Introducción a la simulación con ModelSim.

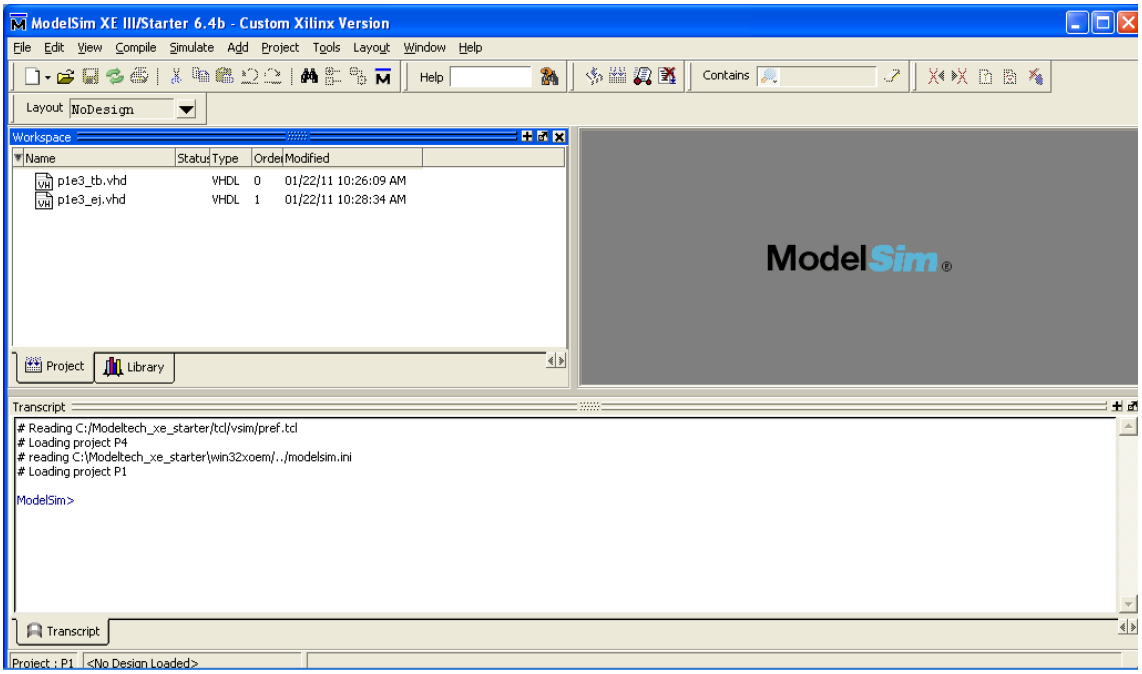
Este es un resumen de la funcionalidad básica del simulador ModelSim.

Model Sim permite muchas cosas más. Puede encontrar más información en *Help->PDF Documentation->Tutorial* y *Help->PDF Documentation-> User´s manual*. Tenga en cuenta que no todas las características de ModelSim están permitidas en una licencia de estudiante.

Crear un proyecto.

Seleccione en la barra de menús *File->New->Project*. Defina nombre de proyecto y path donde va a residir. Deje el resto de opciones en valores por defecto.

Seleccione *Add Existing File* y agregue los ficheros VHDL.



Se pueden agregar o eliminar ficheros del proyecto picando con el botón derecho del ratón en la pestaña *Project* de la ventana *Workspace*. También se pueden agregar ficheros en la entrada de menú *Project*.

Cuando se cierre el ModelSim, se guarda automáticamente el proyecto. Puede invocar el proyecto la siguiente vez seleccionando *File->Open* y seleccionando los tipos de fichero proyecto.

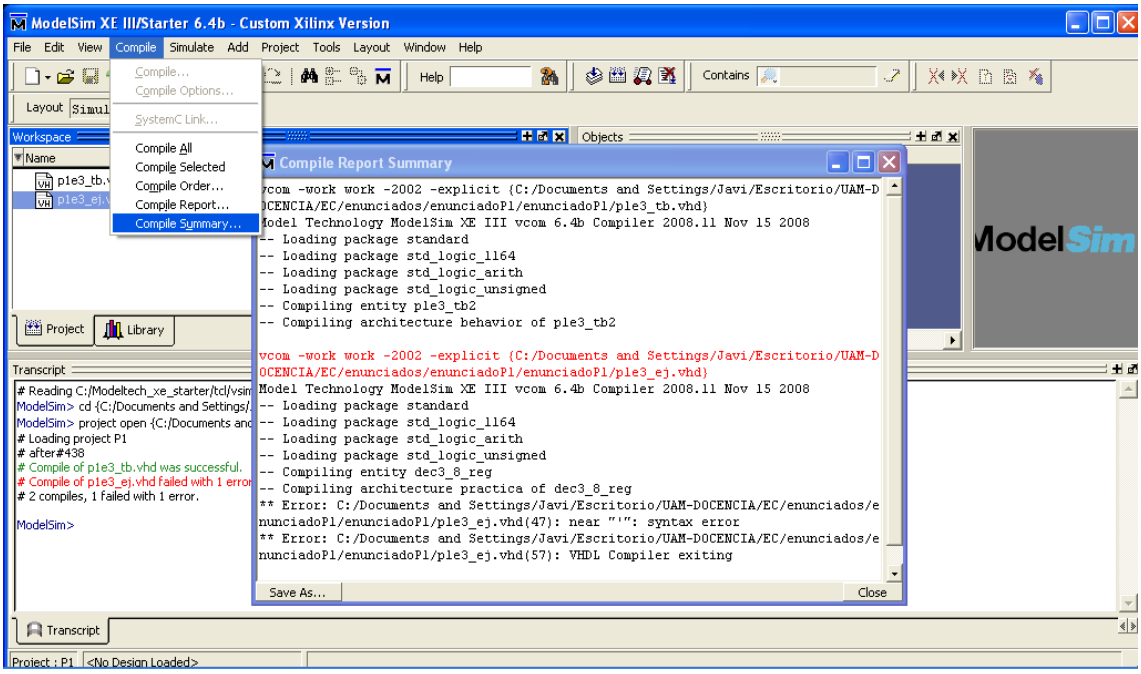
Compilación del proyecto.

Compile->Compile All
Los ficheros que componen el proyecto se encuentra en la pestaña *Project* de la ventana *Workspace*.

En caso de que haya un error, aparecerá una indicación en la ventana *Transcript*.

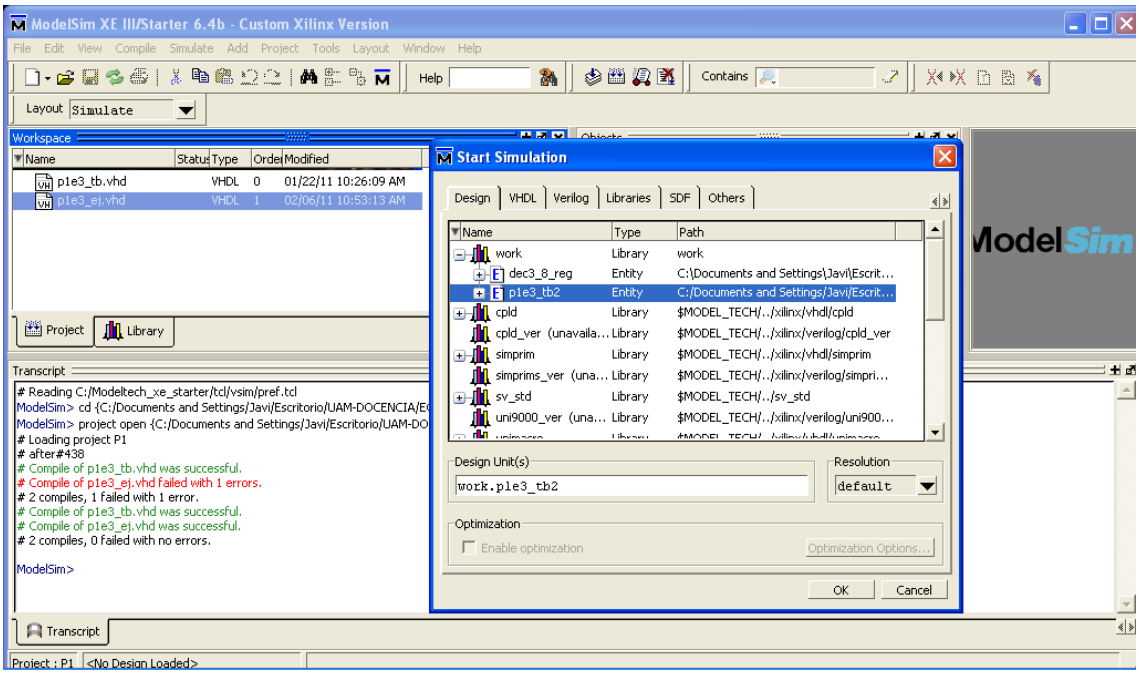
Hacer doble click sobre el mensaje de error o seleccionar *Compile->Compile Summary* hará que se abra una ventana con el listado de los errores. Abrir el fichero y editarlo picando en el fichero correspondiente en la lista que aparece en la pestaña *Project*, ventana *Workspace*.

Recompile hasta que no tenga más errores de sintaxis.

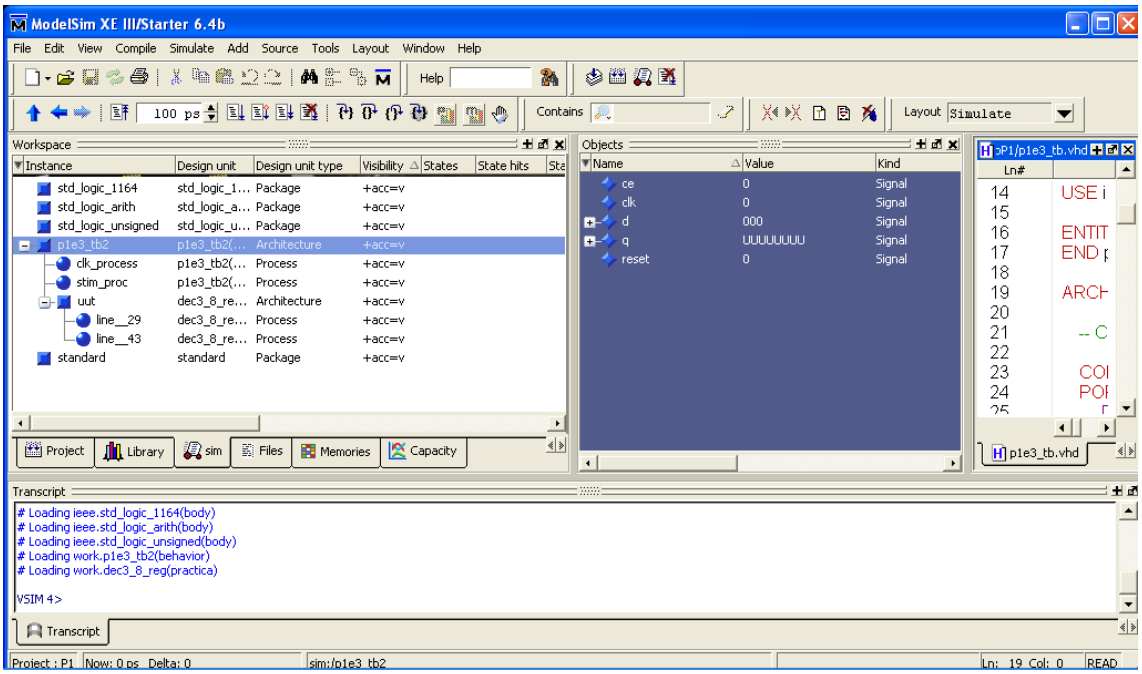


Simulación del diseño.

Seleccionar *Simulate->Start Simulation*
Se abrirá un diálogo donde se seleccionara el diseño a simular, dentro de la librería *work*. El diseño a simular será el “testbench”.



Se abrirán nuevas pestañas y ventanas dentro del área de trabajo. Ver más abajo:

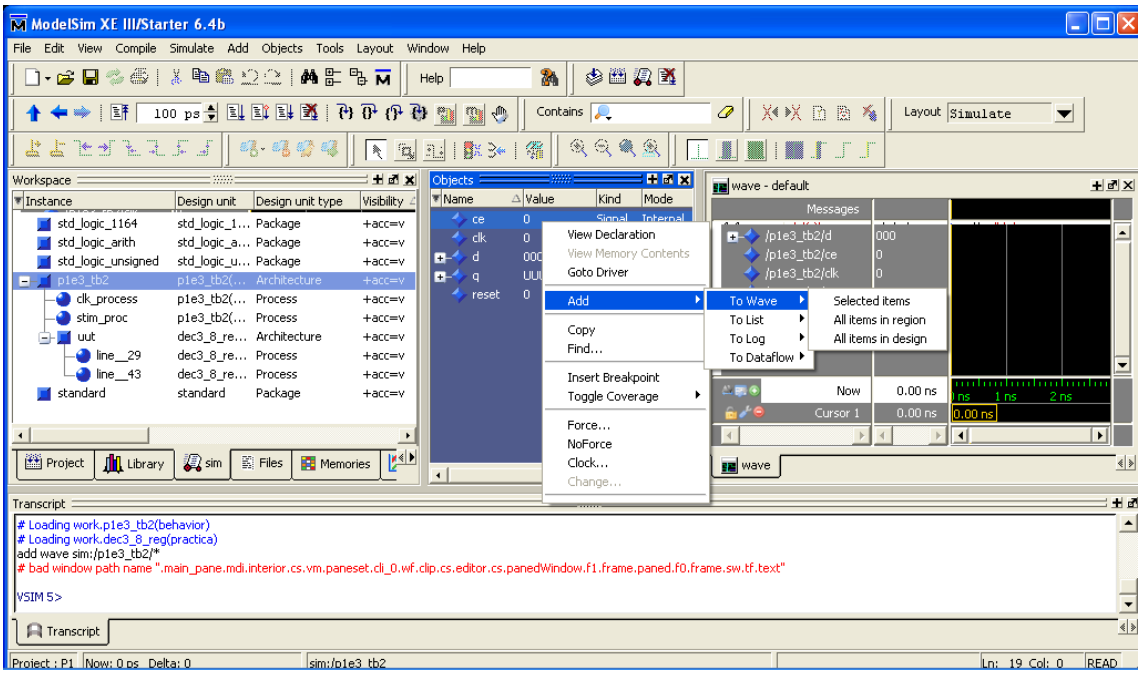


La pestaña *sim* en la ventana *Workspace* tiene una jerarquía de los objetos VHDL (packages, componentes, procesos,…)del diseño. Si se pica dos veces con el botón izquierdo del ratón sobre el objeto, se abrirá el fichero que contenga ese objeto posicionado en su declaración.

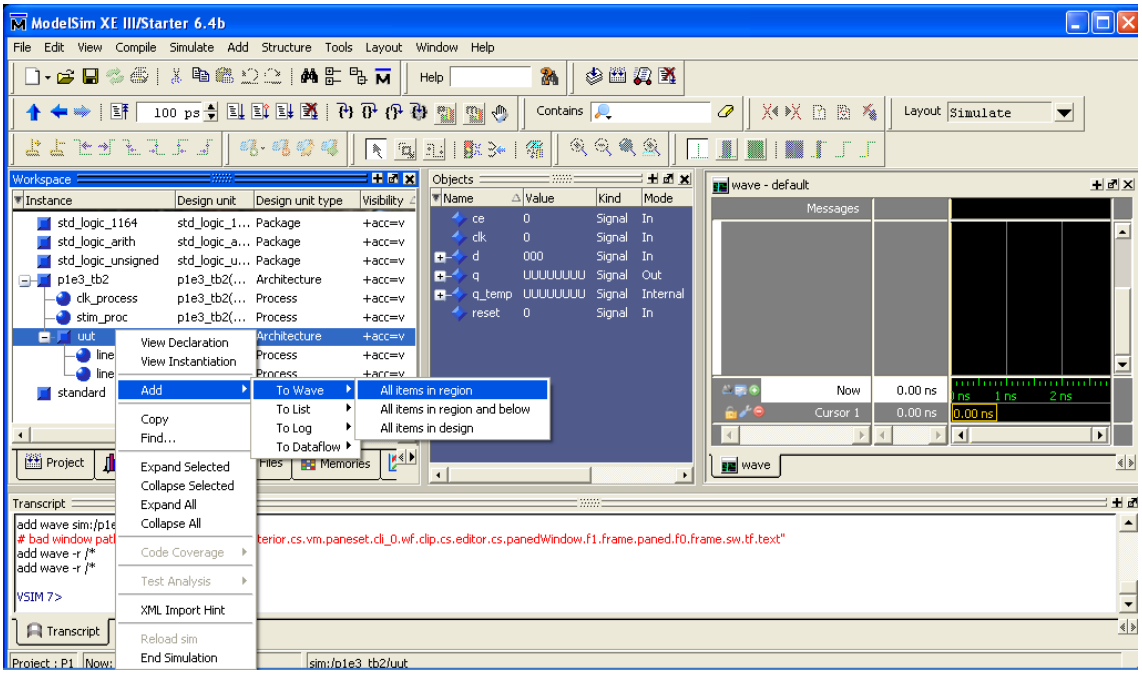
En la figura anterior, en la pestaña *sim,* ventana *Workspace*, las líneas *clk_process* y *stim_proc* debajo de *p1e3_tb2* corresponden a los procesos que con esas etiquetas se encuentran en *p1e3_tb.vhd*. A su vez, *uut* corresponde a la instanciación del componente *dec3_8_reg*. Es posible abrir esa línea, dando acceso a los procesos que hay en ese componente (fichero *p1e3.vhd*); como en este caso no tienen etiqueta, se identifican con el número de línea.

Agregando señales para la simulación.

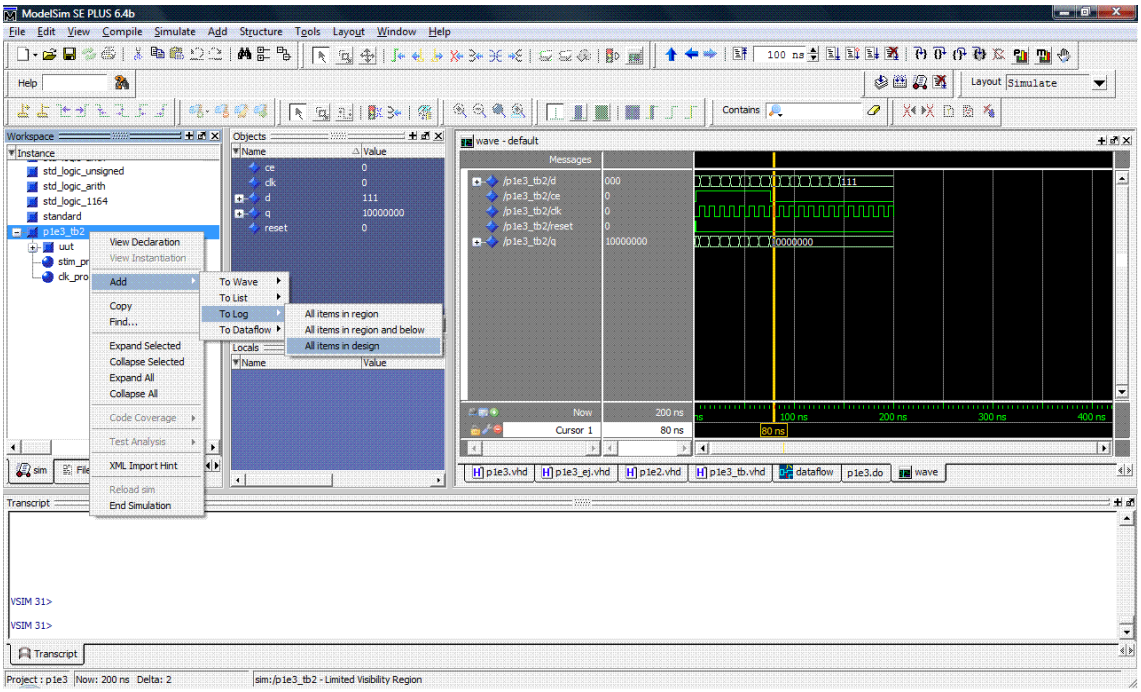
Si se selecciona *p1e3_tb2*, en la ventana *Objects* aparecen todas las señales que hay en ese bloque. Se pueden agregar esas señales a la ventana *wave* (que es donde se van a ver los resultados de la simulación) seleccionando las señales en la ventana *Objects* y con el botón derecho del ratón seleccionando a *Add ->To wave-> Selected Items* (o también *All ítems in region*). Esta operación también se puede hacer de forma similar seleccionando el objeto en la pestaña *sim*, ventana *Workspace*.



NOTE que si selecciona uut en la ventana sim, las señales que aparecen en la ventana Objects no son las mismas, sino las pertenecientes a este otro bloque. Así, pueden aparecer señales internas al bloque dec3_8_reg: en este caso aparece la señal *Q_temp*. De esta manera se pueden ver señales que están dentro de otros bloques que no sea el testbench.



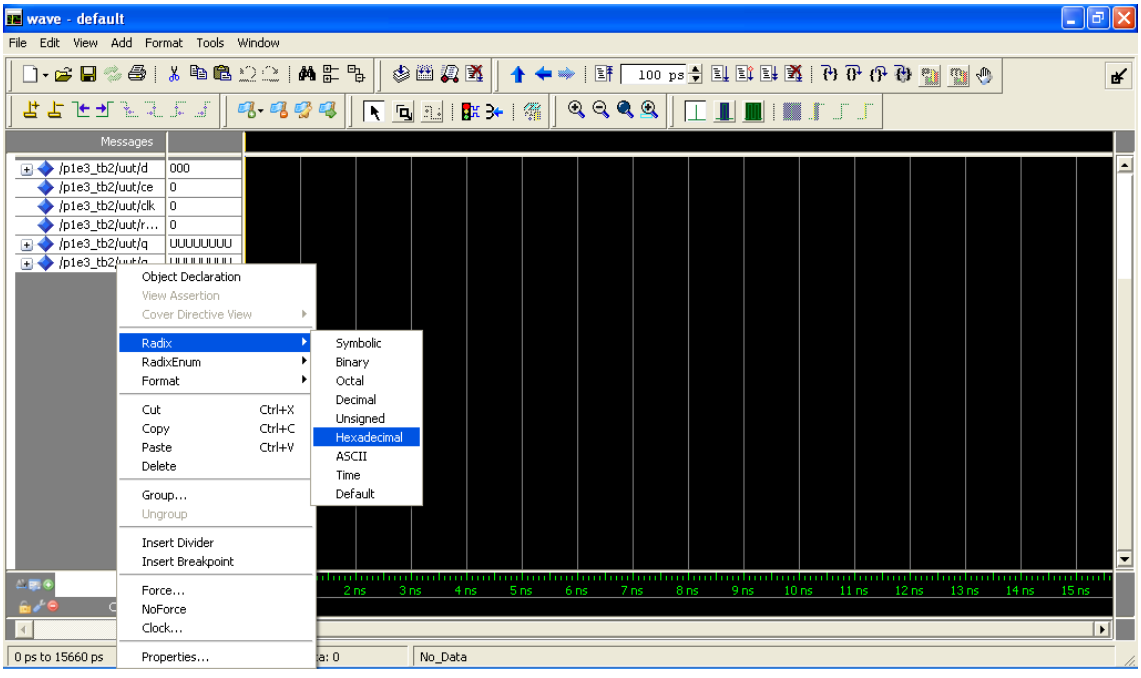
En principio, las señales tienen que estar agregadas en la ventana *wave* antes de iniciar la simulación. Si queremos que el simulador registre señales adicionales del diseño aunque por ahora no las muestre podemos añadir señales al *Log*. Por ejemplo, para que se registren todas las señales del diseño y que en cualquier momento podamos añadir más a la ventana de ondas sin tener que re-simular podemos ponernos sobre el testbench en la pestaña *sim* del *Workspace* y con el botón derecho del ratón seleccionar *Add->To Log->All ítems in design*.



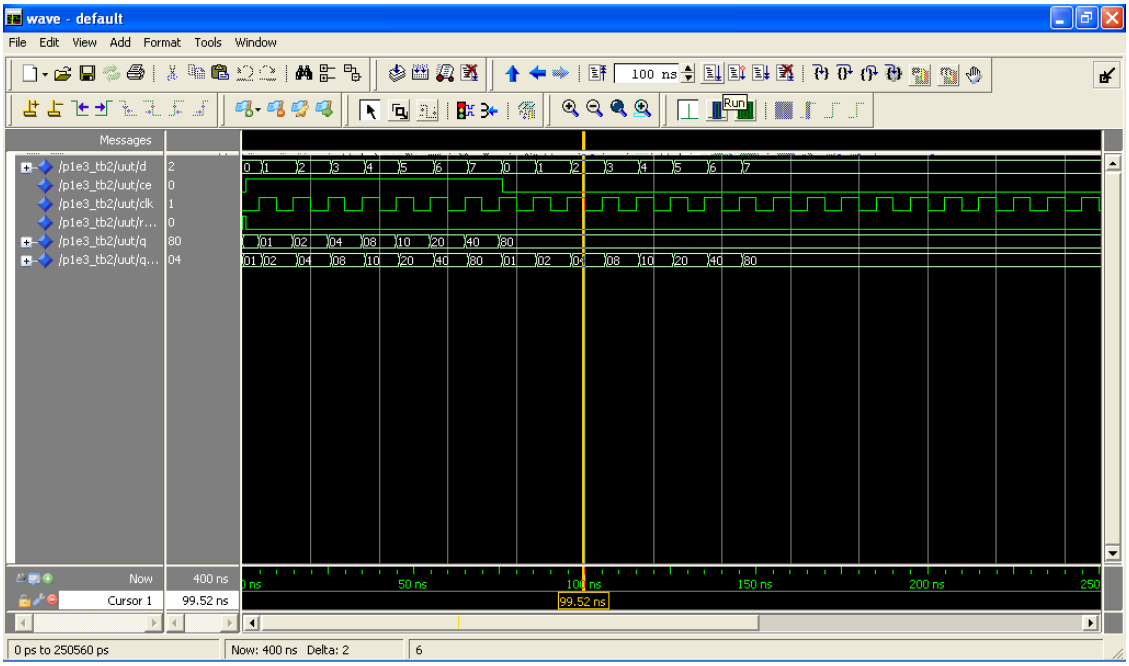
Si tenemos visualizadas bastantes señales puede ser útil dividirlas por grupos, insertando divisores con nombre, mediante el comando de menú *Add ->Divider* o directamente con el botón derecho en la zona de señales, *Insert Divider*. Las señales pueden también organizarse desplazándolas abajo o arriba mediante el ratón.

Controlando la simulación.

Una vez que están agregadas las señales a la ventana Wave, se puede definir representarlas en hexadecimal seleccionando las señales y con el botón derecho del ratón, *Radix-> Hexadecimal*. De la misma manera se pueden seleccionar otros formatos como decimal sin signo (*Unsigned*), etc. Los nombres de las señales adoptan un formato similar a los nombres de los ficheros en un directorio: */p1e3_tb2/uut/q* identifica la señal q dentro del bloque *uut*, dentro del bloque *p1e3_tb2*.

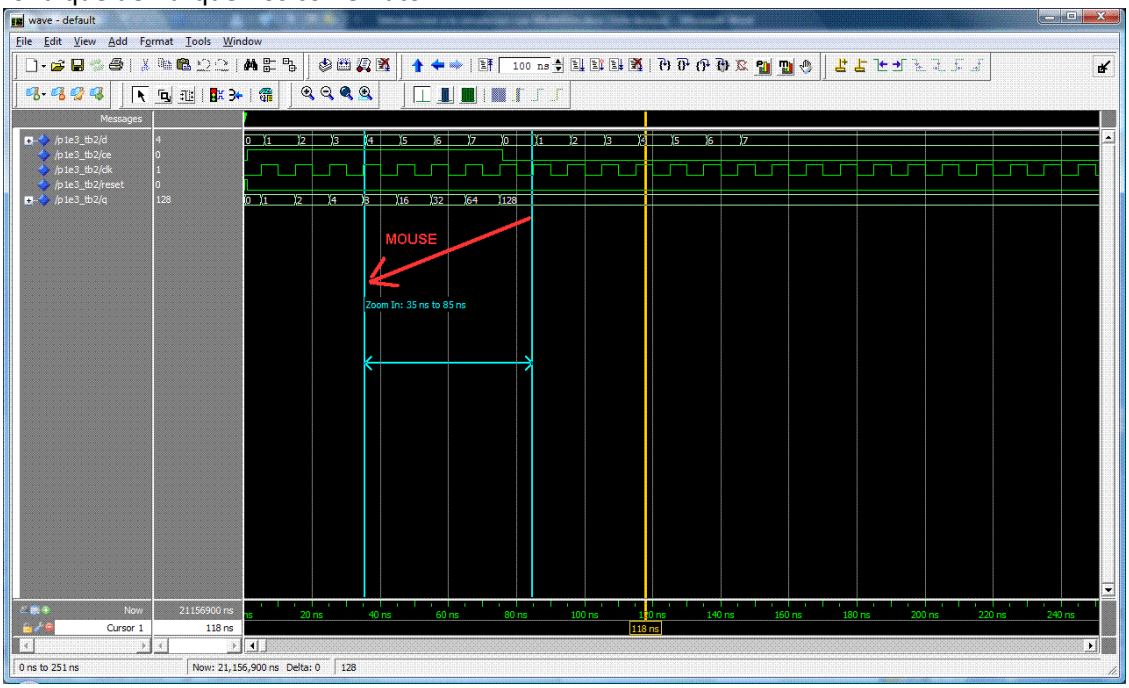


Para simular, pique en los botones *Run* o *Run –all*. El primero avanza la simulación el periodo de tiempo del formulario adyacente (ej.: 100 ns). El segundo avanza la simulación hasta que ésta termine en el testbench (si no hay un comando de parada de simulación, esta seguirá indefinidamente; en este caso se puede parar la simulación con la opción *Break* del menú *Simulate* o el botón equivalente).



Cuando un *assert* falla, aparecerá una flecha en la fila de *Messages* de la ventana de *wave*. Pasando el ratón sobre la flecha, sale el mensaje correspondiente del *assert* fallido, que también saldrá en la ventana de *Transcript*. La ventana de *Transcript* es la encargada de presentar todos los mensajes de texto que se generan en la simulación.

Para hacer *zoom* en una zona de la simulación, aparte de los botones con un dibujo de una lupa, es útil utilizar la tecla *Ctrl* a la vez que se hace un "*drag*" con el botón izquierdo del ratón. De esta forma, según la dirección a la que nos movamos se hará *zoom* hacia fuera (*out*), dentro (*in*) o vista completa (*fit*), siendo particularmente útil para hacer *zoom in* en la zona que demarquemos con el ratón.



Para reiniciar la simulación (por ejemplo para agregar más señales si no se registraron) pulse en el botón reiniciar, y mantenga todas las ondas que anteriormente había agregado.

Salvar el formato de las ondas

Es posible salvar el formato que damos a nuestra visualización de ondas en un fichero, guardando la información sobre señales mostradas, "radix" utilizadas y otras características del visor de ondas como uso de colores, separadores, etc. Para hacer esto basta con darle al botón de salvar en disco estando la ventana de ondas activa. Por defecto esto creará un fichero "wave.do", que es un fichero de script de ModelSim con los comandos necesarios para configurar la ventana de ondas, y que podrá ser cargado en este ventana con el comando de menú *Load*.

Comandos y scripts

La ventana *Transcript*, aparte de visualizar los mensajes del simulador, se comporta como una consola de comandos. Además de ejecutar los comandos propios de ModelSim, entiende muchos comandos habituales de Linux. Por ejemplo, podemos cambiarnos de directorio con *cd*, listar el contenido del directorio con *ls*, etc. Es también interesante conocer que con las teclas de flecha arriba y flecha abajo podemos movernos por los comandos ejecutados anteriormente, de forma que por ejemplo repetir la última simulación puede ser tan sencillo como pulsar la techa flecha arriba y luego la tecla *Enter*.

Una característica útil de ModelSim es que permite utilizar ficheros de script para lanzar sus comandos, ficheros que normalmente se nombran con la extensión ".do" (por ejemplo la herramienta *ISE* de Xilinx genera varios de estos, con diferentes extensiones ".*do", es la manera que tiene *ISE* de decirle a *ModelSim* qué hacer). Esto nos puede ser útil para lanzar rápidamente la compilación y simulación de un diseño incluso sin necesidad de tener un proyecto creado a tal efecto. A continuación se muestra un ejemplo de este tipo de scripts, que entre otras cosas hace uso del script de configuración de la ventana de ondas *wave.do* antes comentado. En el script se usa también el comando "*log –r /**" para que el simulador grabe todas las señales y se puedan añadir nuevas señales a la ventana de ondas sin tener que volver a simular.

Para ejecutar un script basta hacer en la ventana *Transcript*:
do <nombre_fichero_script>.

```
# Simulation script
# ------------------------------------

# Create library
vlib work

# Compile files
vcom -explicit -93 "p1e3.vhd"
vcom -explicit -93 "p1e3_tb.vhd"

# Run simulation. Resolution 1 ps, avoid nodes disappearing due to optimization.
vsim -t 1ps -novopt -lib work p1e3_tb2

# Record information about all signals
log -r /*

# Load waveforms
do wave.do

# Open windows
view wave
view structure
view signals

# Run full simulation
run –all
```

P1e3_tb2.vhd

```
USE ieee.std_logic_arith.ALL;

ENTITY p1e3_tb2 IS
END p1e3_tb2;

ARCHITECTURE behavior OF p1e3_tb2 IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT dec3_8_reg
    PORT(
         D : IN  std_logic_vector(2 downto 0);
         CE : IN  std_logic;
         CLK : IN  std_logic;
         Reset : IN  std_logic;
         Q : OUT  std_logic_vector(7 downto 0)
        );
    END COMPONENT;

   --Inputs
   signal D : std_logic_vector(2 downto 0) := (others => '0');
   signal CE : std_logic := '0';
   signal CLK : std_logic := '0';
   signal Reset : std_logic := '0';

  --Outputs
   signal Q : std_logic_vector(7 downto 0);

  constant CLK_period : time := 10 ns;
  constant espera : time := 1 ns;
  constant Ninput: integer := 3;

BEGIN
```

```
  -- Instantiate the Unit Under Test (UUT)
    uut: dec3_8_reg PORT MAP (
           D => D,
           CE => CE,
           CLK => CLK,
           Reset => Reset,
           Q => Q
         );

    CLK_process :process
    begin
          CLK <= '0';
          wait for CLK_period/2;
          CLK <= '1';
          wait for CLK_period/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
                  --Inicializacion de todo
          D <= "000";
          CE <= '0';
          Reset <= '1';
          wait for espera;

          -- Chip enable
          Reset <= '0';
          CE <= '1';
          for i in 0 to 7 loop
                  D <= conv_std_logic_vector (i,Ninput);
                  wait until CLK = '1';
                  wait for espera;
          end loop;

          -   Mas estimulos aquí …

     wait;
    end process;

END;
```

P1e3.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity dec3_8_reg is
    Port ( D : in  STD_LOGIC_VECTOR (2 downto 0);
           CE : IN  std_logic;
           CLK : IN  std_logic;
           Reset : IN  std_logic;
           Q : out  STD_LOGIC_VECTOR (7 downto 0));
end dec3_8_reg;

architecture Practica of dec3_8_reg is
signal Q_temp: std_logic_vector(7 downto 0);
begin
  process (D)
  begin
          case D is
                  --Otros casos a incluir
                  WHEN OTHERS => Q_temp <= "10000000";
          end case;
  end process;

  process (Reset, CLK)
  begin
          -- Codigo registro (a incluir)

  end process;

end Practica;
```