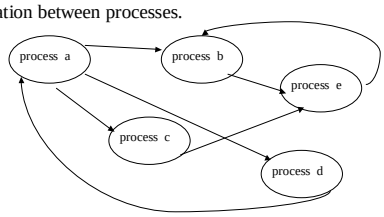
VHDL & the Single Clock Cycle (SCI) Implementation

- We will use the VHDL language to model the SCI MIPS implementation (as well as other MIPS versions)
- VHDL is a Hardware Description Language (HDL)
  - A logic synthesis tool can take a VHDL description and synthesize a netlist of logic gates (hardware!) that implements the description
- We will use the ModelSim-XE simulator to compile and execute the VHDL model
- VHDL looks similar to a programming language but execution of VHDL is VERY DIFFERENT from executing a normal sequential program
  - VHDL describes hardware, which executes in parallel! (concurrently)
  - A programming language executes sequentially

VHDL Processes, Signals

VHDL models consist of processes, with signals used to carry information between processes.



A process is like a small program that is executed whenever an 'event' occurs on one of its input signals. An 'event' is any change in value of the signal.

VHDL Execution

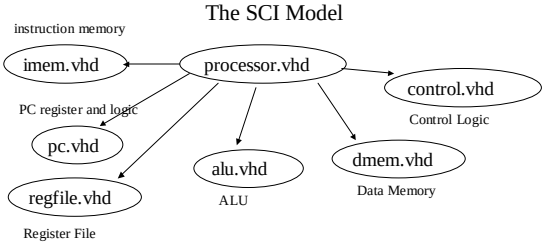
1. The VHDL simulator maintains an event queue, which is simply a time-ordered listing of events and the signals that they are connected to.

2. At time 'X', the simulator looks at all events scheduled for that time, and executes any processes that are 'triggerred' by those events (a trigger is an event occurring on input signal to a process).

3. Executing a process may generate new events on the process output signals. These new events are placed on the time queue.

4. When all events for time 'X' have been executed, the simulator advances time to the next the event on the event queue and the sequence is repeated.

The SCI Model



Auxiliary files:

define.vhd : A VHDL package that contains MIPS opcode definitions. Used by *control, ALU.*

mips_components.vhd : A VHDL package that contains component definitions for *imem, pc, regfile, alu, dmem, control.* Used by processor.vhd.

VHDL Vocabulary

- A description of a hardware component in VHDL consists of *entity* and *architecture* declarations.
  - The *entity* describes the input/output signals (ie. the interface)
  - The *architecture* describes the behavior (contains the process statements)
  - In our files, both entity and architecture are in the same file (ALU entity and architecture are in alu.vhd)
- A VHDL *package* consists of common procedures, functions, constants, component declarations used by one or more entities or other packages
  - *define.vhd* and *mips_components.vhd* contain VHDL packages used by other entities in the SCI model

The VHDL Language

- You will only need to know a very small subset of the VHDL language to do the external assignments in this class
- See the class notes on 'Combinational Logic in VHDL" at http://www.erc.msstate.edu/~reese/EE4743
- You can get by in this class with knowing how to do:
  - Signal assigments ile., Y <= A or B;
  - When/Else concurrent statement: Y = '1' when (A =B) else '0';
  - 'if' and 'case' statements inside of processes
  - How to add an input or output signal to a model
- By just reading the models, you can see the format of these statements.

Using ModelSIM-XE with the SCI Model

- Install ModelSIM-XE and get a valid license file from Xilinx
- Download the modelsim_sci.zip file from the class WWW page. This contains the VHDL models for the MIPS SCI.
  - When you unzip this archive, you will have a directory tree with: modelsim/sci, modelsim/utilities
  - The 'utilities directory contains some additional VHDL packages needed by the SCI model, you should never have to edit these files and it is already compiled.

Using ModelSIM-XE with the SCI Model (cont).

- Use a text editor to edit the file modelsim/sci/sci.mpf and make the following changes:
  - Change the line 'others = C:/data/Modeltech_xe/… ' to reflect the installation directory of ModelTech_xe on your system (probably just C:/Modeltech_xe/…)
  - Change the line "utilties = H:/EE4714/modelsim/utilities/work " to reflect the location of 'modelsim/utilities' directory on your disk.
  - The 'sci.mpf' is a project description file. A project is simply a collection of files. Any changes to a 'project' via the Modelsim interface are made to this file.
- Run ModelSim XE . Use the File → Change Directory menu choice to change to the 'modelsim/sci' directory.
- Use the File → Open Project menu and load the sci/sci.mpf file. If presented with a load Design window, load the 'processor' entity.

Compilation, Execution

- To compile all files in a project, choose the Design → Compile Project menu choice
- One way to edit a file is to use Options → Edit Project
  - Once the Edit project window pops up, click on the down arrow in the 'Source File' field to see a list of files in the project. Choose one, then click on the 'Edit' button. After making changes to the file, you can use the 'Compile' button in the Edit project window to compile the file
- To simulate the SCI design, first use Design → Load New Design and load the processor entity.
  - You should see the 'vsim>' prompt appear.
  - Type 'do sci.do'. This is a command file that will cause a wave window to appear with the various signals displayed. You may need to resize the wave window.
  - Type 'run 400 ns' to run the simulation for 400 ns. Use the 'Zoom Full' menu choice in the Wave window to see the entire waveforms.

Edit, Compile, Debugging

- After editing and compiling a file, you must do 'Design → Load New Design' to load the altered design, this is not done automatically.
- There are several windows available via the 'View' menu that are useful for debugging
  - Structure – this shows you a hierarchy of all components in a design. Clicking on a component will update other windows like Source, Signals, etc to show the details for that component.
  - Source – source file window. Clicking on line number puts a breakpoint at that line number.
  - Signals – shows the current values of all signals for the currently selected component.
  - Wave – waveform display window. Look at the 'sci.do' command file to see the command for adding signals to the waveform window.

Comments on *imem.vhd*

- *imem.vhd* contains the instruction memory model
  - imem expects a file called 'program.obj' to be in the sci directory
  - program.obj defines the initial instruction memory contents and is read at VHDL startup
  - The format of each line of program.obj is
    'mem_loc mem_contents'
  - where mem_loc is a decimal number, mem_contents is a 32-bit binary value representing a MIPS instruction. 'mem_loc' is a WORD address, not a byte address.
- A perl script utitility (/home/reese/bin/spim2obj.pl) can be used to transform a mips assembly file (test.s) to an object file.
  - Verify that your assembly file can be loaded by XSPIM with no errors.
  - Then execute "/home/bin/spim2obj.pl myfile' where you leave the ".s" extension off (*myfile.s* should be in the current directory). The 'spim' program must be on your path.
  - A new file called 'myfile.obj' will be produced. Copy this to the modelsim/sci directory and rename it to 'program.obj'.

Comments on *dmem.vhd*

- *dmem.vhd* contains the data memory model
- dmem.vhd expects a file called "datamem.txt" to be in the modelsim/sci directory
  - *datamem.txt* defines the initial data memory contents
  - Data memory is first filled with zeros, then the *datamem.txt* is read
  - The format of this data memory file is the same as the instruction memory file
  - There is no utility for creating this file, simply edit it manually and set the desired initial data contents

Initial Program provided with ZIP file

- The initial 'program.obj' file provided with the zip archive has the following contents for the first four instructions:

  Word 0 (byte 0) : jmp word 8
  Word 8 (byte 32): addi $3, $0, 0x7FF0
  Word 9 (byte 36): addi $2, $0, 4
  Word 10 (byte 40): lw $4, 0($2)

- The clock period is set in processor.vhd to be 100 ns. After 400 ns, all 4 instructions are executed, and the following registers are changed:
  - $3 = 0x7FF0
  - $2 = 4
  - $4 = 0xFFFFFFF (see contents of datamem.txt, this is the value of word location 1).

External Assignment

- The control.vhd file only implements instructions addi, add, beq, jmp, lw
- Add support for instructions bne, sw, and, andi, or, ori, sub, slt, slti
  - An assembly test program will be provided later that can be used as an 'acid' test for your changes to the SCI model. You can also write your own test for these changes.
  - You will only need to modify 'control.vhd' to support these instructions.
- For an extra challenge, try adding support for 'jr'. You will have to modify more than 'control.vhd'!!!

ModelSim for Unix

- Download the zip file modelsim_sci.zip (the same file as PC users)
  - Unzip via "unzip modelsim_sci.zip".
- This creates directories modelsim/sci, modelsim/utilities
- Edit the file modelsim/sci/sci.mpf and make the following changes:
  - Change:
    others = C:/data/Modeltech_xe/win32xoem/../modelsim.ini
    TO:
    others =/opt/ecad/mentor/pctools/modelsim/default/modeltech/bin/../modelsim.ini
  - Change: *utilities = H:/EE4713/modelsim/utilities/work*
    TO: *utilities = ../utilities/work*
  - Change: *OptionFile = c:/data/Modeltech_xe/examples/vlog.opt*
    TO: *OptionFile = /opt/ecad/mentor/pctools/modelsim/default/modeltech/ examples/vlog.opt*
- To place modelsim on your path, do "swsetup modelsim".
- To execute modelsim, change directory to "modelsim/sci" and type "qhsim &"
  - The brings up the Modelsim interface which is the same for all users.