INTRODUCTION TO VLSI DESIGN WITH SYSTEM ON CHIP DESIGN REUSE:

A TUTORIAL FOR STUDENTS

by

Frank J. Ventura Jr.

An Applied Project in Partial Fulfillment
of the Requirements for the Degree
Master of Science in Technology

ARIZONA STATE UNIVERSITY EAST

March 2005

INTRODUCTION TO VLSI DESIGN WITH SYSTEM ON CHIP DESIGN REUSE:

A TUTORIAL FOR STUDENTS

by

Frank J. Ventura Jr.

has been approved

March, 2005

APPROVED:

______________________________________ ,Chair

Dr. Narciso F. Macia

____

Dr. Albert L. McHenry

____

Dr. Don Cottrell

Supervisory Committee

ACCEPTED:

____

Dr. Lakshmi V. Munukutla
Department Chair

ABSTRACT

Due to the absence of a formal class in VLSI design at ASU East, the need arose for a hands-on tutorial that would introduce the students to VLSI design, emphasizing System-on-a-Chip design and the concept of design reuse.

When UET 513 – Introduction to VLSI Design was taught at ASU East, The Western Design Center Inc. (WDC) supported this class by donating industry used software tools and their microprocessor Intellectual Property (IP) for the class to use in learning the concepts of VLSI Design. WDC continues to support the students at ASU East that are interested in VLSI design, by offering internships to give the students experience in the VLSI design flow. Students interested in using this tutorial will be able to work at WDC's office and have access to the software tools and technology. The students will be following the design flow as used by WDC. The students will be exposed to the following design tools: ViewDraw for schematic entry, Silos for Verilog HDL simulation, ICED for laying out an IC, and PSPICE for electrical characterization. These tools, used in conjunction with the microprocessor technology of WDC, provide the students with a hands-on experience in VLSI design methodology. The students will also learn the value of design reuse by utilizing the standard cell library created by WDC.

The tutorial that was created introduces the students to the tools, concepts, methodology, and history of VLSI design. The students will gain hands-on experience by performing exercises related to each step of the VLSI design flow as it is used in industry. Thus, the students' introduction to VLSI design with system-on-a-chip design reuse will be complete.

DEDICATION

This project is dedicated to my parents, Frank and Lorraine Ventura, and my family for all of their support.

ACKNOWLEDGEMENTS

I would like to thank my committee, Dr. Narciso F. Macia, Dr. Albert L. McHenry and Dr. Don Cottrell for your patience, support and input on this project.

Special thanks to William D. Mensch Jr. of The Western Design Center, Inc. for allowing me to use the 65xx technology and WDC's design tools, and for all the support, guidance and technical assistance that you have provided to me.

I would also like to thank the following people, who in some way were involved in getting me through this project: Lou Prado, Lars Dannemann, Dianne Mensch, Jack Daniels, Dr. Rajeswari Sundararajan, Linda Deutsch, Four Peaks, Prakash Moparthi, Sam Buca, Anna Wales and Betsy Allen.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Background

A major factor in the electronics industry today is to make devices that are small and fast, and to get these devices to market in the fastest time possible. With the many advances in process technology over the years, it is now possible to integrate whole systems on a single chip. In the past, a system consisted of a printed circuit board that had on it one chip for RAM (Random Access Memory), one chip for ROM (Read Only Memory), one chip for a microprocessor, one chip for I/O (Input/Output) capabilities, and perhaps chips for A to D (Analog to Digital) converters, D to A (Digital to Analog) converters, and UART's (Universal Asynchronous Receiver Transmitter). Today, one chip can contain a microprocessor core, RAM, ROM, I/O, A to D and so on. Thus we get the name System-on-a-Chip, (SOC). This is all possible due to VLSI (Very Large Scale Integration).

There are many aspects to VLSI design. There is the design of the system itself, which today is widely done using HDL's (Hardware Description Languages). The schematic entry method is also used in system design. This is widely done by dropping in reusable Intellectual Property (IP) cores into the design. These IP cores are proven to be reliable by the manufacturer and can help speed up the design process. Then there is the verification of the design, running simulations to determine if the design is working as expected. There is the layout of the design where the layers that make up the circuit are physically defined. Layout can be done by placing and routing the design manually, or it can be done by an automatic place and route tool. After the design has been laid out and verified, the final step is to target the design for a manufacturing process and send the design out to be fabricated. Typically, prototypes are fabricated first, in small quantities, to keep expenses down in case the design is faulty. After evaluating the performance of prototype devices, the design needs to be reworked if there are problems, or sent for fabrication in larger quantities if the design is performing as expected. Figure 1.1 shows a flowchart for the system-on-chip design process that is used by The Western Design Center, Inc. (WDC). A class in VLSI design is instrumental in giving the student an introduction to the theory and tools that are used in the development of an SOC.

In the early days of system chip design, Negative Metal Oxide Semiconductor (NMOS) was the primary process used. The number of transistors on a single chip was in the thousands and the gate size was .5 microns or larger. The entire design process was very time-consuming as the circuit layout was manually compared to the circuit's schematics.

Over the years, the advent and refinement of HDL's has allowed designers to develop the chip by describing the design's hardware. Complementing HDL's, many tools were developed for design automation. These tools could place and route a design, perform a design rule check of the design, and simulate the design to ensure proper functionality.

Today, Complimentary Metal Oxide Semiconductor (CMOS) is the prevalent process used and transistor counts on a single chip are in the millions. The gate sizes of these transistors are .25 microns or less. Design automation has made the design of complex system chips faster and more efficient, cutting down on production time and increasing the time to market.

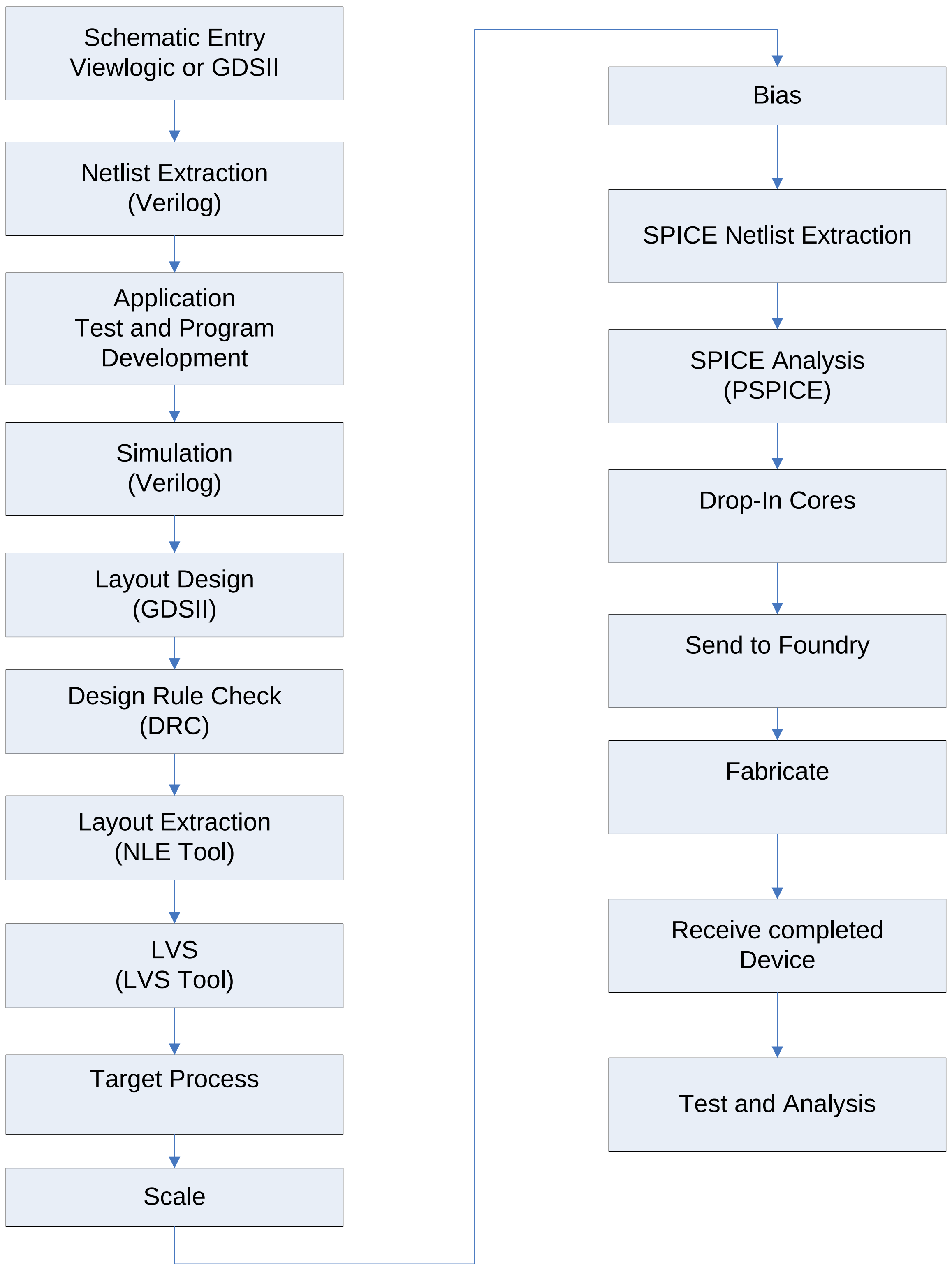


Figure 1.1 WDC flow chart for SOC design, manufacturing and test

1.2 Problem

The original intent of this project was to develop a much needed laboratory manual that was to accompany the class UET 513 – Introduction to VLSI Design. This was a class that had been taught at ASU East for five years, without any structured lab to coincide with the theory presented. Unfortunately, UET 513 is no longer offered at ASU East, leaving a void for students interested in pursuing a career in VLSI design.

The focus of this project has shifted from an accompanying lab manual to a hands-on tutorial that will introduce students to the concept of reusable IP and to some of the tools that are used in creating a SOC. This tutorial will provide the student with the necessary means in understanding the VLSI design flow.

1.3 Scope

The tutorial will introduce the students to the SOC design flow and the technology used at The Western Design Center Inc. WDC is located in Mesa, AZ, and was responsible for donating the computers, software tools, and IP that was used in UET 513. The intent of this tutorial is to provide an ***introduction*** to VLSI design. Each of the sections will introduce the student to a different software tool, or program, used in the SOC design flow. It is important to note that only the basic elements of the tools will be presented to the student. Again, the objective of the tutorial is to *introduce* the student to the tool, not make them experts in using the tool. The student will use technology already available to them, showing them the value of design reuse. This will give the student an appreciation of the tools available to them, and help them understand how VLSI design is done today.

1.4 Assumptions

It is assumed that senior level undergraduate students and graduate students, will be using this tutorial, and will have had little or no prior experience in VLSI design, but have some knowledge of semiconductor terminology and theory. It is also assumed that a tutorial is necessary to assist the student in learning the tools and techniques used in the SOC design flow. Furthermore, it is assumed that the students will be using this tutorial at the offices of WDC, where the tools will be available for their use. If the student would like to perform this tutorial, the student must contact WDC, sign a Non-Disclosure Agreement (NDA), and arrange with WDC times that the student can go to the WDC office to use the design tools.

1.5 Sequence of Presentation

Chapter 2 contains:

- Separate sections for each step of the design flow
- Introductory information about each step
- Introduction to the design tool used in each step
- Simple hands-on example demonstrating the usage of each tool

Chapter 3 contains:

- The tutorial using the components of the W65C122S SOC

Chapter 4 contains:

- Conclusions and recommendations

The Appendix contains:

- Information regarding Internships at The Western Design Center
- Contact information for The Western Design Center, Inc.

2. VLSI Design Flow

2.1 Design entry

The first step in the VLSI design flow is creating, or entering, a design. “The purpose of design entry is to describe a microelectronic system to a set of electronic design automation (EDA) tools” (4:327)*. There are two ways to accomplish this task. The first way is to enter the design via schematic entry, or schematic capture, where gates, symbols and interconnects are drawn using a computer program. The second way is to enter the design using a Hardware Description Language (HDL) such as Very High Speed Integrated Circuit Hardware Description Language (VHDL) or Verilog HDL. With this method, the designer describes the hardware by using software, and the code can be written using any text editor. After the code has been written, the code must be tested and debugged.

In the early 1980s, before the advent of schematic design entry tools, schematics were drawn using a graphics editor or drawn by hand. This provided a picture of the schematic with no functionality behind the schematic. From this picture, the designer manually coded the netlist of the circuit that was used for simulation. So in reality, HDLs were used before schematic entry tools for circuit design.

The Western Design Center, Inc. (WDC) has an interesting approach for their core designs in that they use the mask design editor to create a detailed “floor planning” schematic. The schematic can be edited and plotted on any Graphic Design System II (GDSII) editor, making it available to a wide range of users. GDSII is an Electronic Design Automation (EDA) industry standard binary format of the design that is used to transfer mask design data to a wafer fabrication shop (FAB).

* Number in parenthesis indicates the reference at the end of this document.

The Verilog structural HDL is manually created by WDC and this corresponds to the GDSII schematic. This approach provides for complete control of naming every node manually, and helps in the mask design and debug phase of the design.

2.1.1 Schematic entry

There are several powerful software programs that can be used to enter a design using schematic entry. Cadence Design Systems and Mentor Graphics make some of the more popular, widely used ones. With each of these programs, the designer can use the manufacturers' standard cell library, or import a custom cell library to draw the design. Figure 2.1 shows an example of a design that has been drawn using ViewDraw. ViewDraw is the schematic entry program in Mentor Graphics eProduct Designer (ePD) design suite. The schematic is the address decoder of the W65C122S.

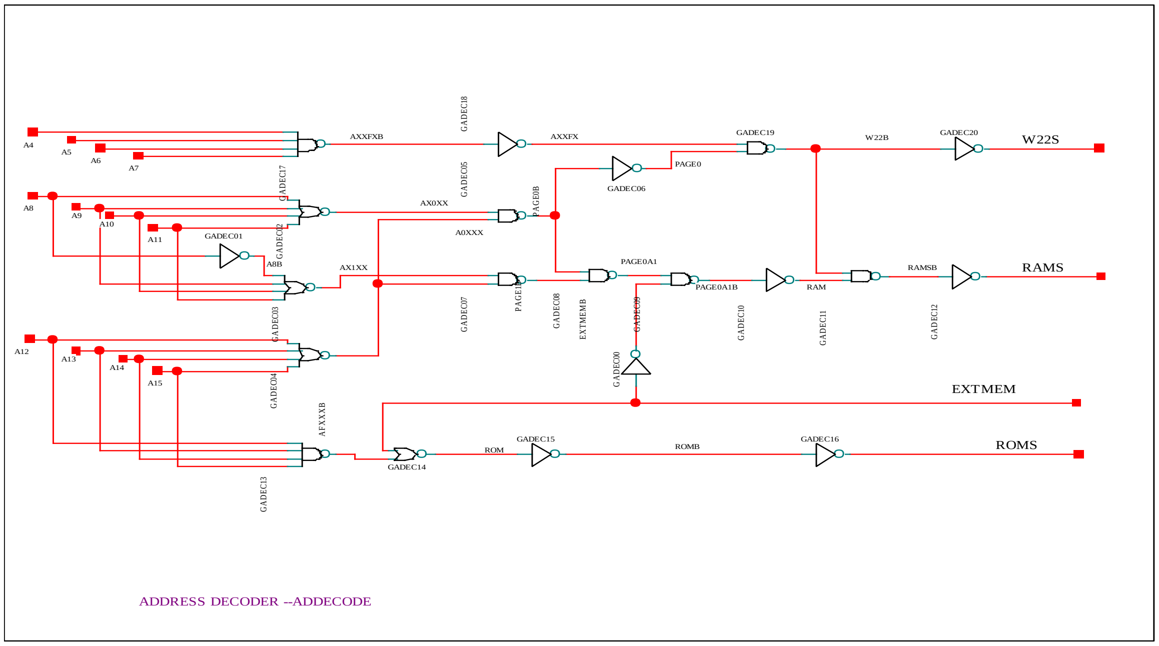


Figure 2.1 Example of schematic entry – W65C122S address decoder

As can be seen in Figure 2.1, the schematic is a graphical representation of the design. This method is sometimes preferred because it provides an easily understood picture of the circuit. More importantly, it is not just a picture of the design, but a functional description of the design. The main goal in doing schematic design entry is to obtain an output file that can be used to simulate the circuit. Once the schematic is finished, the designer can extract a netlist of the design. This netlist is the output file that is used for circuit simulation. Circuit simulation will be discussed in section 2.2.

2.1.2 Hardware Description Languages (HDLs)

There are two main HDLs used for design entry in industry today. One is VHDL and the other is Verilog HDL. While both packages are excellent at modeling hardware structures, there are differences in each one. Therefore, choosing the package to use depends on personal preferences, EDA tools available, business and marketing issues (4:10). Figure 2.2 shows a comparison of VHDL and Verilog HDL for a Serial Adder/Subtractor. For an in-depth comparison of the two languages, see (4:10-14).

2.1.2.1 VHDL

VHDL "can be used to model a digital system at many levels of abstraction, ranging from the algorithmic level to the gate level" (1:1). With VHDL, the designer can describe the system in concurrent or sequential fashion, and may wish to include timing characteristics in the design.

There are four different ways to express the architecture of a system. The first is the structural method, where the system is "described as a set of interconnected components" (1:14). The second method is the data flow method, which uses concurrent signal assignment statements. The third method is the behavioral method, where the system is described “as a set of statements that are executed sequentially in the specified order” (1:18). The last method is mixed style modeling which incorporates a mixture of the three previously mentioned methods.

2.1.2.2 Verilog HDL

Described as an easy to learn and use HDL, Verilog HDL is a general-purpose HDL whose syntax is very similar to C and PASCAL. As with VHDL, systems can be described at different levels of abstraction. “The designer can define a hardware model in terms of switches, gates, RTL (register transfer level), or behavioral code” (3:7).

The highest level of abstraction is the behavioral level. At this level, the designer is concerned with describing the behavior of the circuit, and not how the circuit will be implemented using gates (3:115). At the data flow level, the system is described by specifying the actual data flow between registers, with knowledge on how this data is processed in the overall design. At the gate level, the system is described in terms of the individual logic gates and the interconnections between them. Finally, the switch level is the lowest level of abstraction, where the design “can be implemented in terms of switches, storage nodes, and interconnections between them” (3:16).

The designer is also able to mix each level of abstraction in the design. The combination of behavioral and data flow is commonly called RTL. When higher levels of abstraction are used, the design is more flexible and technology independent. Designs are more technology dependent and inflexible when lower levels of abstraction are used (3:16).

VHDL

```
library IEEE:
use IEEE.STD_Logic_1164.all; IEEE.Numeric_STD.all;

entity ADD_SEQ is
   port (Clock, Reset: in  std_logic;
   ParaLoad, Serialin, EnableShiftAdd: in std_logic;
   CoeffData: in  unsigned(7 downto 0);
   ParallelOut: out unsigned(7 downto 0));
end entity ADD_SEQ;

architecture RTL of ADD_SEQ is
   component FULL_ADD
      port (A, B, Cin: in std_logic; Sum, Cout: out std_logic;
   end component:
   signal ShiftRegA, ShiftRegB: unsigned(7 downto 0);
   signal Sum, Cout, HoldCout: std_logic;
begin
   REG_AB: process (Clock)
   begin
       if rising_edge(Clock) then
      -----------------------
      --- Shift register A
      -----------------------
      if (ParaLoad = '1') then
         ShiftRegA <= CoeffData;
      elseif (EnableShiftAdd = '1') then
         Shift_RegA <= rotate_right(ShiftRegA, 1);
       end if;
      -----------------------
      --- Shift register B
      -----------------------
      if (EnableShiftAdd = '1') then
          ShiftRegB <= rotate_right(ShiftRegB ,1);
      end if;
     end if;
end process REG_AB;
ParallelOut <= ShiftRegB;
      ----------------------------
      --- Single bit full adder
      ----------------------------
FA1: FULL_ADD port map
   (A => ShiftRegA(0), B => ShiftRegB, 0),
     (Cin => HoldCout, Sum => Sum, Cout => Cout);
      -------------------------------------
      --- Hold carry out for next add
      -------------------------------------
HOLD_COUT: process (Clock, Reset)
begin
   if (Reset = "0") then
      HoldCout <= '0';
   elseif rising_edge(Clock) then
      if (EnableShiftAdd = '1') then
         HoldCout <= Cout;
      else
         HoldCout <= HoldCout;
         end if:
      end if;
     end process HOLD_COUT;
 end architecture RTL;
```

Verilog

```
module ADD_SEQ
     (Clock, Reset, ParaLoad, CoeffData, Serialin,
      EnableShiftAdd, ParallelOut);
        input            Clock, Reset;
        input            ParaLoad, Serialin, EnableShiftAdd;
        input     (7:0) CoeffData;
        output    (7:0) ParallelOut;

         reg             ShiftRegA_LSB;
         reg      (7:0) ShiftRegA, ShiftRegB;
         wire            Sum, Cout;
         reg             HoldCout;

     always @(posedge Clock)
        begin: REG_AB
           //----------------------
           //  Shift Register A
           //----------------------
           if (ParaLoad)
              ShiftRegA = CoeffData;
           else if (EnableShiftAdd)
                begin
                   ShiftRegA_LSB = ShiftRegA(0);
                   ShiftRegA = ShiftRegA >> 1;
                   ShiftRegA(7) = ShiftRegA_LSB;
                end
            //------------------------
            // Shift Register B
            //------------------------
            if (EnableShiftAdd)
               begin
                 ShiftRegB = ShiftRegB >>1;
                 ShiftRegB(7) = Sum;
               end
         end
         assign ParallelOut = ShiftRegB;
   //-----------------------------
   // Single bit full adder
   //-----------------------------
   FULL_ADD FA1
    (.A(Serialin), .B(ShiftRegA(0)),
    .Cin(HoldCout),
    .Sum(Sum), .Cout(Cout));

    //-----------------------------------
    // Hold carry out for next add
    //-----------------------------------
    always @(posedge Clock or negedge Reset)
       begin: HOLD_COUT
         if (!Reset)
           HoldCout = 0;
         else if (EnableShiftAdd)
             HoldCout = Cout;
         else
             HoldCout = HoldCout;
       end

 endmodule
```

Figure 2.2 Comparison of VHDL and Verilog for a serial adder/subtractor

2.1.3 Introduction to ViewDraw

As previously mentioned, The Western Design Center, Inc. uses ViewDraw for their schematic entry tool. ViewDraw is part of the Mentor Graphics ePD tool suite. This section will provide a basic introduction to ViewDraw, introducing the design environment, WDC's standard cell library, and the schematic editor.

2.1.3.1 Introduction to creating a schematic in ViewDraw

The first step in the process of drawing a schematic is to create a project where all libraries and design files are stored. To create a project, double-click on the eProduct Designer 2004 icon on the desktop. This will launch the ePD Dashboard, where the project is created and the project hierarchy is stored. After opening up Dashboard, your screen will look similar Figure 2.3.

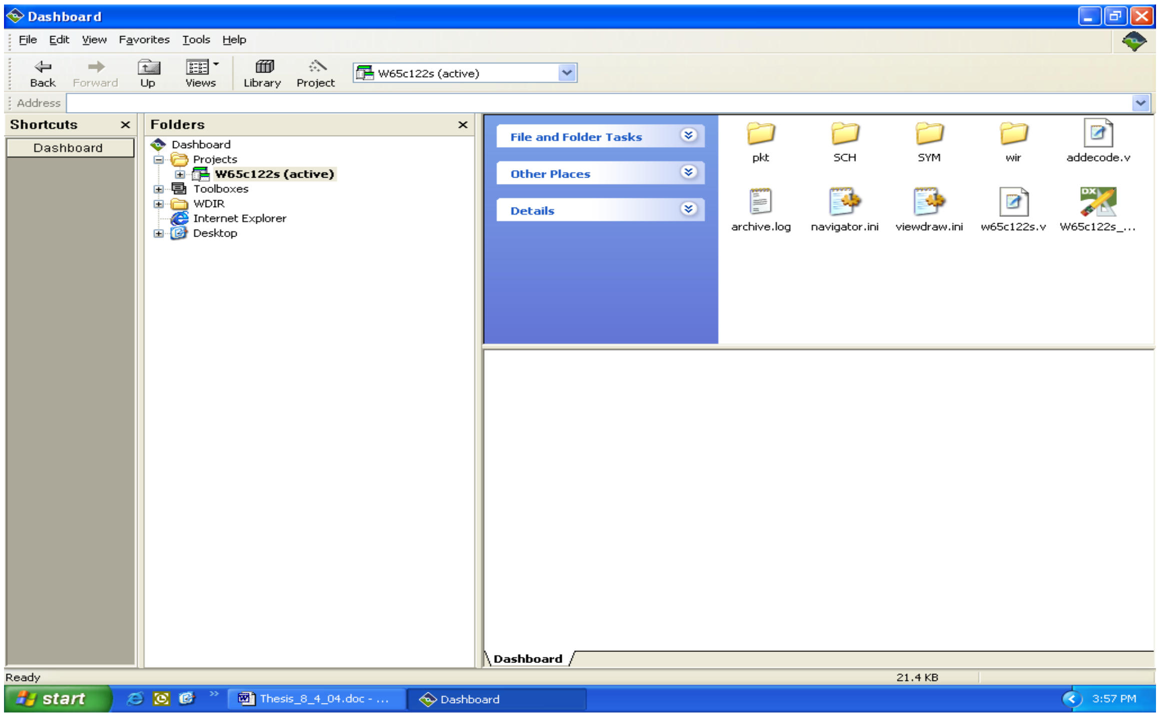
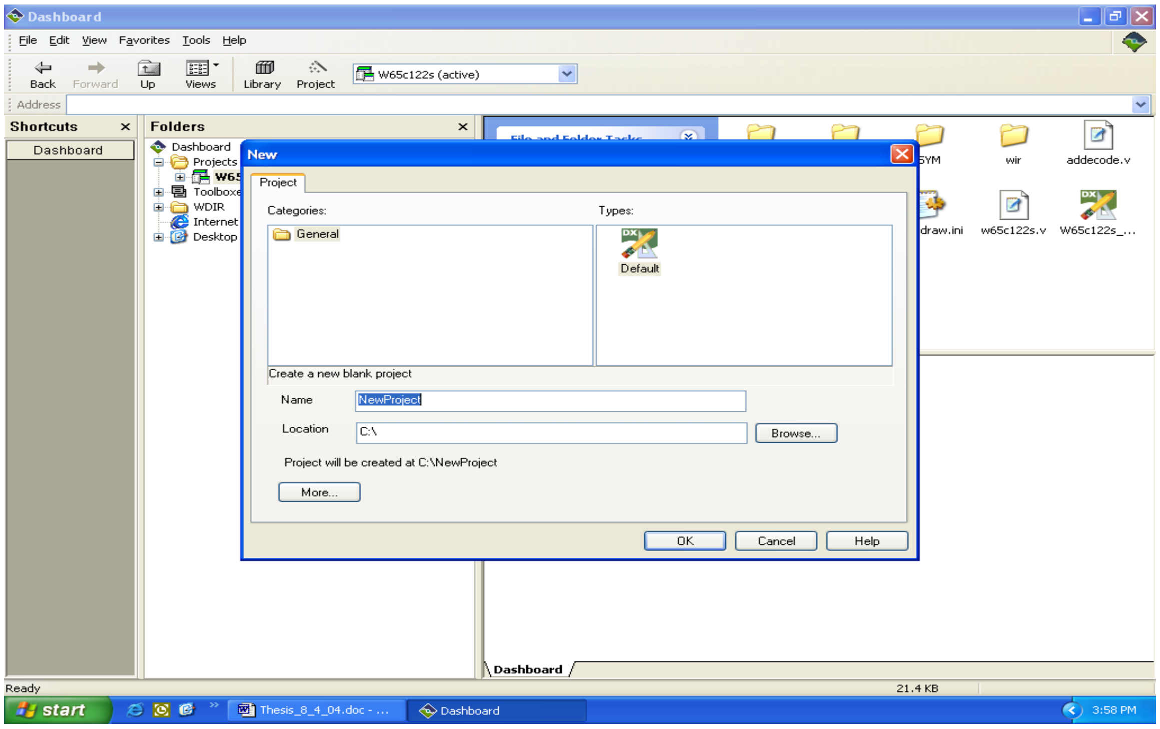


Figure 2.3 ePD dashboard main window

Click on File → New → Project. Your screen will look similar to Figure 2.4.



Categories: General

Types: Default

Create a new blank project

Name: NewProject

Location: C:\ Browse...

Project will be created at C:\NewProject

More...

Figure 2.4 ePD new project window

In the Name field, type in a name for the project. For this example, type in VDExample for ViewDraw Example. Notice that the text below the Location field states that the project will be created in C:\VDExample. This can be changed by using the Browse button if desired, but for this example, we will not change this directory. Next, click OK. Your screen will look similar to Figure 2.5.

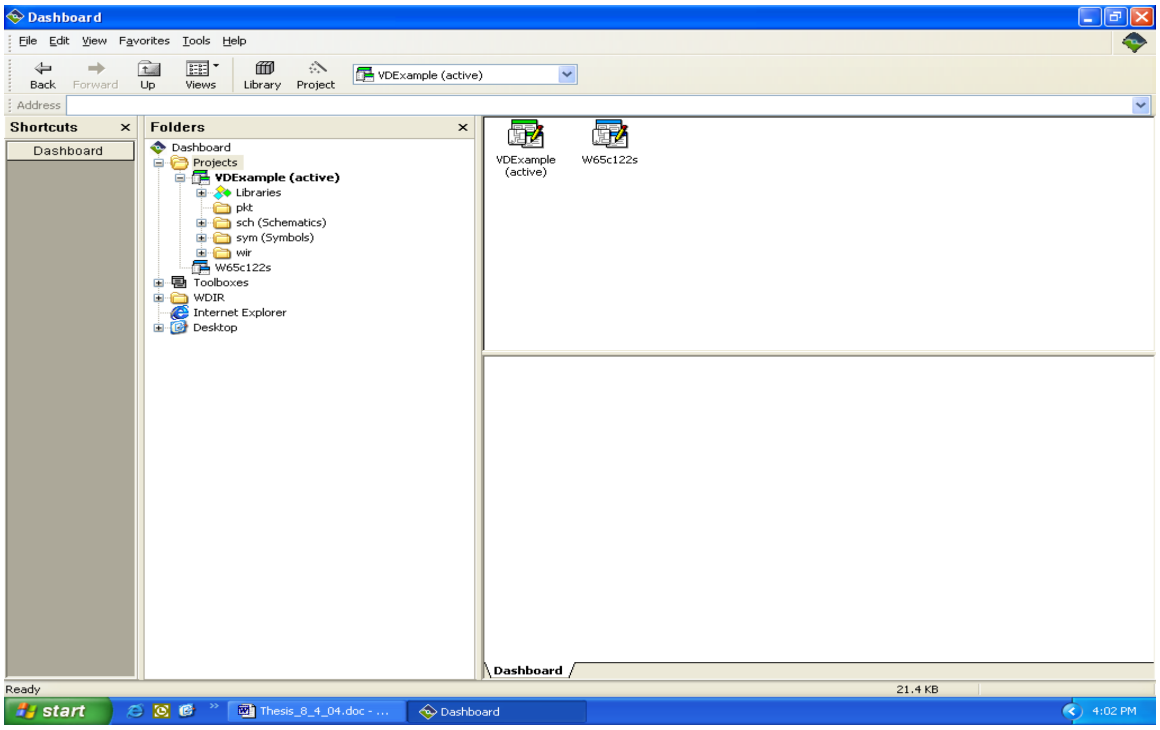
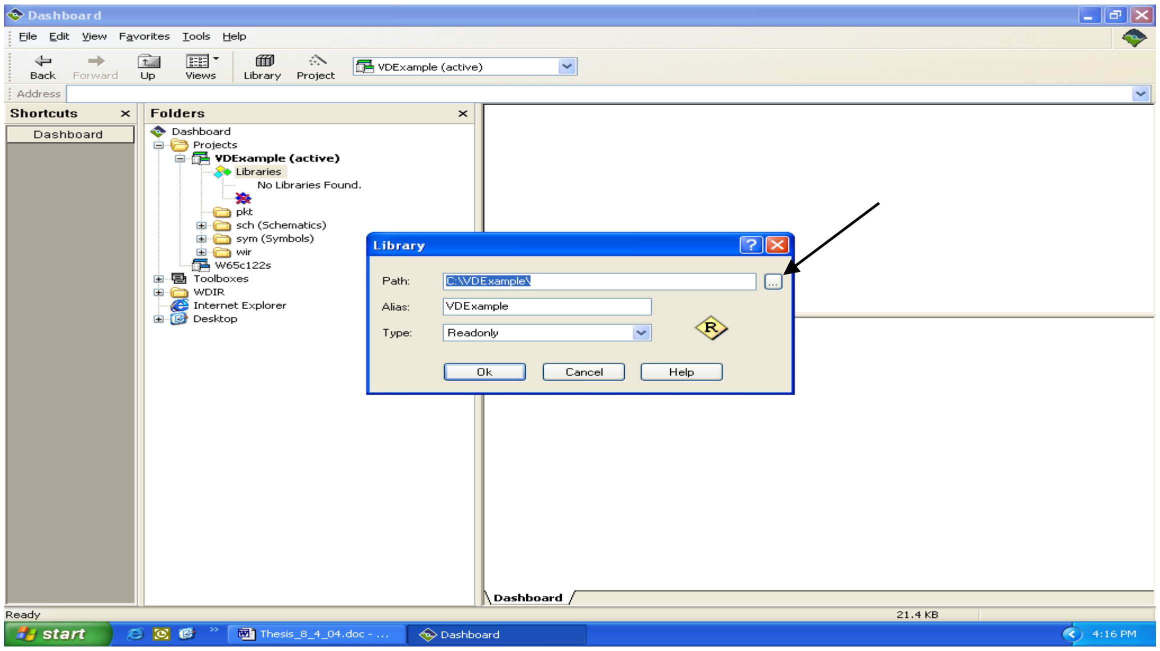


Figure 2.5 ePD dashboard view of projects

Figure 2.5 shows the Dashboard after the creation of the project VDExample. Notice that VDExample is listed in bold and has the word active next to it. This indicates that VDExample is the active project and any work done will be saved in the VDExample project hierarchy.

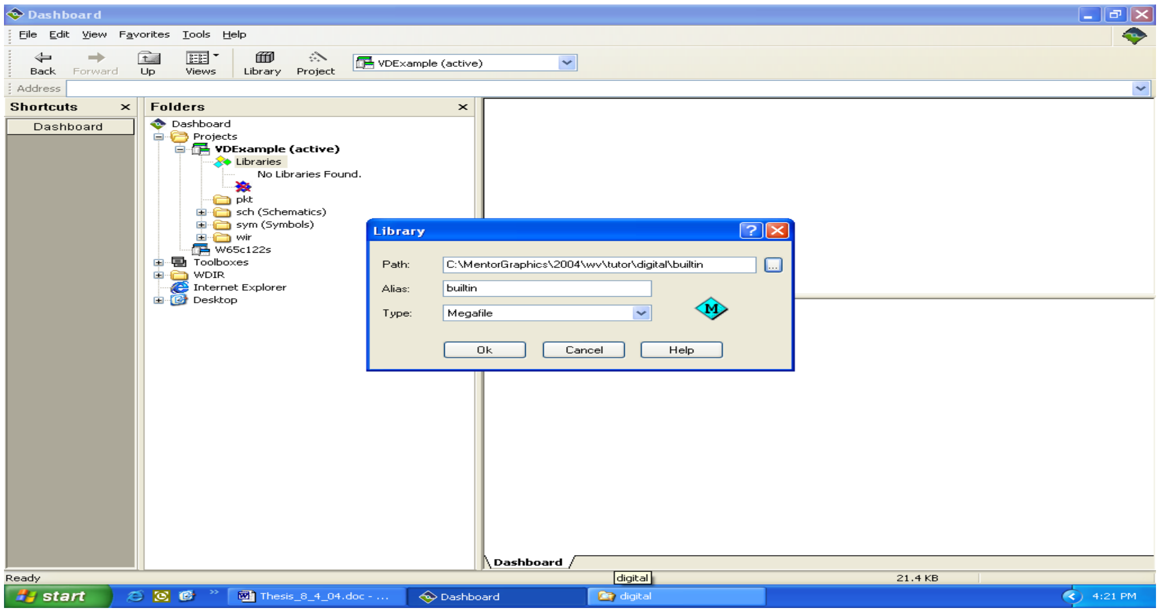
The next step is to import the library or libraries that we will use to create a schematic. The library will have all of the schematic symbols that will be placed onto the schematic drawing. For the purposes of this example, we will be using the builtin library that is standard with ViewDraw. To add this library, click on the Library icon on the toolbar. Your screen will look similar to Figure 2.6.



Path: C:\VDExample\
Alias: VDExample
Type: Readonly

Figure 2.6 ePD add library window

Click on the Browse button next to the Path field, as shown in Figure 2.6, and browse to the following directory: C:\MentorGraphics\2004\wv\tutor\digital. Select the builtin directory and click OK. Your screen will look similar to Figure 2.7.



Path: C:\MentorGraphics\2004\wv\tutor\digital\builtin
Alias: builtin
Type: Megafile

Figure 2.7 ePD library window

Click OK. Your screen will now look similar to Figure 2.8.

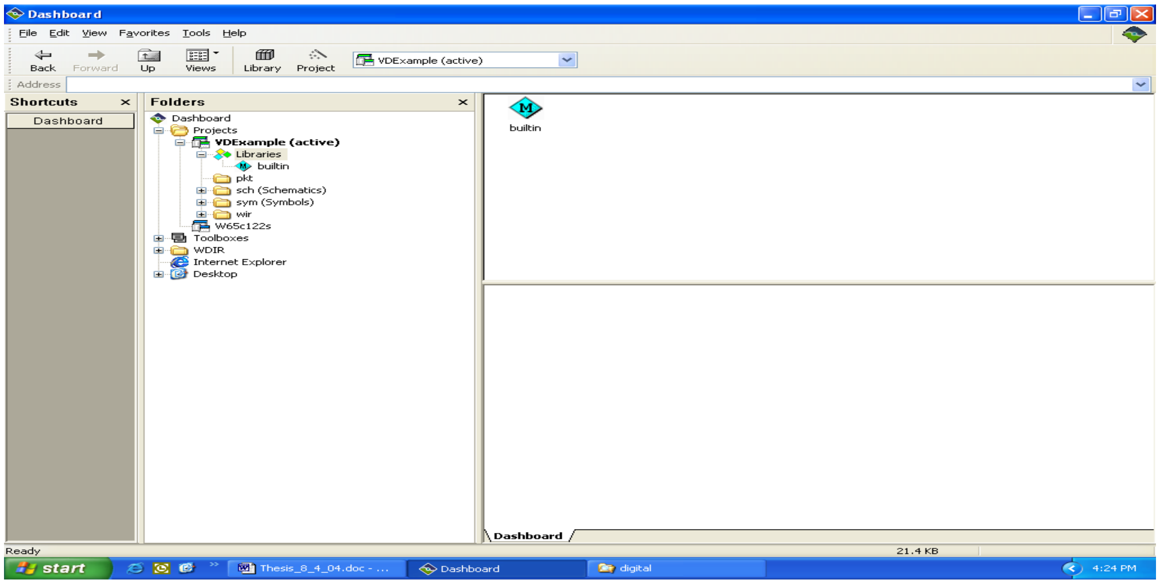


Figure 2.8 ePD dashboard view with builtin library added

Notice now that the builtin library is listed under the Library directory, indicating that the library has been successfully added to the project. We are now ready to begin entering our schematic into ViewDraw. To do this, click on the project VDExample (active). Your screen will look similar to Figure 2.9.

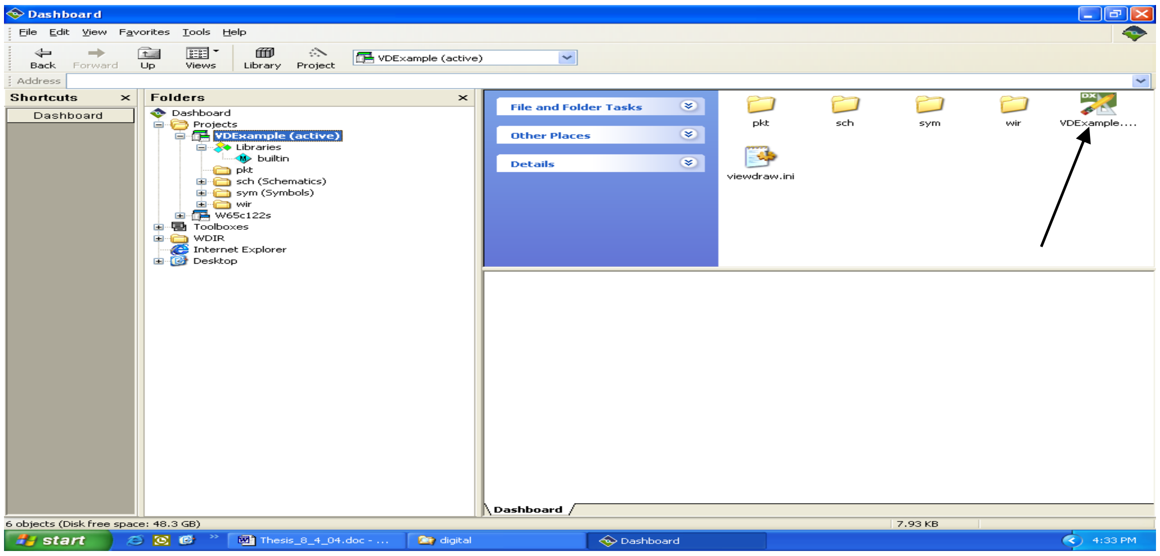


Figure 2.9 ePD dashboard view of complete VDExample project

Double-click on the VDExample.dproj icon, as shown in Figure 2.9. This will automatically open up ViewDraw. **Note**: The software protection key, called a dongle, must be attached to the parallel port of the computer in order for ViewDraw to open. If ViewDraw is not opening, ask the system administrator at WDC to assist you. After ViewDraw opens, your screen will look similar to Figure 2.10.

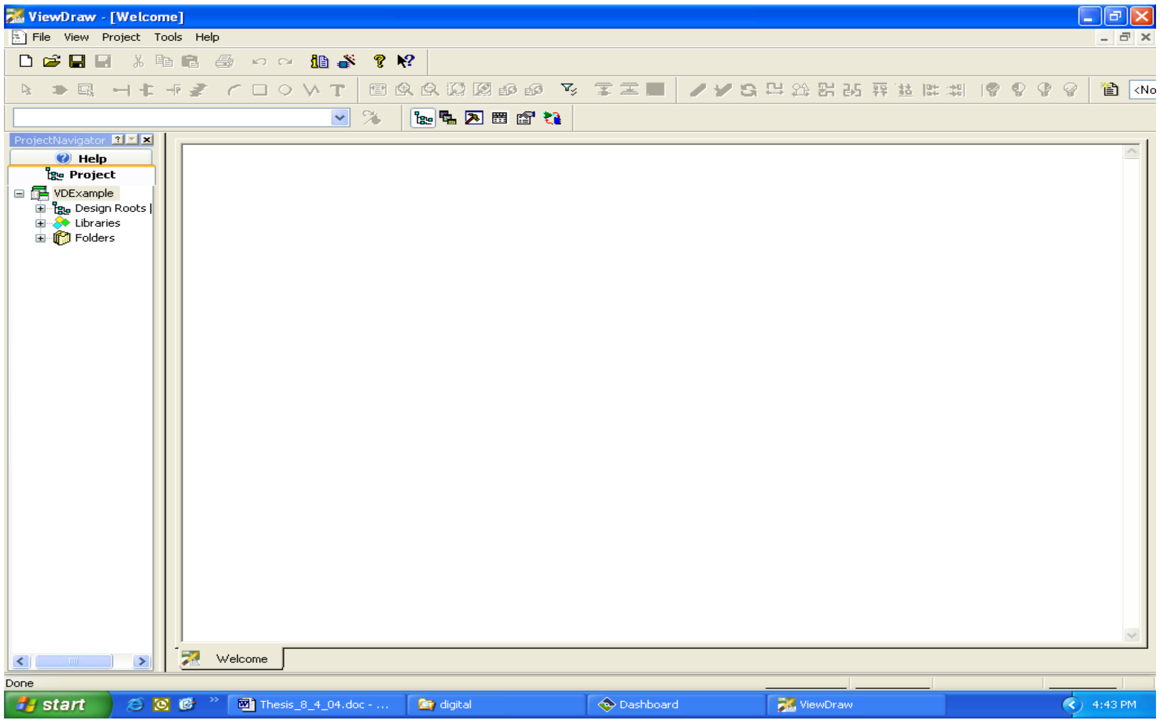
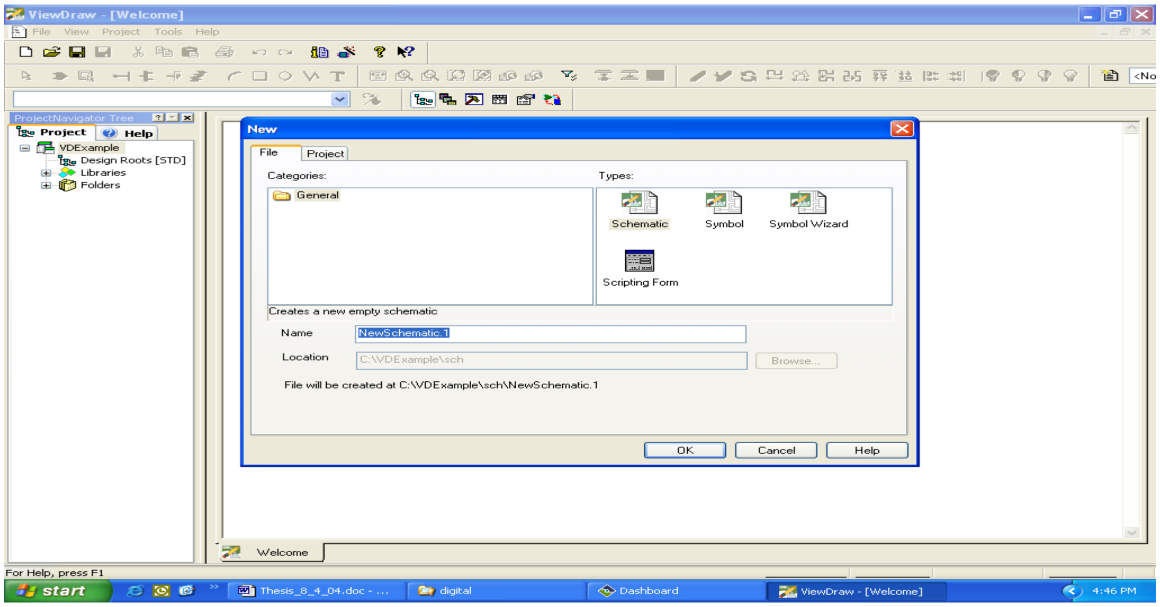


Figure 2.10 ViewDraw main welcome window

The first step is to create a new schematic drawing. To do this, click File → New. Your screen will look similar to Figure 2.11.



Creates a new empty schematic

Name: NewSchematic.1

Location: C:\VDExample\sch

File will be created at C:\VDExample\sch\NewSchematic.1

Figure 2.11 ViewDraw new schematic window

This window has Schematic selected as the default. If for some reason Schematic is not selected, click on the Schematic icon to select it. In the Name field, type in a name for the schematic. For this example, type in VDExample and click OK. Your screen will look like Figure 2.12.

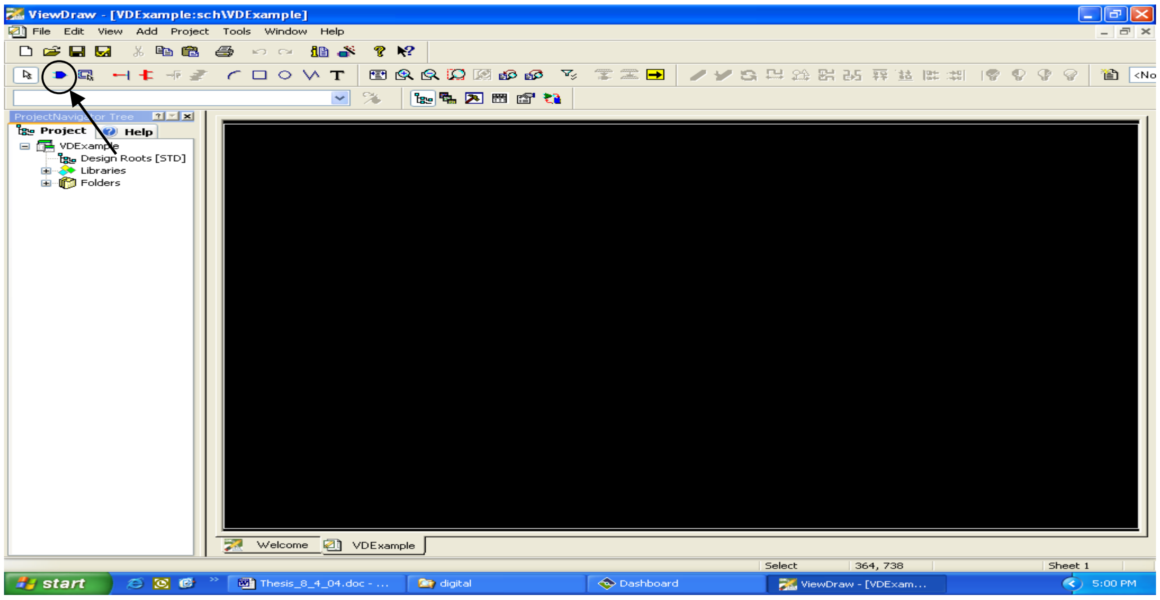


Figure 2.12 ViewDraw schematic window

We are now ready to begin drawing a schematic. To draw a schematic, you have to pick components from the library, place them onto the schematic, and connect the pins by drawing wires from pin to pin. There are 2 ways to add a component. The first way is by clicking Add → Component from the Standard (File) toolbar. The second way is by clicking on the Component icon on the toolbar, shown in Figure 2.12. Using either method to add a component, your screen will look like Figure 2.13.

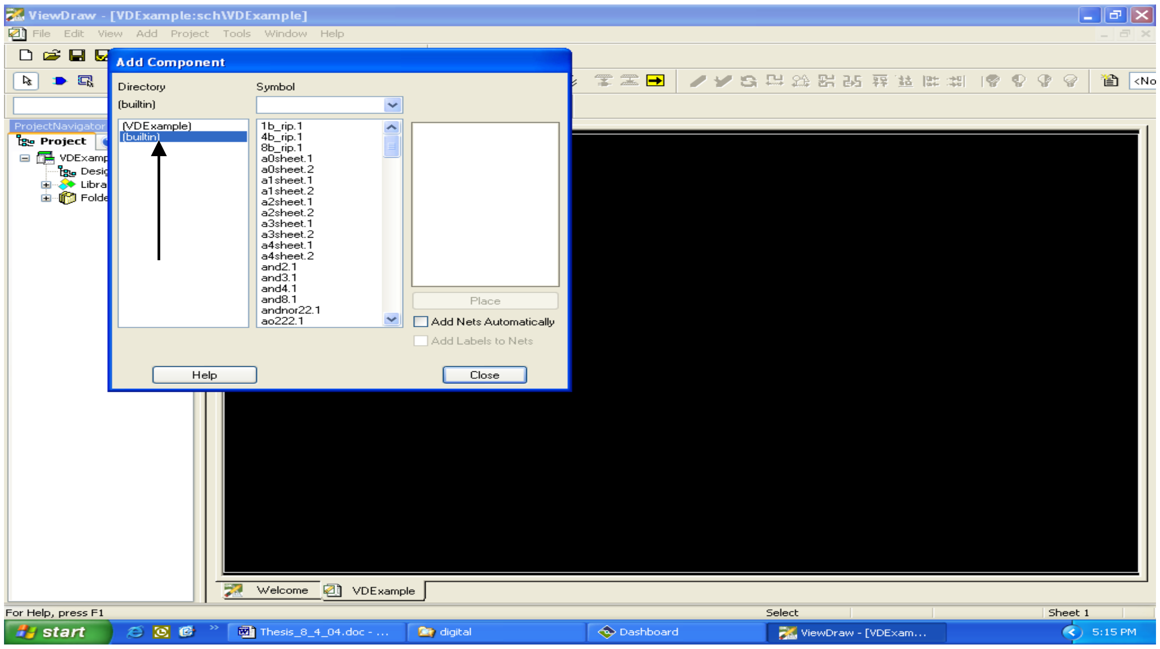


Figure 2.13 ViewDraw add component window

This window shows the available symbols in the builtin library. If the symbols are not listed, click on the builtin directory as shown in Figure 2.13. When a symbol is selected, a preview of the symbol is displayed in the preview window, as shown in Figure 2.14.

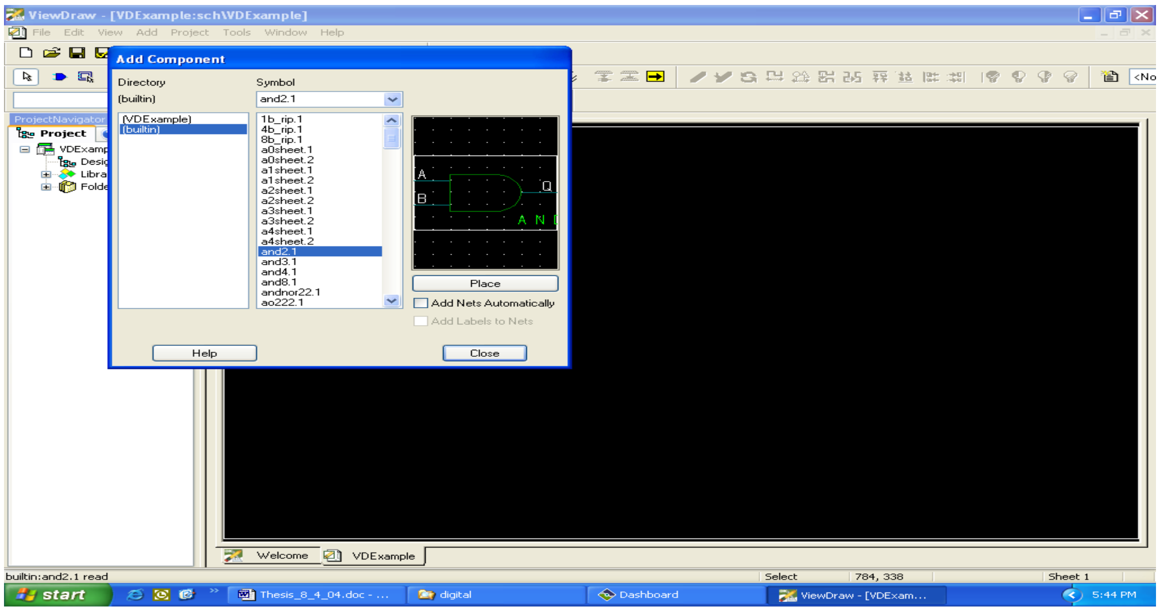


Figure 2.14 ViewDraw add component with preview window

Here, a 2-input, AND gate is selected and the preview of the symbol is shown. To place the symbol onto the drawing, click the Place button and drag the mouse to the drawing window, as shown in Figure 2.15.

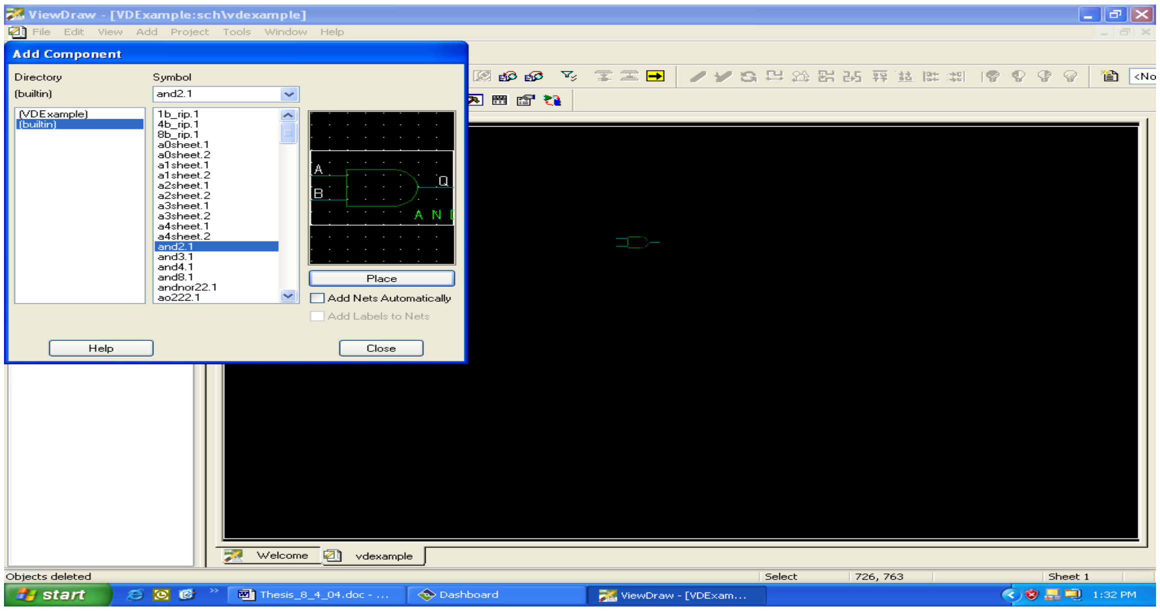


Figure 2.15 Symbol placement on schematic

Click the left mouse button to place the symbol onto the drawing area. Now, place a few more parts onto the drawing area. For this example, place two more 2-input, AND gates onto the drawing area. When complete, click the Close button on the Add Component window. Your screen will look similar to Figure 2.16.

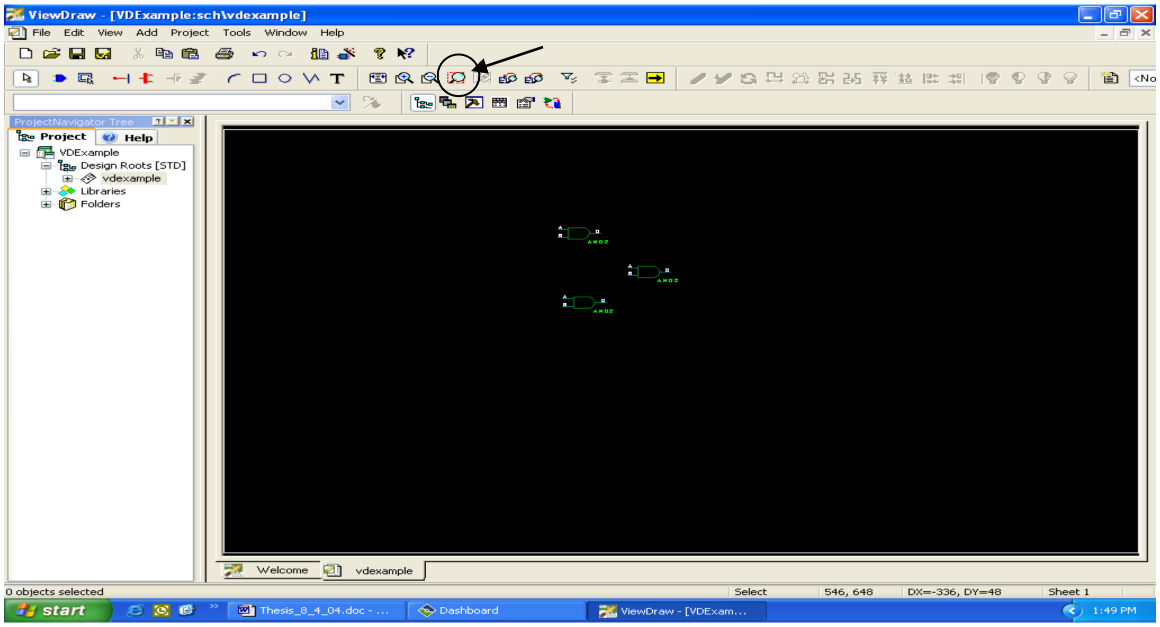


Figure 2.16 ViewDraw schematic window with added components

Next, zoom into the area of the three AND gates by clicking the Zoom Area icon on the toolbar as shown in Figure 2.16. Click this icon once to select it, then, using the mouse, draw a square around the three AND gates. To do this, place the cursor somewhere in the upper left corner of the top AND gate. Click and hold the left mouse button. Keeping the mouse button depressed, move the mouse down and to the right. You will see that a box is displayed as you move the mouse, showing the area that will be zoomed in on. Once you have the box around all three AND gates, release the mouse button. You will see that the AND gates have been magnified and that a grid now appears on the drawing. Your screen will look similar to Figure 2.17.

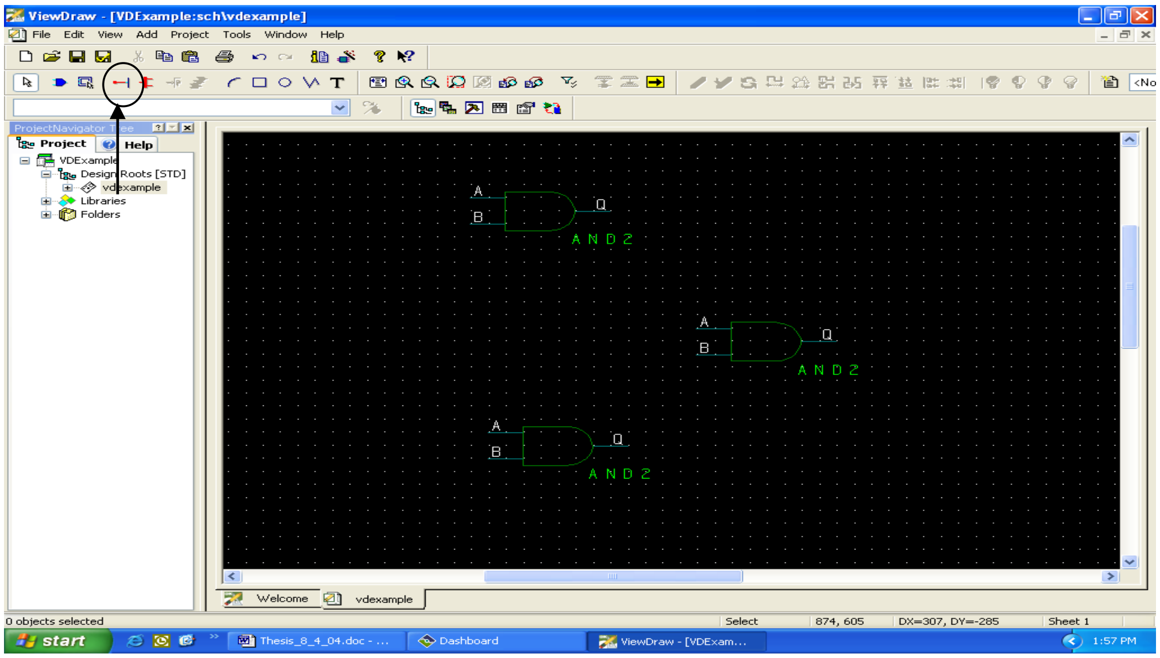


Figure 2.17 Schematic zoom view

Now, with the drawing grid in view, you can line up the symbols anyway that you want. To move the symbols around, place the cursor over the symbol you want to move, click and hold the left mouse button, and drag the symbol to the desired location. Release the mouse button when the symbol is at the desired location.

After the symbols are lined up, the next step is to connect the pins together using the wire command. Click on the Wire icon, as shown in Figure 2.17, to activate this mode. For this example, we will connect the A inputs of two AND gates to each other and the B inputs to each other. The outputs of each AND gate will connect to the inputs of the third AND gate. The purpose of this example is not to make a functional schematic, but to introduce you to the basic functions of ViewDraw.

In Wire mode, place the cursor at the A input of the topmost AND gate. Click and hold the left mouse button. Drag the cursor to the left for a few grid spaces. You will see that a wire is being drawn as you move the cursor. Releasing the mouse button will end the wire at that location. Note that you are still in Wire mode. Now, place the cursor at the end of the wire you just drew. Click and hold the left mouse button and drag the cursor to the grid line that is associated with the A input of the bottom left AND gate. Release the mouse button to end the wire here. Again, place the cursor at the end of this wire, click and hold the left mouse button, and drag the cursor to the A input pin of the AND gate. Repeat this process to connect the B inputs together, as well as connecting the outputs of the left AND gates to the inputs of the right AND gate. When complete, your screen should look similar to Figure 2.18.

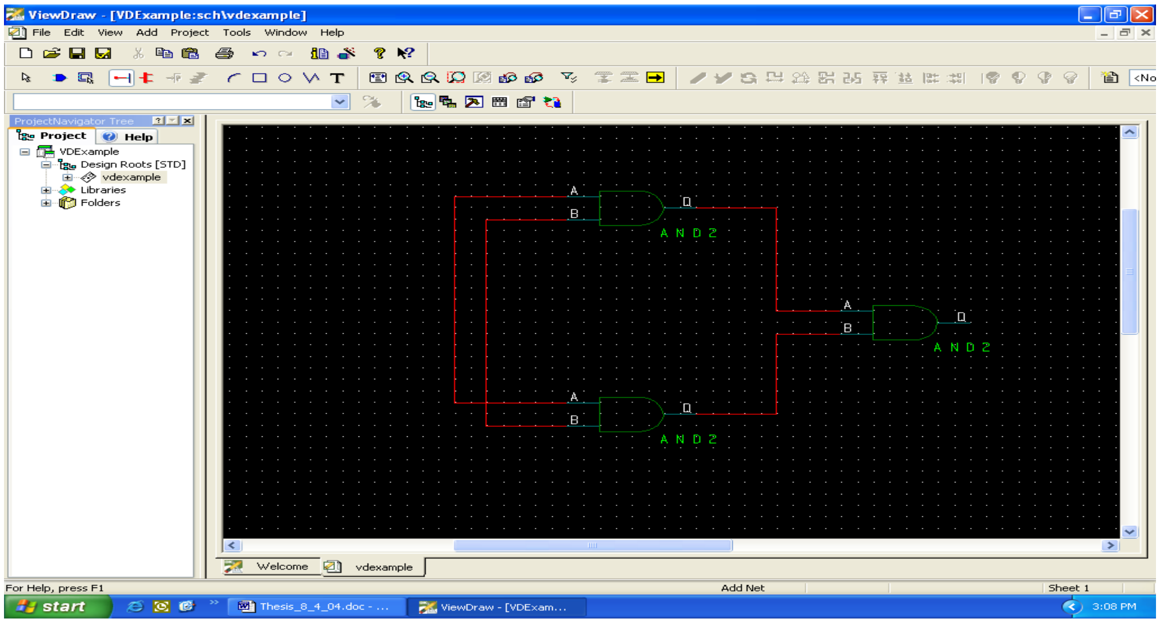


Figure 2.18 Example of schematic with components connected

The next step is placing and connecting power and ground pins to the schematic. As discussed before, go to the Add Component window, select the **gnd** and **pwr** symbols and place them onto the schematic. Place the symbols and connect them up as shown in Figure 2.19.

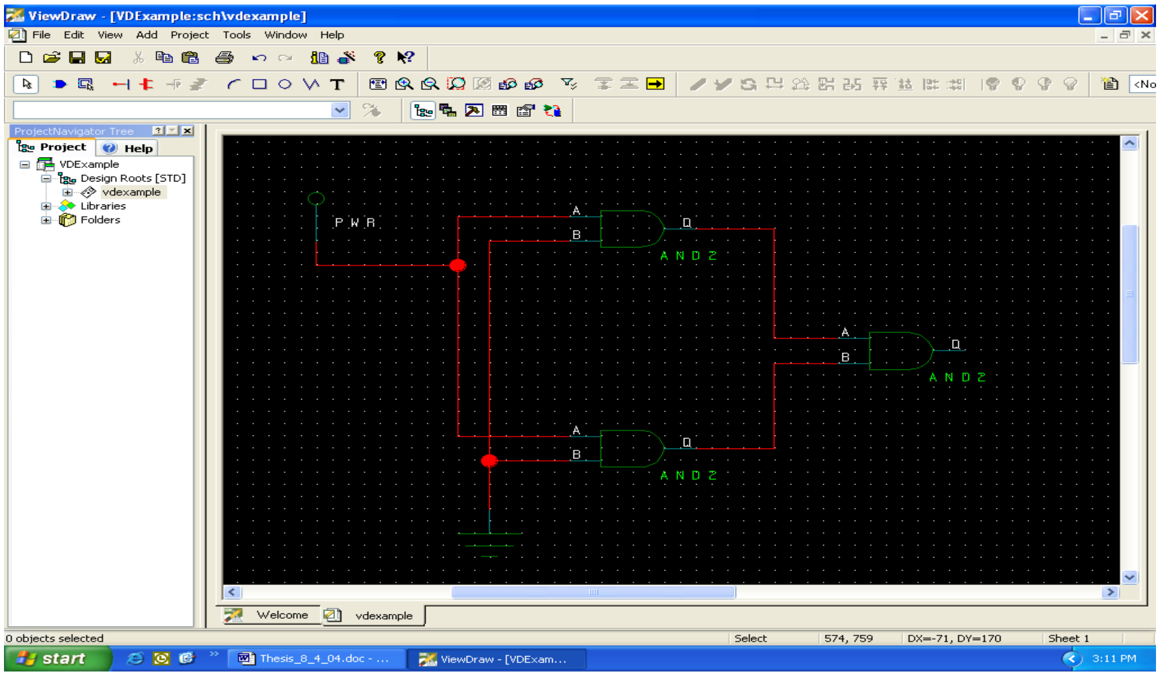


Figure 2.19 Example of completed schematic

Next, we want to name the internal nets. For this example, we will name the power and ground nets as VDD and VSS respectively, and the outputs of the two AND gates OUT1 and OUT2. By default, ViewDraw automatically names any net connected to a **pwr** or **gnd** symbol VDD and VSS. However, for this exercise, we will rename these nets so that the name is visible on the schematic.

To name a net, place the cursor on any segment of the net and double-click the left mouse button. Do this for any segment of the net connected to the **pwr** symbol. A pop out window will appear and your screen will look like Figure 2.20.

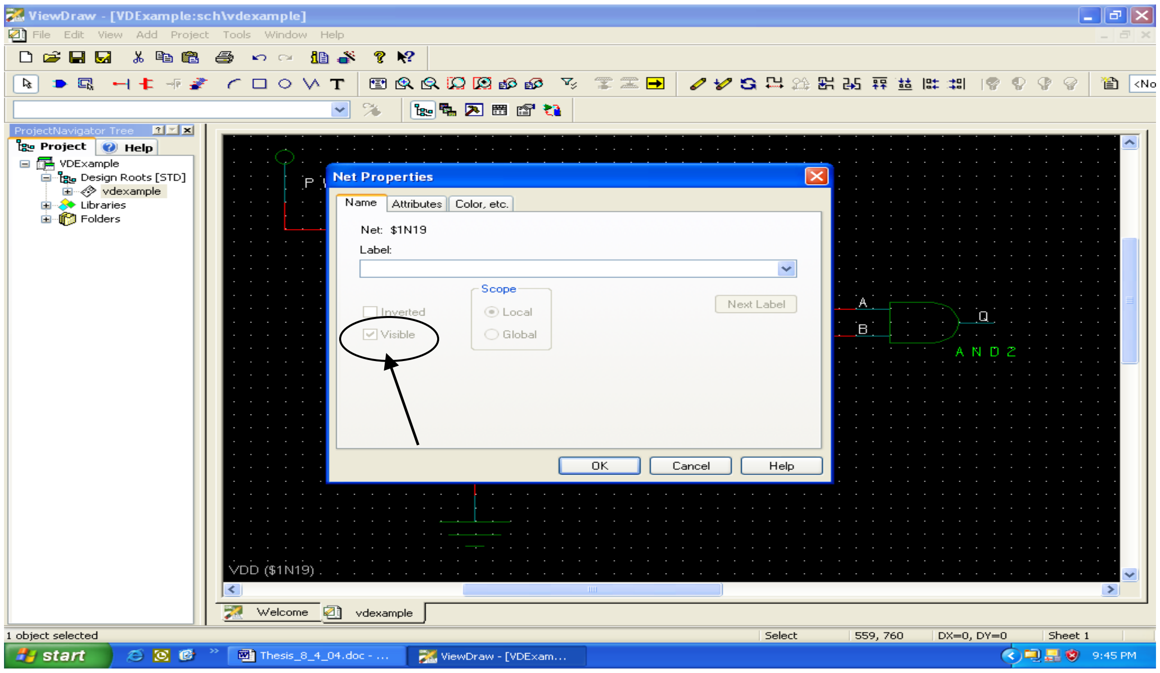


Figure 2.20 Net properties window

In the Label field, type in VDD, verify that the Visible box is checked as shown in Figure 2.20, and press OK. Your screen will now look similar to Figure 2.21.

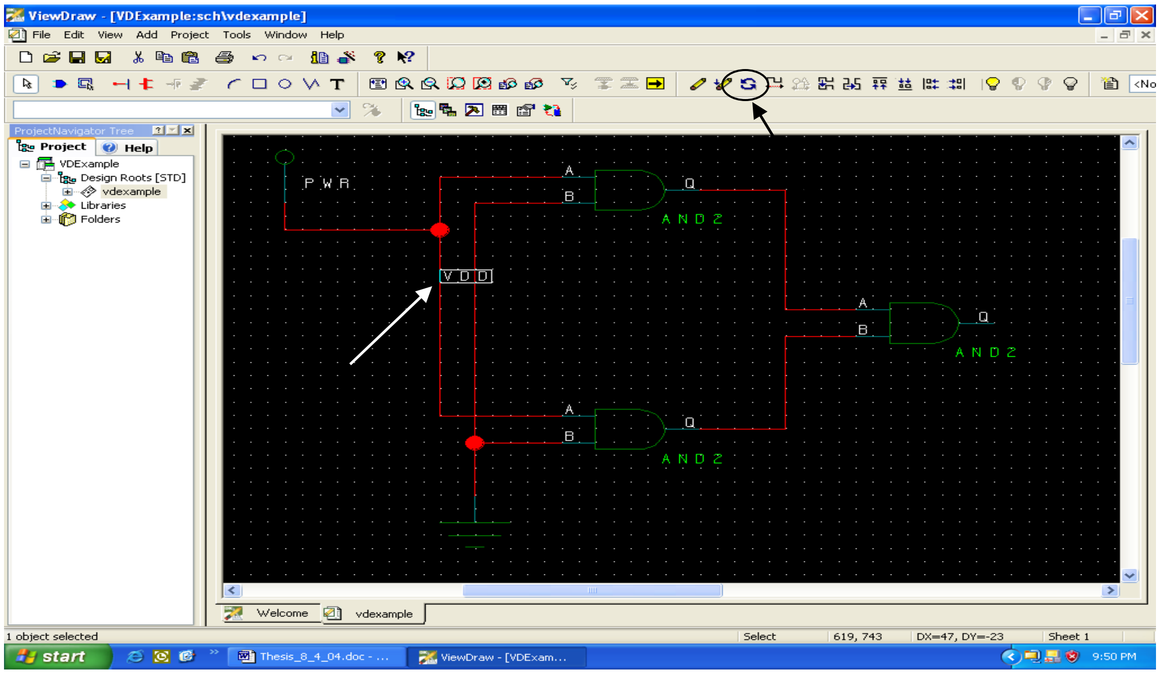


Figure 2.21 Schematic with net names visible

With the net name visible, you may want to move the location of the name. To do this, click on the name to highlight it as shown in Figure 2.21. Then, move the cursor over the name, press and hold the left mouse button, and drag the name to a good location. It may also be desirable to rotate the name. To rotate the name, select the name as described above and click on the Rotate icon, as shown in Figure 2.21. This will rotate the name 90 degrees each time this icon is clicked.

Follow the same procedure to name the three remaining nets. When completed, your screen will look similar to Figure 2.22.

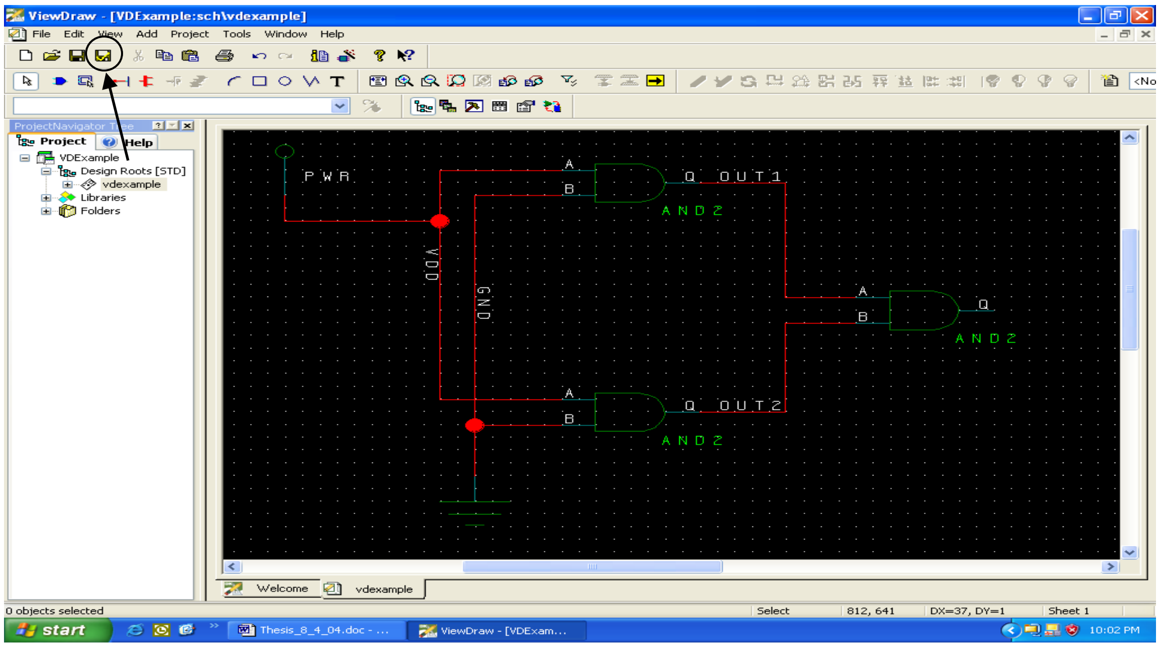


Figure 2.22 Schematic with all net names visible

The final step in this example is saving and checking the schematic for errors. To save and check the schematic, click on the Save and Check icon on the toolbar, as shown in Figure 2.22. ViewDraw will save the schematic and check the design for any errors, such as unconnected pins or un-named nets. If any errors are found, an error message window will appear informing you of the errors and where they are on the schematic.

ViewDraw is a very powerful tool and has many more capabilities that are not in the scope of this example to discuss. It is recommended that the user gets more familiar with ViewDraw by using the online help features and getting started tutorials that are available within the tool itself.

2.2 Design simulation

After the design has been entered using either of the above-mentioned methods, the design needs to be checked to verify its functionality. In the past, prototypes of the circuit were built and used to check the circuit. This method was feasible if designs were small and standard parts were used. With the complexity of today's circuits, prototyping is impractical. Therefore, for complex designs utilizing SOCs, ASICs, (Application-Specific Integrated Circuits) and FPGAs (Field Programmable Gate Arrays), a simulator is used to verify the designs functionality.

"Simulation is the fundamental and essential part of the design process for any electronic based product. Simulation is the process of verifying the functional characteristics of models at any level of behavior, from high levels of abstraction down to low levels" (4:14).

The simulator itself is a software tool that is used to simulate hardware models. Many times it is part of a design package, such as ePD, or can be a standalone tool such as Silos. With ePD, the designer can enter a design using schematic entry and perform a simulation of the design using one tool. Silos is strictly a Verilog HDL simulator, with no provisions for schematic entry.

Although The Western Design Center, Inc. uses ePD for schematic entry, they do not have a license for the simulation tool that is part of ePD. WDC uses Verilog as their choice for design entry and they use Silos for all their simulations. This does not mean that schematics entered using ViewDraw cannot be simulated. ViewDraw has a utility called Verilnet, which is a Verilog netlister. When the Verilnet utility is run on a schematic, it creates a Verilog description of the schematic, called a netlist. The netlist describes all the components in the design and their interconnections. The netlist also includes the model parameters for the devices used in the design. This netlist is then used as the top level design that is used by the Silos Verilog simulator.

Along with the top level design or netlist, the simulator requires a stimulus file in order to perform a simulation. The stimulus file, or test bench, is a Verilog model that invokes the top level design and drives the different signals in the design. The test bench will usually have a clock defined in it and it will exercise each input with respect to the clock. For example, if the design is a two bit adder, the test bench will place a pattern of 1's and 0's on the adder's inputs. The output of the adder for the various input patterns can be stored in an output file, and this output file can be examined to verify that the adder is working properly.

The results can also be analyzed by viewing the waveforms that are generated by the simulator. By observing the waveforms, the designer can verify if operations are taking place when they are supposed to and if there are any conflicts. Using the adder example, if input A and input B are added together, the correct result should show up on the Sum output signal. If not, then there is a problem with the design and the design needs to be evaluated to find the problem. Once the functionality of the design has been verified, the designer can proceed to the next step in the design flow, the layout of the design. This is discussed in Section 2.3.

2.2.1 Introduction to Silos

The Western Design Center, Inc. uses Verilog as their primary design entry format and WDC uses Silos as their Verilog simulator of choice. Silos is a product of Silvaco and can be used as a component in their suite of design tools, or used as a standalone tool. WDC uses Silos as a standalone tool. This section will give a brief introduction to Silos, describing the creation of a project, adding files to the project, performing a simulation and viewing the simulation results of a simple Verilog design.

2.2.1.1 Introduction to simulating a design using Silos

The first step to performing a simulation is to create a Silos project. All files used for the design and simulation are stored within this project. These files include the top level Verilog model of the design, the test bench files and any supporting library files. To begin, create a directory on the C:\ drive of the computer called C:\Silos_Example. This will be where we will create the project and store the project files. Open up Silos by double-clicking on the Silos icon on the desktop. Note that Silos is protected by a hardware dongle, and this dongle must be attached to the computer's parallel port in order for Silos to open. After opening Silos, your screen will look like Figure 2.23.

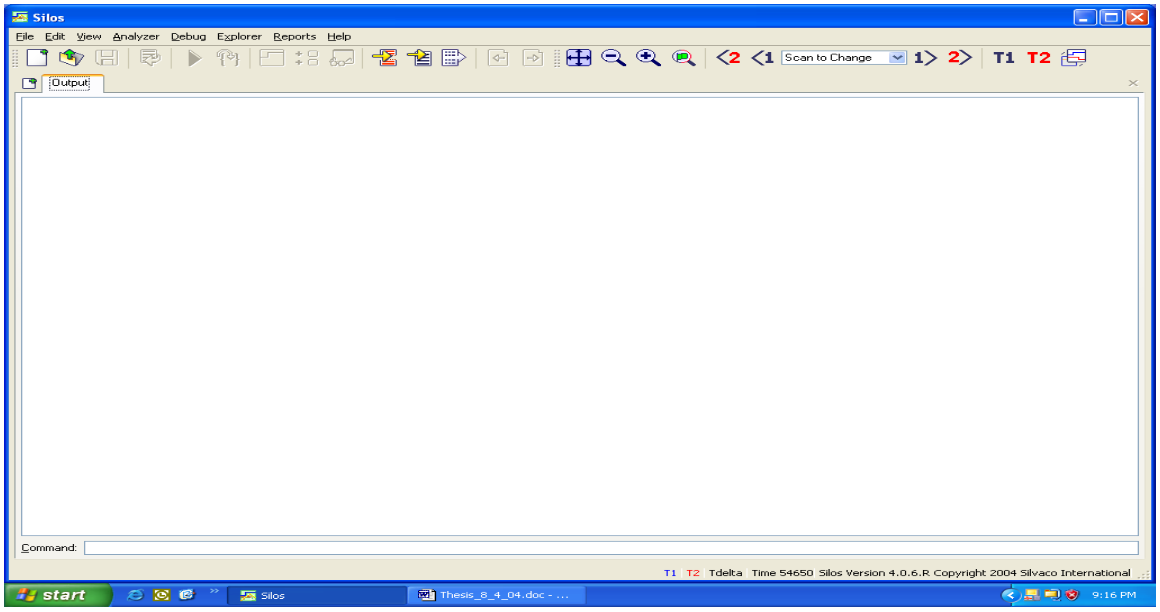

Silos
File Edit View Analyzer Debug Explorer Reports Help
Scan to Change
T1 T2
Output
Command:
T1 T2 Tdelta Time 54650 Silos Version 4.0.6.R Copyright 2004 Silvaco International
start
Silos
Thesis_8_4_04.doc - ...
9:16 PM

Figure 2.23 Silos main window

Next, create a new project by clicking File → New Project from the menu toolbar. This will cause the Create New Project window to appear and your screen will look like Figure 2.24.

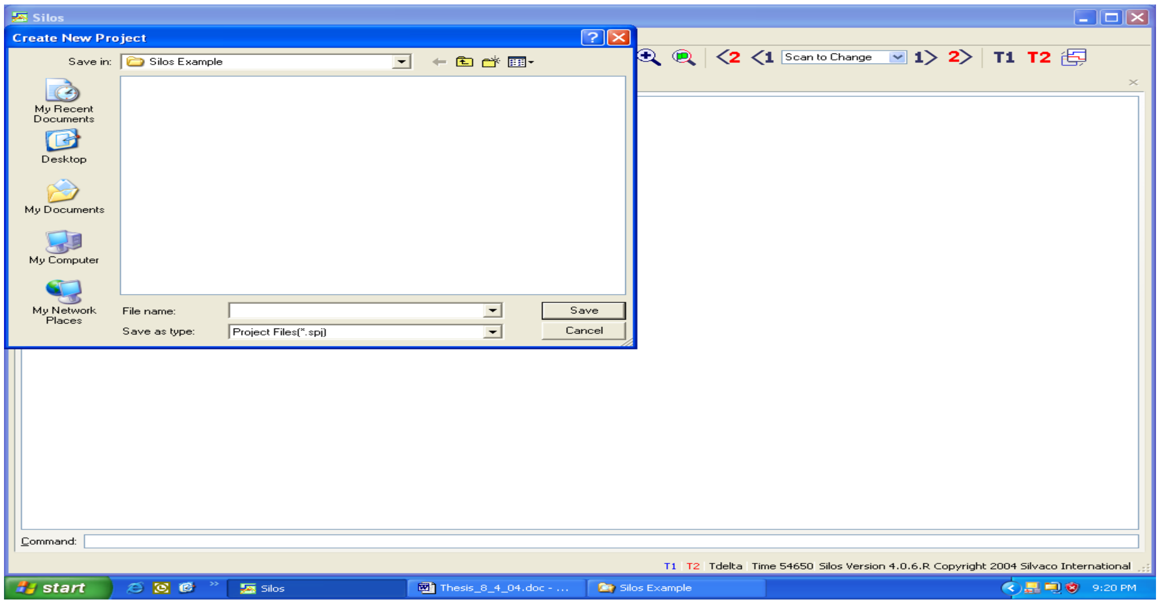


Figure 2.24 New project window in Silos

In the File name field, type in a name for the project. For this example, call the project Silos_Example. The extension of the file name for a project is .spj. Browse to C:\Silos_Example and click Save. After clicking Save, a Project Properties window appears and is shown in Figure 2.25.

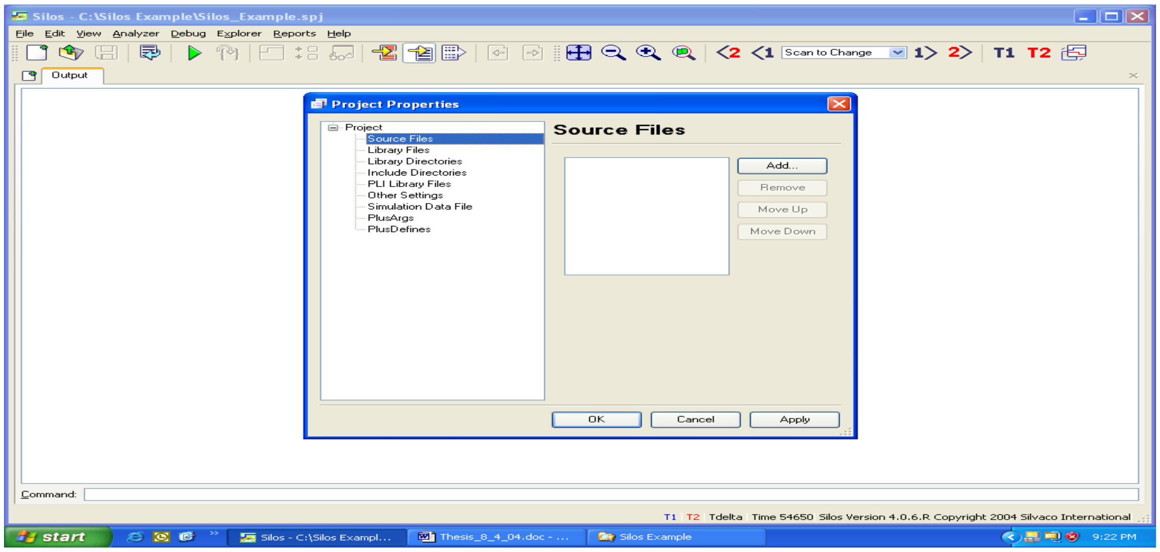


Figure 2.25 Silos project properties window

The Project Properties window will show all of the files that are associated with the project. At this time, we do not have any files to add to the project, so click Cancel to close this window. The screen will return back to the main Silos window.

Next, we must create the source and test bench files for our project. To demonstrate this, we will create the Verilog top level module and test bench of a 4-bit full adder. This example is taken from (7:206-208). The text for the files can be written in any text editor, such as WordPad or Notepad. Silos also has a text editor to enter source code. For this example, we will use the Silos text editor.

To create a new file in Silos, click File → New. This will open a blank source window as shown in Figure 2.26.

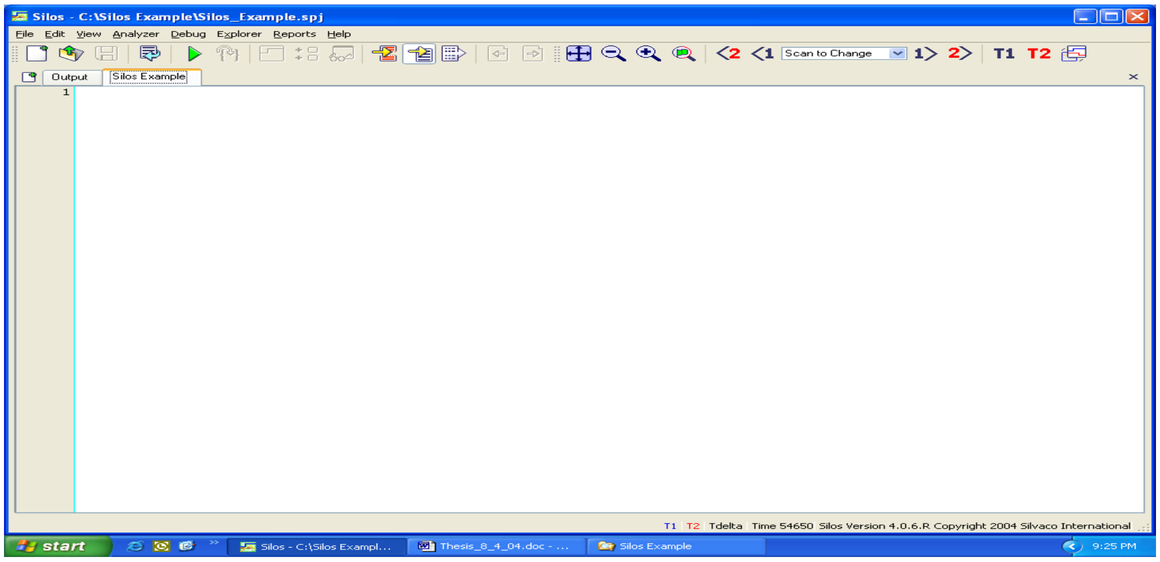

Silos - C:\Silos Example\Silos_Example.spj
File Edit View Analyzer Debug Explorer Reports Help
<2 <1 Scan to Change 1> 2> T1 T2
Output
Silos Example
T1 T2 Tdelta Time 54650 Silos Version 4.0.6.R Copyright 2004 Silvaco International
start
Silos - C:\Silos Exampl...
Thesis_8_4_04.doc - ...
Silos Example
9:25 PM

Figure 2.26 Silos text editor window

Type the source code shown in Figure 2.27 into the text field. Notice that as you type, the editor changes the color of comments and Verilog reserved words, and adds line numbers.

```
module fourbitadder(sumout, carryout, ain, bin, cin, clock);

output [3:0] sumout;
output carryout;
input [3:0] ain, bin;
input cin, clock;
wire [3:0] ain, bin, sumout_tmp;
wire cin, carryout_tmp;
reg [3:0] sumout, ain_tmp, bin_tmp;
reg carryout, cin_tmp;

always @(posedge clock) begin
        carryout = carryout_tmp;
        sumout = sumout_tmp;
        cin_tmp = cin;
        ain_tmp = ain;
        bin_tmp = bin;
end
        assign {carryout_tmp,sumout_tmp} = ain_tmp + bin_tmp + cin_tmp;
endmodule
```

Figure 2.27 4-bit adder Verilog model (adder.v)

When you are done typing in the code, save the file to C:\Silos_Example and name the file adder.v. Next, create a new file as described above and type in the test bench code shown in Figure 2.28.

```
module testbench;
        wire [3:0] sumout;
        wire carryout;
        reg [3:0] ain, bin;
        reg cin, clock;
        integer i, j; parameter cycle = 100;
        fourbitadder INST(sumout, carryout, ain, bin, cin, clock);
        // adder4 INST(sumout, carryout, ain, bin, cin, clock
        initial clock = 0; //non-synthesizable clock
        always #(cycle/2) clock = ~clock; //generator

        always @(posedge clock) begin
                cin = 0; ain = 0; bin = 0;
                for ( i=0; i <= 15; i = i + 1) begin
                        #cycle ain = i;
                        for (j = 0; j <= 15; j = j + 1)
                        #cycle bin = j;
                end
                #cycle cin = 1;
                for (i = 0; i <= 15; i = i + 1) begin
                        #cycle ain = i;
                        for (j = 0; j <= 15; j = j + 1)
                                #cycle bin = j;
                end
                        #cycle $finish;
                end
                initial begin
                 $monitor("%0d ", $time,, "clock = ", clock,
                        " cin = ", cin,
                        " ain[0] = ", ain[0],
                        " ain[1] = ", ain[1],
                        " ain[2] = ", ain[2],
                        " ain[3] = ", ain[3],
                        " bin[0] = ", bin[0],
                        " bin[1] = ", bin[1],
                        " bin[2] = ", bin[2],
                        " bin[3] = ", bin[3],
                        " s[0] = ", sumout[0],
                        " s[1] = ", sumout[1],
                        " s[2] = ", sumout[2],
                        " s[3] = ", sumout[3]);
                end
        endmodule
```

Figure 2.28 Test bench for the 4-bit full adder (test_bench.v)

When you are done typing in the code, save the file to C:\Silos_Example and name the file test_bench.v. We are now ready to add the two files to the project that we created earlier.

To add the files to the project, click on Edit → Project Properties. Your screen will look like Figure 2.29.

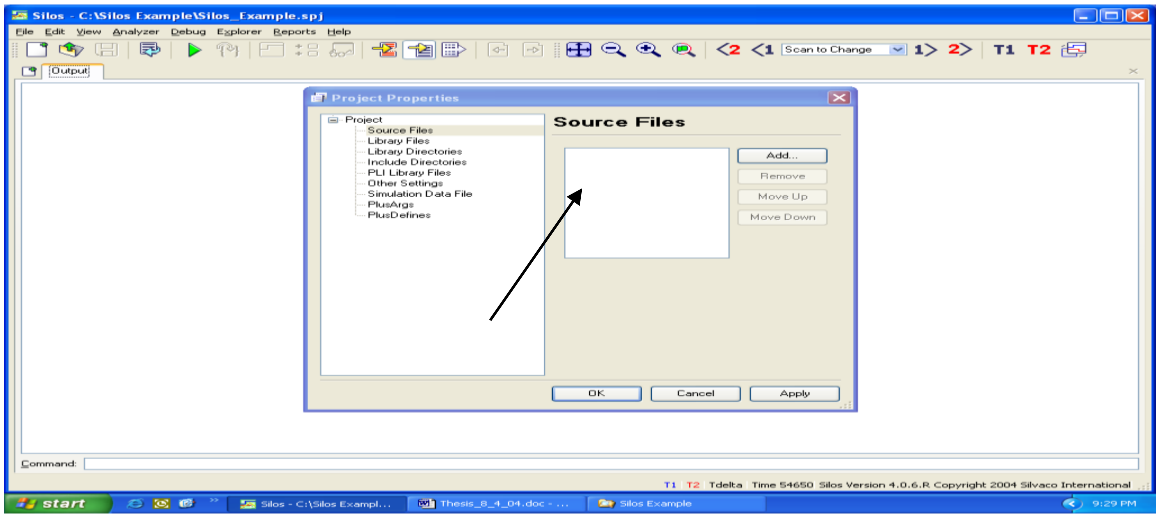


Figure 2.29 Silos project properties window

Next, click the Add button. This will bring up a file browser window, which defaults to the C:\Silos_Example directory. The two files that we created adder.v and test_bench.v are listed. There are two ways to add these files into the project; the quick method is to select each file while holding down the CRTL key, then click Add. This will add both files to the project at the same time. Alternately, you can select each file individually, then click Add, and repeat the process for the other file. After adding the files to the project, they are placed in the Source Files window, indicated in Figure 2.29. After adding both files to the project, your screen will look similar to Figure 2.30.

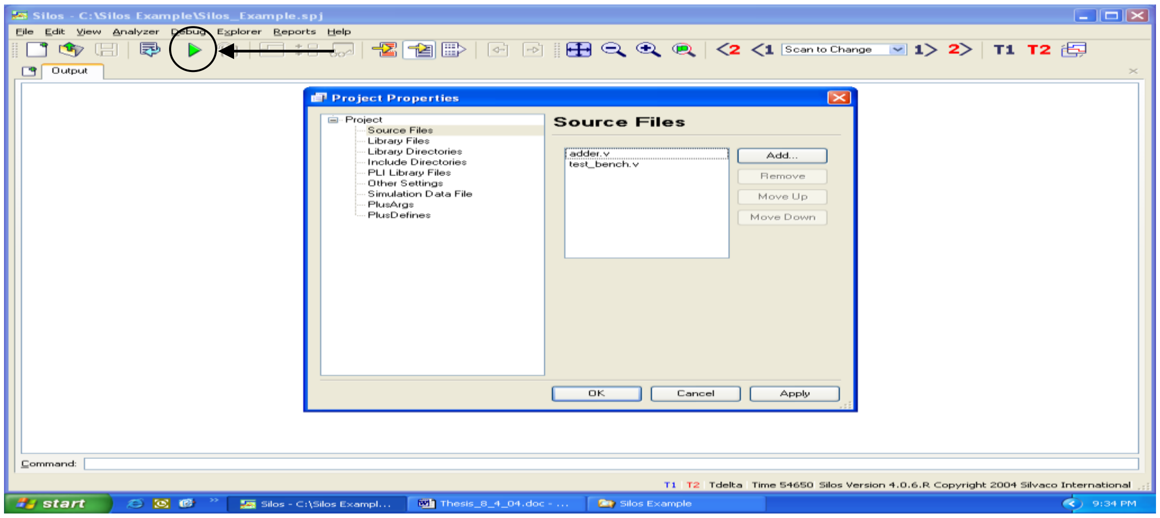
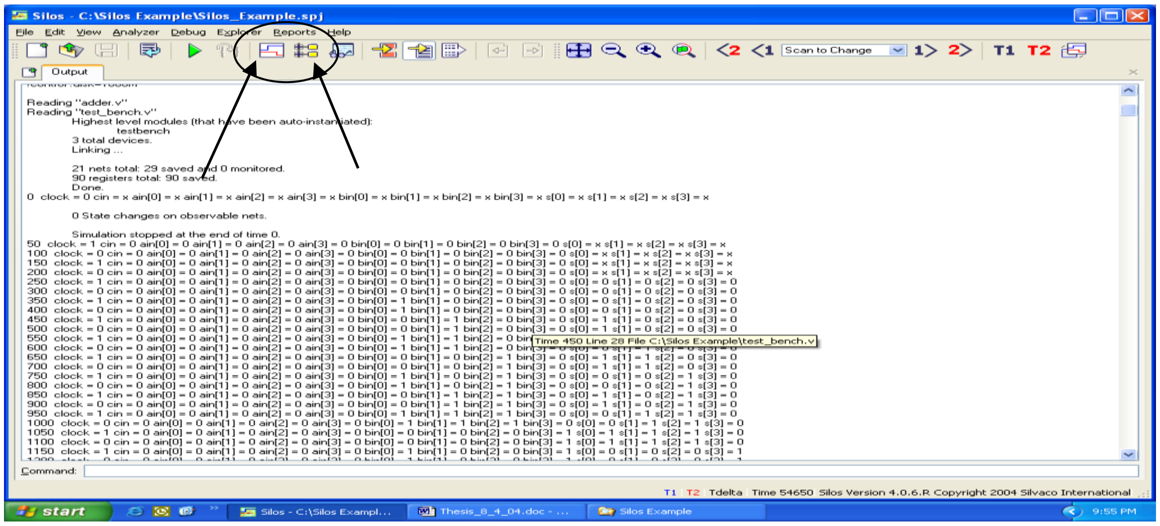


Figure 2.30 Silos project properties window with added files

Click OK to close the Project Properties window. The screen will return to the main Silos window.

Now that the files have been added, the next step is to simulate the project. To do this, click the Start Simulation icon as indicated in Figure 2.30. If there are no errors in the source or test bench files, the simulation will begin and your screen will look similar to Figure 2.31.


Silos - C:\Silos Example\Silos_Example.spj
File Edit View Analyzer Debug Explorer Reports Help
Scan to Change
Output
Command:
T1 T2 Tdelta Time 54650 Silos Version 4.0.6.R Copyright 2004 Silvaco International
start
Silos - C:\Silos Exampl...
Thesis_8_4_04.doc - ...
Silos Example
9:55 PM

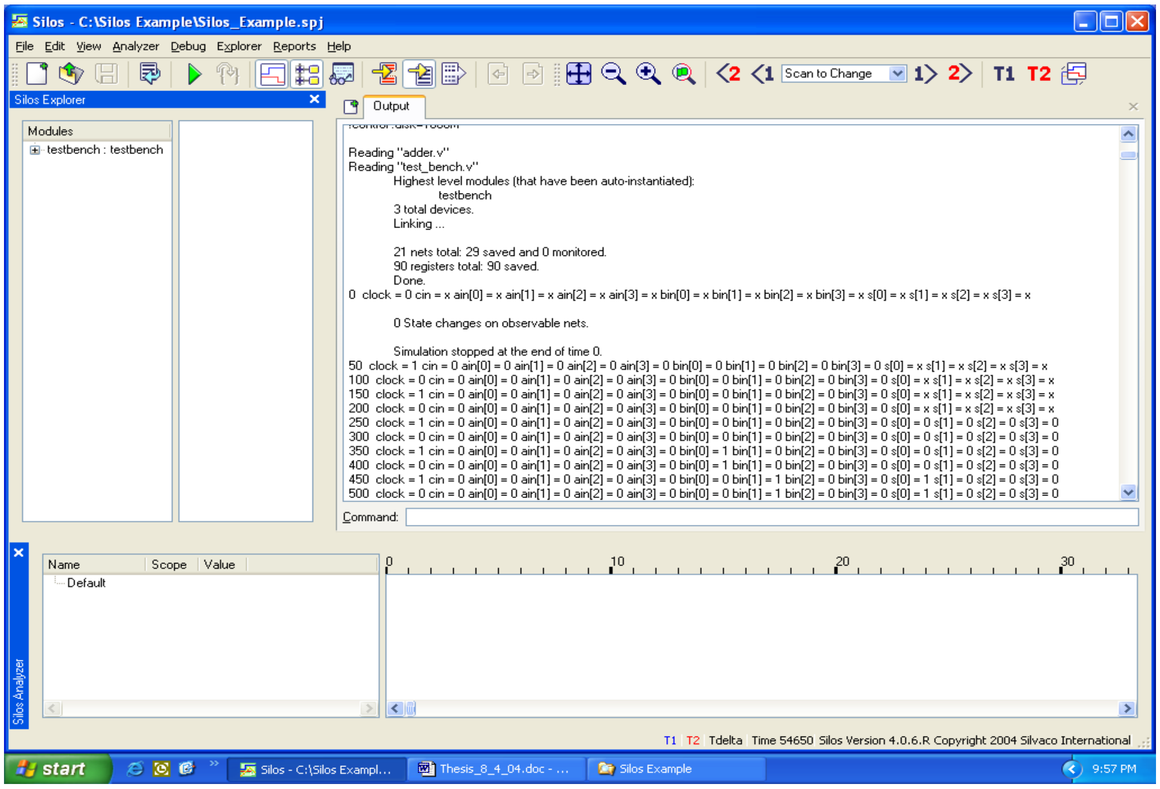
```
Reading "adder.v"
Reading "test_bench.v"
          Highest level modules (that have been auto-instantiated):
                    testbench
          3 total devices.
          Linking ...

          21 nets total: 29 saved and 0 monitored.
          90 registers total: 90 saved.
          Done.
0  clock = 0 cin = x ain[0] = x ain[1] = x ain[2] = x ain[3] = x bin[0] = x bin[1] = x bin[2] = x bin[3] = x s[0] = x s[1] = x s[2] = x s[3] = x

          0 State changes on observable nets.

          Simulation stopped at the end of time 0.
50  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = x s[1] = x s[2] = x s[3] = x
100  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = x s[1] = x s[2] = x s[3] = x
150  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = x s[1] = x s[2] = x s[3] = x
200  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = x s[1] = x s[2] = x s[3] = x
250  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 0 s[3] = 0
300  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 0 s[3] = 0
350  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 0 s[3] = 0
400  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 0 s[3] = 0
450  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 1 bin[2] = 0 bin[3] = 0 s[0] = 1 s[1] = 0 s[2] = 0 s[3] = 0
500  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 1 bin[2] = 0 bin[3] = 0 s[0] = 1 s[1] = 0 s[2] = 0 s[3] = 0
550  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 1 bin[2] = 0 bin Time 450 Line 28 File C:\Silos Example\test_bench.v
600  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 1 bin[2] = 0 bin[3] = 0 s[0] = 0 s[1] = 1 s[2] = 0 s[3] = 0
650  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 1 bin[3] = 0 s[0] = 1 s[1] = 1 s[2] = 0 s[3] = 0
700  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 1 bin[3] = 0 s[0] = 1 s[1] = 1 s[2] = 0 s[3] = 0
750  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 0 bin[2] = 1 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 1 s[3] = 0
800  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 0 bin[2] = 1 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 1 s[3] = 0
850  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 1 bin[2] = 1 bin[3] = 0 s[0] = 1 s[1] = 0 s[2] = 1 s[3] = 0
900  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 1 bin[2] = 1 bin[3] = 0 s[0] = 1 s[1] = 0 s[2] = 1 s[3] = 0
950  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 1 bin[2] = 1 bin[3] = 0 s[0] = 0 s[1] = 1 s[2] = 1 s[3] = 0
1000  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 1 bin[2] = 1 bin[3] = 0 s[0] = 0 s[1] = 1 s[2] = 1 s[3] = 0
1050  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 1 s[0] = 1 s[1] = 1 s[2] = 1 s[3] = 0
1100  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 1 s[0] = 1 s[1] = 1 s[2] = 1 s[3] = 0
1150  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 0 bin[2] = 0 bin[3] = 1 s[0] = 0 s[1] = 0 s[2] = 0 s[3] = 1
```

Figure 2.31 Silos simulation result window

If an error was made in typing the source or test bench files, Silos will indicate the file name, error and the line number where the error is found. Debug the files as needed until the simulation runs correctly. The simulation will take a few seconds to complete.

Figure 2.31 shows the beginning stage of the simulation. Silos reads in the source and test bench files, checks them for errors, and then runs the simulation if no errors are found. The output shows the status of the clock, ain inputs, bin inputs and the sum outputs. This pattern continues until the $finish line in the test bench file is executed.

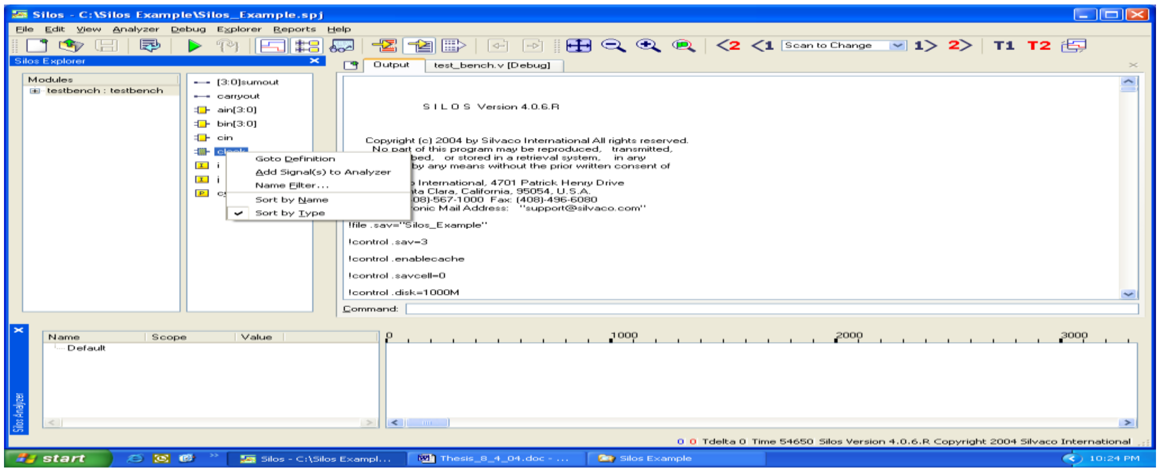
Next, we want to look at some of the signals in a waveform view. To do this, we need to open the Explorer and Analyzer windows within Silos. To do this, click on the Analyzer and Explorer icons, as shown in Figure 2.31. After clicking each icon, your screen will look similar to Figure 2.32.


Silos - C:\Silos Example\Silos_Example.spj
File Edit View Analyzer Debug Explorer Reports Help
Scan to Change
T1 T2
Silos Explorer
Output
Modules
testbench : testbench
Command:
Name Scope Value
Default
Silos Analyzer
0 10 20 30
T1 T2 Tdelta Time 54650 Silos Version 4.0.6.R Copyright 2004 Silvaco International
start
Silos - C:\Silos Exampl...
Thesis_8_4_04.doc - ...
Silos Example
9:57 PM

```
Reading "adder.v"
Reading "test_bench.v"
          Highest level modules (that have been auto-instantiated):
                    testbench
          3 total devices.
          Linking ...

          21 nets total: 29 saved and 0 monitored.
          90 registers total: 90 saved.
          Done.
0  clock = 0 cin = x ain[0] = x ain[1] = x ain[2] = x ain[3] = x bin[0] = x bin[1] = x bin[2] = x bin[3] = x s[0] = x s[1] = x s[2] = x s[3] = x

          0 State changes on observable nets.

          Simulation stopped at the end of time 0.
50  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = x s[1] = x s[2] = x s[3] = x
100  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = x s[1] = x s[2] = x s[3] = x
150  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = x s[1] = x s[2] = x s[3] = x
200  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = x s[1] = x s[2] = x s[3] = x
250  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 0 s[3] = 0
300  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 0 s[3] = 0
350  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 0 s[3] = 0
400  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 1 bin[1] = 0 bin[2] = 0 bin[3] = 0 s[0] = 0 s[1] = 0 s[2] = 0 s[3] = 0
450  clock = 1 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 1 bin[2] = 0 bin[3] = 0 s[0] = 1 s[1] = 0 s[2] = 0 s[3] = 0
500  clock = 0 cin = 0 ain[0] = 0 ain[1] = 0 ain[2] = 0 ain[3] = 0 bin[0] = 0 bin[1] = 1 bin[2] = 0 bin[3] = 0 s[0] = 1 s[1] = 0 s[2] = 0 s[3] = 0
```

Figure 2.32 Silos analyzer and explorer window view

In the Explorer window, the module testbench is listed. Click the module name to select it. The signals that can be viewed are displayed. To add a signal to the Analyzer, select a signal to highlight it, then right-click over it to display a pop out menu. For example, select the signal clock to highlight it and then right-click the mouse. The screen should look like Figure 2.33.

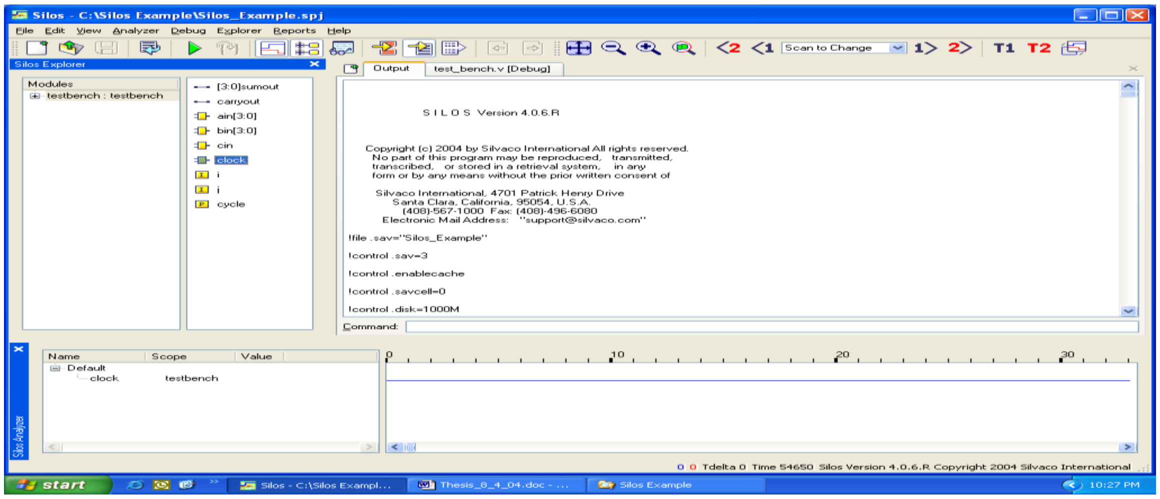


```
                S I L O S  Version 4.0.6.R


     Copyright (c) 2004 by Silvaco International All rights reserved.
       No part of this program may be reproduced,   transmitted,
                  bed,   or stored in a retrieval system,    in any
                  by any means without the prior written consent of

                  o International, 4701 Patrick Henry Drive
                  ita Clara, California, 95054, U.S.A.
                  08)-567-1000  Fax: (408)-496-6080
                  onic Mail Address:   "support@silvaco.com"

!file .sav="Silos_Example"

!control .sav=3

!control .enablecache

!control .savcell=0

!control .disk=1000M
```

```
0 0 Tdelta 0 Time 54650 Silos Version 4.0.6.R Copyright 2004 Silvaco International
```

Figure 2.33 Adding signals to the analyzer

In the pop out menu, click on the Add Signal(s) to Analyzer option. Your screen will now look similar to Figure 2.34.

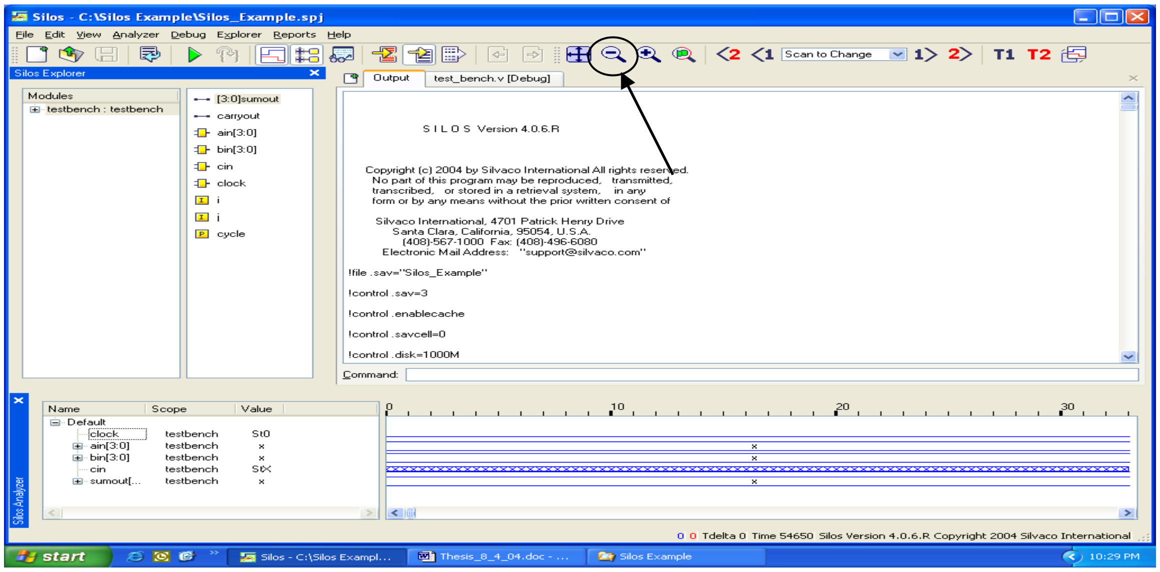


```
                S I L O S  Version 4.0.6.R


     Copyright (c) 2004 by Silvaco International All rights reserved.
       No part of this program may be reproduced,   transmitted,
       transcribed,   or stored in a retrieval system,    in any
       form or by any means without the prior written consent of

        Silvaco International, 4701 Patrick Henry Drive
            Santa Clara, California, 95054, U.S.A.
              (408)-567-1000  Fax: (408)-496-6080
          Electronic Mail Address:   "support@silvaco.com"

!file .sav="Silos_Example"

!control .sav=3

!control .enablecache

!control .savcell=0

!control .disk=1000M
```

```
0 0 Tdelta 0 Time 54650 Silos Version 4.0.6.R Copyright 2004 Silvaco International
```

Figure 2.34 Analyzer window with clock signal added

Repeat this procedure to add the ain, bin, cin and sumout signals to the Analyzer. Your screen should now look similar to Figure 2.35.



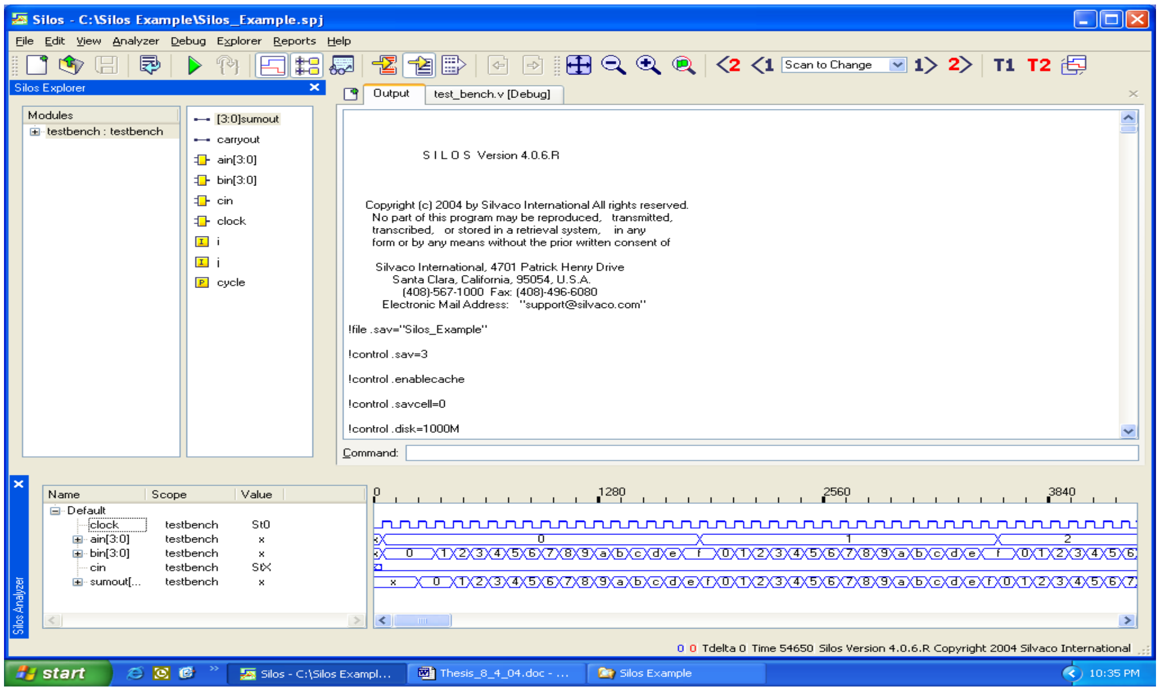
```
                 S I L O S  Version 4.0.6.R


    Copyright (c) 2004 by Silvaco International All rights reserved.
      No part of this program may be reproduced,   transmitted,
      transcribed,   or stored in a retrieval system,    in any
      form or by any means without the prior written consent of

       Silvaco International, 4701 Patrick Henry Drive
           Santa Clara, California, 95054, U.S.A.
              (408)-567-1000  Fax: (408)-496-6080
        Electronic Mail Address:   "support@silvaco.com"

!file .sav="Silos_Example"

!control .sav=3

!control .enablecache

!control .savcell=0

!control .disk=1000M
```

Name	Scope	Value
Default		
clock	testbench	St0
ain[3:0]	testbench	x
bin[3:0]	testbench	x
cin	testbench	StX
sumout[...	testbench	x

```
0 0 Tdelta 0 Time 54650 Silos Version 4.0.6.R Copyright 2004 Silvaco International
```

Figure 2.35 Analyzer window all signals added

Figure 2.35 shows only a small portion of the complete test. To view the signals in a different time scale, use the Zoom Out icon, shown in Figure 2.35. There is also a Zoom Full icon which will show all of the transitions for the whole time duration. However, this is not practical for analysis, as the signals are all compressed and not viewable. Click the Zoom Out icon a few times so that your screen looks similar to Figure 2.36.



```
                    S I L O S  Version 4.0.6.R



     Copyright (c) 2004 by Silvaco International All rights reserved.
       No part of this program may be reproduced,   transmitted,
       transcribed,   or stored in a retrieval system,    in any
       form or by any means without the prior written consent of

        Silvaco International, 4701 Patrick Henry Drive
            Santa Clara, California, 95054, U.S.A.
               (408)-567-1000  Fax: (408)-496-6080
          Electronic Mail Address:   "support@silvaco.com"

!file .sav="Silos_Example"

!control .sav=3

!control .enablecache

!control .savcell=0

!control .disk=1000M
```

Name	Scope	Value
Default		
clock	testbench	St0
ain[3:0]	testbench	x
bin[3:0]	testbench	x
cin	testbench	StX
sumout[...	testbench	x

```
0 0 Tdelta 0 Time 54650 Silos Version 4.0.6.R Copyright 2004 Silvaco International
```

Figure 2.36 Analyzer window with signals viewable

Note that the values of the registers are displayed within the waveform. This makes it easier to debug the project. Using the waveform view, the designer can monitor each signal with respect to the clock and analyze the output to see if the desired result is achieved. If the simulation results are correct, the design can move to the next step in the process. If the simulation produces incorrect results, the source files need to be debugged, corrected and re-simulated until proper functionality is achieved.

At this point, we have created a project, entered in the source code and test bench code, ran a simulation and viewed the simulation results. There are other features of Silos that are not in the scope of this introduction to discuss. It is recommended that the user get more familiar with Silos by utilizing the online help and tutorials available within Silos.

2.3 Physical layout of the design

The next step after design simulation is to do the physical layout of the design. The layout is done using an IC layout editor program. It is here that the design is laid out, as it will appear on silicon. The N and P regions of the transistors are defined, the polysilicon interconnect layers and the metal layers are drawn. In most cases, more than one layer is used. In this case, vias are used to provide interconnects between the layers. Figure 2.37 shows the physical layout of the Address Decoder used in the W65C122S.

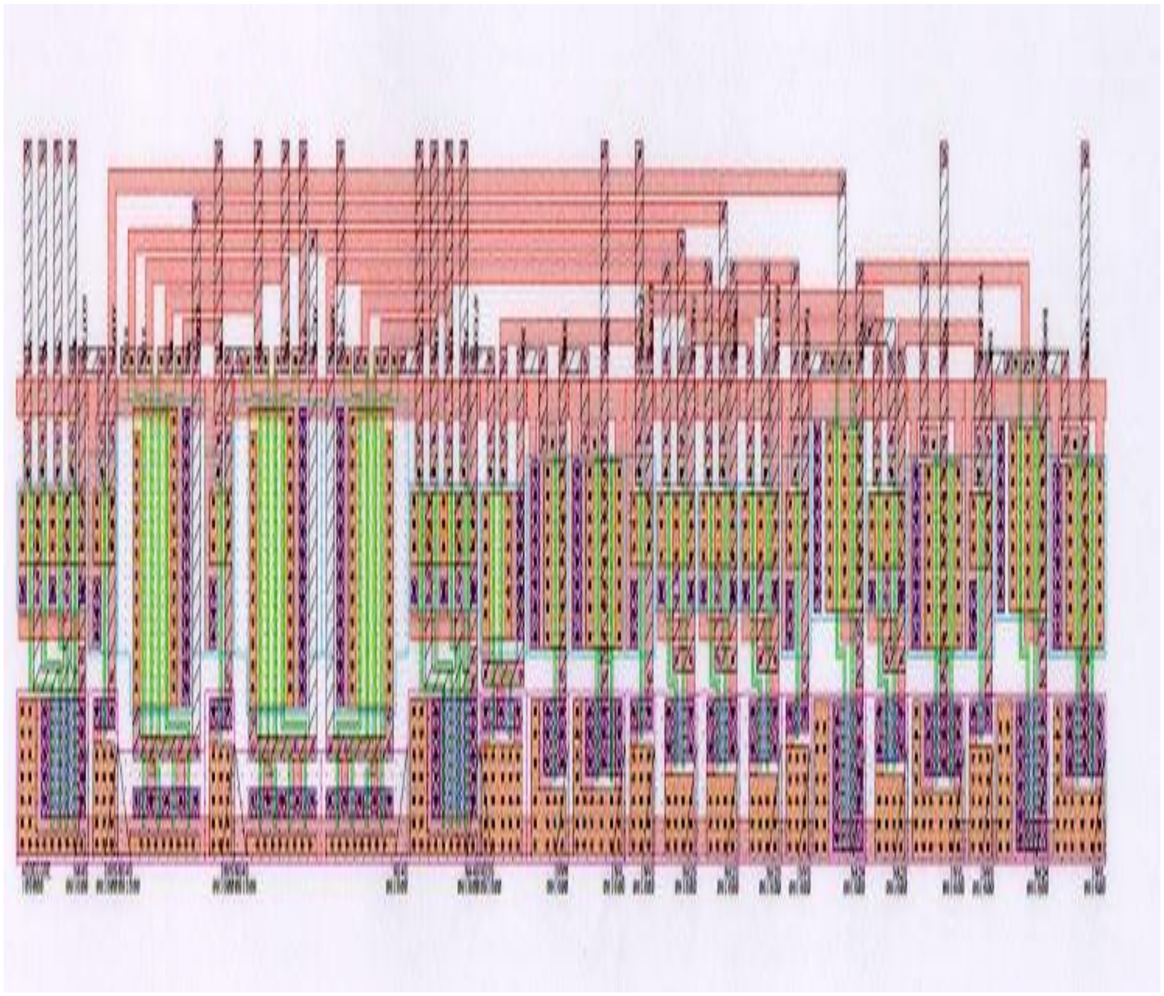


Figure 2.37 Physical layout of the W65C122S address decoder

Once the layout has been completed, the layout must have a DRC (Design Rule Check) done on it. This is to insure that all design rules for the chosen technology have been followed. It checks for the correct spacing between poly lines, and the correct width of both poly and metal lines to name a few. Table 2-1 gives the American Microsystems Inc. Semiconductor (AMIS) standard diffusion layer design rules for their CMOS .5μ process.

Table 2-1 AMIS .5micron process diffusion layer design rules

Rule Name	**Rule Description**	**Rule**	**Units**	**Rule Type**	**Notes**
DIFSP	Min DIFfusion SPacing	0.90	μm	*	
DIFW	Min DIFfusion Width	0.50	μm	*	Resistors less than 0.8 μm wide do not meet Parametric Specs and are not modeled accurately in simulation.
TBEOND	TuB Enclosure Of N-Diffusion	0.00	μm	*	Well Tie Only
TBEOPD	TuB Enclosure Of P-Diffusion	1.50	μm	*	
TBNDSP	TuB to N-Diffusion SPacing	1.50	μm	*	
TBPDSP	TuB to P-Diffusion SPacing	0.00	μm	*	Substrate Tie Only
TRANW	Minimum TRANsistor Width	0.80	μm	*	

(Rule Type: * Required, ** Recommended, Checked, *** Suggested, NOT Checked)

Once the layout has been completed and the DRC passes, the designer extracts a netlist of the layout, called NLE (NetList Extraction). The netlist describes the nodes and interconnects of the design. This netlist is used to perform an LVS (Layout versus Schematic) check. It is here that the schematic netlist is compared to the layout netlist. If the netlists match, the design can go to the final simulation before being sent out for fabrication. If the netlists do not match, the layout and the schematic need to be rechecked and corrected accordingly. In most cases, the mismatch lies within the layout, as the design has already been verified by simulation. For example, if in the schematic point A is connected to point B, but in the layout point A is connected to point C, the LVS will detect the error and point the designer to the problem area.

WDC, as well as many other companies, have their own standard cell library and design rules that they use for their designs. These cells (NAND gates, NOR gates, I/O Pads, etc…) are laid out individually according to the design rules and then placed into a library. When a new circuit needs to be laid out, the parts are picked from the library and then placed into the layout drawing. “Placement is the task of placing modules adjacent to each other to minimize area or cycle time” (6:431).

After the cells or modules have been placed, the modules need to be connected together. There are two methods to connect up, commonly referred to as routing, the modules. The first method is to manually route the interconnections to each module, which can be time consuming, but can better optimize the design by minimizing the lengths of the interconnections. The second method is automatically routing the interconnections. The automatic routing tools need input files or algorithms in order to guide the routing process. There are tools that will also automatically place the cells or design blocks, as well as perform automatic routing. Automatic place and route tools are very powerful, high-end tools, therefore very expensive. WDC uses the manual place and route method to layout a design.

It is important to note that all of WDC’s designs are done using WDC’s proprietary, 2-micron retargetable design rules. Once the design is laid out using the WDC design rules, the DRC is done against these rules. If the DRC passes, NLE and LVS can be performed on the design. Once the design passes DRC and LVS, the design can then be retargeted to a particular foundry for fabrication using the foundries design rules. This retargeting process will be discussed in Section 2.4.

2.3.1 Introduction to ICED

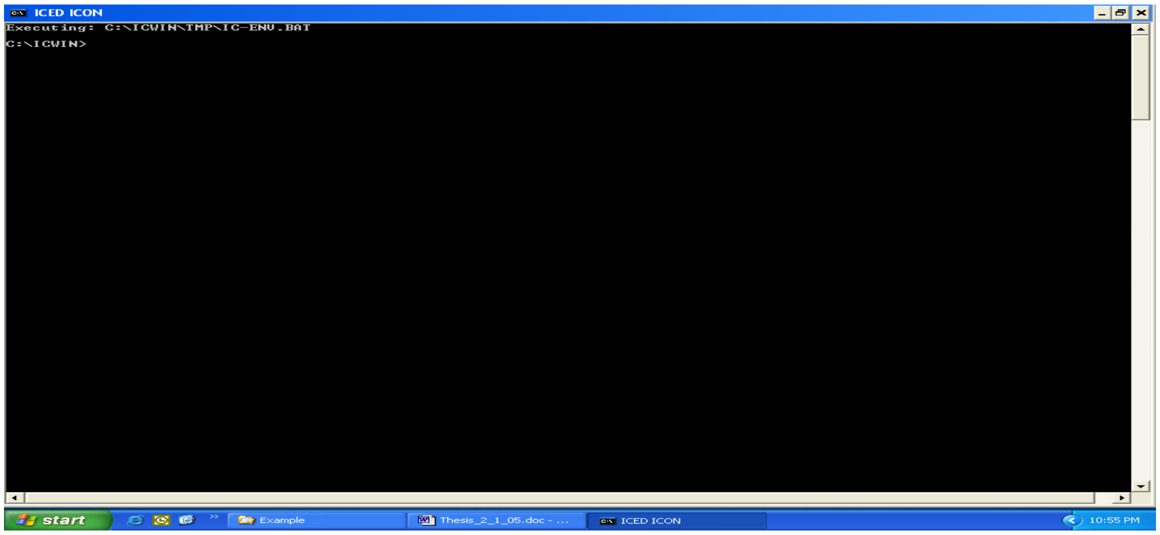
There are many IC Layout tools available, some are very costly and very powerful tools and some are free to use for non-commercial purposes. WDC uses the IC EDitor (ICED) layout software. This software is widely used in the industry and it is also used to perform the DRC and LVS on the design.

Laying out a design is a very complex process and will take some time to learn how to do it properly. Teaching how to do layout is beyond the scope of this tutorial. However, this section will introduce the user to the ICED software, and discuss some of the basic operations that will be needed in Chapter 3. This section will not go into performing a DRC or LVS on a design.

2.3.1.1 Working with ICED

As with the other software programs previously discussed, ICED is protected by a hardware key attached to the parallel port of the computer. Verify that this key is attached before working with ICED.

To begin, double-click on the ICED icon on the desktop of the ICED computer. You will get a DOS prompt as shown in Figure 2.38. ICED is designed to run using a DOS environment. We will utilize the C:\ICWIN\TUTOR directory for the remainder of this exercise. At the DOS prompt, change the working directory by typing: **cd tutor**. The DOS prompt should now read C:\ICWIN\TUTOR. Now, type: **del *.*** to erase the contents of the TUTOR directory.

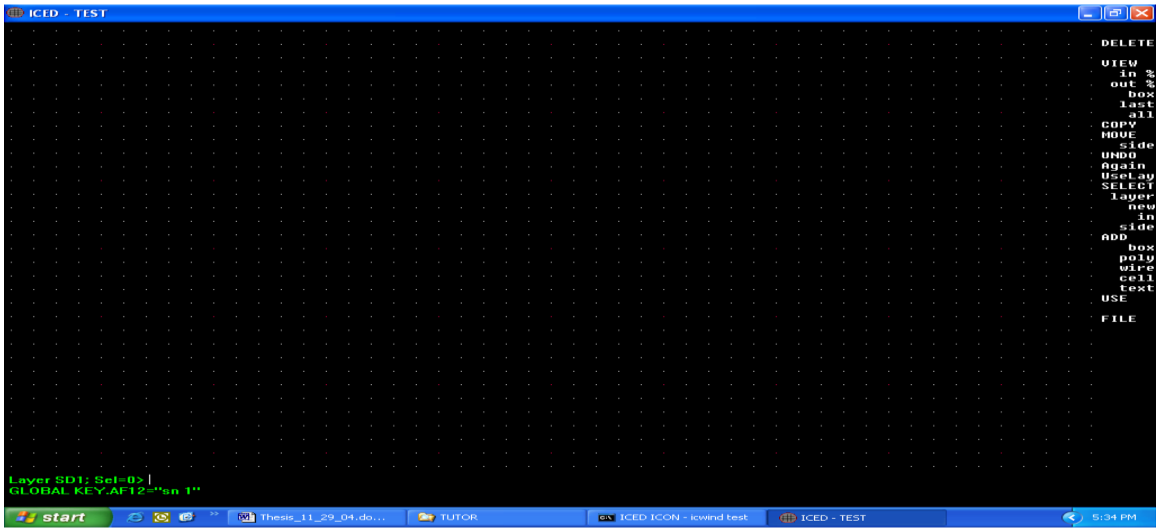


```
Executing: C:\ICWIN\TMP\IC-ENV.BAT
C:\ICWIN>
```

Figure 2.38 ICED ICON DOS window

As previously mentioned, WDC has their own set of design rules that define the layers for the N-Well, Poly and Metal layers to name a few. These layer definitions are located in a command file that ICED calls when it is invoked.

To open ICED, you must call this command file and provide a cell name, for example, TEST. At the DOS prompt, type: **icwind test.** Your screen will look like Figure 2.39.

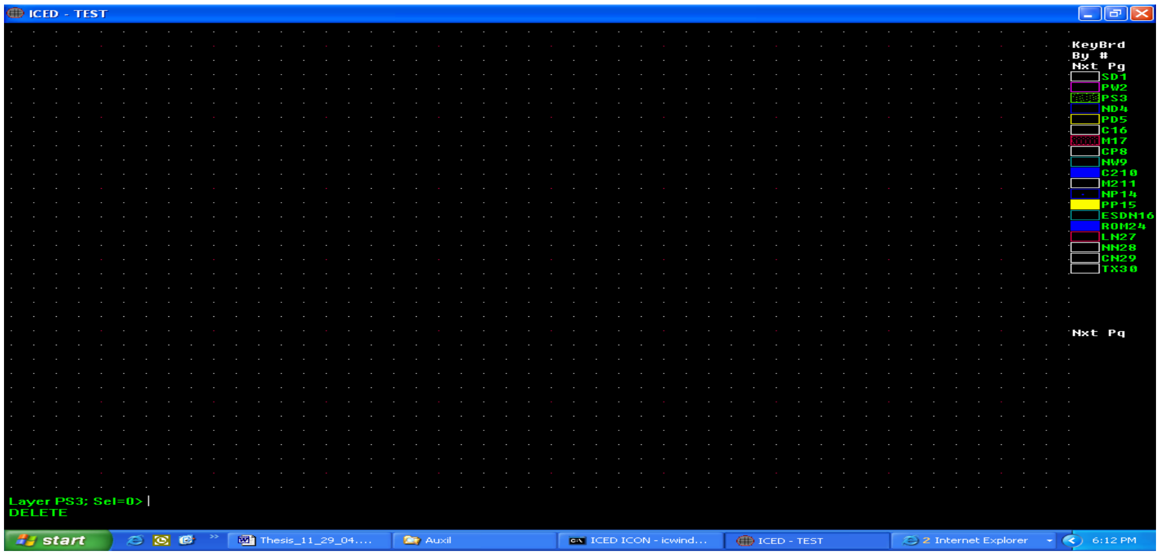


```
Layer SD1; Sel=0> |
GLOBAL KEY.AF12="sn 1"
```

Figure 2.39 ICED editor window

ICWIND is the name of the command file that will start ICED and call all of the command files needed, including the file with the layer definitions.

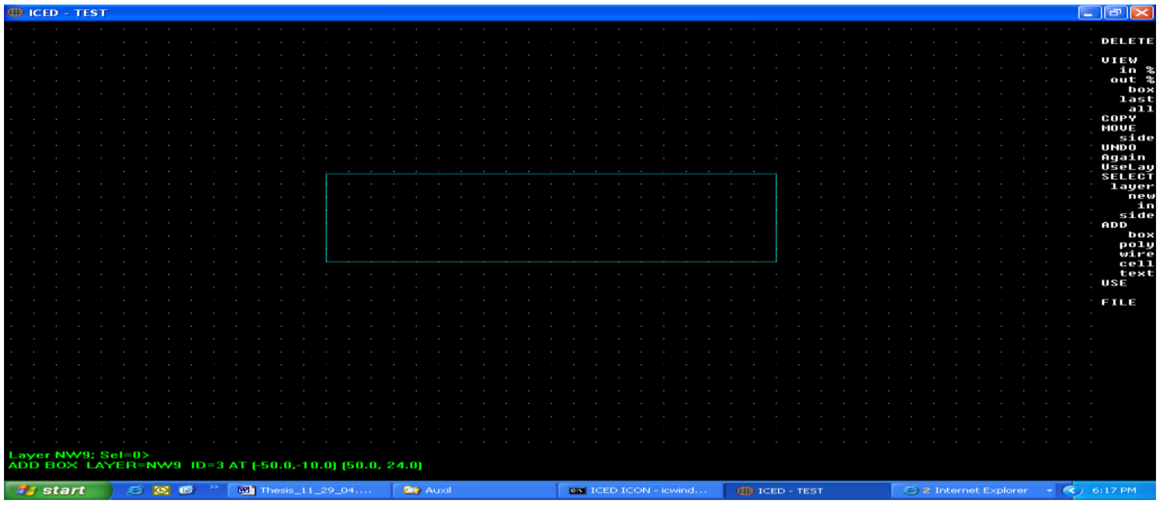
We are now ready to start drawing a cell. For this example, we will draw an NMOS transistor, introducing you to WDC’s layers and some basic functions of ICED. First, click the UseLay option from the menu on the right side of the window. Your screen will look like Figure 2.40.



```
Layer PS3; Sel=0> |
DELETE
```

Figure 2.40 ICED editor window showing layer menu

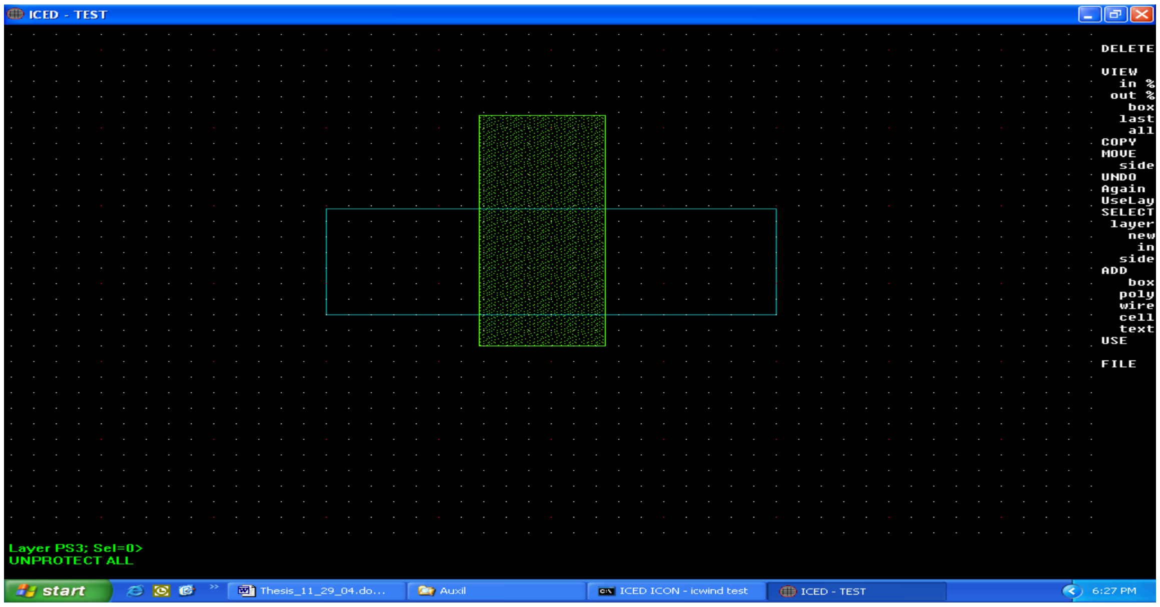
The menu now shows the WDC layers. Click the layer called NW9, as this layer is the N-Well layer. Notice that the menu items have changed. To add a box for the N-Well, click Box from the menu. Draw a box in the drawing window, similar to the one shown in Figure 2.41. We are not concerned at this point with the size of the box, as we are not following any design rules. If we were drawing an actual gate to be used in a design, we would have to follow the design rules and draw the N-Well, and other layers, following these rules.



```
Layer NW9; Sel=0>
ADD BOX  LAYER=NW9  ID=3 AT (-50.0,-10.0) (50.0, 24.0)
```

Figure 2.41 ICED editor window showing N-Well

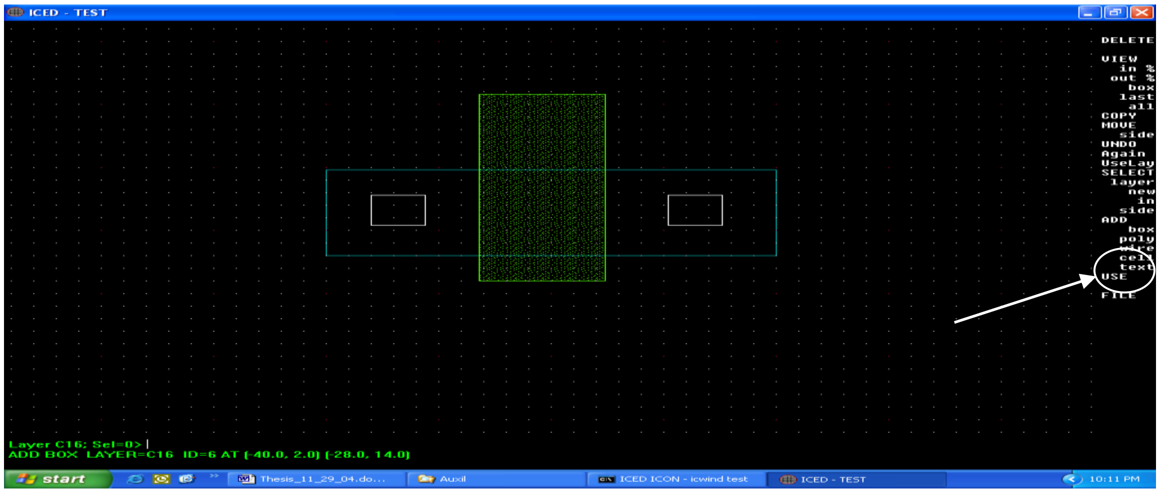
Next, we want to draw a poly line. As before, click UseLay, then PS3 for the poly layer. Add a box to the drawing as shown in Figure 2.42.



```
Layer PS3; Sel=0>
UNPROTECT ALL
```

Figure 2.42 ICED editor window showing N-Well and Poly layers

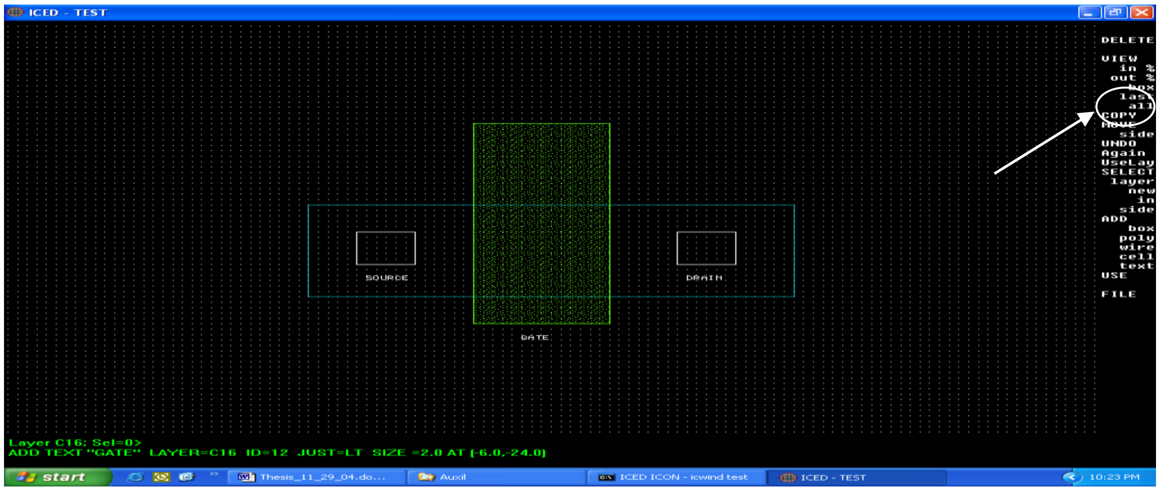
Next, we will add contacts to the source and drain regions. Again, click UseLay and select layer C16 for the contact layer. Add a box to the source and drain regions as shown in Figure 2.43.



```
Layer C16; Sel=0> |
ADD BOX  LAYER=C16  ID=6 AT (-40.0, 2.0) (-28.0, 14.0)
```

Figure 2.43 ICED editor window showing NMOS transistor

The last thing that we want to do is add some text to the drawing. To do this, Under the Add menu, click the Text option as shown in Figure 2.43. At the bottom of the screen will be a prompt reading Enter Text:. Type in Source and press Enter. This will add a box in a random location on the drawing. Using the mouse, move the text box to the source area and place the text box under the contact and press the left mouse button to anchor it. Do the same for the Drain and place the text for the Gate at the bottom of the poly line. When finished, your screen should look similar to Figure 2.44.



```
Layer C16; Sel=0>
ADD TEXT "GATE"  LAYER=C16  ID=12  JUST=LT  SIZE =2.0 AT (-6.0,-24.0)
```

Figure 2.44 ICED editor window with text on drawing

Now, under the View menu, click All, as shown in Figure 2.44. This will reformat the drawing window and expand the drawing to maximize the view.

This concludes the introduction to ICED. At this point, if we were drawing this to follow a set of design rules, a DRC would have to be run to ensure that all rules were correctly followed. If a schematic of the transistor was created, we could also do an LVS to make sure that our schematic netlist matched the layout netlist.

This section discussed a few basic operations and briefly touched on the capabilities of ICED. Students are encouraged to follow the comprehensive tutorial provided by ICED in order to gain a better understanding of ICED and learn a few more of the basic functions. The tutorial is located in C:\ICWIN\DOC.

To exit out of ICED and save all of the work done, type Exit at the prompt in the lower left corner of the editor window and press Enter. This will save the design in the Tutor directory and the file will be called test.cel. The ICED ICON DOS window is still open at this point, and can be closed by typing Exit at the DOS prompt.

2.4 Scaling and sizing the design (Retargeting)

Once all of the above steps are complete, the design is ready to be prepared for fabrication. In order for fabrication to take place, a foundry and one of their processes have to be chosen as a target for the design. As an example, for the device to be fabricated using the AMIS .5μ process, the transistors have to be scaled accordingly and the AMIS .5μ design rules must be followed.

An existing design can also be retargeted for a different process within the same foundry. If a design was first manufactured using a foundries .25μ technology, it can now be manufactured using their .18μ technology by scaling the transistors.

Sizing means to adjust the widths and spacing of the lines after they have been scaled, with the result being that they pass the targeted processes DRC's. The Western Design Center, Inc. has a unique approach when it comes to sizing a design. WDC has developed an Excel spreadsheet that has their design rules entered in one column. Another column is where the design rules for the targeted process are entered. The spreadsheet is then run and any errors detected will be reported. If design rule violations exist, further scaling and/or selective biasing may be required. Sometimes rule violations are not critical and can be ignored. In this way, a design that is done using WDC's design rules can be retargeted to any manufacturing process.

"The main benefits of scaling are (1) smaller device sizes and thus reduced chip size, (2) lower gate delays, allowing higher frequency operation, and (3) reduction in power dissipation" (2:120). Due to the smaller device sizes, more devices can fit on a wafer, resulting in a higher yield per wafer. For more information on scaling MOS transistors, see (6:250-255).

Once the scaling, sizing and layer mapping to the targeted process have been completed, this information is then put into files that ICED will use to DRC the design for the targeted process. When the DRC passes, LVS must be run on the design to ensure there were no errors caused in the retargeting process. When the retargeted design passes the DRC and LVS, the design is now ready for PSPICE simulation to check the electrical and timing characteristics of the design.

2.5 Electrical characterization and timing analysis

In this part of the design flow, the circuit is checked for its electrical characteristics and more intensive timing analyses are performed. The electrical characterization shows the operating voltages and currents throughout the design. The timing analysis will show switching times, set-up times, hold times and provide information on how fast the design will run. The timing analysis will also show the critical paths in the design. A critical path is the path that has the longest delay. If speed is a concern, the critical paths can be redesigned to minimize any delays.

Timing analysis can only be done after the layout, DRC, LVS and any retargeting steps have been completed. WDC uses PSPICE to perform timing analysis on designs. PSPICE, a Cadence Design Company product, is one of the more prevalent tools used to perform electrical characterizations on circuits.

In order for a PSPICE simulation to be performed, a netlist that can be read by PSPICE must be created. This PSPICE netlist is created from the netlist that was extracted from the layout using ICED. The netlist format created by ICED is not readable by PSPICE and needs to be converted into a PSPICE-readable netlist. WDC has created a PERL (Practical Extraction and Report Language) script that performs this conversion. Once the PSPICE netlist has been created, the PSPICE simulation can be performed.

2.5.1 Description of PSPICE simulation setup

This section will describe the files used for performing a PSPICE simulation and how to set up the PSPICE simulation. A PSPICE simulation takes about 10 hours to complete and is beyond the scope of this document to perform. The purpose of this section is to introduce WDC’s concepts behind setting up a PSPICE simulation.

2.5.1.1 The PSPICE simulation files

This section lists each of the files required to perform a PSPICE simulation and provides a description of each file.

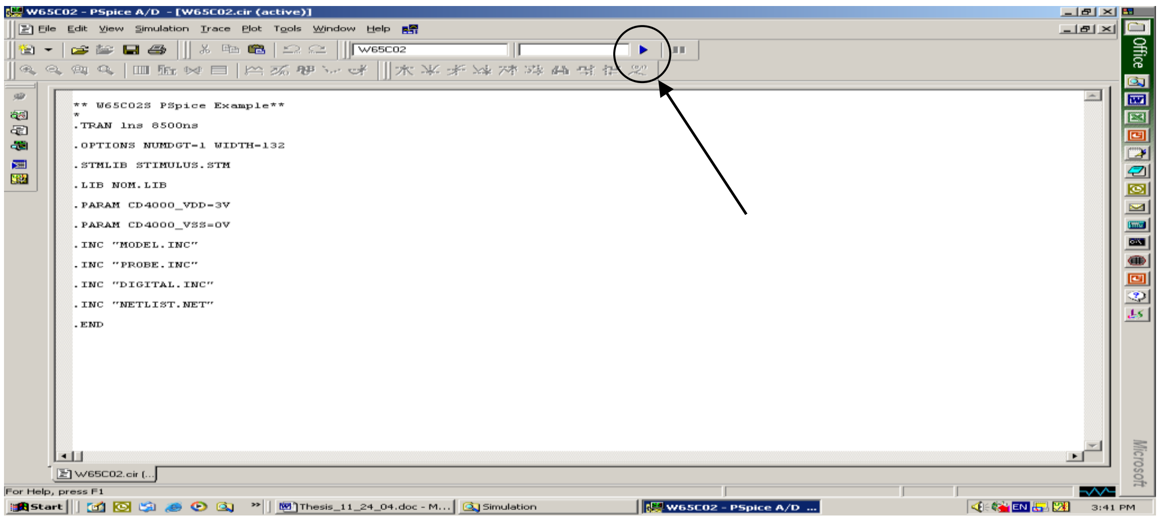
- Filename.cir – This CIRcuit file is the top level or project file for the simulation. This file contains all of the files required for simulation.
- Digital.inc – This file contains the Memory (RAM), A to D and D to A converters, buffers and logic circuitry that is used for the test setup.
- Model.inc – This file contains the transistor and capacitor model parameters that are used for simulation. These model parameters are the target foundries model parameters. For example, if the device is to be manufactured on the TSMC .5µ process, this file would contain the TSMC .5µ transistor and capacitor models.
- Probe.inc – This file contains the nodes that we want to look at and analyze after the simulation has completed. Both analog and digital nodes are specified here.
- Stimulus.stm – This is the file where the input signals needed for the simulation are defined. WDC uses a table to drive the inputs, referred to as table driven inputs.
- Netlist.net – This file contains the actual netlist of the design that is used for the simulation. This is the file that was created by converting the layout netlist using the PERL script.
- Ram.ihx – This file is used to program the RAM with the test data. The test data in this file is in the Intel hex format.

2.5.1.2 Setting up the PSPICE simulation

As previously mentioned, preparing for and running a simulation takes quite a long time and is beyond the scope of this tutorial. However, in order to gain an understanding of the process, the following section will describe the steps required for running a simulation using the W65C02S microprocessor as an example.

WDC has a folder on their file server called W65C02S, and in this folder are subfolders pertaining to various design steps for the W65C02S. The directory F:\W65C02S\PSpice\Example\Simulation contains the following files: W65C02.cir, digital.inc, model.inc, probe.inc, stimulus.stm, netlist.net and ram.ihx, as well as PSPICE generated files. These are the files that are required for simulation and these files have been described in Section 2.5.1.1. The transistor and capacitor model parameters that were used for the W65C02S simulation are the parameters of the Taiwan Semiconductor Manufacturing Corporation (TSMC) .5μ process. If a different TSMC process is to be used, the model parameters in model.inc will have to be changed while all other files remain the same. Copy the F:\W65C02S\PSpice\Example\Simulation directory to your C:\ drive so as not to alter the source directory.

To begin, the file W65C02.cir is opened by double-clicking it. This automatically opens PSPICE and Figure 2.45 shows the opening window.


W65C02 - PSpice A/D - [W65C02.cir (active)]
File Edit View Simulation Trace Plot Tools Window Help
W65C02
W65C02.cir
For Help, press F1
Start
Thesis_11_24_04.doc - M...
Simulation
W65C02 - PSpice A/D ...
3:41 PM
Office
Microsoft

```
** W65C02S PSpice Example**
*
.TRAN 1ns 8500ns
.OPTIONS NUMDGT=1 WIDTH=132
.STMLIB STIMULUS.STM
.LIB NOM.LIB
.PARAM CD4000_VDD=3V
.PARAM CD4000_VSS=0V
.INC "MODEL.INC"
.INC "PROBE.INC"
.INC "DIGITAL.INC"
.INC "NETLIST.NET"
.END
```

Figure 2.45 PSPICE main window

Figure 2.45 shows the contents of the .CIR file and lists the time that the simulation is to run to, parameter declarations, and file names for running the simulation.

As previously mentioned, the simulation will take about 10 hours to complete, therefore it will not be performed here. However, the following will describe the process for completeness. A simulation is started by clicking on the blue arrow as shown in Figure 2.45. There will be some activity in the bottom right hand corner of the PSPICE window. It will give the percentage completed of the simulation and the time in μs where the simulation is currently running at.

When the simulation is completed, PSPICE creates a data file, a vector file and other files that are associated with the simulation. To view the waveforms, open the data file W65C02.dat by double-clicking it. Figure 2.46 shows the data (waveform) window.

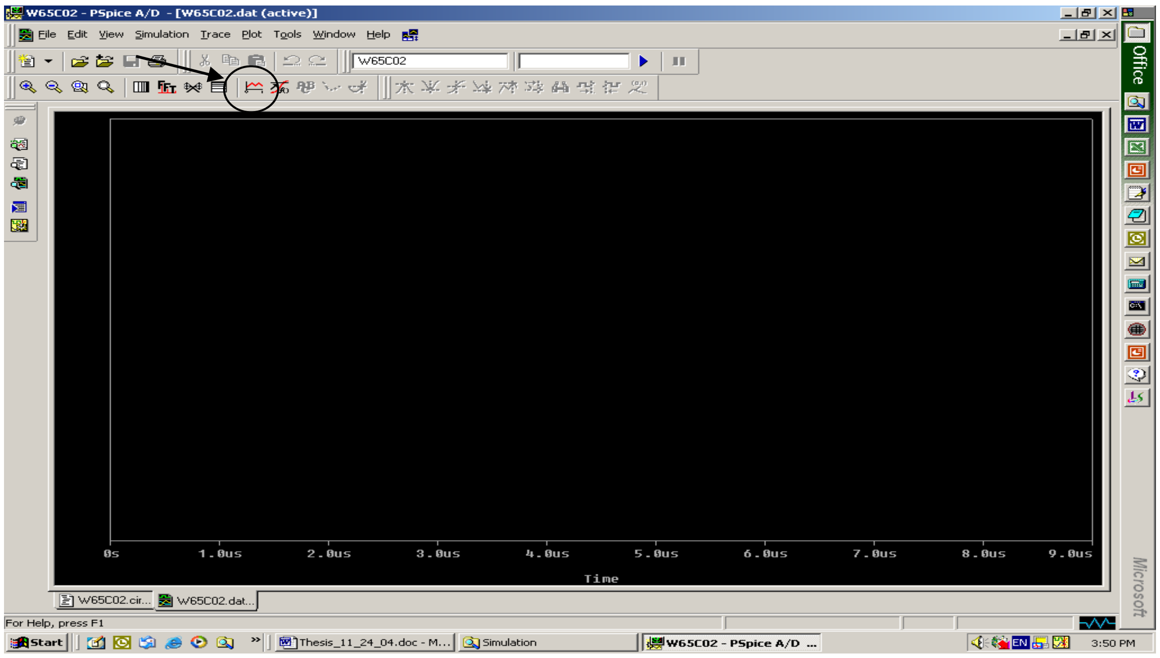


Figure 2.46 PSPICE waveform window

To add the signals that you want to analyze, click the Add Trace icon as shown in Figure 2.46. This will cause a pop-out window to appear and your screen will look like Figure 2.47.

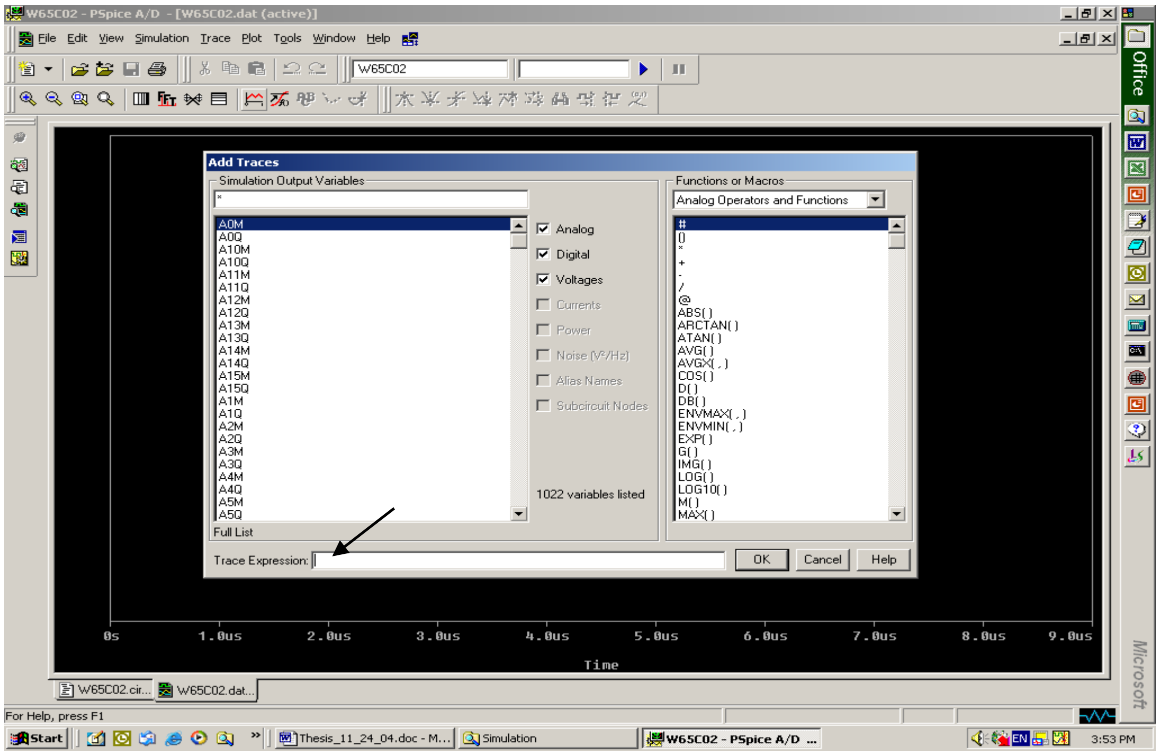


Figure 2.47 PSPICE add trace window

The nodes that were defined in the probe.inc file appear in the left column of the window while mathematical functions appear in the right hand column. To view a signal, locate the desired signal from the left column and double click it. This will place it in the Trace Expression field indicated in Figure 2.47. For this example, select the following signals: D0Q, D1Q, D2Q, D3Q, D4Q, D5Q, D6Q, D7Q and PHI2Q. After selecting the desired signal(s), click the OK button to add these signals to the Waveform window. After adding the desired signals, your screen will look similar to Figure 2.48.

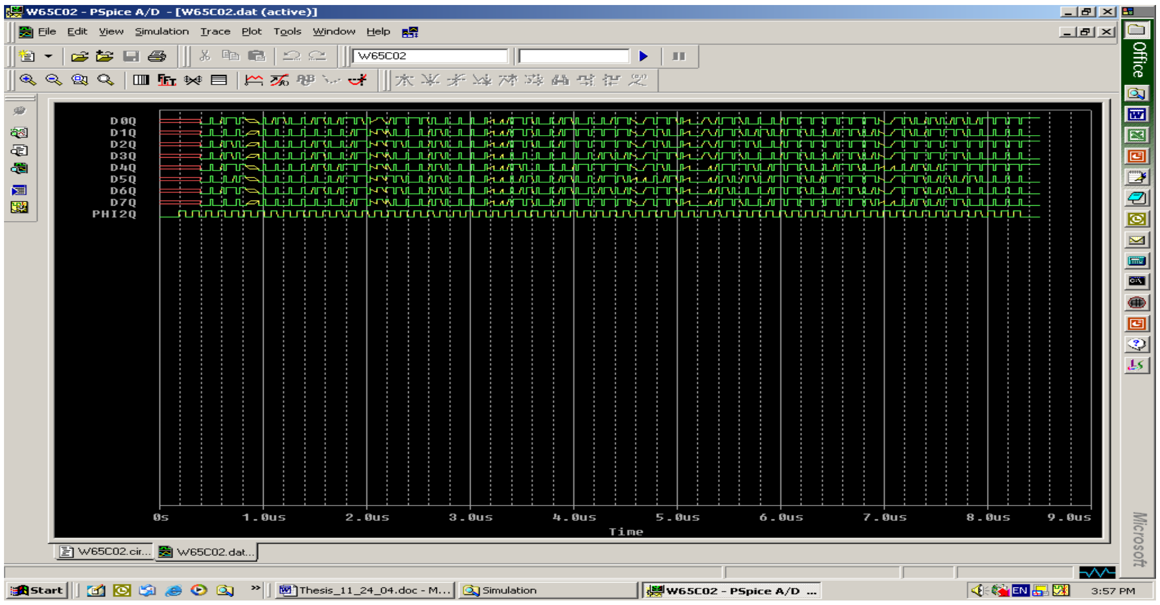

W65C02 - PSpice A/D - [W65C02.dat (active)]
File Edit View Simulation Trace Plot Tools Window Help
D0Q
D1Q
D2Q
D3Q
D4Q
D5Q
D6Q
D7Q
PHI2Q
0s
1.0us
2.0us
3.0us
4.0us
5.0us
6.0us
7.0us
8.0us
9.0us
Time

Figure 2.48 PSPICE waveform window with signals displayed

Figure 2.48 only shows the data bus and clock signals for reference purposes. WDC has a 'gold standard' output file that all new simulations are compared to. All of the signals to be viewed can be obtained from this 'gold standard' output file. The waveforms of the new simulation are compared to the 'gold standard' simulation waveforms. If the waveform patterns match, the design is working properly. If the patterns do not match, the differences need to be analyzed further and modifications may need to be made to the layout, design rules or transistor parameters and the simulation must be run again.

At this point, if desired, add more digital and analog signals to get a feel for how the signals look in order to get more familiar with PSPICE. Close PSPICE when finished.

After the PSPICE simulation has been successfully completed, the design is now ready to be sent to the foundry for fabrication.

2.5.2 Further comments on retargeting and simulating a design

Retargeting and preparing a design for simulation is a very involved process and will take some time to gain a complete understanding of the process. WDC has created a procedure document that goes into more detail on the retargeting steps and steps required for performing a PSPICE simulation. Some of these steps were described in brief in the previous sections. Students are encouraged to go through this procedure document and perform the steps outlined in order to gain a better understanding of the process.

The following list shows the steps that WDC uses for retargeting a design. Note that the list has three (3) DRC steps. One is for the core (W65C02C or other microprocessor/microcontroller), one is for the pad ring and the last is for the complete design (core connected to the pad ring). WDC has a pad ring that they have designed for all of the I/O, power and ground pads for their microprocessors/microcontrollers. The procedure is as follows:

1. Select the target process
2. Enter target rules into WDC Retargetable Design Rules Excel Spreadsheet for cores
3. Convert layers from WDC layers to the targeted process layers
4. Scale using scale factor from Excel spreadsheet for cores
5. Bias using bias calculations from Excel spreadsheet
6. DRC core using the targeted process core design rules
7. Enter target rules into WDC Retargetable Design Rules Excel Spreadsheet for pad ring

8. Scale ring using scale factor from Excel spreadsheet for pad ring
9. DRC pad ring using the targeted process design rules for I/O
10. Wire the core to the ring
11. DRC using the targeted process design rules
12. Extract the SPICE netlist
13. LVS using extracted netlist and chip netlist
14. Perform PSPICE simulation
15. Ship completed design to the foundry
16. Receive chips back from foundry
17. Test chips for functionality

2.6 Fabricating the design

The last step in the design flow is sending the completed design out for fabrication. As was previously discussed, a particular foundry and manufacturing process is targeted for the design to be fabricated on. There are many factors associated with choosing a foundry and process. For example, it is desirable to have the design run at a foundry that has a good reputation and high yields for their processes.

Cost is another factor in choosing a process. In general, the smaller the transistor geometry, the more expensive it will cost to fabricate due to high mask set costs. A mask set for a design that is run on a .5µ process can cost around $50,000 - $100,000 dollars while a mask set for a .18µ process can be as high as $1,000,000 or greater.

It is also desirable to prove out the designs functionality before sending it out for full production fabrication. Many foundries offer what they call a Wafer Shuttle, where there are different designs from different companies on one wafer. This is called a Multi

Project Wafer (MPW). MPW's are a low-cost option for obtaining a limited number of samples that can be evaluated for correct functionality. Typically, the number of chips received in a MPW run is 40. The cost for a MPW run is $6000 - $10,000 dollars, depending on the foundry. By having multiple projects on one wafer, the mask set costs are divided by the different companies, hence the low cost.

If the samples received from a MPW run are functional, an order for full production wafers can be placed. Typically, a foundry requires that a minimum order of 25 wafers be placed for a production run. If a MPW run was not done, it would be very painful and costly to learn that a device did not work coming out of a production run.

2.6.1 The MOSIS Integrated Circuit fabrication service

MOSIS is a foundry service that started in 1981 and is based out of the University of Southern California. MOSIS offers both low-cost prototyping services and small volume dedicated runs. MOSIS is available to commercial businesses, government agencies and educational institutions.

MOSIS is a partner with many different foundries and they run different processes on a monthly basis. This gives businesses a low cost option to have their designs run on different foundries utilizing the MPW runs. For example, if a company wants to fabricate their design on the TSMC .35μ process, they can go to MOSIS to get 40 pieces from a MPW run, verify that the design works properly on this process and then go directly to TSMC for production. The cost for a MPW run at MOSIS is $5500 for educational institutions and $6500 for commercial businesses. Along with fabrication services, MOSIS also offers device packaging services. WDC utilizes MOSIS to prove out a design before committing to production.

Depending on the chosen process, fabrication time is roughly 8 to 10 weeks. When a design is submitted to MOSIS, a check of the project is performed to verify that the design meets the criteria of the chosen process. If there are errors detected, MOSIS support will contact the company to inform them of the problems. Once all problems are resolved, the design is put in the queue for fabrication.

Once fabrication is complete, packaged parts, if this service was requested, are returned to the company and the company can begin testing the parts. WDC uses MOSIS only for fabrication of the design. WDC uses a different vendor for packaging the parts.

2.7 Testing the packaged parts

Once the packaged parts have been received, the parts need to be thoroughly tested before being released for sale to customers. There are two ways that WDC tests their packaged parts. One is in-house on a developer board or prototype board and the other is on an automated tester.

2.7.1 In-house testing

WDC has a developer board for each of their microprocessors and microcontrollers. These boards interface to the PC and test programs can be downloaded into the on-board RAM and run to test the boards' functionality. If the programs run correctly with the new parts installed, the new parts can be considered acceptable and released for sale. However, while an in-system test is good practice, it is always good to run the parts on an automated tester to exercise the parts with more vigorous test programs over different voltage and frequency ranges.

2.7.2 Testing on automated testers

After performing some in-house testing to see that the new parts are working correctly, WDC runs production tests on a Sentry automated tester at a testing facility in Phoenix. Test programs have been written for these testers to test all aspects of the parts. These tests range from writing to and reading from all registers, testing all op-codes and checking timing characteristics, for example. The parts are tested at operating voltages of 1.8V to 6V and operating frequencies ranging from 1MHz to 14MHz. Parts are also tested at higher frequencies to determine the maximum frequency at which the parts will run. When the parts pass all of the tests on the automated testers, they are then considered functional and are made available as production samples to customers.

At this point, the parts have been tested in an in-house system and on the automated testers. As a final test, samples of the new devices are sent to customers to validate in their own systems to verify the parts' functionality. There is always a possibility that the previous tests have missed something or perhaps the test programs need to be modified to reflect timing changes introduced by the new fabrication process. When customers report that the new devices are working in their systems, the devices are released for sale as fully tested production parts.

3. System on a Chip Design Tutorial

3.1 Introduction to the W65C122S System Chip

The W65C122S is a fully static 8-bit System on a Chip microcomputer that is comprised of the W65C02S microprocessor, W65C22S Versatile Interface Adapter (VIA) for I/O, RAM and ROM.

The W65C22S provides programmed control of up to two peripheral devices on Ports A and B. These bi-directional I/O ports provide direct interfacing between the microprocessor and peripheral units. There is 4KB of ROM and 256 bytes of RAM on the chip. There are also eight available pins that can be used to further customize the W65C122S. For example, a UART or an A to D and D to A converter can be added into the W65C122S. Table 3.1 shows the features of the W65C122S and Table 3.2 shows the system memory map.

The projects and files used in the W65C122S tutorial are located on the WDC file server under F:\Tutor122. Make a copy of this directory and place it on the C:\ drive of the computer you are using so as not to change the source directory.

Table 3.1 – Features of the W65C122S

MPU Core	W65C02S
ROM	4096 bytes
RAM	256 bytes
I/O Core	W65C22S
Timer/Counters	2
Synchronous Serial Port	1
Low Power Modes	WAI, STP
Power supply	2.8V-5.5V
Package type	68 PLCC

Table 3.2 System memory map

Address	Label	Function
FFFE-FFFF	IRQ	IRQB vector
FFFC-FFFD	RESET	RESETB vector
FFFA-FFFB	NMI	NMIB vector
F000-FFF9	ROM	On chip ROM
0100-01EF	RAM	On chip RAM (Page 1) Stack
00F0-00FF	VIA	On chip VIA
0000-00EF	RAM	On chip RAM (Page 0)

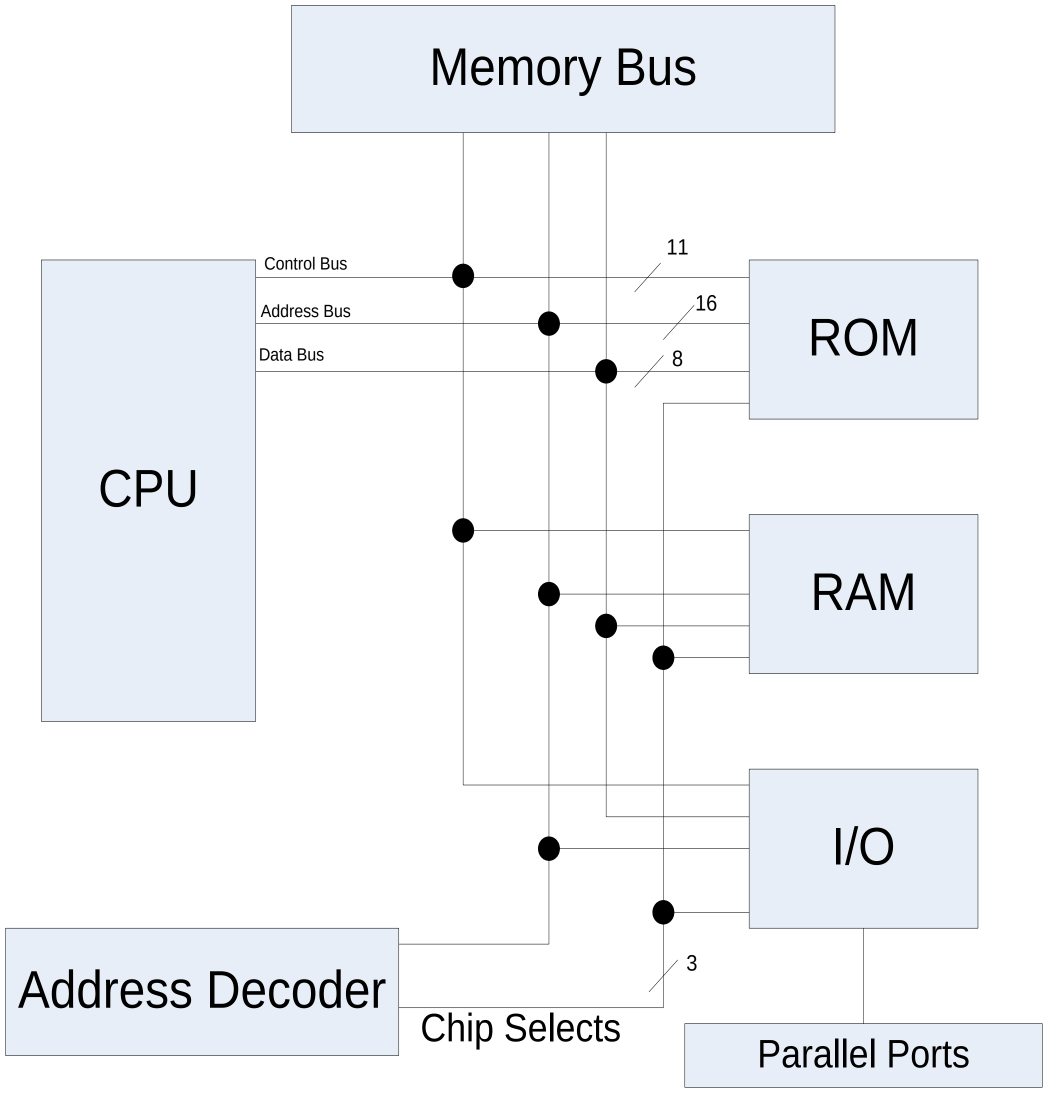


Figure 3.1 Block diagram of the W65C122S

3.2 W65C122S pin descriptions

3.2.1 Address Bus (A0-A15)

The 16–bit Address Bus (A0-A15) is used to address memory and I/O.

3.2.2 Bus Enable (BE)

The Bus Enable (BE) input signal controls the address, data and the RWB (Read/Write Bar) buffers. When BE enable is high, the RWB, data and address buffers are active. When BE is low, these buffers are in the high impedance state and they may be driven by external circuitry such as Direct Memory Access (DMA) Circuitry. This is an asynchronous signal.

3.2.3 Control Lines (CA1, CA2)

The Control Lines (CA1, CA2) serve as the interrupt inputs or handshake outputs for Port A. Each line controls an internal interrupt flag with a corresponding interrupt enable bit. CA1 also controls the latching of Input Data on Port A. (See W65C22S Data Sheet)

3.2.4 Control Lines (CB1, CB2)

The Control Lines (CB1, CB2) serve as interrupt inputs or handshake outputs for Port B. These two control lines control an internal interrupt flag with a corresponding interrupt enable bit. These lines also can be configured as a serial data port under control of a Shift Register. (See W65C22S Data Sheet)

3.2.5 Data Bus (D0-D7)

The Data Bus (D0-D7) is used for data exchanges between microprocessors, memory and peripherals.

3.2.6 External Memory (EXTMEMB)

The External Memory (EXTMEMB) input controls the memory map. When EXTMEMB is high, the on-chip RAM and ROM are accessed at the memory locations in Table 3.2. When EXTMEMB is low, the External memory is accessed at these addresses.

3.2.7 FCLK

This is the main clock source for the system and is driven from an oscillator.

3.2.8 Interrupt Request (IRQB)

The Interrupt Request (IRQB) signal requests that an interrupt sequence begin within the microprocessor. The IRQB is sampled at the falling edge of PHI2 operation. If the Interrupt disable flag (I) in the processor status register is zero, the current instruction is completed and the interrupt sequence begins during PHI2 low. The IRQB is a level sensitive interrupt and therefore must remain in the active low state when RDY is low and IRQB is recognized when RDY goes high and the current instruction is completed. The following sequence occurs:

- The microprocessor stores the program counter and the status register to the stack.
- The microprocessor sets the Interrupt disable flag high (I) so that no further interrupts are recognized
- The Program Counter (PC) is loaded from location FFFE and FFFF. This is the start of the interrupt handler routine. (See W65C02S Data Sheet)

3.2.9 No Connect (NC)

The No Connect (NC) pins are not connected internally and may be connected externally as will occur in the W65C134DB Developer Board for testing. These pins may be used for new features such as UARTS, A to D and D to A. (See W65C134S Data Sheet)

3.2.10 Non-maskable Interrupt (NMIB)

The Non-maskable Interrupt (NMIB) input is an edge sensitive interrupt that is sampled at the falling edge of PHI2. After the current instruction is completed, the interrupt sequence begins. The program counter is loaded with the vector from FFFA and FFFB (high byte). Since this is a non-maskable interrupt, another interrupt will occur while the microprocessor is servicing one. No interrupt can occur if NMIB is low and a negative going edge has not occurred since the last non-maskable interrupt. (See W65C02S Data Sheet)

3.2.11 Peripheral Data Port A (PA0-PA7)

The Peripheral Data Port A (PA0-PA7) is an 8-bit bi-directional bus used for data transfer of data, control, and status information between the W65C122S and a peripheral device. The input data maybe latched into register A of the VIA using the CA1 pin. (See W65C22S Data Sheet)

3.2.12 Peripheral Data Port B (PB0-PB7)

The Peripheral Data Port B (PB0-PB7) is an 8-bit bi-directional bus. The output signal on the PB7 may be controlled by Timer 1 while Timer 2 may be programmed to count pulses on PB6. (See the W65C22S Data Sheet)

3.2.13 PHI2

The PHI2 clock signal is generated from FCLK and provides the timing for the system.

3.2.14 Power Supply (VDD, VSS)

The Power Supply (VDD, VSS) are the positive (VDD) and ground (VSS) power pins. The W65C122S has two VDD pins, one for the core, Pin 36, and one for the pad ring, Pin

61. Likewise, there are two VSS pins, one for the core, Pin 53, and one for the pad ring, Pin 27. The core and buffer ring power dissipation can be measured independently.

3.2.15 Read/Write (RWB)

The Read/Write (RWB) signal indicates that the microprocessor is reading or writing from memory or I/O bus. When in the high state the microprocessor is “reading” from memory or I/O. In low state, the microprocessor is writing to the addressed memory location. (See W65C02S Data Sheet)

3.2.16 Ready (RDY)

When the Ready (RDY) signal is pulled low, the processor will stop in its current state and will remain in the state until the RDY line goes high.

3.2.17 Reset (RESB)

The Reset (RESB) signal resets the microprocessor stopping all operation and resets the VIA. (See W65C02S and W65C22S Data Sheets) RESB should be held low for at least two cycles after VDD reaches operation voltage from a power down. A positive transition of the signal returns the microprocessor to full operation. The interrupt flag is set, the decimal mode is cleared and the program counter is loaded with the vector from locations FFFC and FFFD (See W65C02S Data Sheet)

3.2.18 Synchronize (SYNC)

The Synchronize (SYNC) signal is high when the microprocessor is fetching an opcode. (See W65C02S Data Sheet)

3.3 Working with the W65C122S schematic using ViewDraw

In this section, we will be working with the W65C122S using ViewDraw. The complete W65C122S schematic has already been completed and verified. In Section 2.1.3 you were introduced to some basic operations using ViewDraw. In this exercise, we will be opening an existing project that contains the complete W65C122S schematic, examining and explaining the different blocks that make up the schematic and extracting the Verilog netlist of the W65C122S that will be used for the Verilog simulation using Silos.

3.3.1 Introduction to the W65C122S Schematic

Browse to C:\Tutor122 and double-click on the W65c122s_Tutorial.dproj file to open the project in ViewDraw. Your screen should look like Figure 3.2.

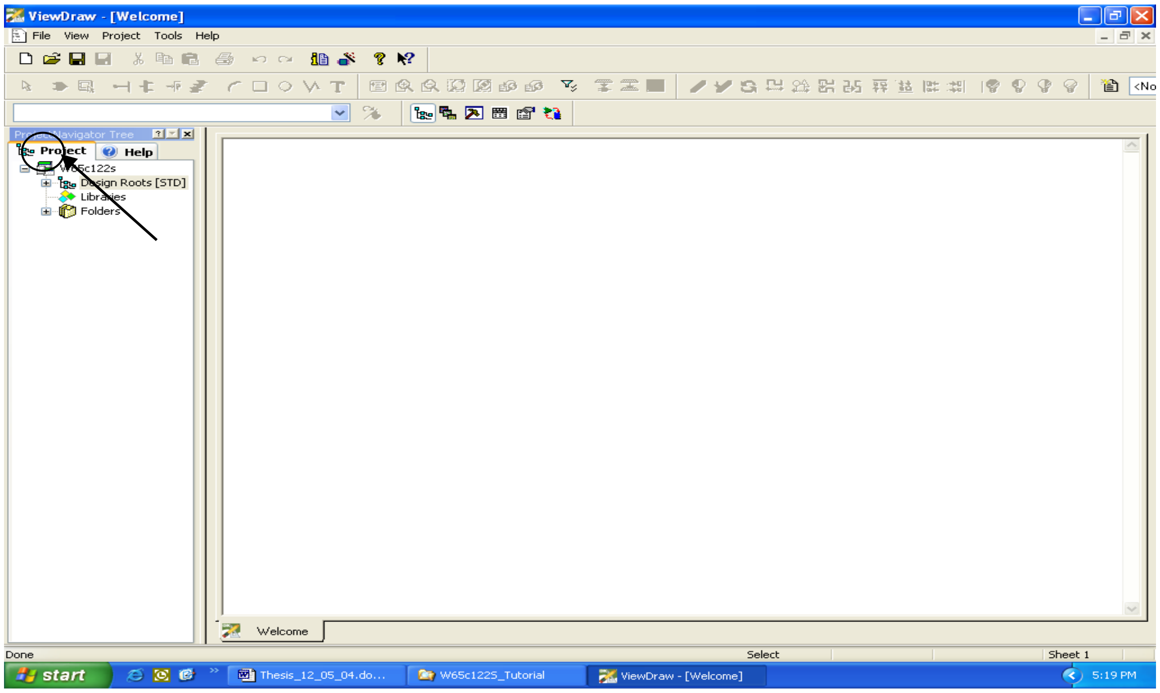


Figure 3.2 W65C122S ViewDraw project window

Open the Design Roots menu by clicking on the plus (+) sign to the left of it, as shown in Figure 3.2. This will display W65C122S with a green symbol next to it. Click on W65C122S to open the schematic. Once opened, your screen will look like Figure 3.3.

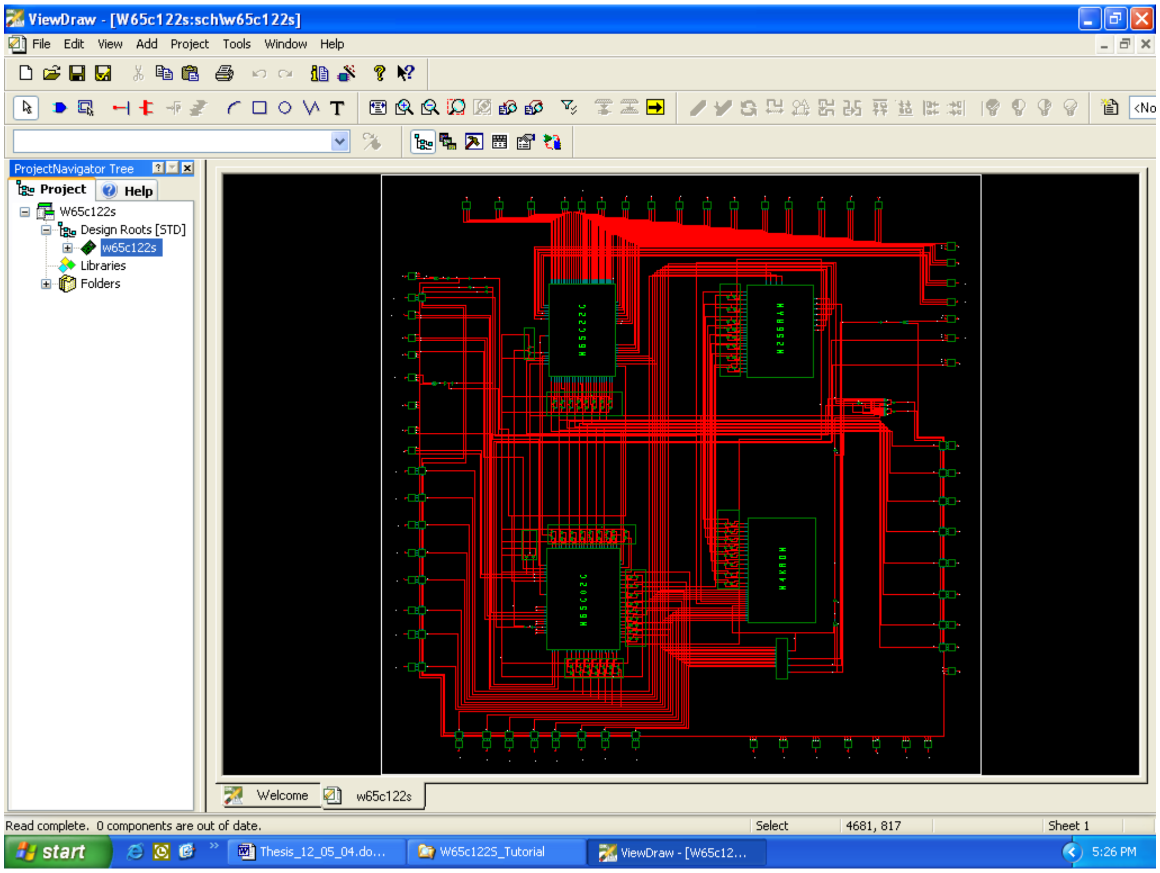


Figure 3.3 W65C122S schematic in ViewDraw

This is the complete schematic of the W65C122S. Notice that there are four (4) main blocks in the schematic. The blocks are: the W65C22C VIA core for I/O (top left), W256RAM which is a 256 byte RAM module (top-right), W65C02C microprocessor core (bottom left) and W4KROM which is a 4KB ROM module.

The outer ring of the schematic is comprised of the I/O and power pads making up the pins of the device. Use the zoom window feature and zoom in on the block that is located under the W4KROM block. This is the Address Decoder block. You will also notice some random logic gates around the schematic that are required for some signals. At the bottom right corner of the schematic, there are eight (8) unconnected pins. These pins are available for the addition of other blocks that can further enhance and customize the W65C122S. For example, a FLASH RAM block can be connected to these pins for more memory, or perhaps A to D and D to A converters can be added.

The blocks for the W65C02C, W65C22C, W256RAM and W4KROM are just symbol representations and they each have a Verilog file associated with them to describe their functionality. For example, click on the W65C02C symbol on the schematic to highlight it. Next, right-click on the symbol to display a pop-out menu which shows the available options for working with the symbol. This is shown in Figure 3.4.

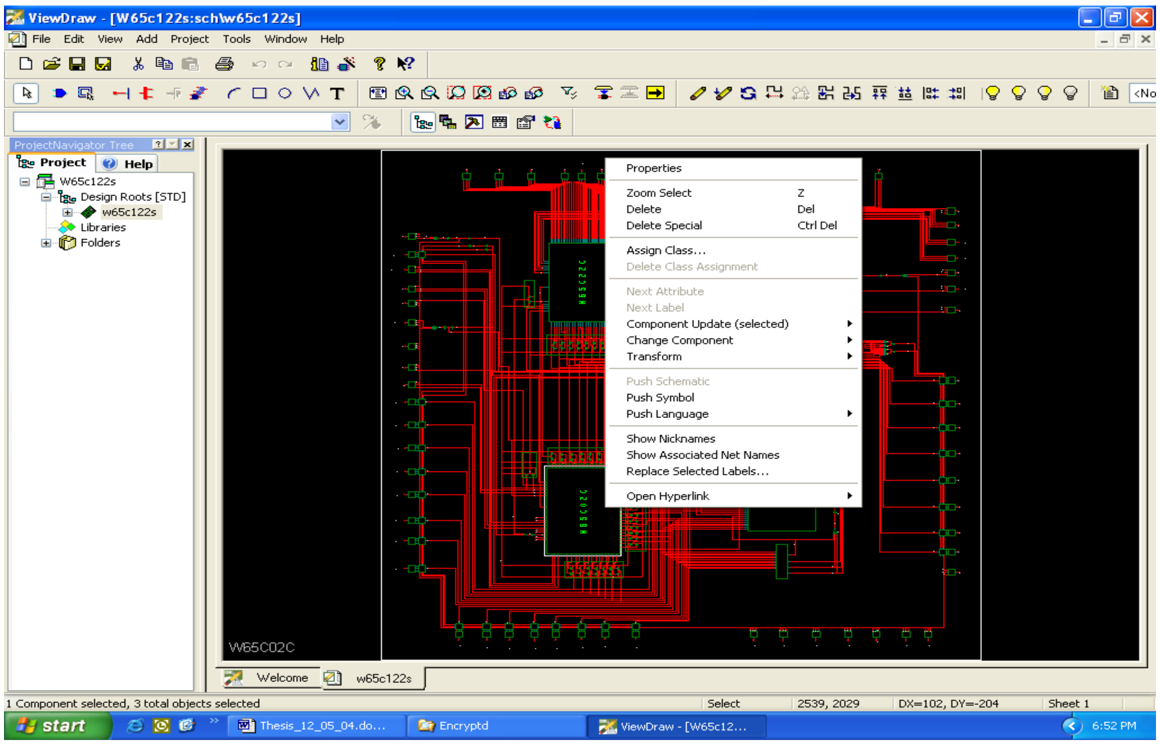


Figure 3.4 Symbol pop-out menu

First, select the Push Symbol option from the list to open the symbol for the W65C02C. Your screen will look like Figure 3.5.

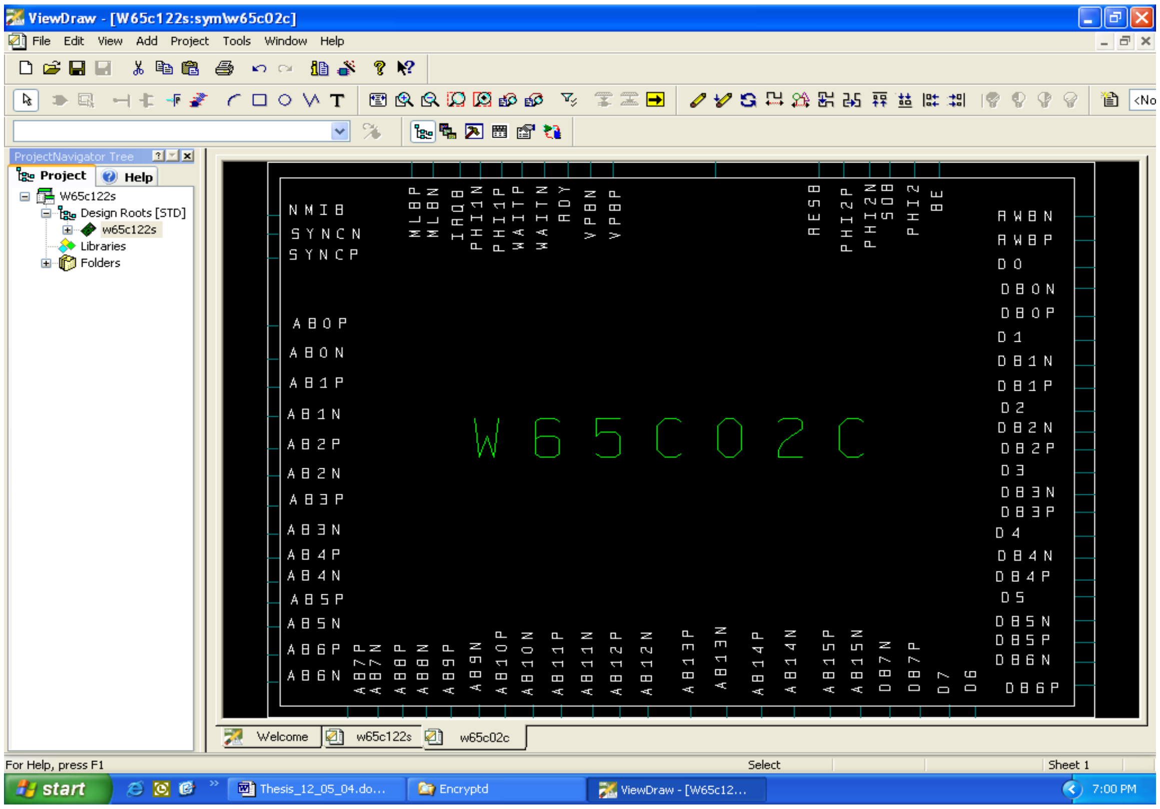
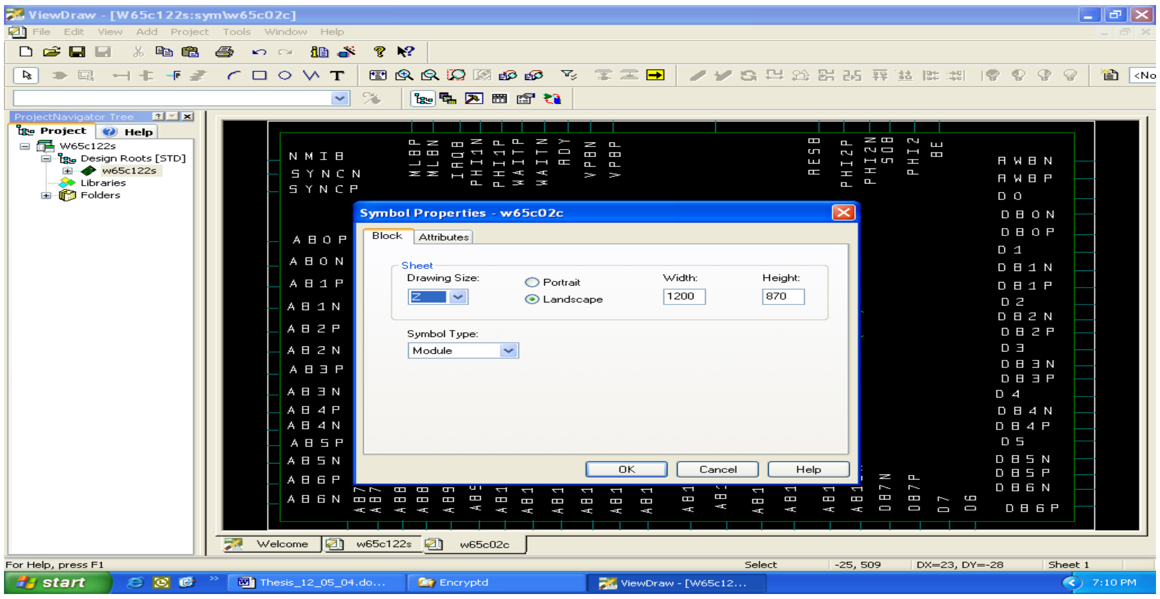


Figure 3.5 W65C02C ViewDraw symbol

All of the pins that are associated with the W65C02 core are placed on the symbol and labeled accordingly. As previously mentioned, there is a Verilog file that is associated with this symbol to describe its functionality. To view the properties of the symbol, right-click in the area to the left or to the right of the symbol to display another pop-out menu and select the Properties option. After doing so, your screen will look like Figure 3.6.



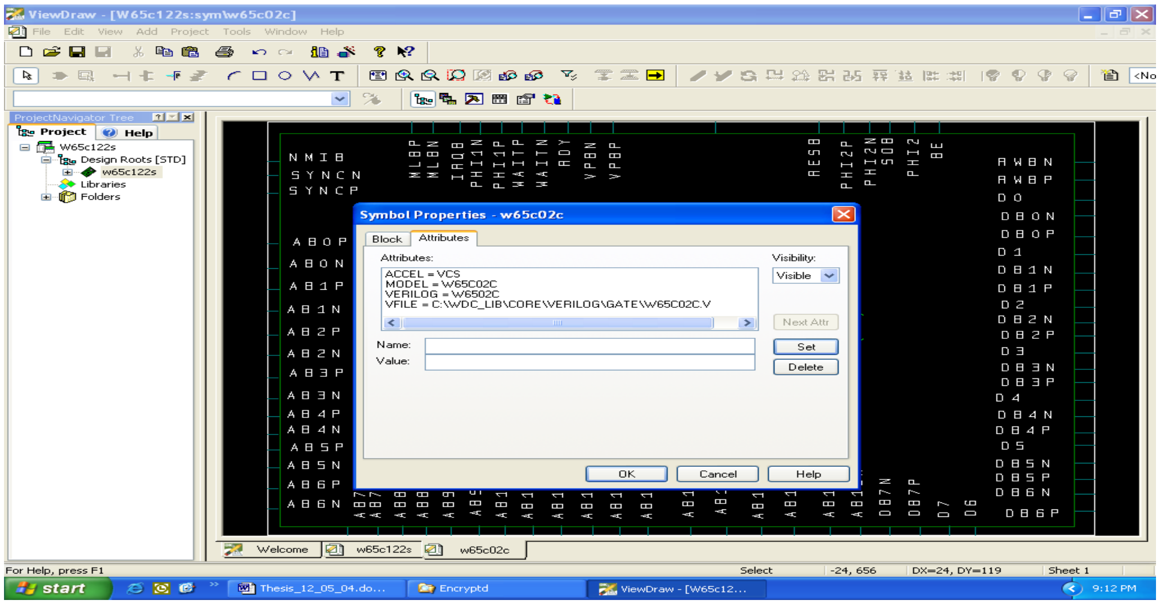
Symbol Properties - w65c02c

Sheet
Drawing Size: Z
Portrait [✓ Landscape]
Width: 1200
Height: 870

Symbol Type: Module

Figure 3.6 Symbol properties window

The Block window defines the drawing size for the symbol and the type of symbol. Use the pull down arrows to see the available drawing sizes and Symbol types. For more information on the different symbol types, consult the online help feature. Next, click on the Attributes tab. Your screen will look like Figure 3.7.



Symbol Properties - w65c02c

Attributes:

```
ACCEL = VCS
MODEL = W65C02C
VERILOG = W6502C
VFILE = C:\WDC_LIB\CORE\VERILOG\GATE\W65C02C.V
```

Visibility: Visible

Figure 3.7 Symbol attribute window

There are a number of different attributes that can be assigned to a symbol. For a complete list and description of the types of symbol attributes, consult the online help feature. In Figure 3.7, the attribute to note is the VFILE attribute. This is the attribute that shows the path to the Verilog file that defines the W65C02C. After viewing the attributes, click Cancel to return to the symbol view, then close out of the symbol view to return to the W65C122S schematic.

Now that we have viewed the symbol attributes and saw that a Verilog file is associated with the symbol, we want to take a look at the Verilog file, W65C02C.v. To do this, select the W65C02C symbol as previously described and right-click to display the pop-out menu. Now, select Push Language →Verilog. This will open a window and display the Verilog source code for the W65C02C. Due to the proprietary nature of the source code, it will not be included in this document. For the three (3) remaining blocks, you can view the Verilog source and symbol attributes, if desired, by following the procedure described above.

A symbol can either have a Verilog file associated with it to describe its functionality or it can have a schematic to describe its functionality. The four main blocks of the W65C122S schematic have a Verilog file associated with them; however the Address Decoder block has a schematic associated with it. To view the schematic behind the Address Decoder symbol, select the symbol to highlight it, right click on it to open the pop-out menu, and select Push Schematic. Your screen will look like Figure 3.8.

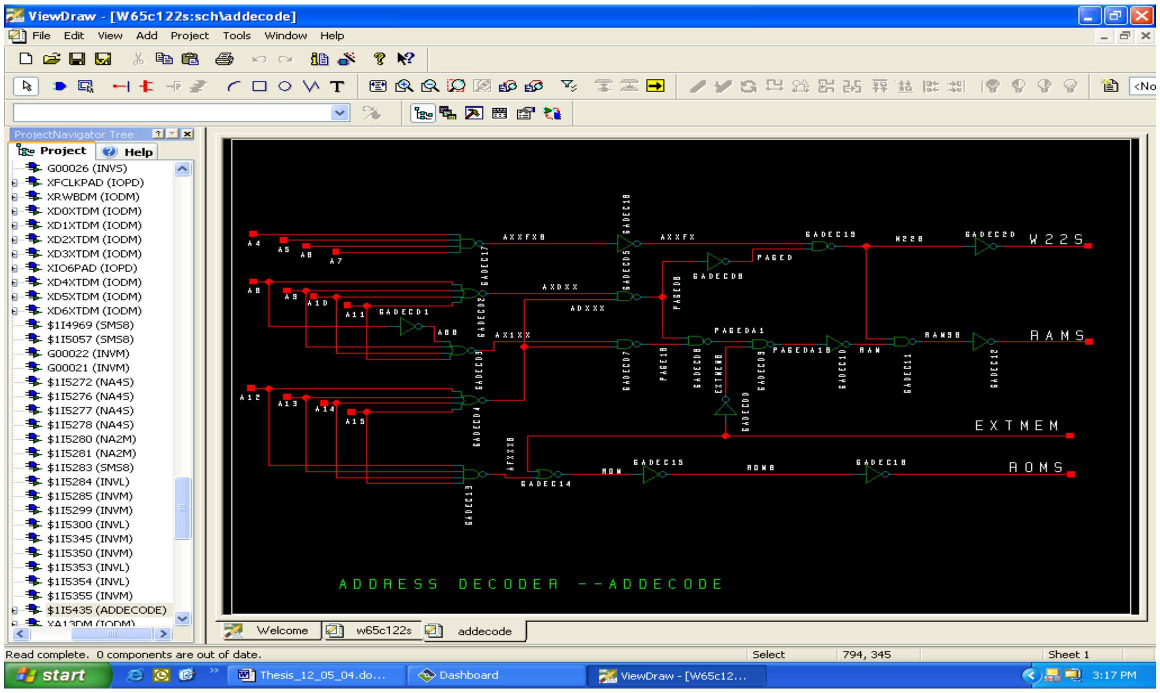


Figure 3.8 Schematic of the address decoder

Figure 3.8 shows the schematic that makes up the Address Decoder. The external signals A4 – A15 are the inputs and the signals W22S, RAMS, EXTMEM and ROMS are the outputs of the decoder. These external connections are the pins that are represented on the Address Decoder symbol which in turn are connected to the main W65C122S schematic. After examining the Address Decoder schematic, close out the window to return to the W65C122S schematic.

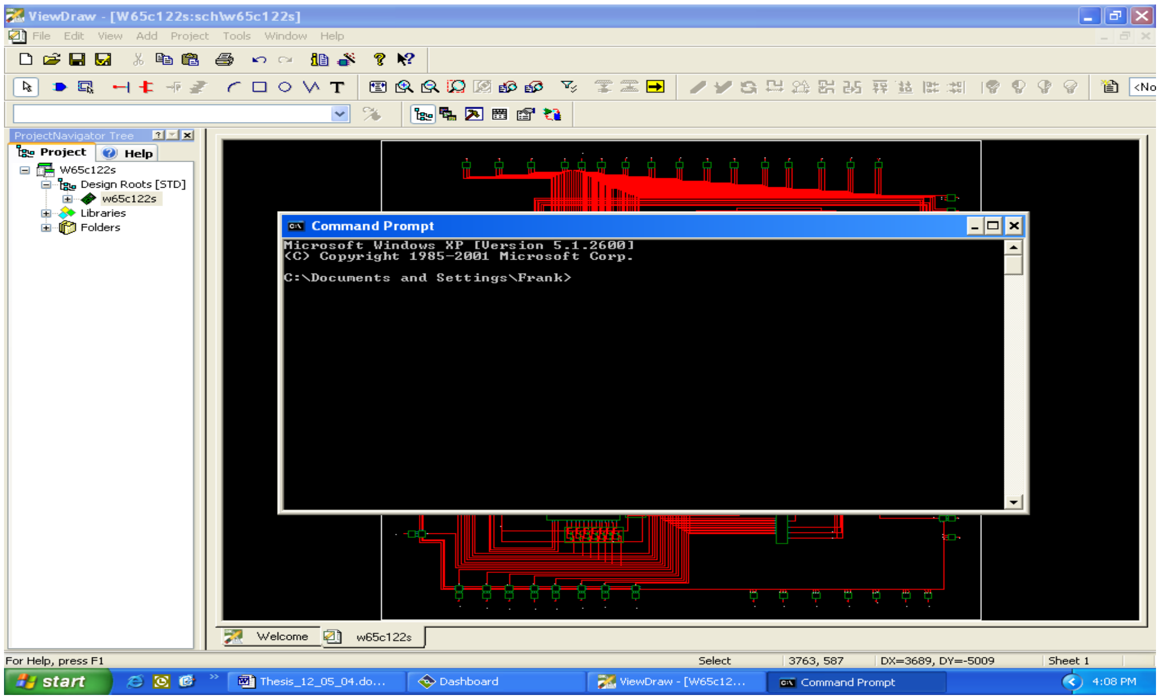
Lastly, around the edge of the schematic are the I/O pads for the signals that connect to the outside world. If you desire, you can view the properties of these pads by following the above procedures.

3.3.2 Extracting a Verilog Netlist of the W65C122S

After the schematic has been completed, it needs to be simulated in order to verify its functionality. In order to do this, a Verilog netlist must be extracted from the schematic, which in turn will be used as an input file to the Silos Verilog logic simulator.

WDC has created a 'gold standard' test bench which will be used to test the extracted netlist. The results of the simulation will then be compared to the 'gold standard' results. If the results match, then we know that there are no errors in the schematic. If errors are found, they need to be analyzed to determine where the error in the schematic is located.

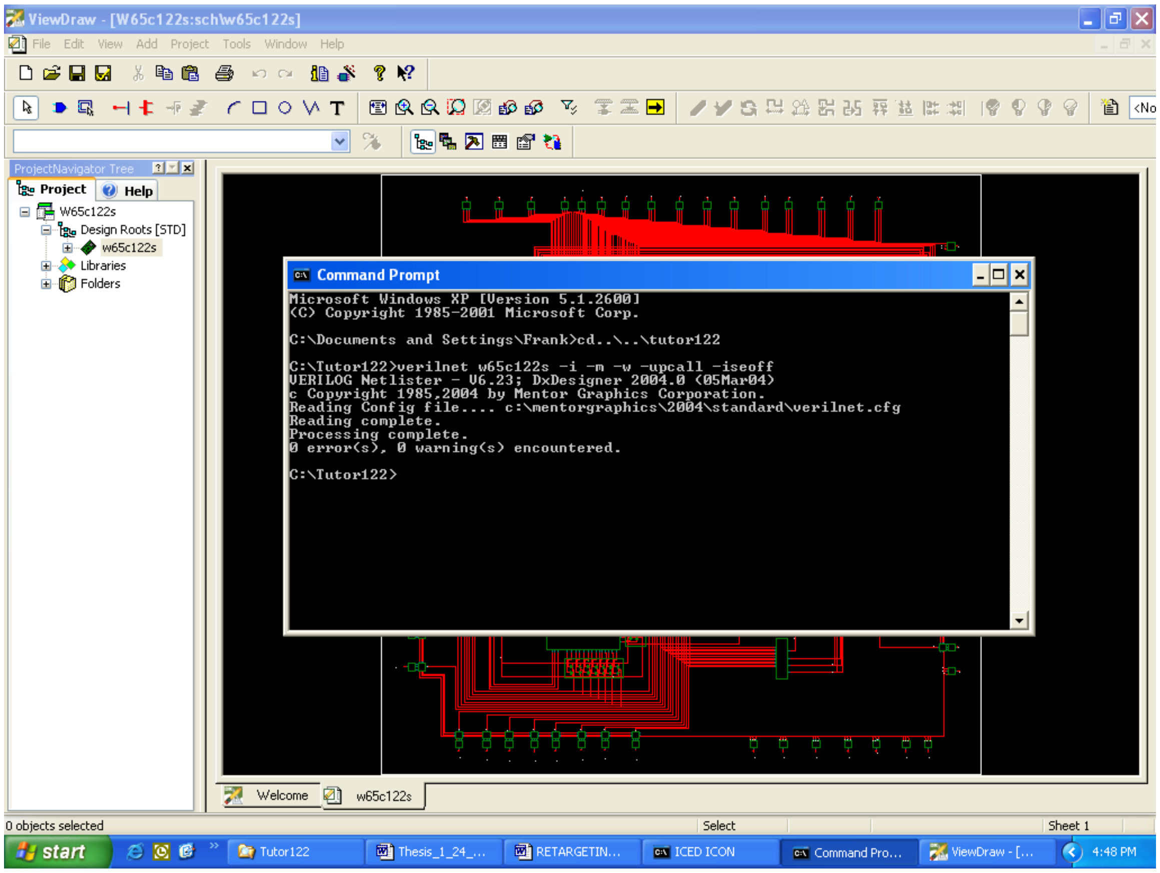
ViewDraw has a utility called Verilnet which we will be using to extract the netlist. This utility is run from a DOS command prompt. Open a DOS window by clicking Start → Programs → Accessories → Command Prompt. Your screen will look similar to Figure 3.9.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Frank>
```

Figure 3.9 DOS window for using Verilnet

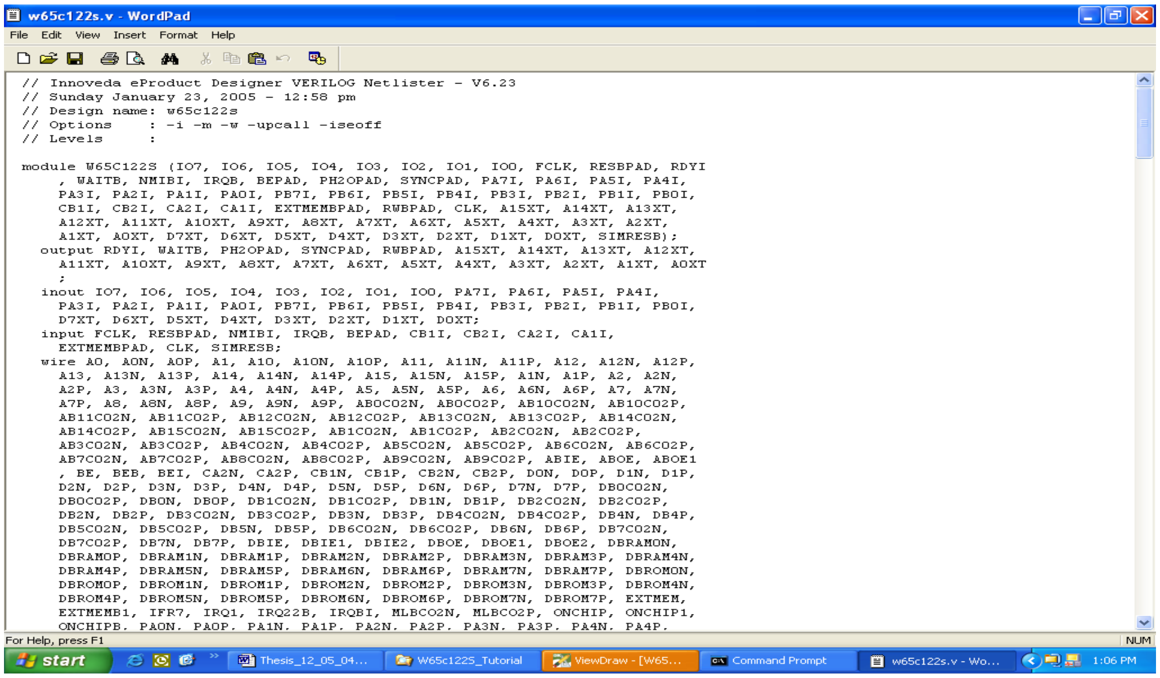
Change the directory to your working directory by typing **cd..\..\tutor122**. At the command prompt type the following line: **verilnet w65c122s -i -m -w -upcall –iseoff** and press Enter. This will run the Verilnet utility. There will be some brief activity as the utility reads in the program and executes. The process only takes a few seconds. When completed, your screen should look like Figure 3.10.



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Frank>cd..\..\tutor122

C:\Tutor122>verilnet w65c122s -i -m -w -upcall -iseoff
VERILOG Netlister - V6.23; DxDesigner 2004.0 (05Mar04)
c Copyright 1985,2004 by Mentor Graphics Corporation.
Reading Config file.... c:\mentorgraphics\2004\standard\verilnet.cfg
Reading complete.
Processing complete.
0 error(s), 0 warning(s) encountered.

C:\Tutor122>
```

Figure 3.10 Verilnet results window

The output file generated is w65c122s.v and is placed in your working directory. Browse to your working directory and open the w65c122s.v file to view it. Figure 3.11 shows a partial listing of the file.



```
// Innoveda eProduct Designer VERILOG Netlister - V6.23
// Sunday January 23, 2005 - 12:58 pm
// Design name: w65c122s
// Options     : -i -m -w -upcall -iseoff
// Levels      :

module W65C122S (IO7, IO6, IO5, IO4, IO3, IO2, IO1, IO0, FCLK, RESBPAD, RDYI
    , WAITB, NMIBI, IRQB, BEPAD, PH2OPAD, SYNCPAD, PA7I, PA6I, PA5I, PA4I,
    PA3I, PA2I, PA1I, PA0I, PB7I, PB6I, PB5I, PB4I, PB3I, PB2I, PB1I, PB0I,
    CB1I, CB2I, CA2I, CA1I, EXTMEMBPAD, RWBPAD, CLK, A15XT, A14XT, A13XT,
    A12XT, A11XT, A10XT, A9XT, A8XT, A7XT, A6XT, A5XT, A4XT, A3XT, A2XT,
    A1XT, A0XT, D7XT, D6XT, D5XT, D4XT, D3XT, D2XT, D1XT, D0XT, SIMRESB);
  output RDYI, WAITB, PH2OPAD, SYNCPAD, RWBPAD, A15XT, A14XT, A13XT, A12XT,
    A11XT, A10XT, A9XT, A8XT, A7XT, A6XT, A5XT, A4XT, A3XT, A2XT, A1XT, A0XT
    ;
  inout IO7, IO6, IO5, IO4, IO3, IO2, IO1, IO0, PA7I, PA6I, PA5I, PA4I,
    PA3I, PA2I, PA1I, PA0I, PB7I, PB6I, PB5I, PB4I, PB3I, PB2I, PB1I, PB0I,
    D7XT, D6XT, D5XT, D4XT, D3XT, D2XT, D1XT, D0XT;
  input FCLK, RESBPAD, NMIBI, IRQB, BEPAD, CB1I, CB2I, CA2I, CA1I,
    EXTMEMBPAD, CLK, SIMRESB;
  wire A0, A0N, A0P, A1, A10, A10N, A10P, A11, A11N, A11P, A12, A12N, A12P,
    A13, A13N, A13P, A14, A14N, A14P, A15, A15N, A15P, A1N, A1P, A2, A2N,
    A2P, A3, A3N, A3P, A4, A4N, A4P, A5, A5N, A5P, A6, A6N, A6P, A7, A7N,
    A7P, A8, A8N, A8P, A9, A9N, A9P, AB0CO2N, AB0CO2P, AB10CO2N, AB10CO2P,
    AB11CO2N, AB11CO2P, AB12CO2N, AB12CO2P, AB13CO2N, AB13CO2P, AB14CO2N,
    AB14CO2P, AB15CO2N, AB15CO2P, AB1CO2N, AB1CO2P, AB2CO2N, AB2CO2P,
    AB3CO2N, AB3CO2P, AB4CO2N, AB4CO2P, AB5CO2N, AB5CO2P, AB6CO2N, AB6CO2P,
    AB7CO2N, AB7CO2P, AB8CO2N, AB8CO2P, AB9CO2N, AB9CO2P, ABIE, ABOE, ABOE1
    , BE, BEB, BEI, CA2N, CA2P, CB1N, CB1P, CB2N, CB2P, D0N, D0P, D1N, D1P,
    D2N, D2P, D3N, D3P, D4N, D4P, D5N, D5P, D6N, D6P, D7N, D7P, DB0CO2N,
    DB0CO2P, DB0N, DB0P, DB1CO2N, DB1CO2P, DB1N, DB1P, DB2CO2N, DB2CO2P,
    DB2N, DB2P, DB3CO2N, DB3CO2P, DB3N, DB3P, DB4CO2N, DB4CO2P, DB4N, DB4P,
    DB5CO2N, DB5CO2P, DB5N, DB5P, DB6CO2N, DB6CO2P, DB6N, DB6P, DB7CO2N,
    DB7CO2P, DB7N, DB7P, DBIE, DBIE1, DBIE2, DBOE, DBOE1, DBOE2, DBRAM0N,
    DBRAM0P, DBRAM1N, DBRAM1P, DBRAM2N, DBRAM2P, DBRAM3N, DBRAM3P, DBRAM4N,
    DBRAM4P, DBRAM5N, DBRAM5P, DBRAM6N, DBRAM6P, DBRAM7N, DBRAM7P, DBROM0N,
    DBROM0P, DBROM1N, DBROM1P, DBROM2N, DBROM2P, DBROM3N, DBROM3P, DBROM4N,
    DBROM4P, DBROM5N, DBROM5P, DBROM6N, DBROM6P, DBROM7N, DBROM7P, EXTMEM,
    EXTMEMB1, IFR7, IRQ1, IRQ22B, IRQBI, MLBCO2N, MLBCO2P, ONCHIP, ONCHIP1,
    ONCHIPB, PA0N, PA0P, PA1N, PA1P, PA2N, PA2P, PA3N, PA3P, PA4N, PA4P,
```

Figure 3.11 Extracted Verilog netlist of the W65C122S schematic

The netlist shows the module instantiation, the input and output signals, internal wire names, instantiations of the pads, cores and random logic gates.

We must also extract the Verilog netlist for the Address Decoder. Remember that the Address Decoder block has a schematic representation, not a Verilog representation. Thus, in order to simulate the complete schematic, a Verilog description of the Address Decoder must also be included. As before, at the command prompt type:

verilnet addecode –i –m –w –upcall –iseoff and press Enter. The output file addecode.v will be placed in your working directory. After extraction completes, close the DOS window.

Now that we have successfully extracted a Verilog netlist of the W65C122S and Address Decoder schematics, there is nothing more that needs to be done in ViewDraw and you can close out this program. The netlists that we have extracted will now be used to simulate the design in Silos.

3.4 Simulating the W65C122S using Silos

In Section 3.3.2, we extracted the Verilog netlist of the W65C122S schematic. We now must simulate this netlist and compare it to the 'gold standard' simulation results to verify its functionality. The logic simulation will be done using Silos and the test bench that was created by WDC.

3.4.1 Working with the W65C122S Silos simulation directory

The Silos project for simulating the W65C122S is located at C:\Tutor122\Verilog\Chip\Viewdraw. Open Silos by double-clicking on the Silos icon on the desktop. Your screen will look like Figure 3.12.

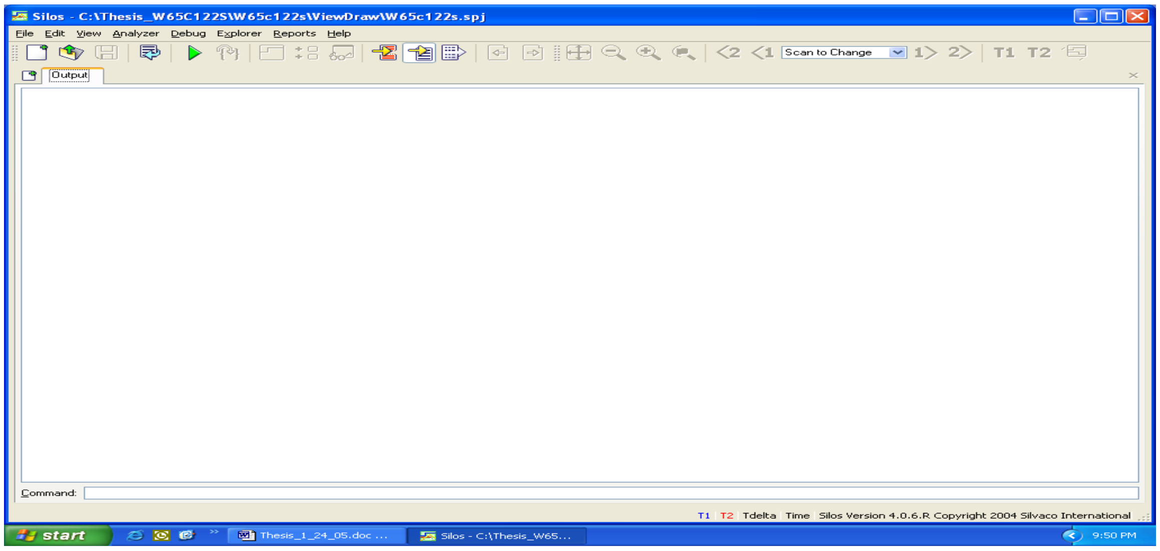
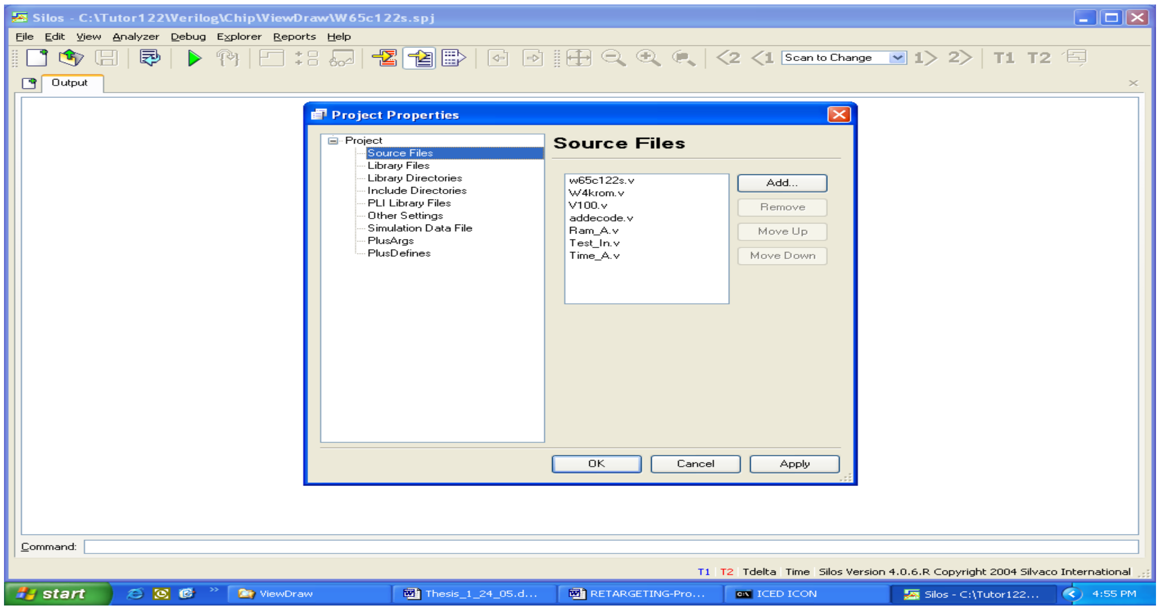

Silos - C:\Thesis_W65C122S\W65c122s\ViewDraw\W65c122s.spj
File Edit View Analyzer Debug Explorer Reports Help
Scan to Change
T1 T2
Output
Command:
T1 T2 Tdelta Time Silos Version 4.0.6.R Copyright 2004 Silvaco International
start
Thesis_1_24_05.doc ...
Silos - C:\Thesis_W65...
9:50 PM

Figure 3.12 Silos main window

Next, open the existing project by clicking File → Open Project. Then browse to C:\Tutor122\Verilog\Chip\ViewDraw and double-click on W65C122S.spj. To begin, click on Edit → Project Properties to view the source and library files that are associated with the simulation. Your screen will look like Figure 3.13.



```
w65c122s.v
W4krom.v
V100.v
addecode.v
Ram_A.v
Test_In.v
Time_A.v
```

Figure 3.13 Silos project properties dialog box for the W65C122S simulation

The source files needed for simulation are:

W65c122s.v – netlist file extracted form ViewDraw schematic

W4krom.v and v100.v – Verilog files describing the ROM data

Addecode.v – netlist of the Address Decoder, extracted from ViewDraw

Ram_a.v – Verilog file containing the RAM data

Test_In.v – Test bench file for testing the internal memory

Time_A.v – Verilog file containing the simulation time

Next, click Library Files to view the library files needed for simulation. The required files are:

W65c02c.enc – Encrypted Verilog file describing the core of the W65c02

W65c22c.enc – Encrypted Verilog file describing the core of the W65c22

W256Ram.v – Verilog file describing the RAM module

Macro.enc – Encrypted Verilog file of the macros required for netlist simulation (description of the transistors that comprise the gates)

StdCell.v – Verilog file describing WDC’s standard cells

StdModul.v – Verilog file describing WDC’s standard modules

Click Cancel to close out of the Project Properties Window.

Also located in C:\Tutor122\Verilog\Chip\ViewDraw are files that are generated by Silos and reference files. The file Test_In.dat contains the results of the simulation. The file Test_In_Gold.dat is the ‘gold standard’ simulation result file that any new simulation is compared to. Test_In.dat is generated by Silos after every simulation. If the file already exists, it will overwrite and replace the original file. Also included in this directory are files called W65c122s_orig.v and Addecode_orig.v. These files are known, good netlists of the W65C122S and Address Decoder schematics and can be used for reference purposes. There is a Readme.txt file that contains some more information about the files in this directory.

3.4.2 Setting up the simulation using the extracted schematic netlist

Prior to running the simulation, copy W65c122s.v and Addecode.v from C:\Tutor122 to C:\Tutor122\Verilog\Chip\ViewDraw. These are the schematic netlists extracted in Section 3.3.2. After copying these files, return to the Silos project window that was opened previously. Your screen will look like Figure 3.14.

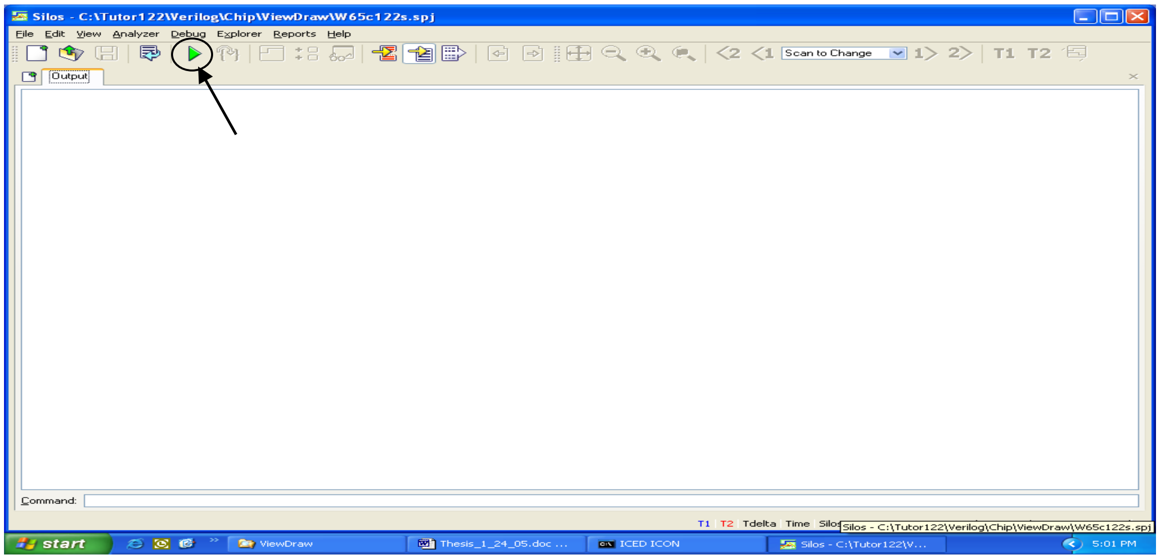
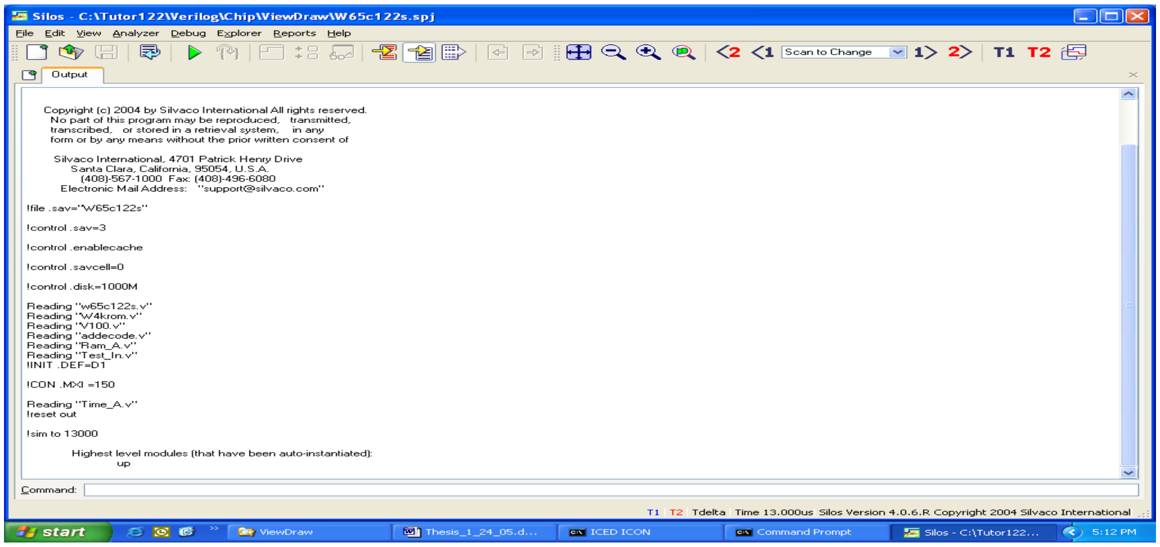

Silos - C:\Tutor122\Verilog\Chip\ViewDraw\W65c122s.spj
File Edit View Analyzer Debug Explorer Reports Help
Scan to Change
Output
Command:
T1 T2 Tdelta Time
Silos - C:\Tutor122\Verilog\Chip\ViewDraw\W65c122s.spj
start
ViewDraw
Thesis_1_24_05.doc ...
ICED ICON
Silos - C:\Tutor122\V...
5:01 PM

Figure 3.14 Silos project window

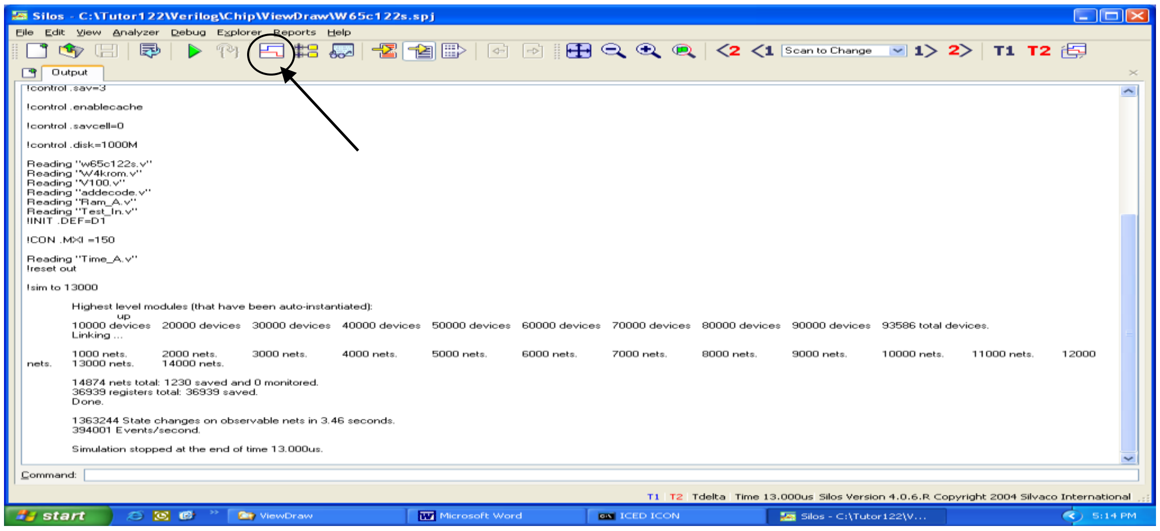
To run the simulation, click on the green arrow icon, illustrated in Figure 3.14. Once the simulation starts, you will see some activity in the window. The files will be read in and checked for errors. If errors are found, the simulation will stop and the errors need to be investigated. Figure 3.15 shows the simulation when it is running. The simulation only takes a few moments to complete.


Silos - C:\Tutor122\Verilog\Chip\ViewDraw\W65c122s.spj
File Edit View Analyzer Debug Explorer Reports Help
Scan to Change
Output
Command:
T1 T2 Tdelta Time 13.000us Silos Version 4.0.6.R Copyright 2004 Silvaco International
start
ViewDraw
Thesis_1_24_05.d...
ICED ICON
Command Prompt
Silos - C:\Tutor122...
5:12 PM

```
   Copyright (c) 2004 by Silvaco International All rights reserved.
     No part of this program may be reproduced,   transmitted,
     transcribed,   or stored in a retrieval system,    in any
     form or by any means without the prior written consent of

      Silvaco International, 4701 Patrick Henry Drive
          Santa Clara, California, 95054, U.S.A.
            (408)-567-1000  Fax: (408)-496-6080
        Electronic Mail Address:   "support@silvaco.com"

!file .sav="W65c122s"

!control .sav=3

!control .enablecache

!control .savcell=0

!control .disk=1000M

Reading "w65c122s.v"
Reading "W4krom.v"
Reading "V100.v"
Reading "addecode.v"
Reading "Ram_A.v"
Reading "Test_In.v"
!INIT .DEF=D1

!CON .MXI =150

Reading "Time_A.v"
!reset out

!sim to 13000

         Highest level modules (that have been auto-instantiated):
                  up
```

Figure 3.15 Screenshot of simulation in progress

After the simulation completes, your screen should look like Figure 3.16.

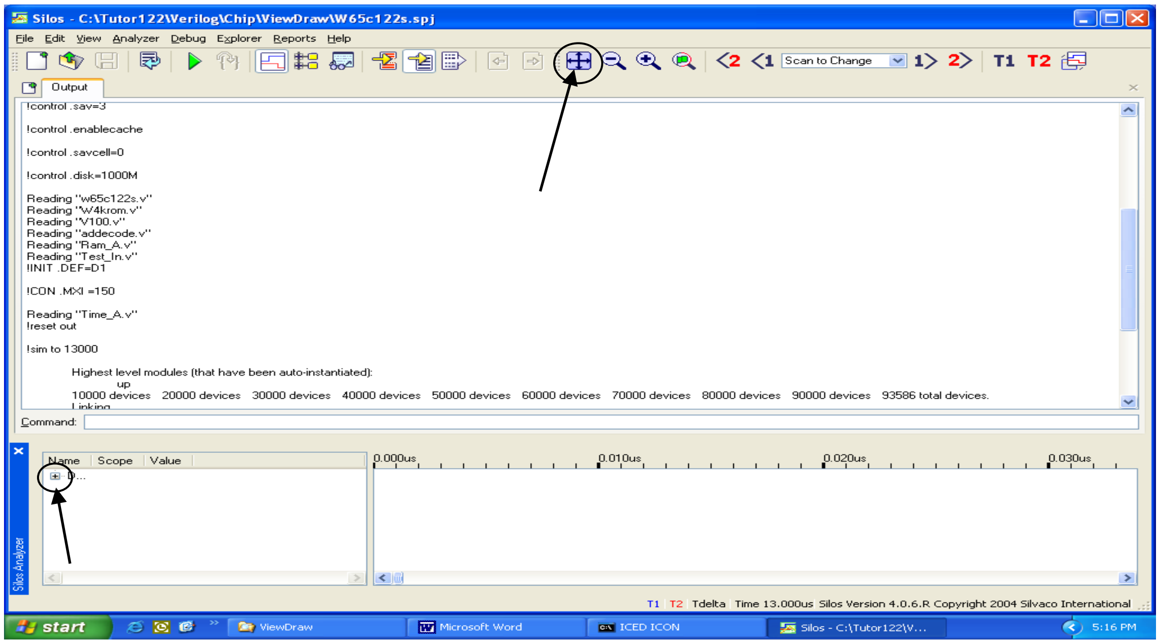


```
!control .sav=3

!control .enablecache

!control .savcell=0

!control .disk=1000M

Reading "w65c122s.v"
Reading "W4krom.v"
Reading "V100.v"
Reading "addecode.v"
Reading "Ram_A.v"
Reading "Test_In.v"
!INIT .DEF=D1

!CON .MXI =150

Reading "Time_A.v"
!reset out

!sim to 13000

         Highest level modules (that have been auto-instantiated):
                  up
         10000 devices   20000 devices   30000 devices   40000 devices   50000 devices   60000 devices   70000 devices   80000 devices   90000 devices   93586 total devices.
         Linking ...

         1000 nets.      2000 nets.      3000 nets.      4000 nets.      5000 nets.      6000 nets.      7000 nets.      8000 nets.      9000 nets.      10000 nets.     11000 nets.     12000
nets.    13000 nets.     14000 nets.

         14874 nets total: 1230 saved and 0 monitored.
         36939 registers total: 36939 saved.
         Done.

         1363244 State changes on observable nets in 3.46 seconds.
         394001 Events/second.

         Simulation stopped at the end of time 13.000us.
```

```
T1  T2  Tdelta  Time 13.000us  Silos Version 4.0.6.R Copyright 2004 Silvaco International
```

Figure 3.16 Screenshot of completed simulation

Next, we want to take a look at the waveforms generated by the simulation. To do this, click on the Analyzer icon, as shown in Figure 3.16. After doing so, your screen will look like Figure 3.17.



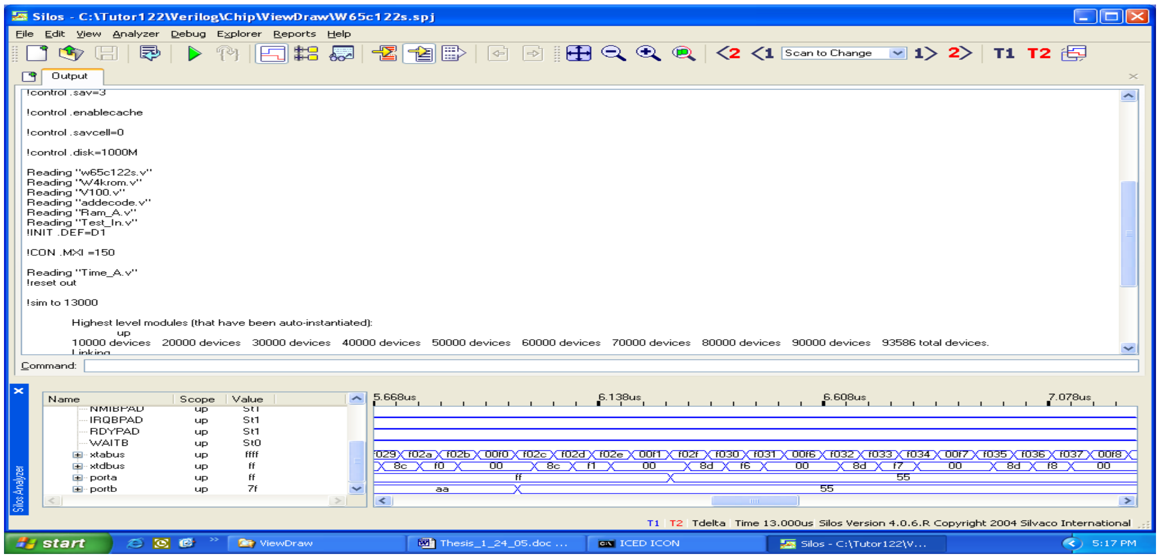
```
!control .sav=3

!control .enablecache

!control .savcell=0

!control .disk=1000M

Reading "w65c122s.v"
Reading "W4krom.v"
Reading "V100.v"
Reading "addecode.v"
Reading "Ram_A.v"
Reading "Test_In.v"
!INIT .DEF=D1

!CON .MXI =150

Reading "Time_A.v"
!reset out

!sim to 13000

         Highest level modules (that have been auto-instantiated):
                  up
         10000 devices   20000 devices   30000 devices   40000 devices   50000 devices   60000 devices   70000 devices   80000 devices   90000 devices   93586 total devices.
```

```
T1  T2  Tdelta  Time 13.000us  Silos Version 4.0.6.R Copyright 2004 Silvaco International
```

Figure 3.17 Silos with data analyzer window

In the left hand side of the Analyzer window, click the plus (+) sign, as shown in Figure 3.17 to expand the signal list. You may have to resize the Name field to see all of the signal names. Once expanded, you will see waveforms on the right hand side of the Analyzer window.

Depending on the zoom factor, there may not be much of the waveforms to see initially. Click on the Zoom Full icon as shown in Figure 3.17 to bring the entire simulation into the viewing window. Click on the Zoom In icon (magnifying glass with the plus (+) sign), three times to magnify the signals for better visibility. Your screen will look similar to Figure 3.18.



```
!control .sav=3

!control .enablecache

!control .savcell=0

!control .disk=1000M

Reading "w65c122s.v"
Reading "W4krom.v"
Reading "V100.v"
Reading "addecode.v"
Reading "Ram_A.v"
Reading "Test_In.v"
!!INIT .DEF=D1

!CON .MXI =150

Reading "Time_A.v"
!reset out

!sim to 13000

        Highest level modules (that have been auto-instantiated):
                up
        10000 devices   20000 devices   30000 devices   40000 devices   50000 devices   60000 devices   70000 devices   80000 devices   90000 devices   93586 total devices.
        Linking
```

```
Name          Scope   Value
   NMIBPAD    up      St1
   IRQBPAD    up      St1
   RDYPAD     up      St1
   WAITB      up      St0
 ⊞ xtabus     up      ffff
 ⊞ xtdbus     up      ff
 ⊞ porta      up      ff
 ⊞ portb      up      7f
```

```
T1  T2  Tdelta  Time 13.000us  Silos Version 4.0.6.R Copyright 2004 Silvaco International
```

Figure 3.18 Simulation waveform view

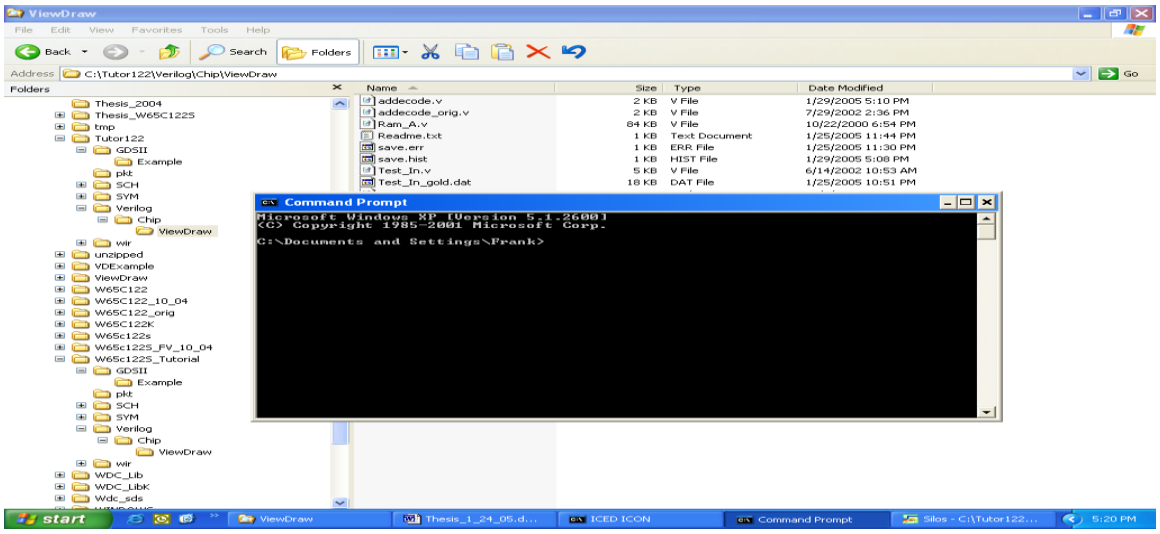
Figure 3.18 shows the signals in the bottom half of the signal list. To view the signals in the top half of the signal list, move the scroll bar in the signal listing window up. We will not be analyzing the signals here, but the signals represented show the contents of the address and data bus as well as some signal pins with respect to the clock. After viewing the waveforms, close the project by clicking File → Close Project.

3.4.3 Comparing the simulation results to the 'gold standard' results

The final step is to compare the results of the simulation using the extracted netlists to the 'gold standard' simulation results. To do this, we will be using the File Compare utility within DOS.

After the simulation completed, the file Test_In.dat was generated by Silos and placed in C:\Tutor122\Verilog\Chip\ViewDraw. This has the results of the simulation in the format specified in the Test_In.v file. To view the Test_In.dat file, open it using WordPad. The file contains the names of the signals and their state at different intervals of time. Close the file after viewing it.

Next, open a DOS window by clicking Start → Programs → Accessories → Command Prompt. Your screen will look similar to Figure 3.19.

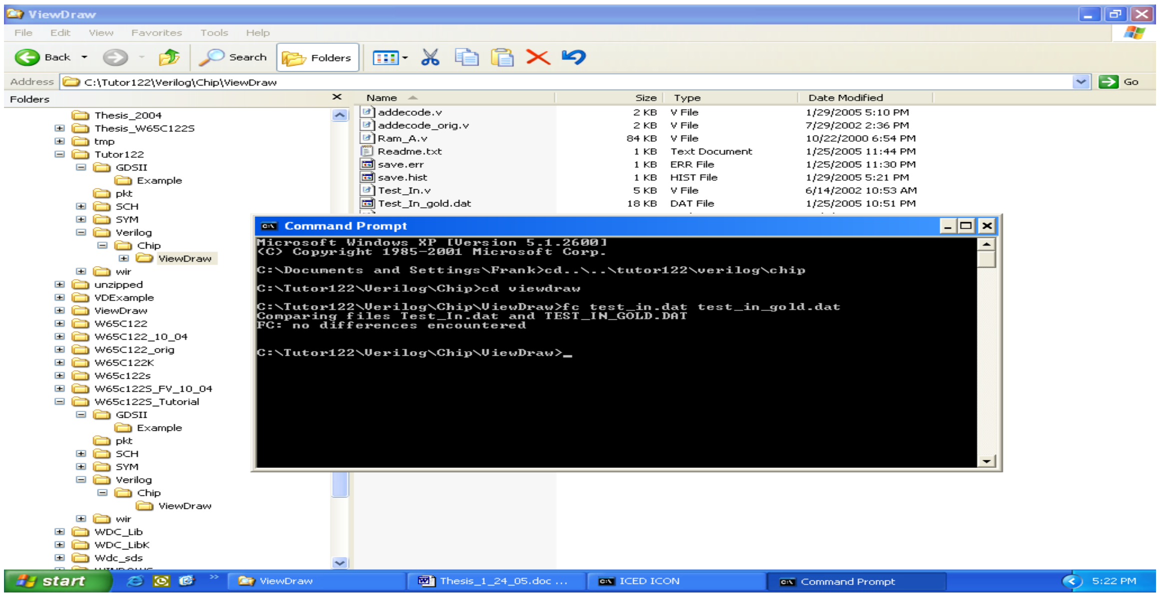


Name	Size	Type	Date Modified
addecode.v	2 KB	V File	1/29/2005 5:10 PM
addecode_orig.v	2 KB	V File	7/29/2002 2:36 PM
Ram_A.v	84 KB	V File	10/22/2000 6:54 PM
Readme.txt	1 KB	Text Document	1/25/2005 11:44 PM
save.err	1 KB	ERR File	1/25/2005 11:30 PM
save.hist	1 KB	HIST File	1/29/2005 5:08 PM
Test_In.v	5 KB	V File	6/14/2002 10:53 AM
Test_In_gold.dat	18 KB	DAT File	1/25/2005 10:51 PM

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Frank>
```

Figure 3.19 DOS window

Change the current directory to C:\Tutor122\Verilog\Chip\ViewDraw. At the command prompt type: **fc test_in.dat test_in_gold.dat** and press Enter. This will execute the file compare utility. The comparison will only take a few seconds. When complete, your screen will look like Figure 3.20.



Name	Size	Type	Date Modified
addecode.v	2 KB	V File	1/29/2005 5:10 PM
addecode_orig.v	2 KB	V File	7/29/2002 2:36 PM
Ram_A.v	84 KB	V File	10/22/2000 6:54 PM
Readme.txt	1 KB	Text Document	1/25/2005 11:44 PM
save.err	1 KB	ERR File	1/25/2005 11:30 PM
save.hist	1 KB	HIST File	1/29/2005 5:21 PM
Test_In.v	5 KB	V File	6/14/2002 10:53 AM
Test_In_gold.dat	18 KB	DAT File	1/25/2005 10:51 PM

Command Prompt

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Frank>cd..\..\tutor122\verilog\chip

C:\Tutor122\Verilog\Chip>cd viewdraw

C:\Tutor122\Verilog\Chip\ViewDraw>fc test_in.dat test_in_gold.dat
Comparing files Test_In.dat and TEST_IN_GOLD.DAT
FC: no differences encountered


C:\Tutor122\Verilog\Chip\ViewDraw>_
```

Figure 3.20 Results of file comparison

The results show that no differences are found between the two files, indicating that the extracted netlist of the W65C122S has no errors and that the design is now ready to be laid out using an IC layout editor. Close out the DOS window.

3.5 Working with the W65C122S using ICED Layout Editor

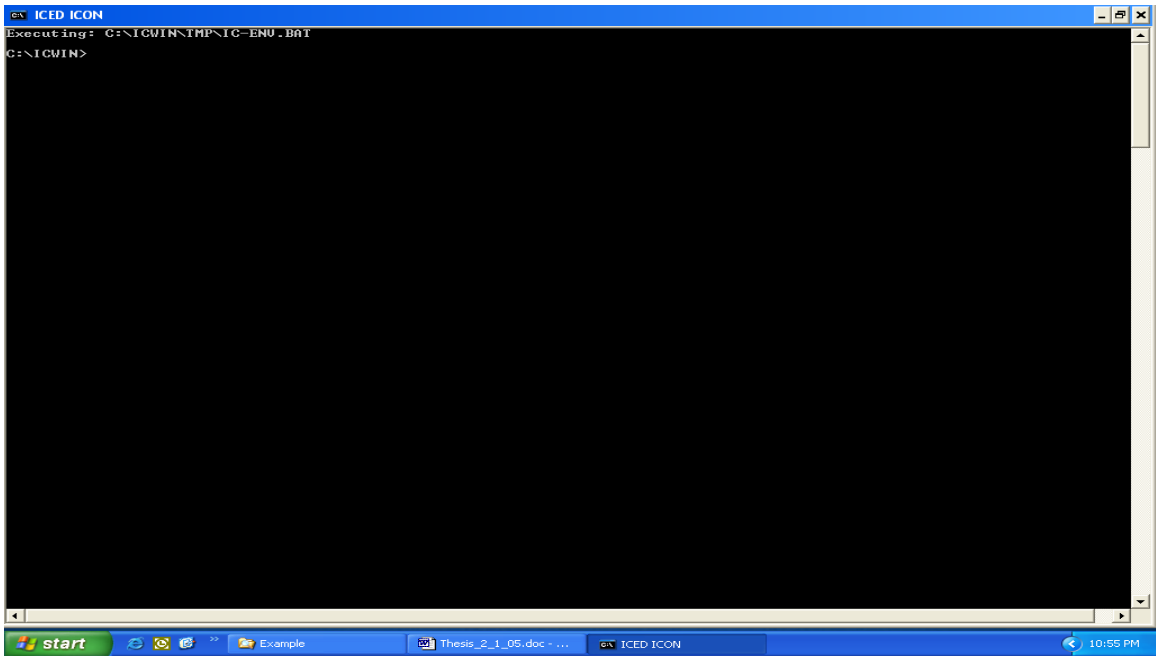
After the design has passed the logic simulation phase, the next step in the design process is to layout the design as it is going to appear on final silicon. There are two methods of laying out a circuit: a) automatic and b) manual place and route. Automatic place and route tools are very costly but take less time to complete the layout. Manually placing and routing the design is more time consuming, but the tools are less expensive.

There are advantages and disadvantages to both. For example, manually placing and routing the design gives more control to the designer as to where the cells are placed and wires are routed, thus resulting in an optimized core. Automatic tools rely on control files to place and route the cells, and though quicker, may not place cells and route connections in the most optimal way. WDC uses the manual place and route method to layout a circuit and uses ICED as the layout editor.

3.5.1 Viewing the W65C122S layout in ICED

The file containing the layout of the W65C122S is located in C:\Tutor122\GDSII\Example and it is called CHIP.SF. The SF stands for Stream File and is a compressed file that contains all of the cells that comprise the W65C122S. To open the file and view the layout, the .SF file must be unstreamed using ICED.

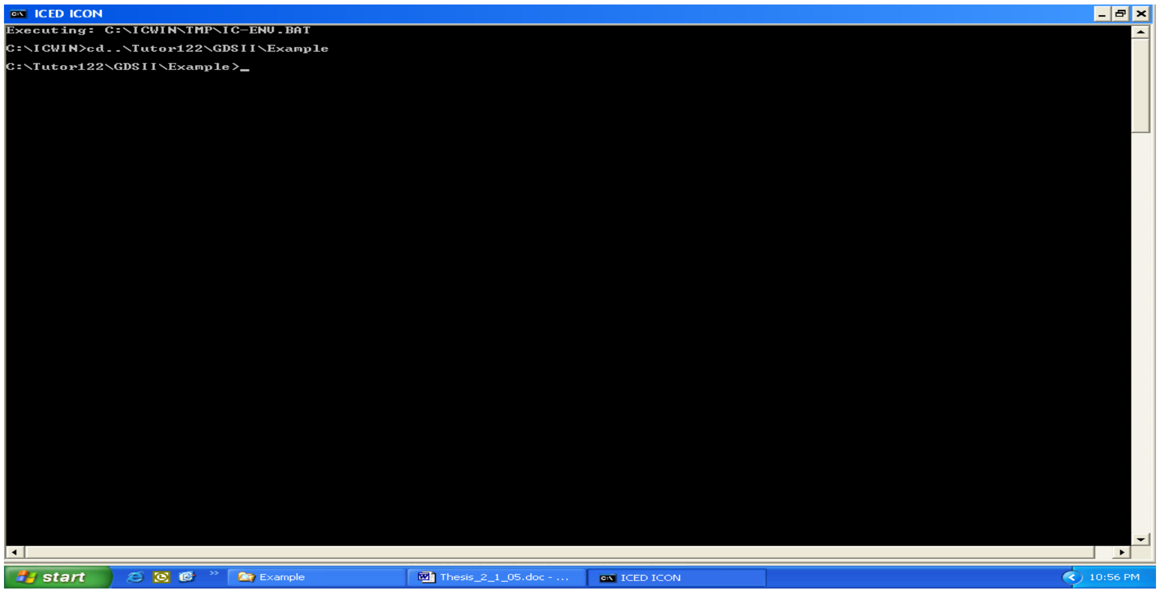
To unstream the CHIP.SF file, first open ICED by double-clicking the ICED icon on the desktop. Your screen will look like Figure 3.21.



```
Executing: C:\ICWIN\TMP\IC-ENV.BAT
C:\ICWIN>
```

Figure 3.21 ICED DOS window

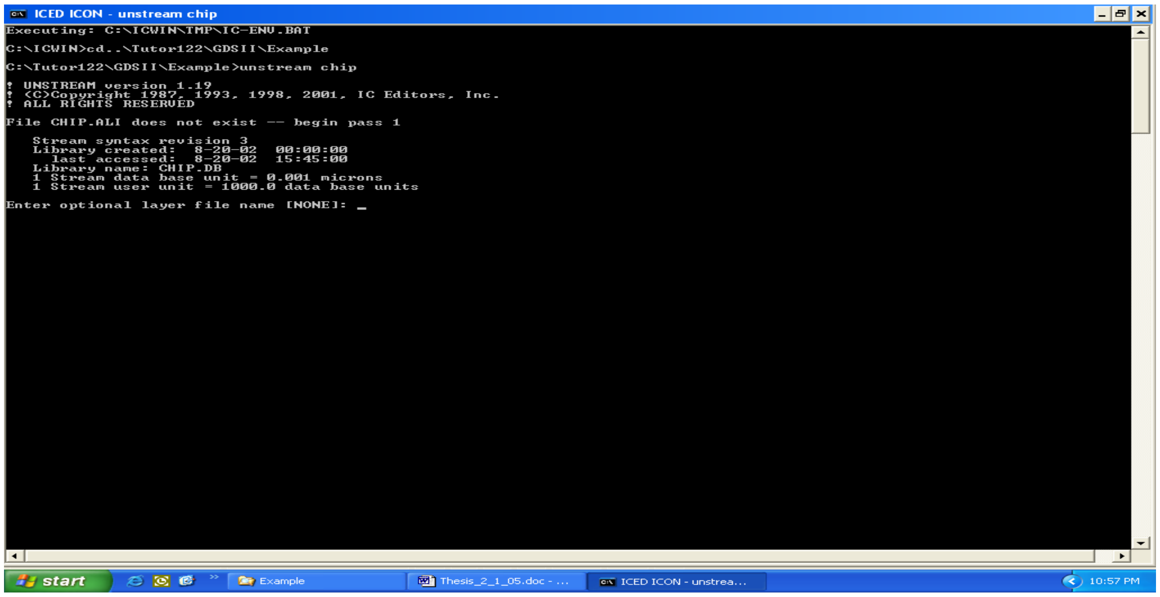
Next, change the directory to the working directory by typing **cd..\tutor122\gdsii\example**. Your screen will look like Figure 3.22.



```
Executing: C:\ICWIN\TMP\IC-ENV.BAT
C:\ICWIN>cd..\Tutor122\GDSII\Example
C:\Tutor122\GDSII\Example>_
```

Figure 3.22 ICED window with working directory

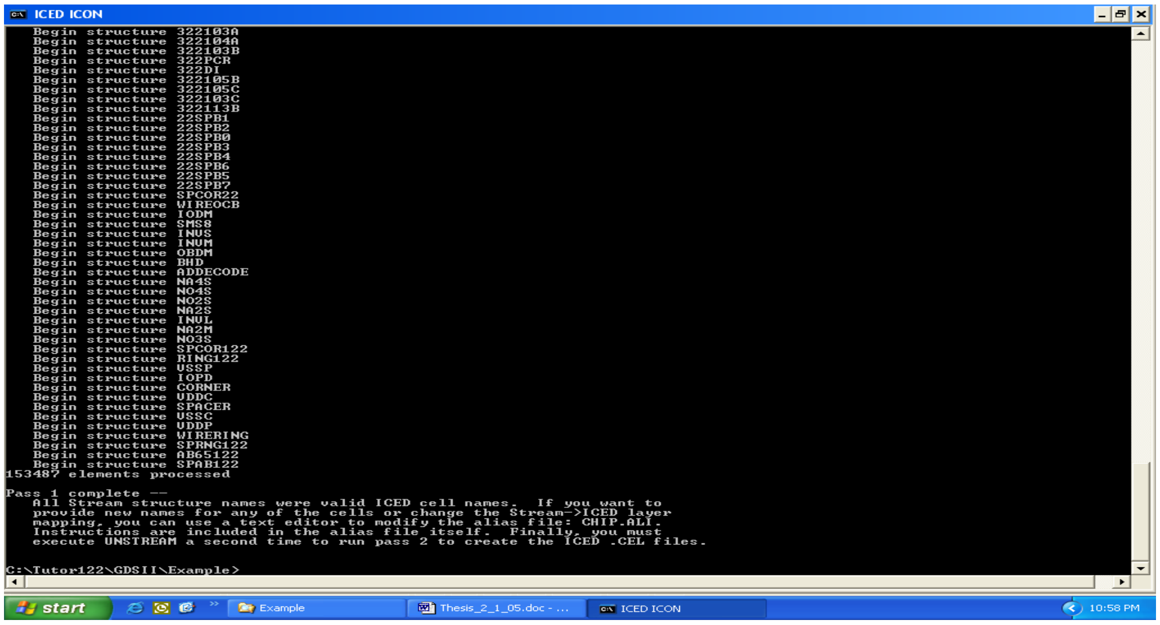
At the command prompt, type: **unstream chip**. Your screen will look like Figure 3.23.



```
Executing: C:\ICWIN\TMP\IC-ENV.BAT
C:\ICWIN>cd..\Tutor122\GDSII\Example
C:\Tutor122\GDSII\Example>unstream chip
! UNSTREAM version 1.19
! <C>Copyright 1987, 1993, 1998, 2001, IC Editors, Inc.
! ALL RIGHTS RESERVED
File CHIP.ALI does not exist -- begin pass 1
   Stream syntax revision 3
   Library created:  8-20-02  00:00:00
     last accessed:  8-20-02  15:45:00
   Library name: CHIP.DB
   1 Stream data base unit = 0.001 microns
   1 Stream user unit = 1000.0 data base units
Enter optional layer file name [NONE]: _
```

Figure 3.23 ICED unstream window

The unstream process takes two passes to complete. Figure 3.23 shows the start of Pass 1. There are three prompts during Pass 1. At each prompt, hit the Enter key as no information is needed for Pass 1. After hitting Enter at the third prompt, you will see some activity in the window as the individual cells are removed from the .SF file. Once the activity stops, your screen will look like Figure 3.24.



```
    Begin structure 322103A
    Begin structure 322104A
    Begin structure 322103B
    Begin structure 322PCR
    Begin structure 322DI
    Begin structure 322105B
    Begin structure 322105C
    Begin structure 322103C
    Begin structure 322113B
    Begin structure 22SPB1
    Begin structure 22SPB2
    Begin structure 22SPB0
    Begin structure 22SPB3
    Begin structure 22SPB4
    Begin structure 22SPB6
    Begin structure 22SPB5
    Begin structure 22SPB7
    Begin structure SPCOR22
    Begin structure WIREOCB
    Begin structure IODM
    Begin structure SMS8
    Begin structure INVS
    Begin structure INVM
    Begin structure OBDM
    Begin structure BHD
    Begin structure ADDECODE
    Begin structure NA4S
    Begin structure NO4S
    Begin structure NO2S
    Begin structure NA2S
    Begin structure INVL
    Begin structure NA2M
    Begin structure NO3S
    Begin structure SPCOR122
    Begin structure RING122
    Begin structure VSSP
    Begin structure IOPD
    Begin structure CORNER
    Begin structure VDDC
    Begin structure SPACER
    Begin structure VSSC
    Begin structure VDDP
    Begin structure WIRERING
    Begin structure SPRNG122
    Begin structure AB65122
    Begin structure SPAB122
153487 elements processed

Pass 1 complete --
    All Stream structure names were valid ICED cell names.  If you want to
    provide new names for any of the cells or change the Stream->ICED layer
    mapping, you can use a text editor to modify the alias file: CHIP.ALI.
    Instructions are included in the alias file itself.  Finally, you must
    execute UNSTREAM a second time to run pass 2 to create the ICED .CEL files.

C:\Tutor122\GDSII\Example>
```

Figure 3.24 DOS window at the completion of pass 1

Next, once again type: **unstream chip** at the command prompt and press Enter. This will start Pass 2 of the unstream process. There are nine prompts to Pass 2. Two of which need information to be provided. The first five prompts are shown below. Only prompts 3 and 5 require the indicated input shown in **BOLD**. Hit Enter at all other prompts. Text under the prompt is information returned by ICED.

Enter ICED user unit size in microns [1.0]:

Enter divisions per unit [1000]:

Are you using UNSTREAM to scale this design or snap it to grid (Y or [N])? **N**

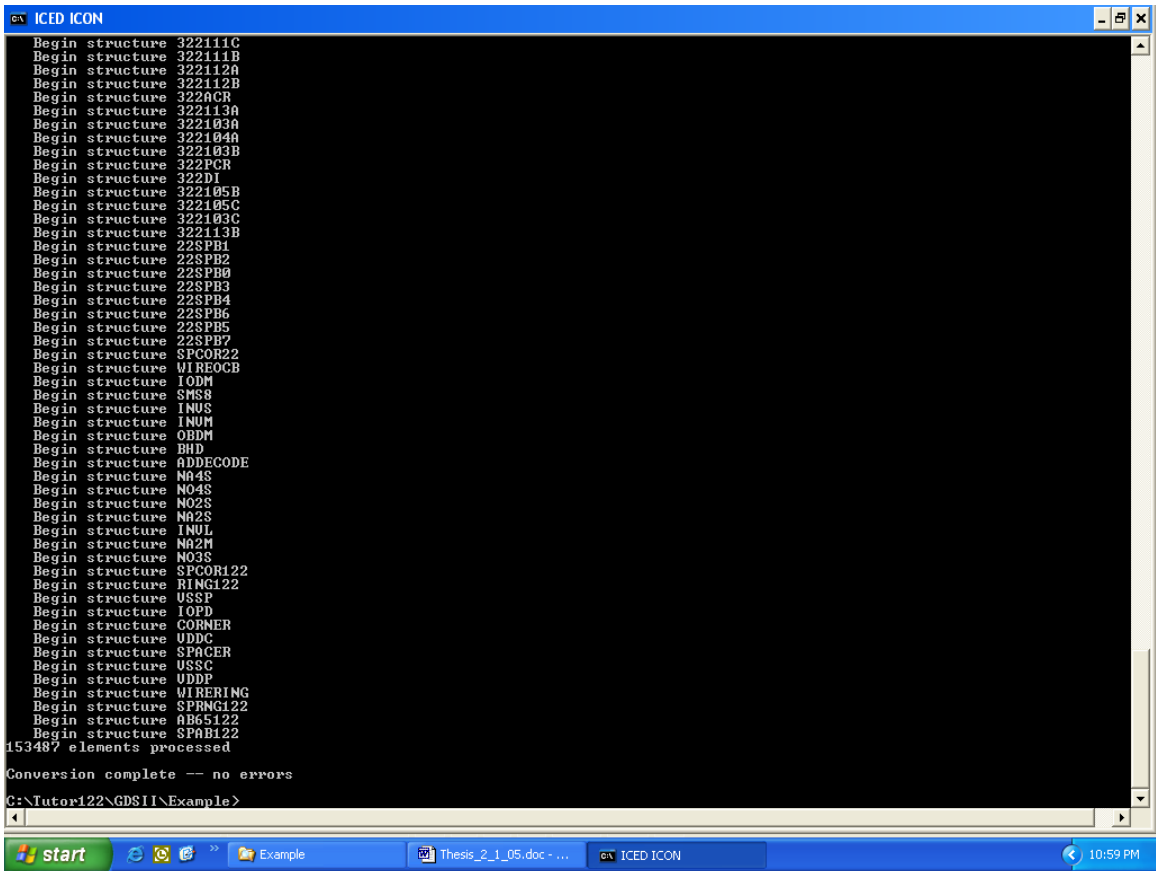
  1 ICED division = 1 Stream data base unit

Enter maximum allowable coordinate rounding error in microns [0.0]:

  Tolerance = 0.0 microns = 0.0 ICED divisions

Font[0] is "GDSII:CALMAFONT.TX" -- Enter character height in microns [1]: **4**

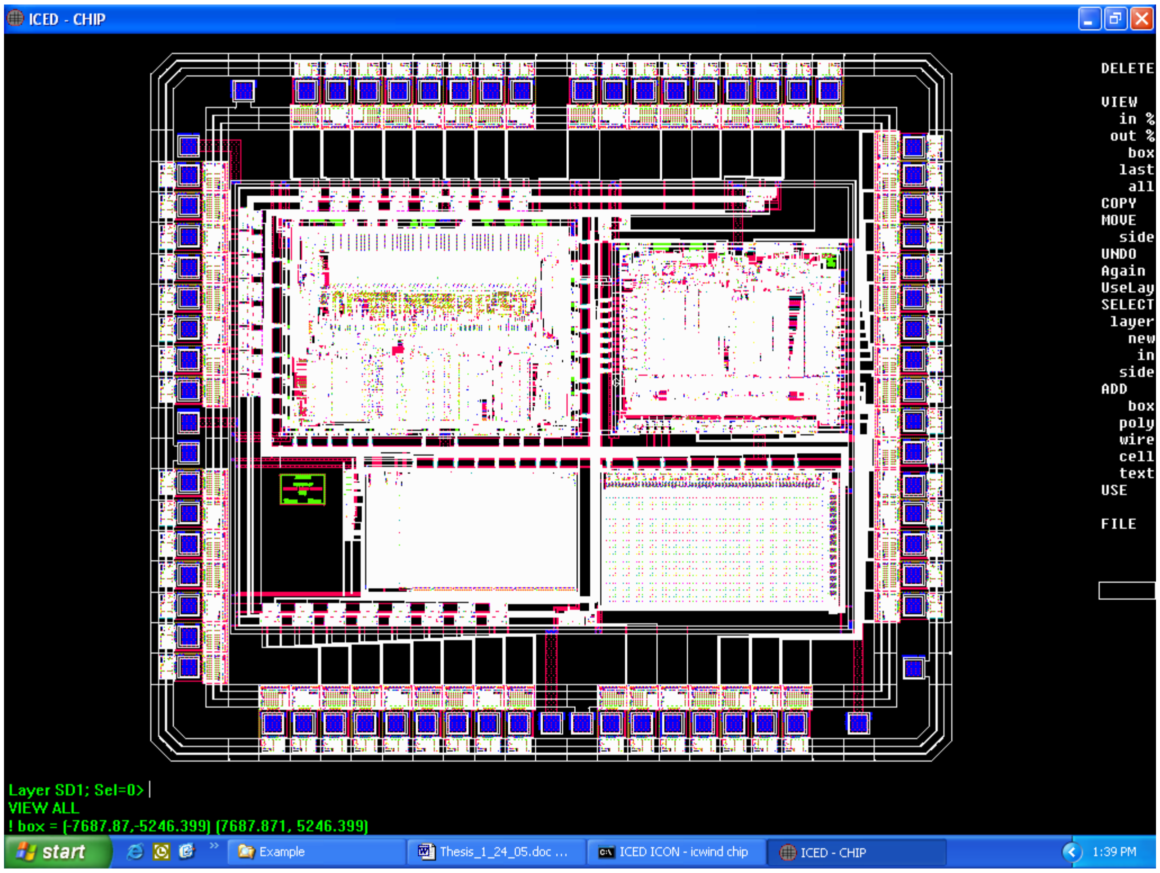
After the last prompt, there will again be some activity on the screen. Once Pass 2 is completed, your screen will look like Figure 3.25.



```
   Begin structure 322111C
   Begin structure 322111B
   Begin structure 322112A
   Begin structure 322112B
   Begin structure 322ACR
   Begin structure 322113A
   Begin structure 322103A
   Begin structure 322104A
   Begin structure 322103B
   Begin structure 322PCR
   Begin structure 322DI
   Begin structure 322105B
   Begin structure 322105C
   Begin structure 322103C
   Begin structure 322113B
   Begin structure 22SPB1
   Begin structure 22SPB2
   Begin structure 22SPB0
   Begin structure 22SPB3
   Begin structure 22SPB4
   Begin structure 22SPB6
   Begin structure 22SPB5
   Begin structure 22SPB7
   Begin structure SPCOR22
   Begin structure WIREOCB
   Begin structure IODM
   Begin structure SMS8
   Begin structure INVS
   Begin structure INVM
   Begin structure OBDM
   Begin structure BHD
   Begin structure ADDECODE
   Begin structure NA4S
   Begin structure NO4S
   Begin structure NO2S
   Begin structure NA2S
   Begin structure INVL
   Begin structure NA2M
   Begin structure NO3S
   Begin structure SPCOR122
   Begin structure RING122
   Begin structure VSSP
   Begin structure IOPD
   Begin structure CORNER
   Begin structure VDDC
   Begin structure SPACER
   Begin structure VSSC
   Begin structure VDDP
   Begin structure WIRERING
   Begin structure SPRNG122
   Begin structure AB65122
   Begin structure SPAB122
153487 elements processed

Conversion complete -- no errors

C:\Tutor122\GDSII\Example>
```

Figure 3.25 DOS window at the completion of pass 2

Now that the unstream process is complete, we can open up the layout and view it. To open the layout, type: **icwind chip** at the command prompt. There will be some activity as ICED reads in the file. When the file opens, it may be zoomed in to a particular area of the design. On the right hand side of the screen is the ICED menu. Under the View menu click All to view the complete design. Your screen will look like Figure 3.26.

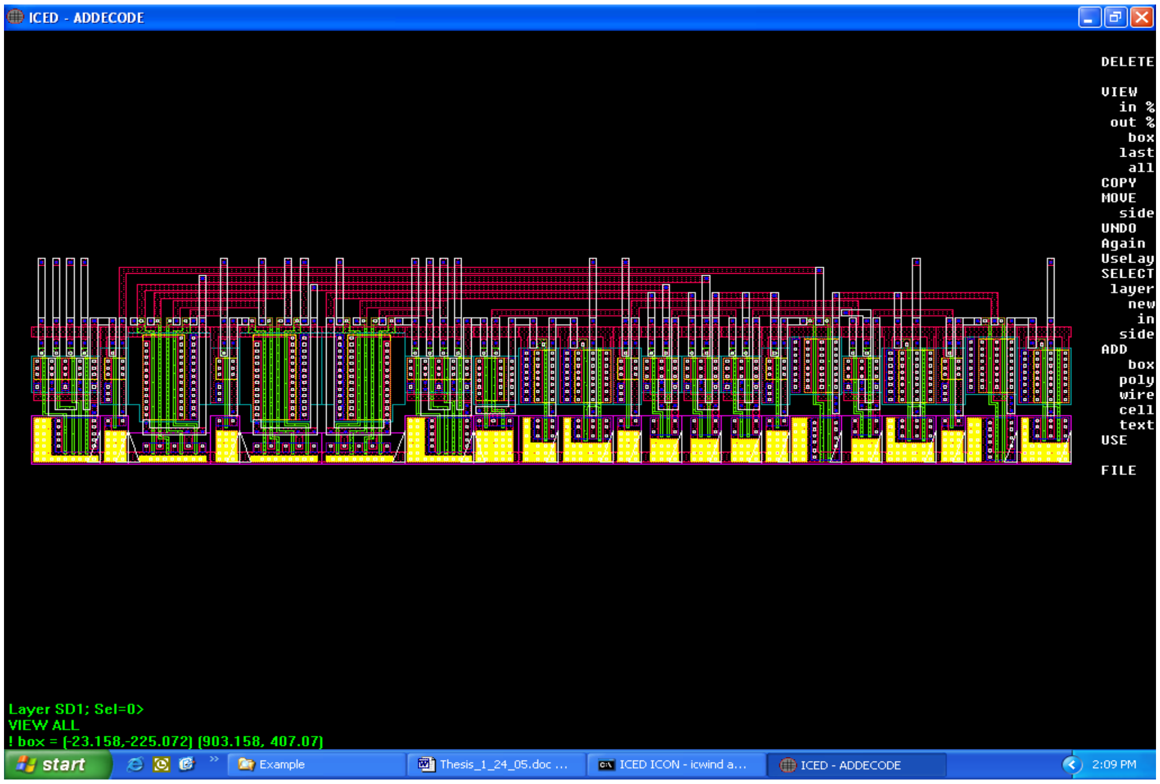


```
Layer SD1; Sel=0>|
VIEW ALL
! box = (-7687.87,-5246.399) (7687.871, 5246.399)
```

Figure 3.26 Complete layout of the W65C122S

Notice that the layout follows the same orientation as the ViewDraw schematic. The block in the top left corner is the W65C02 core, the W65C22 core is in the top right, the W4KROM is at the bottom left and the W256RAM is at the bottom right. The Address Decoder is located at the top left side of the W4KROM.

You can use the View menu to zoom in on different areas of the chip to view the layout in more detail. Alternately, you can close out the current view and open the cells individually to view them. For example, close out of the chip view by clicking File → Leave. At the command prompt, type: **icwind addecode** to view the Address Decoder by itself. Again, you may have to click View → All to see the entire layout. Your screen will look like Figure 3.27.



```
Layer SD1; Sel=0>
VIEW ALL
! box = (-23.158,-225.072) (903.158, 407.07)
```

Figure 3.27 Layout of the address decoder

A list of all the cells that make up the W65C122S can be found in C:\Tutor122\GDSII\Example. All files with a .CEL extension are the cell files and can be viewed by typing **icwind cellname** at the command prompt, as we did for the Address Decoder and Chip.

As previously mentioned, the layout for the W65C122S was manually placed and routed. The cores and standard cells were all laid out previously and are located in a library. When a new design needs to be laid out, the library is placed in the ICED search path and the cells can be added to the design. This is similar to the way a schematic symbol is added to a drawing in ViewDraw.

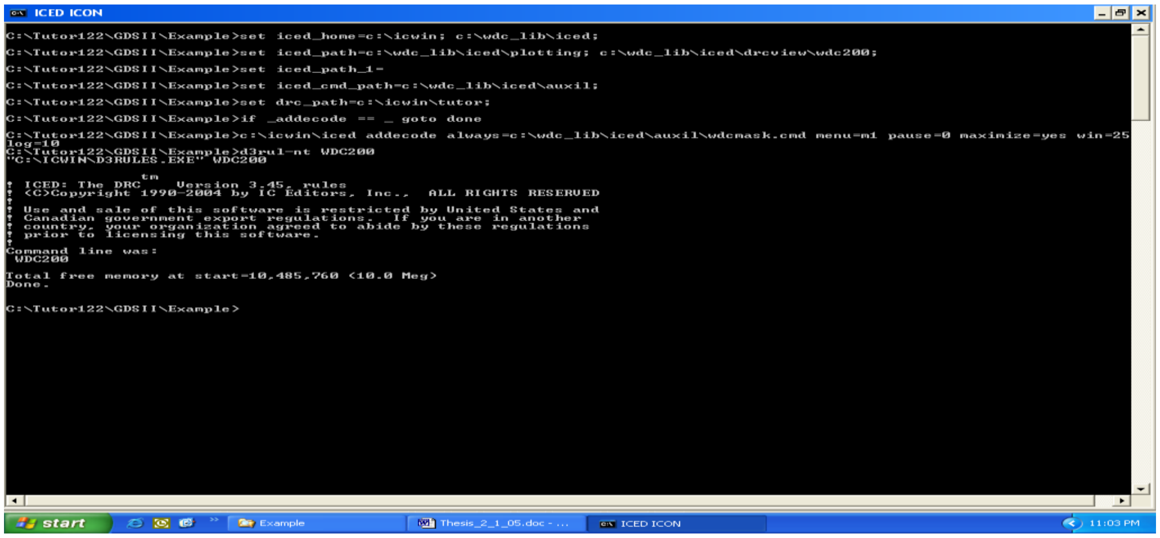
After the design has been laid out, a DRC needs to be done on the design to ensure that all design rules have been followed.

3.5.2 Performing a DRC on the W65C122S Core Layout

There are two DRCs done on the W65C122S layout. One DRC is done on the Core and the other DRC is done on the Ring. The Ring is comprised of the output pads around the chip and the Core is comprised of the W65C02, W65C22 etc… For this discussion, we will only be performing the DRC on the Core.

WDC has created a rules file that describes the layout rules for a 2µ process. All of WDC's designs are laid out using these 2µ design rules. The DRC that we will perform will check for errors against the rules file WDC200.RUL.

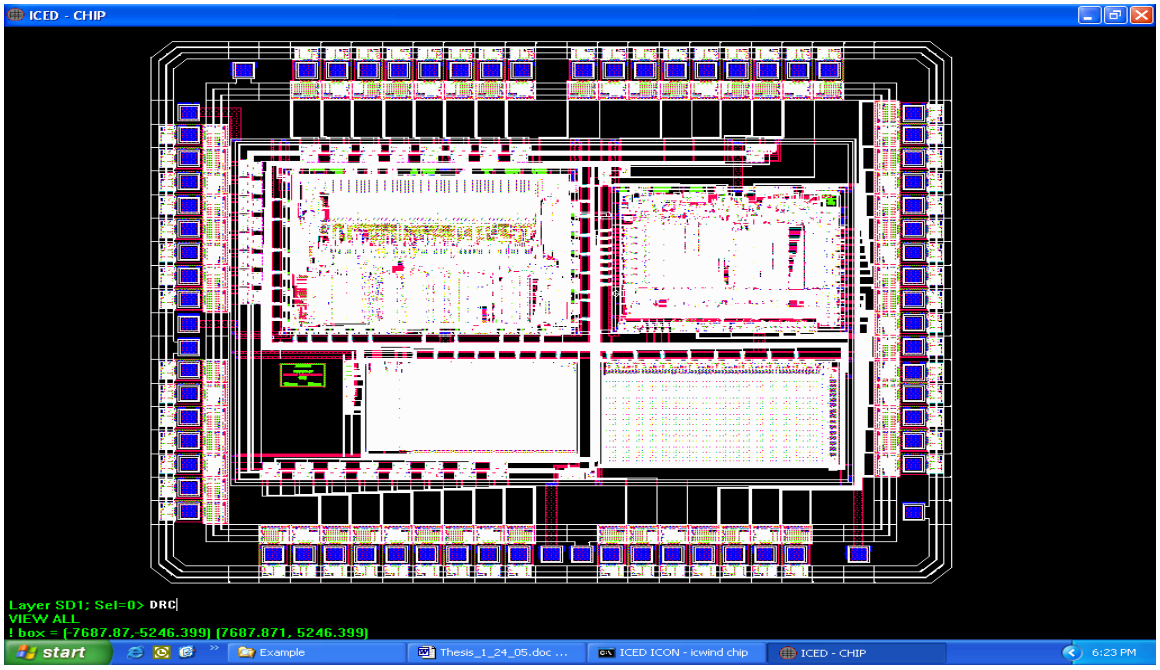
First, the rules file WDC200.RUL must be compiled by ICED. To compile the WDC200.RUL file, at the command prompt type: **d3rul-nt WDC200**. After some brief activity, your screen will look like Figure 3.28.



```
C:\Tutor122\GDSII\Example>set iced_home=c:\icwin; c:\wdc_lib\iced;
C:\Tutor122\GDSII\Example>set iced_path=c:\wdc_lib\iced\plotting; c:\wdc_lib\iced\drcview\wdc200;
C:\Tutor122\GDSII\Example>set iced_path_1=
C:\Tutor122\GDSII\Example>set iced_cmd_path=c:\wdc_lib\iced\auxil;
C:\Tutor122\GDSII\Example>set drc_path=c:\icwin\tutor;
C:\Tutor122\GDSII\Example>if _addecode == _ goto done
C:\Tutor122\GDSII\Example>c:\icwin\iced addecode always=c:\wdc_lib\iced\auxil\wdcmask.cmd menu=m1 pause=0 maximize=yes win=25
log=10
C:\Tutor122\GDSII\Example>d3rul-nt WDC200
"C:\ICWIN\D3RULES.EXE" WDC200
                   tm
! ICED: The DRC     Version 3.45, rules
! (C)Copyright 1990-2004 by IC Editors, Inc.,  ALL RIGHTS RESERVED
!
! Use and sale of this software is restricted by United States and
! Canadian government export regulations.  If you are in another
! country, your organization agreed to abide by these regulations
! prior to licensing this software.
!
Command line was:
 WDC200

Total free memory at start=10,485,760 (10.0 Meg)
Done.

C:\Tutor122\GDSII\Example>
```

Figure 3.28 DOS window after compiling the rules file

Next, open the chip file by typing: **icwind chip** at the command prompt, and view the entire chip by clicking View → All. Then type DRC from the keyboard. Notice that DRC is shown at the command line at the lower left hand corner of the screen, as shown in Figure 3.29.



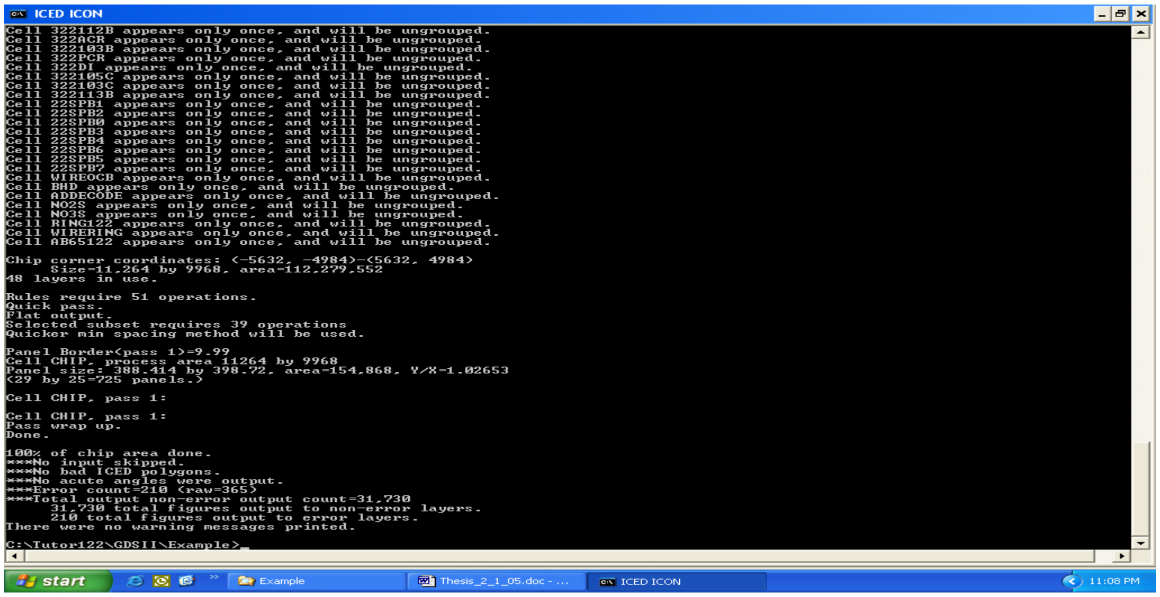
```
Layer SD1; Sel=0> DRC
VIEW ALL
! box = [-7687.87,-5246.399] [7687.871, 5246.399]
```

Figure 3.29 Starting DRC in the chip view

Next, press Enter. You will get a message at the bottom left of the screen stating:

! Output written to file C:\TUTOR122\GDSII\EXAMPLE\CHIP.POK. The .POK file is generated and used by ICED during the DRC process. Now click File → Leave to close out the chip view.

We are now ready to perform the DRC on the design. At the command prompt type: **drc3-nt wdc200 chip output quickpass**. WDC200 is the rules file, chip is the name of the chip, output is the name of the command file that will be generated by the DRC and quickpass is an option for the DRC program that is sufficient to accomplish the task. Next press Enter to run the DRC. There will be some activity in the DOS window as the DRC runs, and it will take a few minutes for the process to complete. When the DRC finishes, your screen will look similar to Figure 3.30.



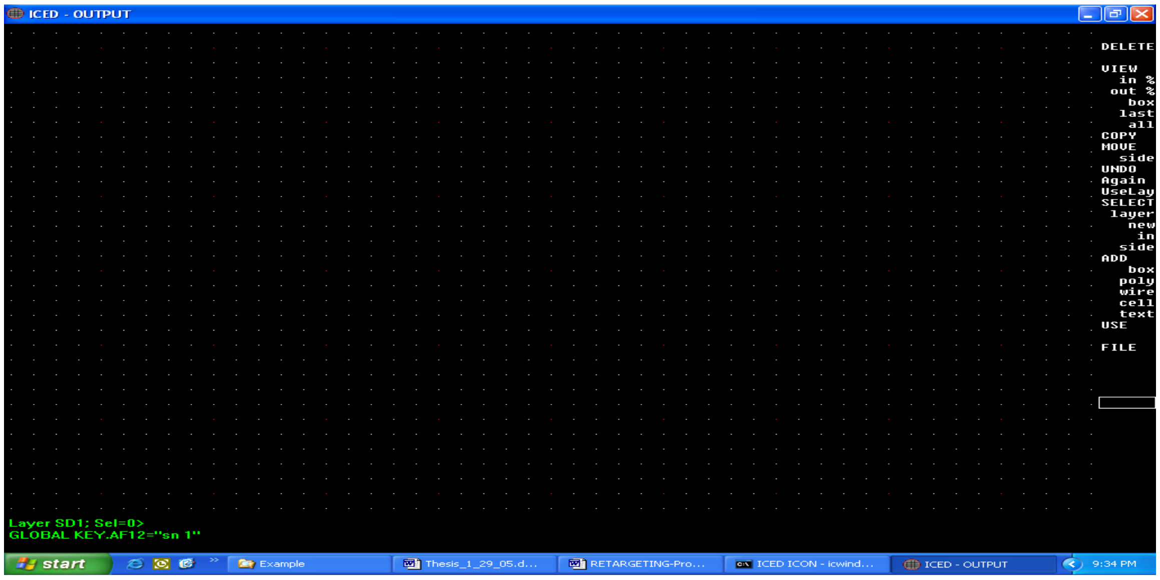
```
Cell 322112B appears only once, and will be ungrouped.
Cell 322ACR appears only once, and will be ungrouped.
Cell 322103B appears only once, and will be ungrouped.
Cell 322PCR appears only once, and will be ungrouped.
Cell 322DI appears only once, and will be ungrouped.
Cell 322105C appears only once, and will be ungrouped.
Cell 322103C appears only once, and will be ungrouped.
Cell 322113B appears only once, and will be ungrouped.
Cell 22SPB1 appears only once, and will be ungrouped.
Cell 22SPB2 appears only once, and will be ungrouped.
Cell 22SPB0 appears only once, and will be ungrouped.
Cell 22SPB3 appears only once, and will be ungrouped.
Cell 22SPB4 appears only once, and will be ungrouped.
Cell 22SPB6 appears only once, and will be ungrouped.
Cell 22SPB5 appears only once, and will be ungrouped.
Cell 22SPB7 appears only once, and will be ungrouped.
Cell WIREOCB appears only once, and will be ungrouped.
Cell BHD appears only once, and will be ungrouped.
Cell ADDECODE appears only once, and will be ungrouped.
Cell NO2S appears only once, and will be ungrouped.
Cell NO3S appears only once, and will be ungrouped.
Cell RING122 appears only once, and will be ungrouped.
Cell WIRERING appears only once, and will be ungrouped.
Cell AB65122 appears only once, and will be ungrouped.

Chip corner coordinates: (-5632, -4984)-(5632, 4984)
     Size=11,264 by 9968, area=112,279,552
48 layers in use.

Rules require 51 operations.
Quick pass.
Flat output.
Selected subset requires 39 operations
Quicker min spacing method will be used.

Panel Border(pass 1)=9.99
Cell CHIP, process area 11264 by 9968
Panel size: 388.414 by 398.72, area=154,868, Y/X=1.02653
(29 by 25=725 panels.)

Cell CHIP, pass 1:

Cell CHIP, pass 1:
Pass wrap up.
Done.

100% of chip area done.
***No input skipped.
***No bad ICED polygons.
***No acute angles were output.
***Error count=210 (raw=365)
***Total output non-error output count=31,730
     31,730 total figures output to non-error layers.
     210 total figures output to error layers.
There were no warning messages printed.

C:\Tutor122\GDSII\Example>_
```

Figure 3.30 DOS window at completion of DRC

The information at the end of the DRC shows the results of the DRC. It will show the error count, non-error count and any warnings generated. In our case, there are 31,730 non-errors and 210 errors in the layout with no warnings. It is important to note that not all errors reported are real errors in the design. ‘False’ errors are generated by the DRC due to text and other information describing the design that does not follow any rules. All errors need to be evaluated to determine if they are real or false errors.

There are two files that are generated by the DRC. The first is OUTPUT.CMD which is a command file that contains the error data found by the DRC and the second is OUTPUT.DLO which contains information about the DRC run and can be viewed in WordPad.

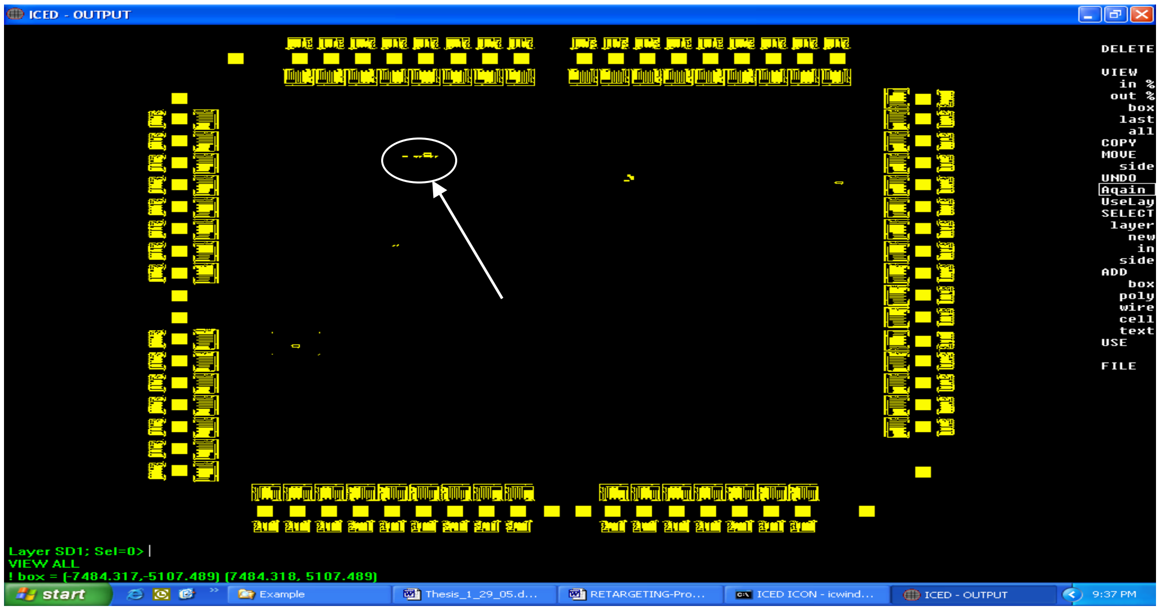
Next we want to view the output file to see the errors that were reported by the DRC. At the command prompt, type: **icwind output** and press Enter. This will open a new cell called output, and your screen will look similar to Figure 3.31.



```
Layer SD1; Sel=0>
GLOBAL KEY.AF12="sn 1"
```

Figure 3.31 New output cell ICED window

Now type: **@output** and press Enter. This will execute the OUTPUT.CMD command file and open the output cell showing the errors. You may have to click View → All to bring the entire chip into view. After doing so, your screen will look like Figure 3.32.

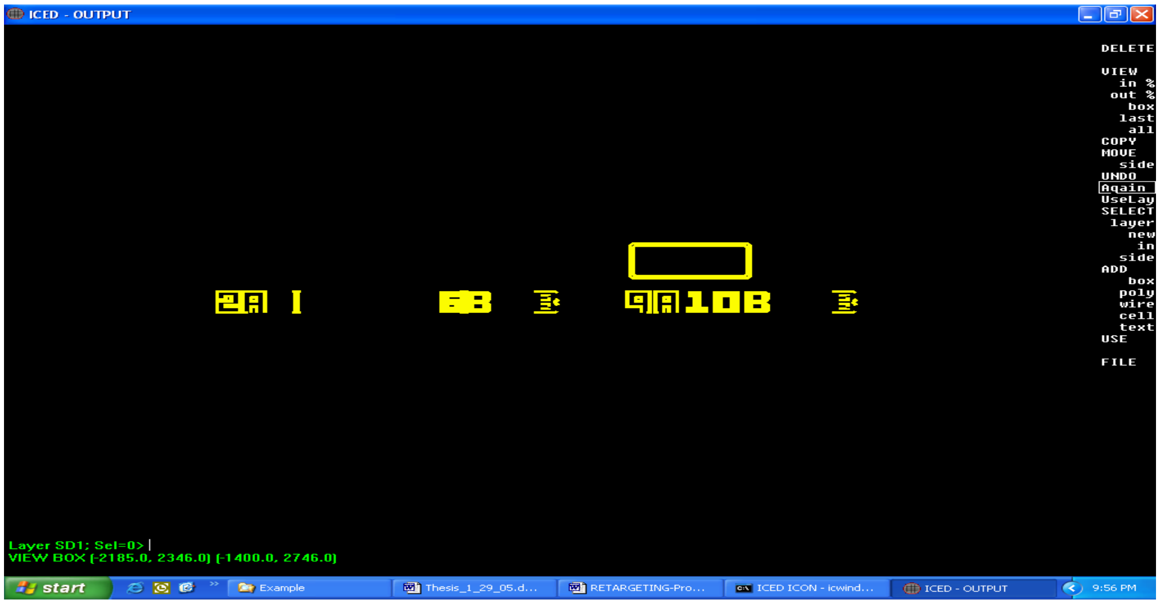


```
Layer SD1; Sel=0>|
VIEW ALL
! box = (-7484.317,-5107.489) (7484.318, 5107.489)
```

Figure 3.32 Output cell showing DRC errors

Figure 3.32 shows the errors detected by the DRC. However, as mentioned previously, this DRC is only for the Core of the chip. As you can see, many of the errors are due to the fact that the Ring is present, thus these can be ignored. In the center of the chip there are some random errors. These need to be checked to see if they are real or false errors.

Zoom in on the area indicated in Figure 3.32 by clicking View → Box. This will display the mouse pointer as an X. Move the pointer to the top left of the area shown in Figure 3.32 and click the left mouse button once. Then move the pointer to the lower right of the area until the box is around the entire area. Then, click the left mouse button once and release. This will zoom in on the selected area and your screen will look like Figure 3.33.



```
Layer SD1; Sel=0> |
VIEW BOX (-2185.0, 2346.0) (-1400.0, 2746.0)
```

Figure 3.33 Close up of DRC errors

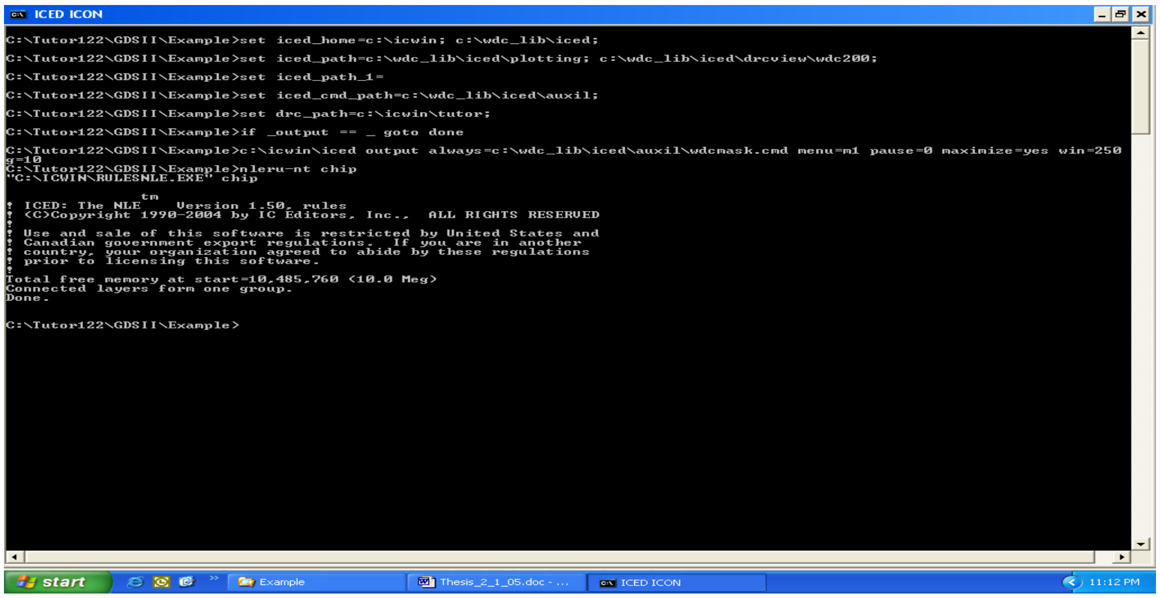
Notice that the area reflects text and other labels that describe the chip. Again, these are false errors and can be ignored. Follow the same procedure for the rest of the errors in the center of the chip. You will notice that all of these are text and markings that are not real errors and can be ignored. After reviewing all of the errors, close the window by clicking File → Leave.

Now that we have performed a DRC on the layout and analyzed all of the errors, the next step is to perform an NLE and LVS on the design.

3.5.3 Running NLE on the design

After the DRC has been performed and everything passes, the next step is to extract the netlist of the layout. The first thing that must be done is to compile the NLE rules files. This rules file is called CHIP.RUL and is located in the working directory. This rules file was created by WDC and has layer information that is read in and used by the NLE tool.

To compile the rules file, at the command prompt, type: **nleru-nt chip** and press Enter. After some brief activity, your screen will look like Figure 3.34.

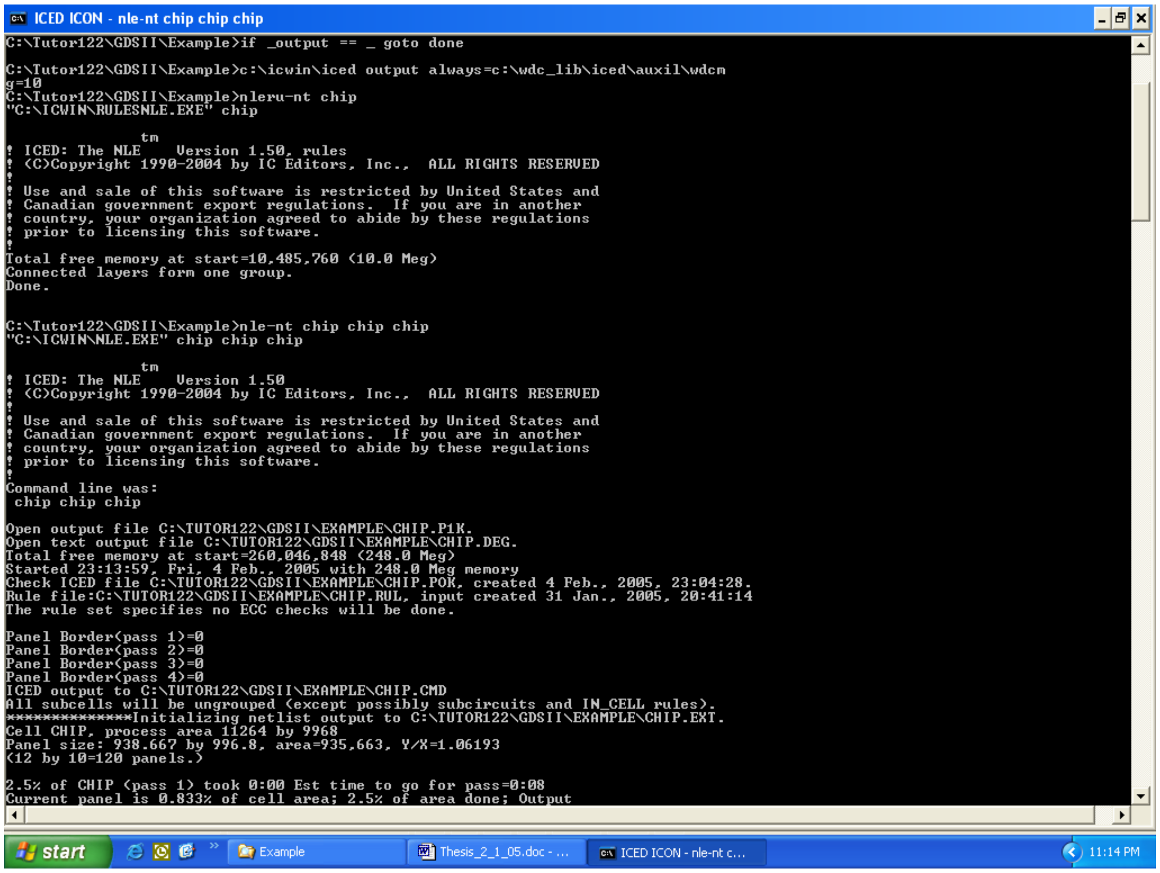


```
C:\Tutor122\GDSII\Example>set iced_home=c:\icwin; c:\wdc_lib\iced;

C:\Tutor122\GDSII\Example>set iced_path=c:\wdc_lib\iced\plotting; c:\wdc_lib\iced\drcview\wdc200;

C:\Tutor122\GDSII\Example>set iced_path_1=

C:\Tutor122\GDSII\Example>set iced_cmd_path=c:\wdc_lib\iced\auxil;

C:\Tutor122\GDSII\Example>set drc_path=c:\icwin\tutor;

C:\Tutor122\GDSII\Example>if _output == _ goto done

C:\Tutor122\GDSII\Example>c:\icwin\iced output always=c:\wdc_lib\iced\auxil\wdcmask.cmd menu=m1 pause=0 maximize=yes win=250
g=10
C:\Tutor122\GDSII\Example>nleru-nt chip
"C:\ICWIN\RULESNLE.EXE" chip

                 tm
! ICED: The NLE    Version 1.50, rules
! (C)Copyright 1990-2004 by IC Editors, Inc.,  ALL RIGHTS RESERVED
!
! Use and sale of this software is restricted by United States and
! Canadian government export regulations.  If you are in another
! country, your organization agreed to abide by these regulations
! prior to licensing this software.
!
Total free memory at start=10,485,760 (10.0 Meg)
Connected layers form one group.
Done.

C:\Tutor122\GDSII\Example>
```

Figure 3.34 Completion of NLE rules file compilation

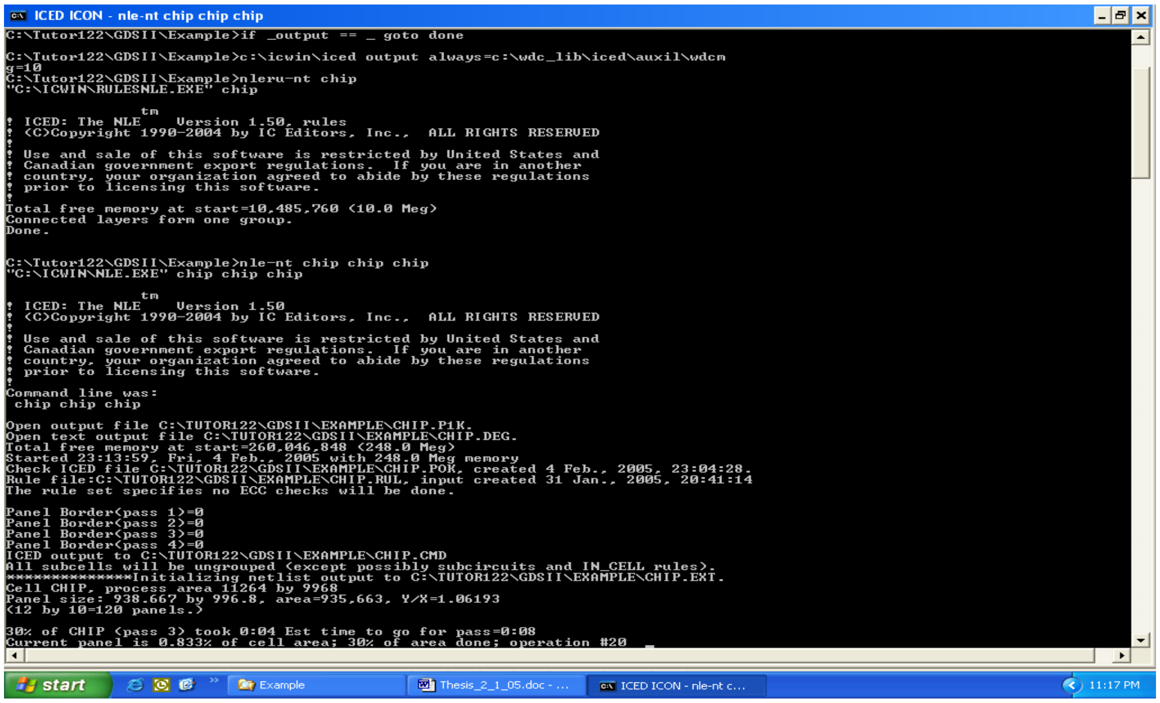
Next, we are ready to run the NLE program. This step will take a few minutes to complete and it is done in four passes. This is because ICED breaks up the design into four panels and extracts the data. To run the NLE, at the command prompt, type: **nle-nt chip chip chip**. In this case, nle-nt calls the NLE program; the first instance of chip is the rules file (chip.rul); the second instance of chip is the cell file (chip.cel) and the third instance of chip is the .POK file (chip.pok) that was generated by the DRC.

As previously mentioned, this step will take a few minutes to complete. Figure 3.35 shows the beginning stage of the extraction; Figure 3.36 shows the extraction after the second pass and Figure 3.37 shows the results after the extraction completes. After entering the above command line, there is no user input required for the duration of the netlist extraction.



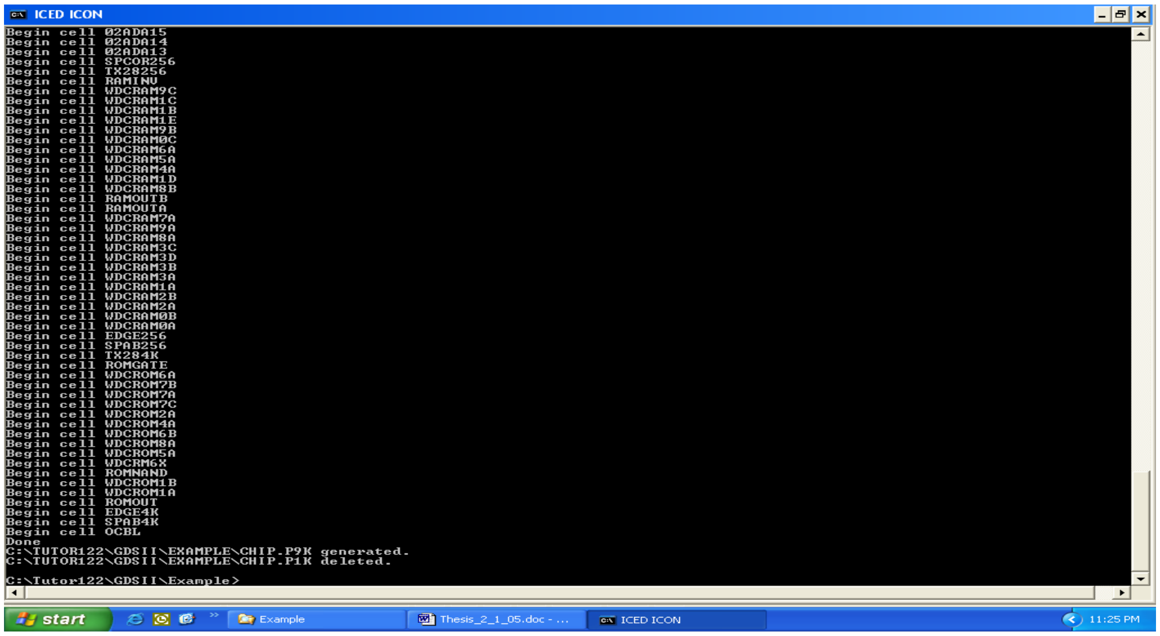
```
C:\Tutor122\GDSII\Example>if _output == _ goto done

C:\Tutor122\GDSII\Example>c:\icwin\iced output always=c:\wdc_lib\iced\auxil\wdcm
g=10
C:\Tutor122\GDSII\Example>nleru-nt chip
"C:\ICWIN\RULESNLE.EXE" chip

                tm
! ICED: The NLE    Version 1.50, rules
! (C)Copyright 1990-2004 by IC Editors, Inc.,  ALL RIGHTS RESERVED
!
! Use and sale of this software is restricted by United States and
! Canadian government export regulations.  If you are in another
! country, your organization agreed to abide by these regulations
! prior to licensing this software.
!
Total free memory at start=10,485,760 (10.0 Meg)
Connected layers form one group.
Done.


C:\Tutor122\GDSII\Example>nle-nt chip chip chip
"C:\ICWIN\NLE.EXE" chip chip chip

                tm
! ICED: The NLE    Version 1.50
! (C)Copyright 1990-2004 by IC Editors, Inc.,  ALL RIGHTS RESERVED
!
! Use and sale of this software is restricted by United States and
! Canadian government export regulations.  If you are in another
! country, your organization agreed to abide by these regulations
! prior to licensing this software.
!
Command line was:
 chip chip chip

Open output file C:\TUTOR122\GDSII\EXAMPLE\CHIP.P1K.
Open text output file C:\TUTOR122\GDSII\EXAMPLE\CHIP.DEG.
Total free memory at start=260,046,848 (248.0 Meg)
Started 23:13:59, Fri, 4 Feb., 2005 with 248.0 Meg memory
Check ICED file C:\TUTOR122\GDSII\EXAMPLE\CHIP.POK, created 4 Feb., 2005, 23:04:28.
Rule file:C:\TUTOR122\GDSII\EXAMPLE\CHIP.RUL, input created 31 Jan., 2005, 20:41:14
The rule set specifies no ECC checks will be done.

Panel Border(pass 1)=0
Panel Border(pass 2)=0
Panel Border(pass 3)=0
Panel Border(pass 4)=0
ICED output to C:\TUTOR122\GDSII\EXAMPLE\CHIP.CMD
All subcells will be ungrouped (except possibly subcircuits and IN_CELL rules).
**************Initializing netlist output to C:\TUTOR122\GDSII\EXAMPLE\CHIP.EXT.
Cell CHIP, process area 11264 by 9968
Panel size: 938.667 by 996.8, area=935,663, Y/X=1.06193
(12 by 10=120 panels.)

2.5% of CHIP (pass 1) took 0:00 Est time to go for pass=0:08
Current panel is 0.833% of cell area; 2.5% of area done; Output
```

Figure 3.35 Initial stage of netlist extraction



```
C:\Tutor122\GDSII\Example>if _output == _ goto done

C:\Tutor122\GDSII\Example>c:\icwin\iced output always=c:\wdc_lib\iced\auxil\wdcm
g=10
C:\Tutor122\GDSII\Example>nleru-nt chip
"C:\ICWIN\RULESNLE.EXE" chip

                   tm
! ICED: The NLE    Version 1.50, rules
! (C)Copyright 1990-2004 by IC Editors, Inc.,  ALL RIGHTS RESERVED
!
! Use and sale of this software is restricted by United States and
! Canadian government export regulations.  If you are in another
! country, your organization agreed to abide by these regulations
! prior to licensing this software.
!
Total free memory at start=10,485,760 (10.0 Meg)
Connected layers form one group.
Done.


C:\Tutor122\GDSII\Example>nle-nt chip chip chip
"C:\ICWIN\NLE.EXE" chip chip chip

                   tm
! ICED: The NLE    Version 1.50
! (C)Copyright 1990-2004 by IC Editors, Inc.,  ALL RIGHTS RESERVED
!
! Use and sale of this software is restricted by United States and
! Canadian government export regulations.  If you are in another
! country, your organization agreed to abide by these regulations
! prior to licensing this software.
!
Command line was:
 chip chip chip

Open output file C:\TUTOR122\GDSII\EXAMPLE\CHIP.P1K.
Open text output file C:\TUTOR122\GDSII\EXAMPLE\CHIP.DEG.
Total free memory at start=260,046,848 (248.0 Meg)
Started 23:13:59, Fri, 4 Feb., 2005 with 248.0 Meg memory
Check ICED file C:\TUTOR122\GDSII\EXAMPLE\CHIP.POK, created 4 Feb., 2005, 23:04:28.
Rule file:C:\TUTOR122\GDSII\EXAMPLE\CHIP.RUL, input created 31 Jan., 2005, 20:41:14
The rule set specifies no ECC checks will be done.

Panel Border(pass 1)=0
Panel Border(pass 2)=0
Panel Border(pass 3)=0
Panel Border(pass 4)=0
ICED output to C:\TUTOR122\GDSII\EXAMPLE\CHIP.CMD
All subcells will be ungrouped (except possibly subcircuits and IN_CELL rules).
**************Initializing netlist output to C:\TUTOR122\GDSII\EXAMPLE\CHIP.EXT.
Cell CHIP, process area 11264 by 9968
Panel size: 938.667 by 996.8, area=935,663, Y/X=1.06193
(12 by 10=120 panels.)

30% of CHIP (pass 3) took 0:04 Est time to go for pass=0:08
Current panel is 0.833% of cell area; 30% of area done; operation #20
```

Figure 3.36 Middle stage of netlist extraction



```
Begin cell 02ADA15
Begin cell 02ADA14
Begin cell 02ADA13
Begin cell SPCOR256
Begin cell TX28256
Begin cell RAMINV
Begin cell WDCRAM9C
Begin cell WDCRAM1C
Begin cell WDCRAM1B
Begin cell WDCRAM1E
Begin cell WDCRAM9B
Begin cell WDCRAM0C
Begin cell WDCRAM6A
Begin cell WDCRAM5A
Begin cell WDCRAM4A
Begin cell WDCRAM1D
Begin cell WDCRAM8B
Begin cell RAMOUTB
Begin cell RAMOUTA
Begin cell WDCRAM7A
Begin cell WDCRAM9A
Begin cell WDCRAM8A
Begin cell WDCRAM3C
Begin cell WDCRAM3D
Begin cell WDCRAM3B
Begin cell WDCRAM3A
Begin cell WDCRAM1A
Begin cell WDCRAM2B
Begin cell WDCRAM2A
Begin cell WDCRAM0B
Begin cell WDCRAM0A
Begin cell EDGE256
Begin cell SPAB256
Begin cell TX284K
Begin cell ROMGATE
Begin cell WDCROM6A
Begin cell WDCROM7B
Begin cell WDCROM7A
Begin cell WDCROM7C
Begin cell WDCROM2A
Begin cell WDCROM4A
Begin cell WDCROM6B
Begin cell WDCROM8A
Begin cell WDCROM5A
Begin cell WDCRM6X
Begin cell ROMNAND
Begin cell WDCROM1B
Begin cell WDCROM1A
Begin cell ROMOUT
Begin cell EDGE4K
Begin cell SPAB4K
Begin cell OCBL
Done
C:\TUTOR122\GDSII\EXAMPLE\CHIP.P9K generated.
C:\TUTOR122\GDSII\EXAMPLE\CHIP.P1K deleted.

C:\Tutor122\GDSII\Example>
```

Figure 3.37 Completion of netlist extraction

After the extraction completes, ICED generates a file called CHIP.P9K. This file contains the results of the extraction and is used during the LVS step.

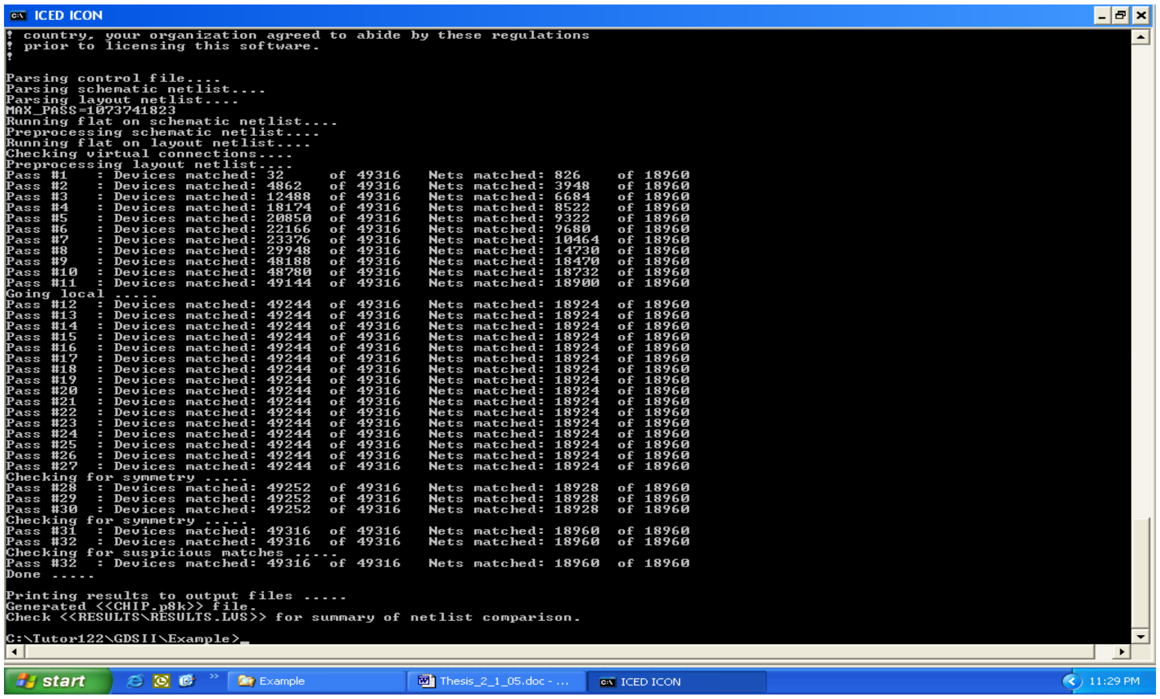
3.5.4 Running LVS on the design

After successfully extracting the netlist of the layout, the next step is to LVS the design, comparing the layout netlist to the schematic netlist. In order for this step to take place, there are some intermediate steps that need to be done that are beyond the scope of this document to perform. However, these steps will be described here to complete the understanding of the process. The LVS step will be performed using files that have already been created and proven by WDC.

The schematic netlist that was extracted in Section 3.3.2 is in Verilog, which is not an acceptable input for LVS. Thus, this netlist must be converted to a file format that can be read in by ICED. The format used is a Circuit Description Language (CDL) and WDC uses a utility called VERTOCDL.EXE to perform the conversion. Once the conversion to CDL is complete, this file needs to be hand modified in preparation for LVS. This hand modification is necessary as some module names need to be renamed and duplicate signal instantiations may need to be removed. The modification of the converted netlist can take a considerable amount of time to complete and then debug if necessary.

The working directory contains a file called CHIP.LVS, which is the CDL netlist of a hand-coded structural Verilog file describing the W65C122S. This is the file that we will be using for LVS. The following files are also required for LVS and these files are located in the working directory: CONTROL.LVS, SCHMODEL.NET, LVS_LAY.NET LAYMODEL.NET and LVS_SCH.NET. The CONTROL.LVS file describes the options and criteria used for the comparison. SCHMODEL.NET specifies the schematic models for the format of the schematic netlist. LVS_LAY.NET includes the layout netlist and layout models used for LVS. LAYMODEL.NET specifies transistor models and parasitic capacitance models. LVS_SCH.NET includes the schematic netlist, schematic models and library files that are used in LVS. All of these files can be viewed using WordPad.

To run LVS, type: **lvs-nt control.lvs lvs_sch.net lvs_lay.net** at the command prompt and press Enter. There will be some activity on the screen as LVS runs. LVS only takes a few seconds to complete.



```
! country, your organization agreed to abide by these regulations
! prior to licensing this software.
!

Parsing control file....
Parsing schematic netlist....
Parsing layout netlist....
MAX_PASS=1073741823
Running flat on schematic netlist....
Preprocessing schematic netlist....
Running flat on layout netlist....
Checking virtual connections....
Preprocessing layout netlist....
Pass #1   : Devices matched: 32     of 49316   Nets matched: 826    of 18960
Pass #2   : Devices matched: 4862   of 49316   Nets matched: 3948   of 18960
Pass #3   : Devices matched: 12488  of 49316   Nets matched: 6684   of 18960
Pass #4   : Devices matched: 18174  of 49316   Nets matched: 8522   of 18960
Pass #5   : Devices matched: 20850  of 49316   Nets matched: 9322   of 18960
Pass #6   : Devices matched: 22166  of 49316   Nets matched: 9680   of 18960
Pass #7   : Devices matched: 23376  of 49316   Nets matched: 10464  of 18960
Pass #8   : Devices matched: 29948  of 49316   Nets matched: 14730  of 18960
Pass #9   : Devices matched: 48188  of 49316   Nets matched: 18470  of 18960
Pass #10  : Devices matched: 48780  of 49316   Nets matched: 18732  of 18960
Pass #11  : Devices matched: 49144  of 49316   Nets matched: 18900  of 18960
Going local .....
Pass #12  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #13  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #14  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #15  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #16  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #17  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #18  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #19  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #20  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #21  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #22  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #23  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #24  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #25  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #26  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Pass #27  : Devices matched: 49244  of 49316   Nets matched: 18924  of 18960
Checking for symmetry .....
Pass #28  : Devices matched: 49252  of 49316   Nets matched: 18928  of 18960
Pass #29  : Devices matched: 49252  of 49316   Nets matched: 18928  of 18960
Pass #30  : Devices matched: 49252  of 49316   Nets matched: 18928  of 18960
Checking for symmetry .....
Pass #31  : Devices matched: 49316  of 49316   Nets matched: 18960  of 18960
Pass #32  : Devices matched: 49316  of 49316   Nets matched: 18960  of 18960
Checking for suspicious matches .....
Pass #32  : Devices matched: 49316  of 49316   Nets matched: 18960  of 18960
Done .....

Printing results to output files .....
Generated <<CHIP.p8k>> file.
Check <<RESULTS\RESULTS.LVS>> for summary of netlist comparison.

C:\Tutor122\GDSII\Example>_
```

Figure 3.38 Results of LVS

Figure 3.38 shows the results of the LVS. Notice that LVS checks for matched devices and matched nets. It is important that all devices and all nets match, as Figure 3.38 shows. This indicates that the schematic or hand-coded netlist matches the layout netlist, resulting in a successful LVS. If LVS indicates that there are unmatched devices and nets, the output files need to be analyzed to determine where the errors are and then they need to be corrected.

Once LVS completes, a Results directory is created in the working directory and all LVS output files are stored here. The LVS output files are: COLLAPSE.LVS which lists all devices that were collapsed into one device, EQUIV.LVS shows the equivalent names of nets between schematic and layout, FILTER.LVS shows filtered nets and devices, LABELS.LVS shows the layout net labels, MATCH.LVS shows the matched nets and devices, NETDEG.LVS shows net information after preprocessing, NETONE.LVS shows any floating nets, PARAM.LVS shows any parameter errors, RESULTS.LVS contains the final results of the comparison, SMETRIC.LVS shows devices that were forced matches, SPICE.LVS is the SPICE netlist and UNMATCH.LVS contains any unmatched devices and nets. All of these files can be viewed using WordPad.

As previously mentioned, all devices and nets must match before the design process can continue. After LVS, the next step in the process is PSPICE analysis to verify the functionality of the chip. SPICE.LVS is the file that will be used by PSPICE for verification. If there are errors in the LVS, then the SPICE netlist will contain errors, thus PSPICE analysis cannot be performed.

3.6 Retargeting the W65C122S to a different foundry process

Sections 3.3 to 3.5 discussed working with the W65C122S as it was designed using WDC's 2µ design rules. However, there are no longer foundries that support designs using this geometry, so the design must be retargeted to a technology supported by a foundry process. For example, .5µ processes remain popular and are cost effective, thus they are still supported by many foundries. This section will discuss the procedures for retargeting the W65C122S to a TSMC .5µ process.

3.6.1 Sizing the WDC 2μ design rules to target design rules

After the design has been drawn using the WDC 2μ design rules and passed the DRC and LVS steps, the design can be retargeted to any desired foundry process. First, the foundry and technology must be selected and in this example, the foundry is TSMC and the process is .5μ. The next step is to acquire the foundries proprietary design rules for the selected process. Typically, a Non-Disclosure Agreement (NDA) must be signed between the foundry and company looking to use the design rules.

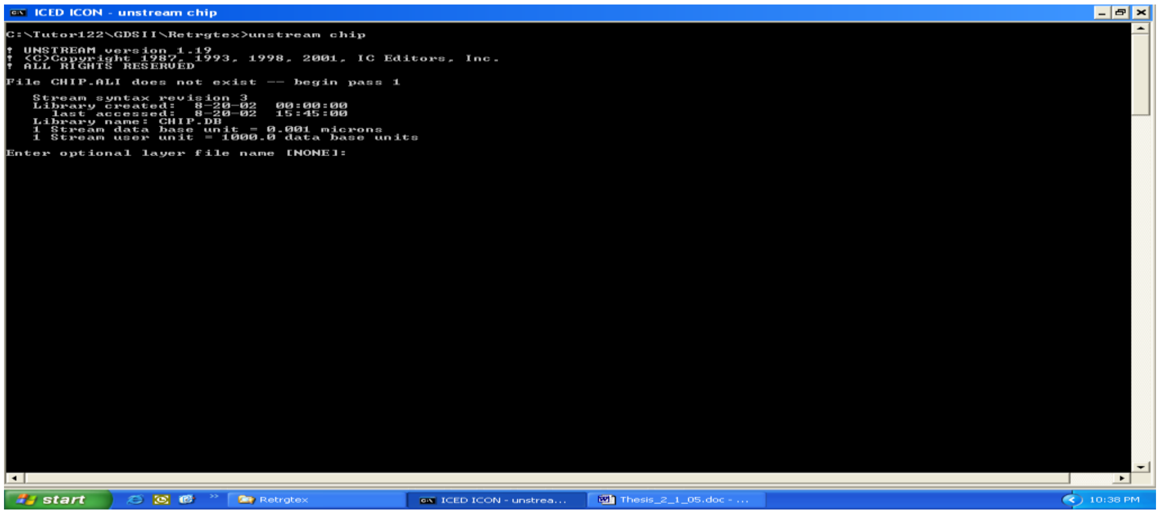
Once the design rules are acquired, the 2μ design rules must be sized to the .5μ design rules. As previously discussed in Section 2.4, WDC uses an Excel spreadsheet to size the 2μ rules to the targeted rules. The 2μ rules are entered into one column and the corresponding targeted .5μ rules are entered into another column. Due to the proprietary nature of these design rules, the spreadsheet will not be shown here. The spreadsheet is then executed and the new layer widths and spaces are displayed and a design rule check is performed. Any errors resulting from the DRC have to be analyzed as some may not be critical. Critical errors need to be corrected by further sizing.

Now that the new target layers have been determined, this information needs to be entered into a file called BIAS.RUL which will be used for the DRC step on the retargeted design.

3.6.2 Scaling the W65C122S using ICED

The next step in the retargeting process is scaling the original design using ICED so that it is redrawn to the target design rules. Scaling is done during the Unstream process where information entered into ICED creates a scaling factor for the .5μ rules.

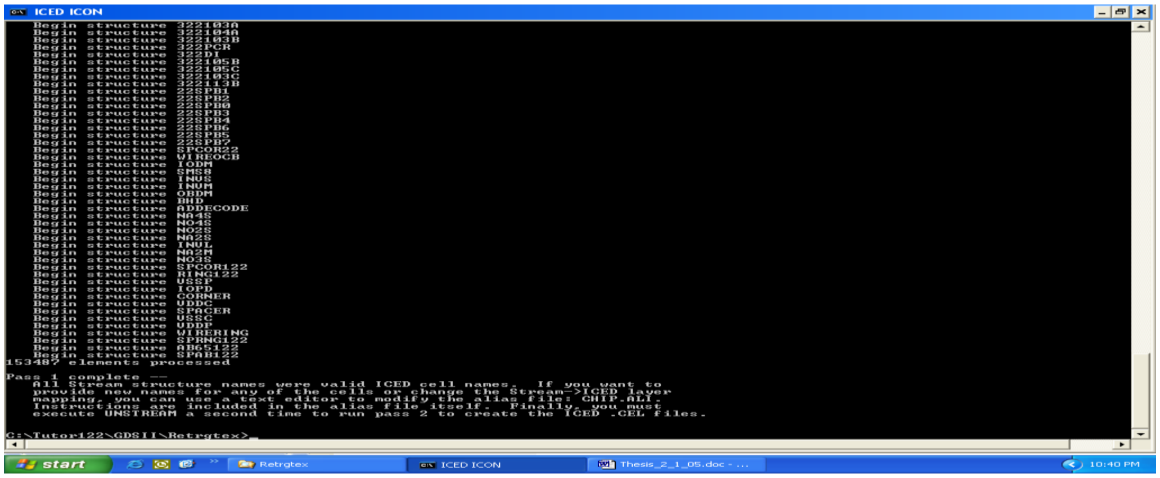
To scale the W65C122S we will be utilizing the CHIP.SF file in the C:\Tutor122\GDSII\Retrgtex directory. Open up ICED and change the default directory to C:\Tutor122\GDSII\Retrgtex. At the command prompt, type: **unstream chip** and press Enter to start the first pass of the Unstream process. Your screen will look like Figure 3.39.



```
C:\Tutor122\GDSII\Retrgtex>unstream chip

! UNSTREAM version 1.19
! (C)Copyright 1987, 1993, 1998, 2001, IC Editors, Inc.
! ALL RIGHTS RESERVED

File CHIP.ALI does not exist -- begin pass 1

   Stream syntax revision 3
   Library created:  8-20-02  00:00:00
     last accessed:  8-20-02  15:45:00
   Library name: CHIP.DB
   1 Stream data base unit = 0.001 microns
   1 Stream user unit = 1000.0 data base units

Enter optional layer file name [NONE]:
```

Figure 3.39 First pass of unstream process

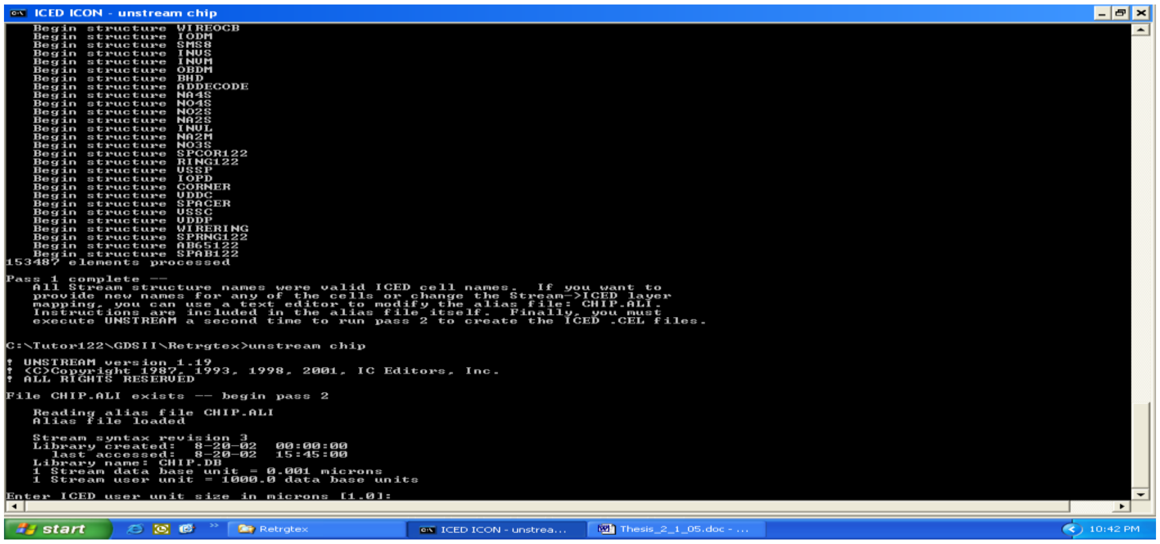
Press Enter for all of the prompts for Pass 1 as there is no input required during Pass 1. After Pass 1 completes, your screen will look like Figure 3.40.



```
   Begin structure 322103A
   Begin structure 322104A
   Begin structure 322103B
   Begin structure 322PCR
   Begin structure 322DI
   Begin structure 322105B
   Begin structure 322105C
   Begin structure 322103C
   Begin structure 322113B
   Begin structure 22SPB1
   Begin structure 22SPB2
   Begin structure 22SPB0
   Begin structure 22SPB3
   Begin structure 22SPB4
   Begin structure 22SPB6
   Begin structure 22SPB5
   Begin structure 22SPB7
   Begin structure SPCOR22
   Begin structure WIREOCB
   Begin structure IODM
   Begin structure SMS8
   Begin structure INUS
   Begin structure INUM
   Begin structure OBDM
   Begin structure BHD
   Begin structure ADDECODE
   Begin structure NA4S
   Begin structure NO4S
   Begin structure NO2S
   Begin structure NA2S
   Begin structure INUL
   Begin structure NA2M
   Begin structure NO3S
   Begin structure SPCOR122
   Begin structure RING122
   Begin structure USSP
   Begin structure IOPD
   Begin structure CORNER
   Begin structure UDDC
   Begin structure SPACER
   Begin structure USSC
   Begin structure UDDP
   Begin structure WIRERING
   Begin structure SPRNG122
   Begin structure AB65122
   Begin structure SPAB122
153487 elements processed

Pass 1 complete --
   All Stream structure names were valid ICED cell names.  If you want to
   provide new names for any of the cells or change the Stream->ICED layer
   mapping, you can use a text editor to modify the alias file: CHIP.ALI.
   Instructions are included in the alias file itself.  Finally, you must
   execute UNSTREAM a second time to run pass 2 to create the ICED .CEL files.

C:\Tutor122\GDSII\Retrgtex>_
```

Figure 3.40 Completion of unstream pass 1

Next, once again type: **unstream chip** at the command prompt and press Enter to begin Pass 2. It is during Pass 2 that the scaling information is entered. Your screen will look like Figure 3.41 at the start of Pass 2.



```
ICED ICON - unstream chip
    Begin structure WIREOCB
    Begin structure IODM
    Begin structure SMS8
    Begin structure INVS
    Begin structure INVM
    Begin structure OBDM
    Begin structure BHD
    Begin structure ADDECODE
    Begin structure NA4S
    Begin structure NO4S
    Begin structure NO2S
    Begin structure NA2S
    Begin structure INVL
    Begin structure NA2M
    Begin structure NO3S
    Begin structure SPCOR122
    Begin structure RING122
    Begin structure VSSP
    Begin structure IOPD
    Begin structure CORNER
    Begin structure VDDC
    Begin structure SPACER
    Begin structure VSSC
    Begin structure VDDP
    Begin structure WIRERING
    Begin structure SPRNG122
    Begin structure AB65122
    Begin structure SPAB122
153487 elements processed

Pass 1 complete --
    All Stream structure names were valid ICED cell names.  If you want to
    provide new names for any of the cells or change the Stream->ICED layer
    mapping, you can use a text editor to modify the alias file: CHIP.ALI.
    Instructions are included in the alias file itself.  Finally, you must
    execute UNSTREAM a second time to run pass 2 to create the ICED .CEL files.

C:\Tutor122\GDSII\Retrgtex>unstream chip

! UNSTREAM version 1.19
! <C>Copyright 1987, 1993, 1998, 2001, IC Editors, Inc.
! ALL RIGHTS RESERVED

File CHIP.ALI exists -- begin pass 2

    Reading alias file CHIP.ALI
    Alias file loaded

    Stream syntax revision 3
    Library created:  8-20-02  00:00:00
      last accessed:  8-20-02  15:45:00
    Library name: CHIP.DB
    1 Stream data base unit = 0.001 microns
    1 Stream user unit = 1000.0 data base units

Enter ICED user unit size in microns [1.0]:
```

Figure 3.41 Second pass of unstream process

The following shows the Pass 2 prompts and the required user response will be in **BOLD**. Press Enter after each entry. Where no input is required, press Enter, shown as **(Enter)**. ICED will display some information lines after certain entries.

Enter ICED user unit size in microns [1.0]: **1**

Enter divisions per unit [1000]: **(Enter)**

Are you using UNSTREAM to scale this design or snap it to grid (Y or [N])? **Y**

Enter the original feature size [1.0]: **2**

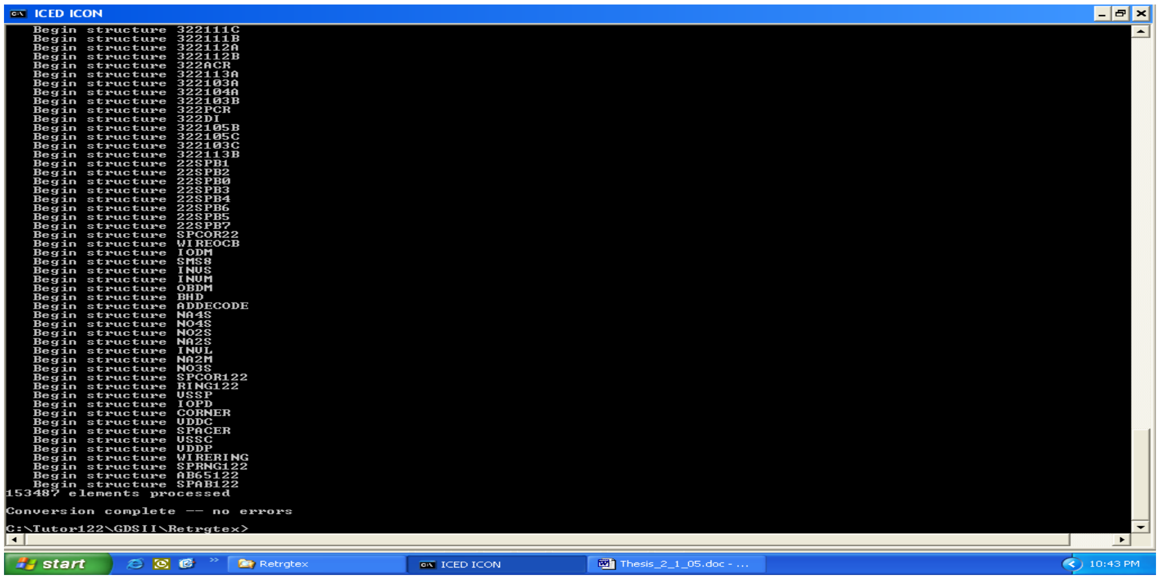
Enter the final feature size [1.0]: **0.5**

Enter snap grid step size in microns [0.001]: **(Enter)**

Enter maximum allowable coordinate rounding error in microns [0.0]: **0.01**

Font[0] is "GDSII:CALMAFONT.TX" -- Enter character height in microns [1]: **4**

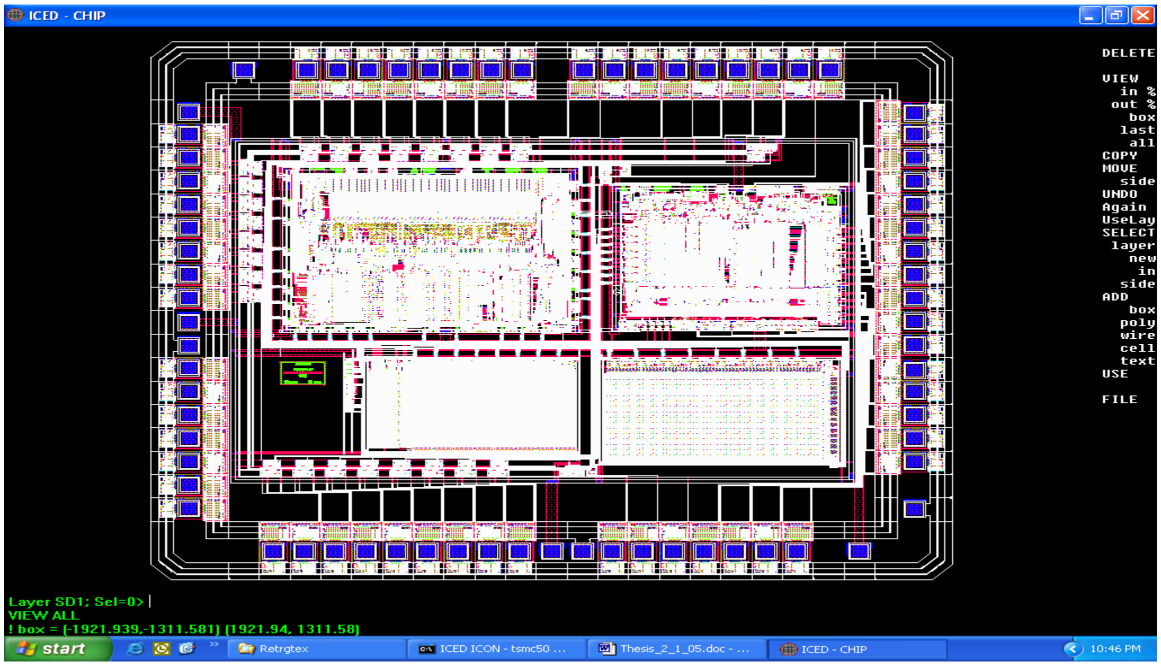
Press Enter for all remaining prompts as no other input is required. After pressing Enter at the final prompt, there will be some activity on the screen while Unstream completes. When completed, your screen will look like Figure 3.42.



```
    Begin structure 322111C
    Begin structure 322111B
    Begin structure 322112A
    Begin structure 322112B
    Begin structure 322ACR
    Begin structure 322113A
    Begin structure 322103A
    Begin structure 322104A
    Begin structure 322103B
    Begin structure 322PCR
    Begin structure 322DI
    Begin structure 322105B
    Begin structure 322105C
    Begin structure 322103C
    Begin structure 322113B
    Begin structure 22SPB1
    Begin structure 22SPB2
    Begin structure 22SPB0
    Begin structure 22SPB3
    Begin structure 22SPB4
    Begin structure 22SPB6
    Begin structure 22SPB5
    Begin structure 22SPB7
    Begin structure SPCOR22
    Begin structure WIREOCB
    Begin structure IODM
    Begin structure SMS8
    Begin structure INVS
    Begin structure INVM
    Begin structure OBDM
    Begin structure BHD
    Begin structure ADDECODE
    Begin structure NA4S
    Begin structure NO4S
    Begin structure NO2S
    Begin structure NA2S
    Begin structure INVL
    Begin structure NA2M
    Begin structure NO3S
    Begin structure SPCOR122
    Begin structure RING122
    Begin structure VSSP
    Begin structure IOPD
    Begin structure CORNER
    Begin structure VDDC
    Begin structure SPACER
    Begin structure VSSC
    Begin structure VDDP
    Begin structure WIRERING
    Begin structure SPRNG122
    Begin structure AB65122
    Begin structure SPAB122
153487 elements processed

Conversion complete -- no errors

C:\Tutor122\GDSII\Retrgtex>
```

Figure 3.42 Completion of unstream pass 2

After the completion of Pass 2 of Unstream, the scaling process is complete. The design has now been retargeted to the TSMC .5µ design rules. Next, view the chip with the TSMC layers by typing: **TSMC50 chip** at the command prompt. TSMC50 is the startup file that calls the file containing the TSMC .5µ layers. After the file opens, click View → All to see the entire chip. Your screen will look like Figure 3.43.



```
Layer SD1; Sel=0> |
VIEW ALL
! box = (-1921.939,-1311.581) (1921.94, 1311.58)
```

Figure 3.43 Chip view with TSMC layers

After viewing the chip, close out of ICED by clicking File → Leave.

3.6.3 Post retargeting steps

Now that the design has been retargeted to the new design rules, DRC must be run on the design to check if there are any design rule violations. The process for DRC is generally the same as was discussed in Section 3.5.2, and will not be performed here.

However, a different rules file is used and the DRC is done on the entire chip, not separately for the core and pad ring. The rules file used is the one created after sizing to the new design rules using the Excel spreadsheet.

After DRC has passed on the retargeted design, the layout netlist must be extracted and then LVS must be performed. Again, the process for NLE and LVS are generally the same as discussed in Sections 3.5.3 and 3.5.4 respectively. All of the required files to complete the DRC, NLE and LVS steps for the TSMC .5µ design rules have been created and tested by WDC.

Finally, after LVS has been successfully completed, the design is ready for functional verification using PSPICE. Electrical characterizations and timing analysis are also done using PSPICE.

3.7 Using PSPICE to verify device functionality

After a successful LVS, we now have a SPICE netlist that can be used for PSPICE analysis. The SPICE netlist as generated by ICED needs to be modified to be accepted by PSPICE, as it contains characters that are recognized as comments in PSPICE and contains only node numbers. Thus, the netlist needs to be modified to remove the comment lines and add node names to make debugging easier.

Similar to the process of modifying the extracted Verilog netlist for NLE and LVS, the preparation and debugging of the PSPICE netlist can be very time consuming and is beyond the scope of this document to perform. Furthermore, setting up the PSPICE simulation files and running the simulation is also beyond the scope of this document. However, this section will discuss the steps required to convert the SPICE netlist and prepare it for simulation, preparation of simulation files and setting up the PSPICE simulation.

3.7.1 Converting the extracted SPICE netlist to PSPICE format

To prepare the “NETLIST.NET” file required by PSPICE, two programs were written to perform the conversion. One program is written in C and the other in PERL. This step must be done on the system at WDC that has a C complier and the PERL program. Both programs will accomplish the same task and the user can decide what program to use. For simplification, this discussion will use the PERL program as the example.

Two text files with the names file1.txt and file2.txt are created in a separate directory which does not need to be in the working directory. The contents of the LABELS.LVS file are copied into file2.txt and the text *.NETLABEL is globally removed from the file. Next, the contents of the SPICE.LVS file are placed in file1.txt and the NLE comments are removed. Lastly, the PERL script is executed and creates a file called file3.txt. This file now contains the netlist of transistors and capacitors with node names instead of node numbers. This file can now be used as an input to PSPICE and is renamed NETLIST.NET.

3.7.2 Setting up the PSPICE simulation

The PSPICE project contains several files required for simulation. The PSPICE project file has a .CIR extension to it. A working directory is set up to store the simulation files and project file. These simulation files are: NETLIST.NET which is the converted SPICE netlist that was generated by LVS, MODEL.INC which contains the model parameters for the transistors and capacitors used in the design. These parameters are from a particular foundry process. PROBE.INC contains the digital and analog nodes that we want to view after simulation. STIMULUS.STM contains the inputs to the microprocessor and other inputs and this file is generated from the Verilog simulation. DIGITAL.INC contains the test setup including voltage levels, memory, A/D and D/A converters and any other required circuitry. RAM.IHX contains the RAM data in Intel Hex format.

Once all of the files have been prepared and placed into the working directory for the PSPICE simulation, the .CIR file is opened which automatically opens PSPICE. The simulation is started from within the PSPICE environment and can take a few hours to complete.

After the simulation has completed, all of the data is stored in a .DAT file. To view the results of the simulation, the waveform window is opened and signals are added to the waveform window. Generally, two waveform windows are opened. One window is used to view the desired digital signals and the other to view the desired analog signals. The digital signals represent those signals that transition from a high-to-low or low-to-high state with respect to the clock. The analog signals represent those signals that have a transient response, and do not change with respect to a clock edge. Figure 3.44 shows both digital and analog signal windows.

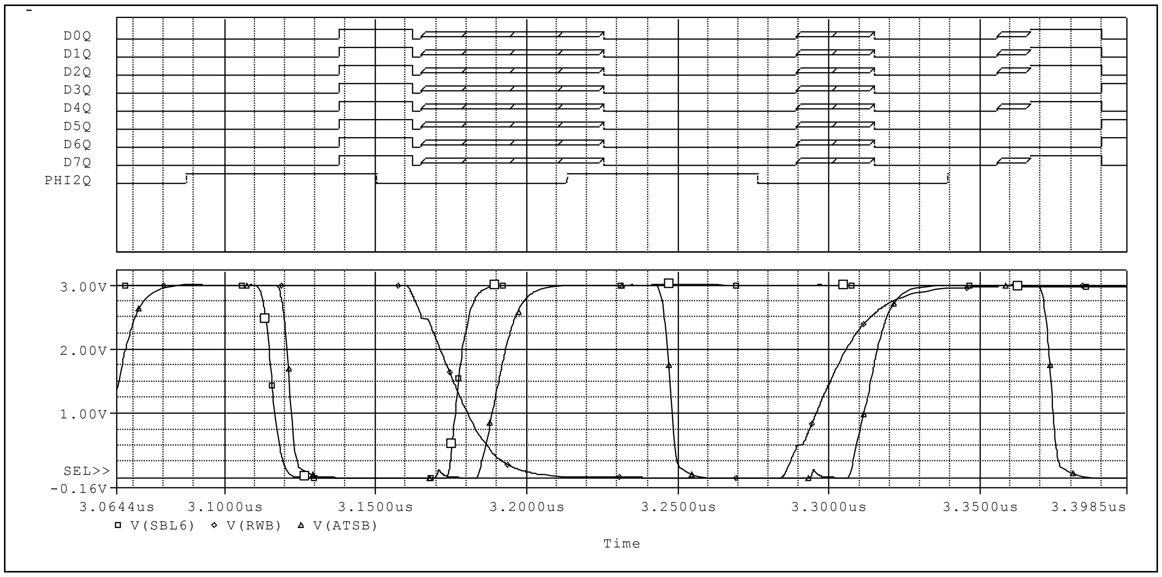


Figure 3.44 PSPICE waveform window with digital and analog signals

Note that waveform windows can be opened at anytime during the simulation and results can be viewed as they are output to the screen. The resulting waveforms need to be analyzed to verify that the design is working as expected.

PSPICE analysis is the final step in the design verification process. This step is critical and time needs to be spent analyzing the simulation results. If something is not performing correctly, it is possible that transistor sizes need to be modified in the netlist and the design re-simulated. If a change to transistor sizes has been made to the simulation netlist, and the problem has been fixed, the layout must also be modified to reflect the changes. This means that the DRC, NLE and LVS steps must be performed again to verify that the layout changes have not violated any design rules and that it still matches the schematic. Then, a new simulation netlist must be created and this netlist must be simulated in PSPICE. The process continues in this way until the design is determined to be working correctly.

As previously mentioned, PSPICE analysis is a critical step in the design flow. The step after functional verification is sending the design out to the foundry to be manufactured. If the design was not thoroughly tested for proper functionality, the chips received from the foundry will not function properly. This will result in time and money being wasted, as the design will need to be re-evaluated; DRC, NLE and LVS performed; functional verification performed and re-sent to the foundry for manufacturing. This will also delay the sale of the chip and market share may be lost.

3.8 Sending the final design to be manufactured

Once the design has gone through functional verification, it is ready to be sent to the foundry to be manufactured. Section 2.6 discussed some options available for manufacturing. One option is manufacturing a limited amount of chips on a multi-project wafer. An MPW run yields approximately forty chips at a fraction of the cost for a full run at the foundry.

This option is beneficial for new designs, especially if the design turns out to be faulty. Spending $6,000.00 for forty chips to find out that the design is faulty is more acceptable than spending $25,000.00 for thousands of faulty chips. A MPW run would be the preferred option for WDC if they were to manufacture the W65C122S, as the design was never sent out to be fabricated.

The other option is sending the design out for a production run at the foundry. This option is usually chosen for designs that have been proven, and just retargeted to a different process. The following section will describe this option as used by WDC for proven designs.

3.8.1 Sending the design to the foundry for full production run

The current foundry used by WDC is TSMC. WDC interfaces with TSMC through a company called Progate, also located in Taiwan. Progate handles all of the contractual paperwork, performs all of the final design checks, sends the final design files to TSMC and returns the completed wafers to WDC.

Using the W65C122S as an example and assuming that the design has been proven on a MPW run, the following describes the process for obtaining a full production run of wafers for the device.

The TSMC .5µ process is chosen and all DRC and LVS steps have been successfully completed. The output file is then streamed (compressed) into a .SF file. All contracts are signed, the .SF file is sent to Progate and funds are transferred to pay for the production. Progate performs the final design checks to ensure there are no conflicts with the targeted process. After Progate completes their process, called tape out, they send the design to TSMC to be manufactured.

Initially, a pilot run of six wafers are manufactured. This pilot run costs approximately $25,000.00. A pilot run is done first so that the design can be tested on silicon before committing to a full production run of twenty-five wafers. Once the pilot run is completed and wafers released to WDC, WDC has the die on three wafers packaged and returned to them. These packaged parts are then tested in house and on automated testers as discussed in Sections 2.7.1 and 2.7.2.

There is a good possibility that even though all design rules have been followed, the design may not work properly on silicon due to the manufacturing process. If testing determines that the design is not working properly, the problem areas need to be analyzed, corrections need to be made to the layout, and the design process repeats itself from the DRC step. WDC has the option at this point to have another pilot run manufactured, or go to full production if they feel confident in the design changes.

If testing determines that the design is working properly, the remaining three wafers are released to WDC, packaged and tested. At this point, the part is available for sale. The amount of parts received depends on the size of the W65C122S die and the size of the wafer. The W65C122S on a six inch wafer would yield approximately 2000 – 2500 good die. Therefore, an entire pilot run can yield approximately 12,000 parts.

WDC can then opt to have a full production run of wafers manufactured depending on chip sales. The cost for a production run of twenty-five wafers is approximately $25,000.00. Notice that a production run costs the same as a pilot run. This is because there are Non-Recurring Engineering (NRE) costs and mask costs involved in the initial phase of manufacturing. Once the design is proven, these NRE and mask costs are no longer incurred.

3.9 Real world application example

This document introduced the student to the concepts of VLSI design, provided an introduction to design tools used in industry and provided hands-on examples using WDC's 65xx technology, introducing the concept of design reuse. This section will relate the concepts discussed in this document to a real world example, providing a better understanding to the value of design reuse.

3.9.1 Licensing and using 65xx IP

The main focus of WDC's business model is licensing reusable IP. All of the cores used in the W65C122S (W65C02, W65C22, W4KROM, 256RAM) are available for companies to license and use in their own custom SOCs. WDC also has IP cores for the W65C816 16-bit microprocessor, W65C134 8-bit microcontroller and W65C265 16-bit microcontroller. All of these cores have been proven to be reliable over the years and are used today in various products such as automotive, consumer and medical to name a few.

Typically, a company will license a microprocessor core from WDC and add other core components around it to develop their own custom chip. These added components can be FLASH, I/O interfaces, A to D and D to A and network interfaces as examples.

For example, let's say that a company wants an 8-bit microprocessor and I/O capabilities, but needs more RAM than the 256 bytes of RAM available on the W65C122S. The company can license the W65C02 and W65C22 from WDC for the microprocessor and I/O. Then, they can either develop their own RAM cell or license a RAM cell from another company to fit their design needs. By licensing and using existing IP, the design can be realized faster and the product can be on the market in a shorter amount of time than if the company was designing everything by themselves.

In summary, using reusable IP for a design will cut down on design costs, shorten the design time, enable the product to be on the market in a timely manner and provide the user with confidence in the design that results from using proven technology.

4. Conclusions and Recommendations

4.1 Conclusions

The original focus of this project was to develop a laboratory manual for UET 513 – Introduction to VLSI Design. The focus was then shifted to a hands-on tutorial in VLSI design utilizing the concept of design reuse when UET 513 was no longer offered. It was realized that a tutorial was needed to provide the student with valuable hands-on experience using the tools involved in developing a SOC. The main concept present throughout this tutorial development was that this is only an *introduction* to the concepts and tools of VLSI design.

The assumption that the student has little or no familiarity with VLSI design was addressed in Chapter 2. Here, background and history of VLSI were presented to give the students an appreciation of how the design methodology has developed over the years. Also presented in Chapter 2 is information on each part of the design flow, from design entry to fabrication. Each step, where applicable, includes an introduction to the tool used in the step, accompanied with a hands-on example. This material gives the student the necessary background information that will help them understand the design process as they are working with the W65C122S SOC presented in Chapter 3.

Chapter 3 introduces the student to the W65C122S SOC, reusable technology and more detailed information about the design steps. Here, the student uses the tools introduced to them in Chapter 2 and works with the W65C122S instead of a simple example. Since the W65C122S has already been designed, the student performs various procedures of the W65C122S design flow and compares their results to known good results. Chapter 3 also relates the design concepts to a real world example and business models used by The Western Design Center, Inc.

Following this tutorial will provide the student with a complete picture of the VLSI design flow; give them experience using design tools that are used in industry; introduce them to WDC's 65xx technology and provide an understanding to the value of using reusable technology.

4.2 Recommendations

As previously stated, this tutorial only provides an introduction to VLSI design. If the student has a desire to learn more about VLSI design, it is recommended that the student refer to sources presented in this tutorial. These sources provide great detail for all the aspects of the VLSI design flow.

It is also recommended that the student consult with the staff at The Western Design Center, Inc. Not only can they provide information regarding VLSI design, but they can also assist the student in understanding the concept of design reuse as it applies to this tutorial and WDC's technology.

An internship with a company doing VLSI, SOC design will also be a valuable experience for the student, enabling them to expand their understanding of VLSI design.

Lastly, it is recommended that the student use this tutorial as a reference and continue where it leaves off. For example, there are eight project pins available on the W65C122S. If a student wants to further their knowledge of VLSI, they can develop a circuit that will be placed onto the system chip using the eight available pins. This will enhance what they have learned following the tutorial, and solidify the concept of design reuse.

REFERENCES

1. Bhasker, J. *A VHDL Primer*. Upper Saddle River: 1999.

2. Boyce, David E., R. Jacob Baker, and Harry W. Li. *CMOS – Circuit Design, Layout, and Simulation*. New York: IEEE Press, 1998.

3. Palnitkar, Samir. *Verilog HDL. A Guide to Digital Design and Synthesis.* Upper Saddle River: Sun Soft Press, 1996.

4. Smith, Douglas J. *HDL Chip Design. A Practical Guide for Designing, Synthesizing and Simulating ASICs and FPGAs using VHDL and Verilog.* Madison: Doone Publications, 2000.

5. Smith, Michael John Sebastian. *Application-Specific Integrated Circuits.* New York: Addison-Wesley, 2000.

6. Weste, Neil H. E., and Kamran Eshraghian. *Principles of CMOS VLSI Design, A Systems Perspective (Second Edition).* New York: Addison-Wesley Publishing Company, 1994.

7. Sternheim, Eli, Rajvir Singh, Rajeev Madhava, and Yatin Trivedi. *Digital Design and Synthesis with Verilog HDL*. San Jose: Automata Publishing Company, 1993.

APPENDIX

A1. Contact information for The Western Design Center, Inc.

The Western Design Center, Inc.
2166 E. Brown Rd.
Mesa, AZ 85213
Ph: 480-962-4545
Fax: 480-835-6442
Contact: William D. Mensch Jr.
E-mail: mensch@westerndesigncenter.com
www.westerndesigncenter.com

A2. Information regarding internships at The Western Design Center, Inc.

WDC offers both paid and unpaid internships. The type of internship is determined between the student and WDC. Prior to a student interning at WDC, a project must be defined that is deemed beneficial for both WDC and the student. A Non-Disclosure Agreement must be signed by the student and WDC before any internship can begin. The NDA is required to protect WDC's technology and IP.

Once a project has been defined and the NDA signed, the student and WDC agree on how many hours and what days the student can come to the WDC office to work on the project. An Internship Agreement defining the scope of the internship is signed by WDC and the student. The student is expected to be self sufficient while working on their project, however, the staff at WDC will be available to assist the student when required.

At the end of the internship, the Internship Agreement is terminated and all work done remains the property of WDC. The student is always bound by the NDA. An internship may or may not lead to a full time position at WDC. This must be discussed between the WDC and the student.