button. You will get a window like this. The green signals can help us to validate the full adder.



Two-Bit Adder

- Design a circuit that takes two 2-bit inputs as unsigned binary sequences, add them, and generate a 3-bit outputs.
- We can design the circuit using gates like what we did for the full adder, but we can also leverage the `numeric_std` library to simplify the process.
 - `numeric_std` library provides numeric types (signed/unsigned) and arithmetic functions.
- We need a vhd file describe the circuit for 2-bit adder, `twoBitAdder.vhd`

```
-- Ying Li
-- A circuit that takes two 2-bit inputs and generate a 3-bit output

-- import useful packages
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all; -- provides numeric types (unsigned/signed) and arithmetic
functions

-- filename must be the same as entity name
-- entity defines the inputs and outputs of the circuit
entity twoBitAdder is
    -- A and B are an array of inputs
    -- F is an array of outputs
    port (
        A, B: in unsigned (1 downto 0); -- array/vector of unsigned type
        F: out unsigned (2 downto 0)
    );
end twoBitAdder;

-- architecture define the circuit
architecture behavior of twoBitAdder is
    begin
        -- extend both 2-bit inputs to 3-bit inputs and then do the addition
        F <= ('0' & A) + ('0' & B);
    end behavior;
```