

# **L09: VHDL Test benches and File Access**

# Overview – topics covered so far...

## 1. Introduction to VHDL

- Use in describing systems

## 2. Describing & modelling systems

- Interface & behaviour
- Signals, events, timing, concurrency
- Use of discrete event simulation

## 3. Basic language constructs

- Std\_logic resolved type
- Signal assignments
- Delays
- Modelling complex behaviour
- Modelling structure

## 4. Use of VHDL to model and simulate sequential logic designs

# Overview for today's lecture...

## 5. Test benches

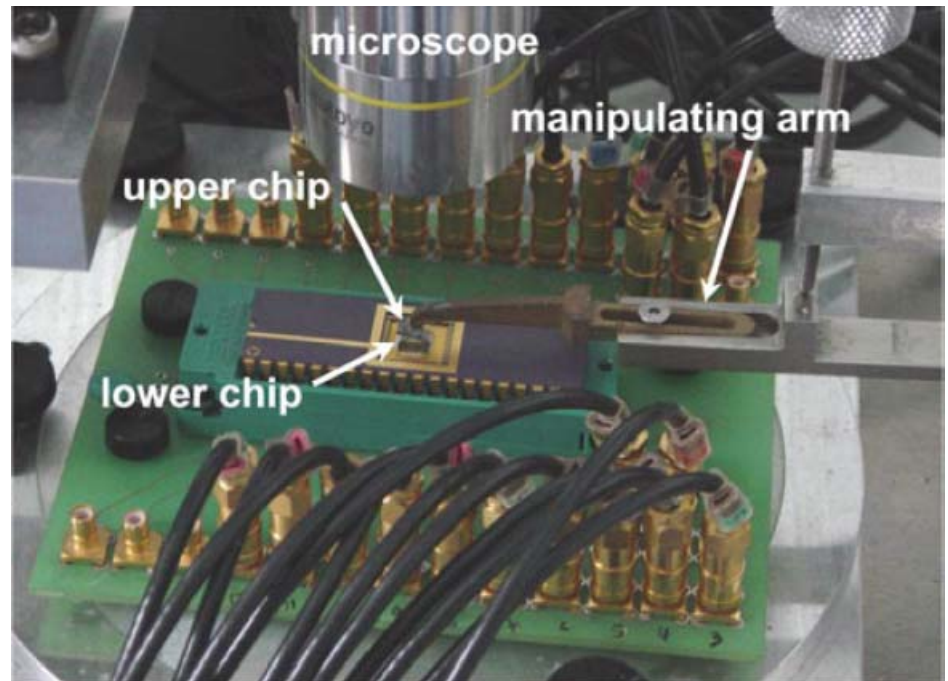
- Examples of generated and hand-crafted test bench code

## 6. File operations

- Examples of commonly used idiom for creating more flexible test benches

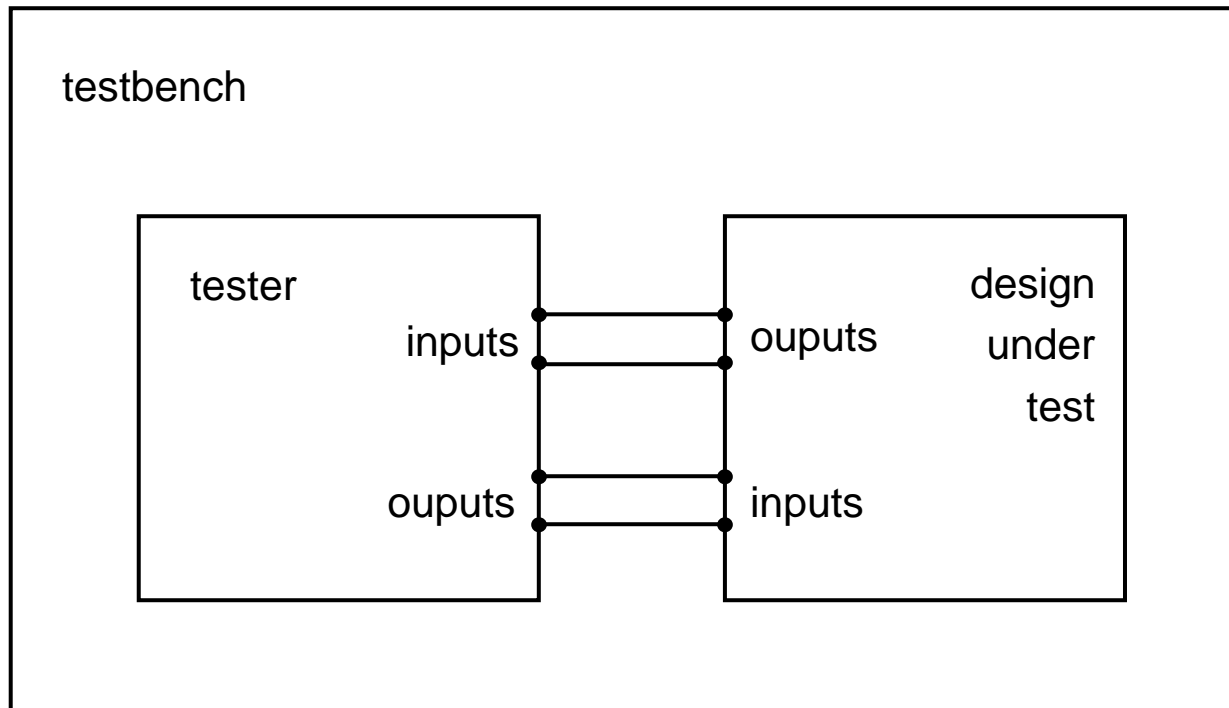
# Test benches

- The term *test bench* derives from the physical testing of a digital system using a test rig (restraints, probes, signal generators, scopes etc.) for stimulating the system, observing the results, and determining whether these deviate from those expected for the given stimuli
- Since a test bench typically is also a digital system, there is some sense in designing these within VHDL as well
- We use test benches to simulate and verify the functionality of our VHDL specifications/designs



# A VHDL Test bench

- A test bench can be structured as the composition of a *design under test* and its tester



# Test bench examples

1. Create a test bench by adding a *VHDL Test bench* to your project
  - Example: full\_adder\fatb\_hand
2. A test bench can also be generated for you when you add a *Test bench Waveform* to your model
  - Example: full\_adder\fatb\_w

# File I/O example

- Use of files is fairly straight-forward
  - Example: [fileio\fileio](#)

# FILE essentials 1

- A **file** is an object like a signal and variable
- The file can be *implicitly* or *explicitly* opened
  - Implicit means at declaration time rather than making an explicit **file\_open** procedure call
  - The file is associated with a **file pointer**, a **mode**, and a **file type** when it is opened
  - The file can be explicitly closed using **file\_close** or left to be closed implicitly when the simulation finishes
  - Procedures relating to files are contained in the **STD** library and **TEXTIO** package found in ...\\xilinx\\vhdl\\src\\std in our setup
  - Procedures that support data of type **STD\_LOGIC\_VECTOR** can be found in the IEEE library in the **STD\_LOGIC\_TEXTIO** package (see ...\\xilinx\\vhdl\\src\\ieee in our case)

# FILE essentials 2

- File I/O is done via **read** and **write** procedures to manipulate an I/O buffer and via **readline** and **writeline** procedures that read, resp. write a buffer line to the file
  - The read/write procedures are *overloaded* for reading objects of different types (TEXT, INTEGER, STD\_LOGIC\_VECTOR)
- The screen (stdout) file is opened by default and associated with the file pointer called **output**
  - Similarly, the keyboard is associated with the **input** file pointer

# Test bench examples

1. Create a test bench by adding a *VHDL Test bench* to your project
  - Example: full\_adder\fatb\_hand
2. A test bench can also be generated for you when you add a *Test bench Waveform* to your model
  - Example: full\_adder\fatb\_w
3. Input of **test vectors** from a file provides greater flexibility and control, and can eliminate tedious recompiling before simulating your models
  - Example: full\_adder\fatb\_file

# Another neat example of file input: Initializing memory

- Recall the memory model you studied in labs: [rw\\_memory](#)
  - Memory was initialized by hard-coding the initial values into the VHDL code
- We could initialize the memory by reading the initial contents from a file when the simulation commences
  - Example: [memory\memory](#)