Dordt College

Engineering Department

OBJECTIVES:

- To learn about safety in the lab
- To become familiar with breadboard connections
- To learn how to use a logic probe
- To observe some basic gates in real hardware
- To be introduced to a simulator and observe simulated gate behavior

EQUIPMENT NEEDED

- 1 FOX kit microprocessor trainer with matching numbered power supply
- 1 logic probe
- 1 each of the following TTL gates,
  '00, Quad 2-input NAND
  '03, Quad 2 input NAND with open collector outputs
  '04, Hex inverter
  '05, Hex inverter with open collector outputs.
- Computer with Quartus II installed (Web Edition or similar. This handout was developed with version 4.1)

PERSONAL SAFETY IN THE LABORATORY

An electronics lab is not a perfectly innocent place. Eye injury and electrocution are remote but potentially serious risks. It is *never* good enough to think to yourself "*I'll be especially careful.*" Accidents happen in spite of being "especially careful." The word "accident" implies something unexpected. Accidents are not necessarily someone's fault or due to negligence. Sometimes things just wear out or break. In addition to being "especially careful," it is much more important that you *manage risks* to reduce the probability of an accident and, should there be an accident, to reduce the consequences. In this lab we have two specific rules to manage the risk:

1.) Never work alone (when there are no others around).

2.) Wear safety glasses with side shields when appropriate (see below).

Risk Management —Reducing the Probability of an Accident

Certain accidents can be made almost impossible. For example, the risk of getting something sharp or abrasive in your eye can be reduced by wearing safety glasses with side shields. The risk of being burned while soldering can be reduced by wearing gloves, a long-sleeved shirt, and other appropriate clothing. Whenever you start a task, think about the risks, and then think about what can be done to prevent the risk.

Risk-Management —Reducing The Consequences of an Accident

Just in case there is an accident, having the proper help and safety equipment ready can reduce the consequences of an accident. For example one should never work alone because if you are disabled you need someone to help you or to call for outside help. A telephone is located in the lab, while it is used mostly for convenience, it is there for safety. Call 8-911 if it is an emergency, otherwise you can call campus security for help or advice. (To call campus security dial 6414, wait for the tone, then dial 22.) A first aid kit is located on the lab bench nearest to the telephone.

Never undertake an unfamiliar procedure without instruction. Consult with a faculty member first so that you may be informed of potential risks, proper work techniques, and related safety procedures. This statement is especially intended for those who might take personal items, like a TV set or CD player for example, into the lab to repair them. No matter what you do, get advice and permission first before undertaking work that is not assigned in a class.

Eye Injury

Perhaps the greatest risk in an electronics laboratory is the risk of permanently damaging your vision. Seemingly innocuous activities like cutting wire, drilling, filing and such can throw sharp bits of metal from the tool or from the work forcefully into your eye. Imagine the painful and permanent injury that can happen if a sharp but tiny scrap of metal, like a fragment of a razor blade, gets stabbed into your eye and you start rubbing the eye to get the irritating thing out. (Are your eyes watering just because you are reading about this?--Good!) Cutting wires can produce exactly such a scrap of metal and can throw it forcefully. If such a scrap manages to slip behind your eyeball the irritation might go away (there are few nerve endings there) but the injury continues. Copper can dissolve (not much but enough) in your eye and it is poisonous to your retina. Blindness can even result. For these reasons you must. . .

WEAR SAFETY GLASSES WITH SIDE SHIELDS WHENEVER YOU. . .

- . . . cut wire, drill, file, saw, hammer, or use any tool like that. The tool itself or the work can fracture.
- . . . solder. Molten solder splashes just like water. Little wires might spring lose from a connection you are attempting to solder and catapult a droplet of solder up at your eye. There are any other number of ways that solder might splash.
- . . . use electrolytic or tantalum capacitors. These can explode like firecrackers if they fail for some reason.

This list could not possibly be comprehensive. Use safety glasses whenever you are in doubt. Side shields are required because some percentage (30 %?) of eye injuries are not self-caused. You need protection from all angles.

Electric Shock

The low voltages used in assigned laboratories reduce the risk of electric shock, but not to the point where you can be careless. Low voltage does not eliminate the risk of *death* due to electric shock. Current kills, not voltage. Our lab supplies can deliver orders of magnitude more current than a lethal dose. In particular, current through your heart is most deadly. It hardly takes any current through your heart to stop your heart or cause fibrillation. If you get a shock, the amount of current that can flow through your body, and where it flows within your body, depends on many factors besides the voltage. Wet hands, a connection to your metal watchband or ring, and other such conditions can render even low voltages dangerous. For this reason, always remove conductive jewelry, and work with only one hand when the circuit is live. Do not work on live circuits over 30 volts in this lab without permission.

Your Responsibility to Others

Your personal activity could accidentally injure another person in the room. When you undertake safety procedures such as wearing safety glasses, protective clothing, or any other protection, make sure others are also protected similarly. Molten solder and sharp fragments of metal can fly for 20 feet or more.

MINIMUM Safety Related Rules

- Avoid electric shock.
  - Use just one hand when probing live circuits to avoid as much as possible a circuit through your heart.
  - Move probes slowly so that you can watch and avoid what they might touch accidentally.
  - Use tools with insulated handles. Grasp them only by the insulated handle
  - Beware of charged capacitors.
  - Remove rings, watches, and other such jewelry. Put them in your pocket or purse.
- Clean up the bench and turn power off before you leave. A mess invites trouble. Equipment left needlessly powered on is a fire hazard, not to mention the waste of power.
- Large, heavy, or valuable items should not be placed on shelves higher than eye level (about 5 feet).
- Probes, wires, power cords, etc. should not be allowed to dangle over the edge of the workbench. They tend to get caught on belt buckles or other clothing, resulting in damaged connections. The equipment attached might even be dragged over the edge, crashing to the floor on a toe.
- Expect electronic parts to be thermally hot. Parts that are normally cool might get hot due to a wiring error or other problem.
- Do not fiddle with unfamiliar equipment You might mess it up for the next user or even break it.
- If something is damaged or broken, inform your instructor so that it can get fixed.
- Use common sense. Do not try pranks or practical jokes.

This list cannot possibly be comprehensive.

FINDING THE EQUIPMENT YOU NEED

Usually the first thing to do is to round up all the parts and equipment you will need for a lab. Ask for assistance if you need help finding something. Remember where things come from because it is your responsibility to clean up your workbench before you leave for the day. You may want to work with the same equipment, especially the microprocessor trainer, from lab to lab so that you are familiar with any particular foibles of the equipment. In that case, take note of a number on the equipment (the capital asset tag for example) so that you can identify it later.

It is easier to find the proper integrated circuits if you know something about the part numbering system employed. This discussion applies mostly to logic gates, not other types of parts, which are more uniquely numbered. In a given part number such as ’00 the apostrophe stands for a coded prefix that is not necessarily specified. This prefix is a code for the manufacturer, the temperature range, and the logic family.

Here are some examples of part complete numbers with explanations:

- Part number SN7400N is manufactured by Texas Instruments (the SN tells this). It is made to commercial specifications (74), contains standard TTL logic gates (No “LS” or other letters between the “74” and “00”), and specifically it contains four two-input NAND gates (00), in a plastic package (N). The logic in this gate can be specified as ’00 because no matter which manufacturer, logic family, or package type, all ’00 parts are a package of four two-input NAND gates.
- The DM74LS157N was manufactured by National Semiconductor (DM) to commercial specifications (74), it belongs to the low-power Schottky family (LS), the logic is quad 2-line to 1-line multiplexers (157), and it is in a plastic package (N). This gate can be used if a ’157 is specified. All ’157 parts are quad 2-line to 1-line multiplexers. (In 1997 National Semiconductor sold this line of products to Fairchild Semiconductor. They continue to manufacture the parts and maintain the web pages, now at http://www.fairchildsemi.com.)

The full part number optionally starts with one or two alphabetic characters to code the manufacturer. For example DM is the code used by National Semiconductor, SN is used by Texas Instruments, and so forth.

Following the alphabetic characters are two digits, either 54 or 74. (Or these numbers start the part number if the manufacturer’s code is absent.) Parts that are built to operate to military specifications are marked 54. These parts are specified to operate properly with case temperatures between -55 and 125 C. Parts marked 74 are built to “commercial” specifications and cost less. The temperature range for these parts is 0 to 70 C.

After the 54 or 74 there usually are a few alphabetic characters that specify the logic family by a code. Typical codes are S for Schottky, LS for low power Schottky, AS for advanced Schottky, ALS for advanced low power Schottky, F for fast Schottky, H for high voltage Schottky, C for CMOS, and HC for high-speed CMOS. If there is no alphabetic character after the “74” or “54,” then the part belongs to the standard family of TTL logic gates. (regular specifications, no adjective like low power, fast, etc.) In most cases, all these logic families are interchangeable except for the “C” and “HC” parts. For the purposes of this course, <u>DO NOT USE “C” OR “HC” PARTS</u>. There are enough differences in these parts to cause problems in your circuits. We also keep the “C” and “HC” parts in separate drawers, but sometimes they get mixed up, so be on the lookout and avoid these.

After the family identifier (S, LS, ALS, HC, etc.) comes the number that identifies the type of gate. This is the number usually specified as ’00 or ’157, etc. The details of these codes can be looked up in a data book or via the World Wide Web. Try http://www.ti.com or http://www.fairchildsemi.com. Look for links to “Products” and then to “Logic.” If you know the part number and want a datasheet, enter a part number into a “part number search” dialog box. Suppose you know you want a type ’00 gate. Fairchild makes the “F” (for “fast”) and “ALS” (for “Advanced Low-Power Schottky”) families, so try part numbers like 74F00 or 74ALS00 in their part number searches. Texas Instruments makes the standard and “LS” (“low-power Schottky”) families, so try part numbers like 7400 and 74LS00 in their part number searches. If you know you want a NAND gate with two inputs but do not know the part number, follow the links for a “parametric search.” Then narrow the search down by specifying as much as you can, for example, you want a NAND gate with 5 volt power supply, TTL inputs and outputs, etc. Alternatively, you can look in a hardcopy data book. (The few data books we have left are getting dog-eared, but sometimes they are faster to use than searching the web.) You will need the pin-out diagram from that data sheet. The pin-out diagram illustrates how the pins of the part are connected to the internal gates.

After the gate identification is the package identification. Examples are N, P and PC which all specify that the package is made of plastic, and D or J which specify that the package is made of a ceramic material.

Most parts are also marked with a date code such as 9327. This example represents a part made in the 27th week of 1993. Watch out for date codes of old parts that might look like part numbers. For example a part with 7407 and SN7400N marked on it is a quad 2-input NAND package, ’00, made in the 7th week of 1974. If a ’07 part is needed then a “7407 SN7400N” is not the right part to use.

THE LOGIC PROBE

Perhaps the most frequently used diagnostic tool for the exercises assigned in this course will be the logic probe. You can use it to quickly determine if a signal is logic 1 or 0 or simply not connected. You can also use it to test power and ground connections since power should test as logic 1 and ground as logic 0.

Power up your logic probe by clipping its power leads to two short jumper wires and plugging the jumper wires into the correct breadboard holes. The red alligator clip gets connected to +5 volts and the black one to GND on the microprocessor trainer. On the logic probe, set the switches to “TTL” and “PULSE.”

When you touch the tip of the logic probe to a point in the circuit, it tests the voltage present. If the voltage is between 0 and 0.8 volts the LO light comes on. If the voltage is between 2.0 and 5.0 volts, the HI light comes on. If the voltage is between 0.8 and 2.0 volts, or if the probe tip is not connected to anything, neither light comes on. Try touching your logic probe to power and ground to observe the different indicator lights in operation.

Sometimes a logic signal is rapidly changing back and forth between one and zero. In this case, the PULSE light comes on. On some types of logic probes the PULSE light comes on and stays on and on other types it blinks. If yours blinks, keep in mind that the rate of blinking has nothing to do with the rate at which the signal is changing. Some logic probes also make audible beeps of different tones so that you do not have to watch the lights. Try touching the logic probe to the signal marked /CLOCK in the very top area of the breadboard. This signal is cycling between 1 and 0 at better than 1 MHz. Observe how your logic probe shows that the signal is not a constant 1 or 0.

FIRST CONNECT AND TEST POWER AND GROUND

When building a circuit on a breadboard, it is a good idea to plug all the chips into the breadboard and wire all power and ground connections first, before wiring logic signals. Just one incorrect or missing power or ground connection can cause bizarre and hard to diagnose problems, possibly even causing smoke to rise! After the power and ground wiring is done and before you start to wire the logic signals, power the breadboard up. Use the logic probe to verify that each power pin is logic 1 and each ground pin is logic 0. Then turn the power off and proceed with the logic wiring. This way the logic wiring, which needs to be easily accessible, ends up above the power and ground wiring. A practical way to switch the power to your circuit on or off is to connect or disconnect the power plug from the Fox kit while leaving the wall wart plugged into the power line.

The breadboard area of the microprocessor trainer is designed to accept small wires and the pins of most integrated circuits. The amount of force you would use if you were inserting a wire or part by hand is the most force you should ever apply, even if you are actually using a pliers. If a part will not insert easily, be sure it is aimed properly into the hole and the lead is not kinked or bent. If the wire or lead is too big to fit through the opening in the breadboard, do not force it. If you do, you will damage the metal contacts in the breadboard. When you insert a whole chip in the board, look carefully to see that all the pins are going in their proper holes and that one or two pins are not buckling or folding under the chip.

Figure 1 illustrates typical power and ground connections. The figure shows one logic chip connected to power and ground. Observe that +5 volts is usually connected to the top left pin of the chip and supplied from one of two rows at the top of the main section of the breadboard. Likewise, ground is typically connected to the lower right corner of the chip and supplied from a row at the bottom. Also make note of the jumpers in the middle of the top and bottom rows used for power. *Without those jumpers, the right half of the breadboard will get no power or ground.* The location of the power pin on the upper left and the ground on the lower right of the chip is typical, but check the data sheet—there are a few exceptions.

Avoid placing wires over top of the integrated circuit chips. It sometimes is helpful to pull a chip out and replace it (if the chip was defective for example). This is easy if there are no wires placed over top of the chip. To pull integrated circuits out of a breadboard, use a special tool called a chip puller. Alternatively, a small screwdriver can be used to gently pry the chip up from either side. Those who insist on using their fingers for this task are likely to fumble and end up with the leads of a chip embedded about 1/8 of an inch underneath a fingernail sometime during the semester. Besides which, using fingers to pull chips tends to bend and weaken the pins so that the next

person who uses the chip finds a pin that buckles or folds upon insertion. Chips do not heal like fingers do. A chip puller is conveniently provided in your lab bench drawer.

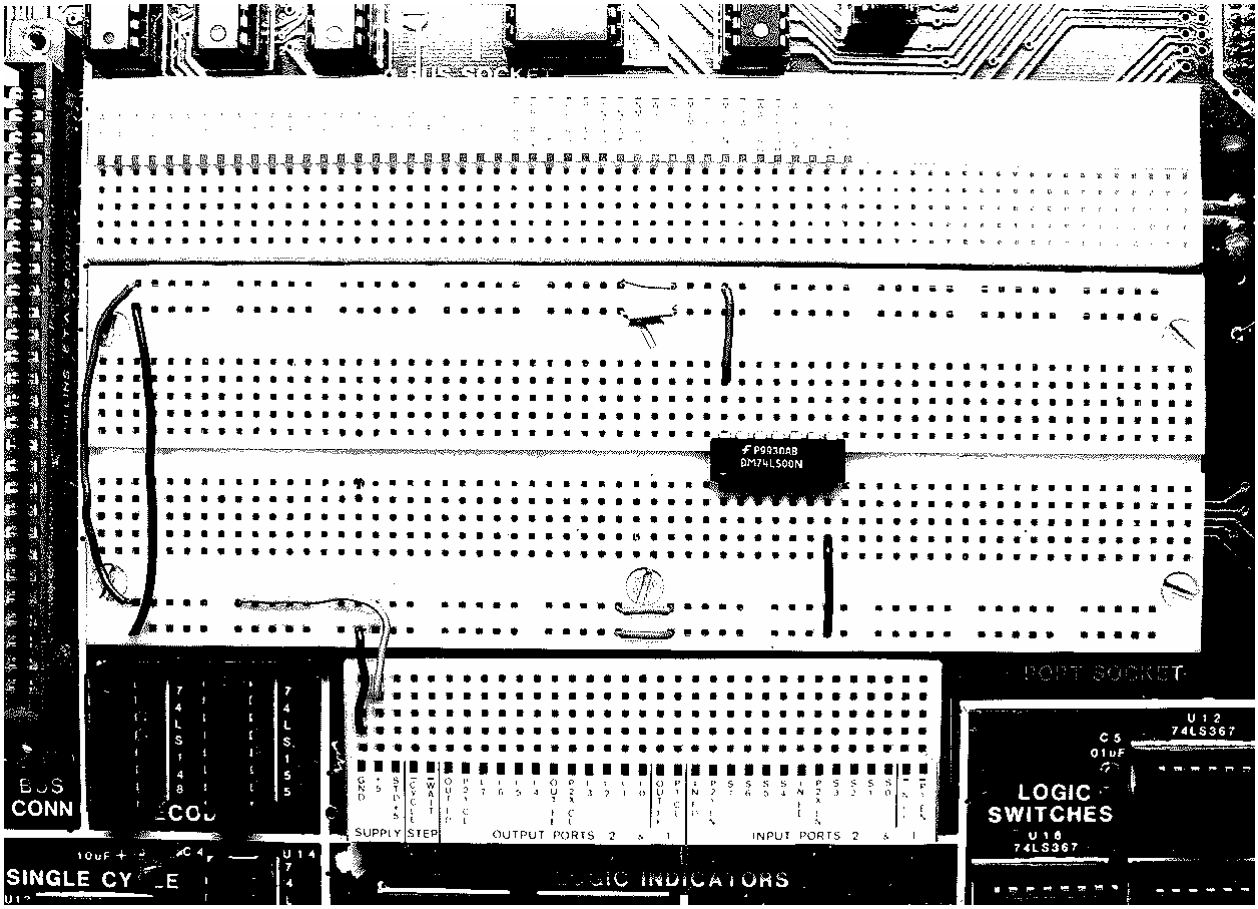

Figure 1. Typical power and ground connections.

THE PORTS OF THE MICROPROCESSOR TRAINER

Source of Logic Signals, PORT 2 Switches

Near the center of the bottom edge of the FOX kit, below the breadboard area but on the green circuit board area, are 16 small switches in two groups of 8. One set of 8 switches is labeled PORT 2 and each switch is numbered 0 through 7 in white on the green background of the printed circuit board. (There may be markings on the body of the switch assembly itself. Ignore these.) These eight switches are provided to generate switch controlled logic values of 1 or 0, which you may use by connecting them to gates in the breadboard area. In order for these switches to work, you need to be sure that nothing is connected to P2X EN and P2Y EN. Touch the logic probe gently into one of the S0 contact holes and turn the number 0 switch on port 2. The logic probe should show high when the switch is rocked toward the top and low when rocked to ward the bottom. You may use the point of a pen, pencil, or small screwdriver to turn the switches. Check the other switches on port 2 to see if they all work.

Indication of Logic Signals, PORT 2 LED's

Along the bottom of the FOX kit, there are 16 red light emitting diodes (LED's). Eight are marked PORT 2 and they can be used to monitor the logic value of signals in the breadboard area. A signal is connected to an LED via the L0 through L7 contacts of PORT 2 near the bottom of the FOX kit. P2X CL and P2Y CL must have no connections to them for this to work. Connect a wire from S0 to L0 and demonstrate that PORT 2 switch 0 now controls PORT 2 LED 0. Now change the connection so that S3 is connected to L7. What happens?

The port 2 LED's do not give as much information as the logic probe. What happens when a PORT 2 LED is connected to GND and then disconnected? What happens when the tip of the logic probe is connected to GND and then disconnected?

What does the PORT 2 LED fail to show? _____

_____

THE BASIC GATE OF THE TTL FAMILY: NAND

The basic gate of the Transistor-Transistor Logic (TTL) is the NAND gate. An inverter is manufactured as a NAND gate with one input omitted (which is equivalent to connecting the omitted input to logic 1 in this case). The object of the remainder of this lab is to observe the actions of NAND gates in real hardware and in a simulation. If you have not already done so, use the World Widw Web to find a data sheet for the type ’00 gate.

’00

Always do wiring (or modify wiring) with the wall-wart disconnected from the Fox kit. (Incomplete or incorrect wiring in the breadboard area can erase software normally stored in the Fox kit. It can be restored, but it wastes time.) With the microprocessor trainer unplugged from its wall-wart power pack, plug a ’00 chip into your breadboard so that it straddles the center groove. Connect power and ground jumper wires. Avoid placing wires over top of the chip so that it can be removed and replaced without disturbing the wiring. Test the power supply connections by plugging the wall-wart into AC line power and into the fox kit, connecting the logic probe to power, and probing the power and ground pins. If both test correctly unplug the wall-wart from the fox kit and proceed, otherwise debug your wiring.

Now choose one of the four NAND gates in the chip and connect its two inputs to S1 and S0. Connect the output to L0. Apply power.

By counting in binary (00, 01, 10, 11) on the PORT 2 switches 1 and 0, try all four combinations of the inputs and observe the corresponding output of the logic gate on LED 0 of PORT 2. Make a truth table of when the LED is on and when it is off. Repeat the experiment but this time hold the logic probe on the output pin of the gate you are using and observe the output on the logic probe. Make notes of when HI is on and when LO is on and if neither is on for some switch settings.

S1	S0	L0
0	0	
1	0	
0	1	
1	1	

S1	S0	Probe Indication
0	0	
1	0	
0	1	
1	1	

Figure 2. Truth tables for the type ’00 NAND gate.

'03

Unplug the wall-wart from the Fox kit, carefully remove the '00 chip and replace it with a '03 chip without otherwise changing the wiring. The '00 and '03 are the same except that the '03 has an open collector output. Repeat the experiments done for the '00 chip and note the differences.

S1	S0	L0
0	0	
1	0	
0	1	
1	1	

S1	S0	Probe Indication
0	0	
1	0	
0	1	
1	1	

Figure 3. Truth tables for the type '03 NAND gate.

Power the circuit down, and add a 1 kΩ resistor from the output of the '03 gate to +5 volts. This resistor is called a "pull-up" resistor and it supplies a weak logic 1 signal at all times. The type '03 gate can overpower the weak logic 1 with a strong logic 0. Power up again and repeat the experiment. Compare the results to the previous experiments.

S1	S0	L0
0	0	
1	0	
0	1	
1	1	

S1	S0	Probe Indication
0	0	
1	0	
0	1	
1	1	

Figure 4. Truth tables for the type '03 NAND gate with pull-up resistor.

'04 and '05

Power down and remove the '03 chip and all the wiring. Then design your own wiring and experimental procedure to investigate the '04 and '05 chips in a style similar to what you did for the '00 and '03 chips.

Unused Inputs

Before removing the '05 chip, disconnect the input (leave the power on for this—it will be OK.). What happens to the output? Unused inputs are said to be *floating*. In the TTL logic family all floating inputs tend to default to logic 1. Did you observe that? Sometimes in the lab it is convenient to ignore an unused input because you can usually rely on the default logical value of a floating input. However, in a production product it is unwise to leave inputs floating because they are susceptible to noise and the default logical input will not be reliable. In the CMOS logic families unused inputs do not have a predictable default. They act like either 1 or 0 depending on manufacturing variations and other factors.

SIMULATION VIA QUARTUS II

Overview of logic simulation

Complicated logic circuits are most easily designed using a simulator rather than a breadboard. The simple circuits we will design in this and the next few weeks do not really need simulation, however they make excellent material for learning how the simulator works. Then, when you need to simulate more complicated circuits before building them, you will better realize how the simulator relates to the hardware to be built.

There are two types of logic simulations available. Most engineers first use a *functional simulation* to verify the logical correctness of a design. This type of simulation only verifies that the logic is set up as intended. It ignores many details of the actual hardware, such as how fast the signals will propagate through the electronics and what the loading effects might be. After the functional simulation is working, a *timing simulation* can be done to test the remaining details. Usually a timing simulation is not attempted until the functional simulation works as expected. This is because a timing simulation is invalid if the functional logic is not correct. Although a timing simulation can reveal functional errors, wading through a long list of irrelevant error messages and other oddities to

find the underlying functional errors is tedious. For the next several weeks we will be studying logic at the functional level. Later in the course we will learn about timing simulations.

The simulator needs a description of the logic to be simulated. This can be given as a schematic diagram (a drawing using symbols for the gates and lines for the wires). The description of the logic can also be given in other forms such as a truth table, an equation, or a hardware description language. For now we will use schematics since it is easier to understand the simulator this way. Modern circuit designs with millions of gates are now much too complicated to draw in schematic diagram form. The schematic would have to be as big as the floor of an average classroom to draw all the millions of gates at a scale that could be seen without using a magnifying glass. For this reason, hardware description languages are also important. A hardware description language is text-based computer code that describes how the logic should be arranged for your circuit. Schematics may still be used for small detailed sections of a circuit, but increasingly, a hardware description language is used instead. We will use primarily schematics in this course, but later we will study the basics of a hardware description language too.

Real logic circuits have real signal inputs and outputs, such as inputs from switches, keyboards, sensors, etc. (e.g. we use S0–S7 of Port 2 on the Fox kit for switch-controlled inputs). There are also outputs to relays, motors, light emitting diodes (LED's, e.g. L0–L7 on the Fox kit), etc. Simulators use special symbols on the schematic to represent inputs and outputs. (The symbols look something like arrows.) Since there are no real switches, keyboards, sensors, etc. for the simulator to connect to, the inputs need to be described somehow. Likewise, since there are no real outputs for the simulator to connect to, there needs to be some way for the simulator to present the output results to the user of the simulator. *Waveform Vector Files* are used for these purposes. With a waveform vector file you can describe a sequence of logical values for input to the simulator and the simulator will show you the resulting sequence of output logic values.

As you work on simulating a logic design you will invariably create many computer files all associated with that one design. This set of files is called a *project*. Logic simulators can keep all project files associated with each other for you. You can assist the process by keeping all the related files in one folder on your computer system.

To begin work on a new simulation project, open Windows Explorer and browse to a drive and folder where you would like to keep your simulation project. (A shortcut to start Windows Explorer: Hold down the "Windows" key while you press the "e" key. Note: Windows explorer is not the same as Internet Explorer.) For example, you might call this first project that you do today "TUTORIAL1." In this lab project we will simulate a simple NAND gate. Create a folder something like this for today's work:

g:\courses\204\lab1\TUTORIAL1

(Note: Certain project names such as "NAND" and "XOR" cause problems since they correspond to filenames used by Quartus II for "primitives." Do not name your project "NAND.")

Starting the Quartus II Software

As with every windows program, there are many ways to start it. To use the Start menu the typical mouse sequence is to click on "Start | Altera | Quartus II" (Or Quartus II Web Edition Lite, etc.) Alternatively, if there is a desktop icon for Quartus II you could double-click it to start Quartus II. Either of these methods is a good way to start Quartus II when you want to create a new project. If you are just getting started, start Quartus II via one of the above methods.

The first time you start Quartus II you should use the New Project Wizard to create a new project ("File | New Project Wizard. . ."). A "New Project Introduction" window might open. If so, read it, then click next. (If it does not open, it is copied below in Figure 5 for your convenience. Someone selected the "don't show" box.) The New Project Wizard will ask, "What is the working directory for this project?" Use the browse button (three dots on it) to select the folder you created for this project, for example:

g:\courses\204\lab1\TUTORIAL1

It is best to browse to the folder rather than type it into the wizard so that you avoid silly typing errors. The Wizard will also ask, "What is the name of this project?" It is usually least confusing if you name the project the same as the folder you created for it, for example, "TUTORIAL1." (The wizard might already have sensibly filled this answer in with "TUTORIAL1." In that case, just let it be.)

The Wizard will ask, "What is the name of the top level entity?" In our case the top level entity will be the schematic, which again is best named the same as the folder, for example, "TUTORIAL1." (The wizard might already have sensibly filled this answer in with "TUTORIAL1." In that case, just let it be.)

When these questions are answered, click "Finish." (You could click "Next," but then take all the defaults right to the end of the wizard.) The new project wizard will then create the necessary project files in the folder you specified. It is usually best to create a separate folder for each project otherwise one project might accidentally use or modify a file from the other project. (Sometimes advanced users do not separate projects into separate folders because they actually want a new project to use files from an older project, but this can be very confusing for a novice.)

Once you have created a project, if you want to re-start Quartus II and work on the project, a better way to start Quartus II is to double-click on the project file name in Windows Explorer. Alternatively, you can start Quartus II as for a new project, but then instead of using the New Project Wizard, from the file menu select "File | Open Project. . ." and browse over to the project file and double-click it. The project file to double-click always has the stylized "blue Q" icon (for Quartus) and the file name ends with the extension *.qpf for ("Quartus project file"). In our example, the project file would be named TUTORIAL1.qpf. It is possible to use "File | Open. . ." to open individual files and edit them outside of the project, but then the simulator probably will not work. It will not be able to find all the files associated with the project.

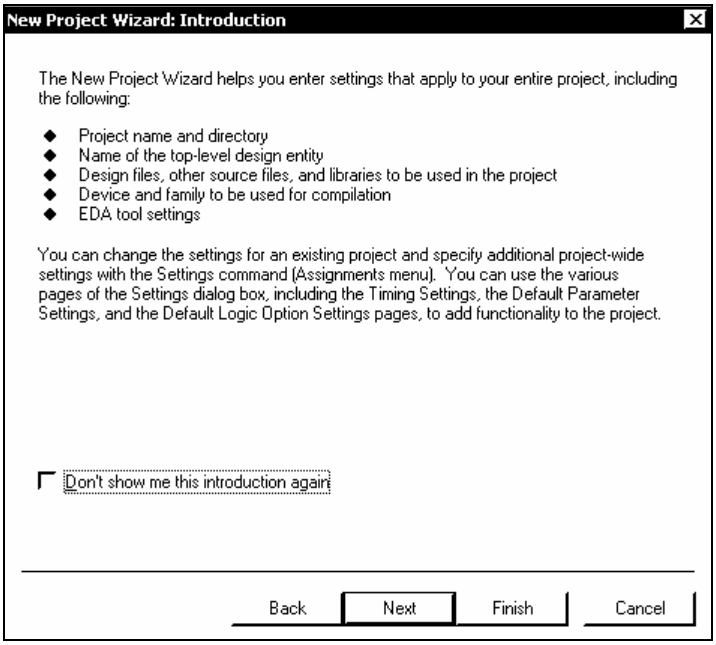

Figure 5. The New Project Wizard's Introductory dialog box.

Creating a Schematic Diagram for Simulation

To start drawing a new schematic diagram in a project, Quartus II must be open and a project must be open within Quartus II (or just created with the New Project Wizard). Then, to start the schematic editor, from the file menu select "File | New. . ." and then select the "Device Design Files" tab and then "Block Diagram/Schematic File." Click "OK." A new window opens in which you will draw the circuit diagram. This window now needs to be named and associated with the project. The easiest way to do this is just to use "File | Save As. . ." and accept all the default answers—except look to be sure "Add to current project" has a check mark. Then click "OK." This process will create a *.bdf (block diagram file) in the project in which the schematic will be stored. (It is an annoyance that "File | Save" does not work for this until you have first done a "File | Save As. . .")

The text below describes how to draw the schematic shown in Figure 6. As you proceed with your work, it is a good idea to occasionally click on the "File Save" icon so that if there is a system crash, power flicker, or similar problem, you will not lose all your work. If ever you are about to embark on some edits you are unsure of and you think maybe you would like to be able to undo all the edits you are about to make, use "File Save As. . ." to

change the name of the file. Then later you can go back to the original filename if you want since all the edits will go into the new file only.

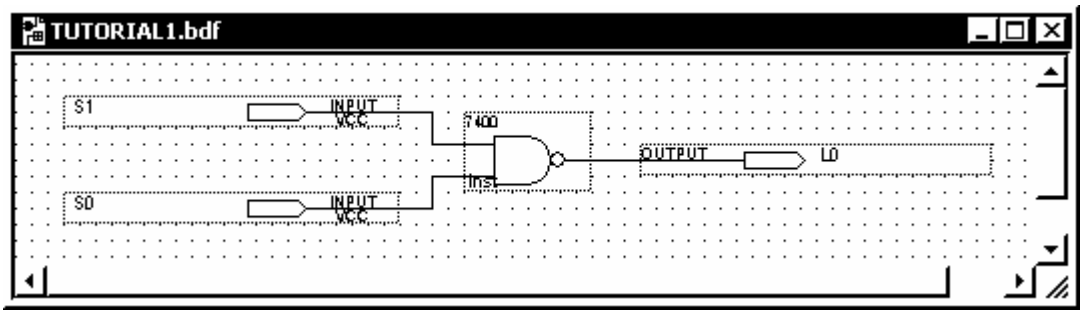

Figure 6. The schematic drawing needed to simulate a type 7400 NAND gate.

Logic gates can be added to the schematic as "symbols" which you select from a library. In the next step you will add a type 7400 NAND gate from the "maxplus2" library to your schematic. In future lab projects, always use symbols from the "maxplus2" library with names that start with "74" for logic gates, otherwise the drawing you make will not model parts that are available in our lab. Also, check to be sure we actually have the part in our lab stock. There are some parts in the maxplus2 library that we do not stock, and likewise, there are parts in our lab stock that are not included in any Quartus II library.

From the edit menu, select "Edit | Insert Symbol. . ." In the dialog box that opens up, in the "Libraries:" window, drill down to "<drive:/filepath>/libraries/others/maxplus2/7400." The symbol for the type 7400 NAND gate will then be shown in the dialog box. Click "OK" and the symbol will be attached to the mouse cursor in the block diagram/schematic file window. Maneuver the symbol to a place near the middle of the window and click once to place it there. You could proceed to place more type 7400 NAND gates with additional clicks. Instead, escape the insertion process by pressing the <ESC> key on the keyboard. There are at least two other ways to escape the insertion process. You may select (click on) the "Selection and Smart Drawing Tool" which is an icon with an arrow on it that looks like a normal mouse cursor. You may also right-click anywhere in the schematic window and then select "Cancel." If the symbol for the logic gate did not get placed exactly where you wanted it, after escaping the insertion process you can move the symbol by dragging it with the mouse. You can delete it by selecting it (click once on it) and pressing the delete key or by right-clicking it and selecting delete. Right-clicking it will also give you options to rotate and flip the symbol, should you want to do that. (Not necessary at this time.)

If you double-click on a non-text area inside a symbol then a new window opens to show you what is in the symbol—that is, all the simpler parts, gates, and functions that were used to build the symbol. If there are more symbols in that window, you can double-click the symbols in that window too. Continuing that process, eventually you will get to what are known as *primitive* symbols which are only described by a text window which gives basic information, such as the name of the primitive. A detailed description of each primitive symbol can be found in the help system under "Help | Contents Tab | Devices and Adapters | Primitives." Primitives are what the library symbols are built from. They are native to the simulator, whereas libraries of symbols can be made and augmented by the user, if desired. Fortunately, most simulators come with very adequate libraries so that new symbols do not usually need to be added to them for most simulation projects. (The descriptions for some symbols are encrypted so that you cannot resell that intellectual property, and some symbols are described with a hardware description language, rather than a schematic. You will not run into these cases in the "74. . ." series of symbols.)

You also need to show the simulator where the inputs and outputs are in the schematic. This is done with *pins* named "input" and "output." They are just primitive symbols. To simulate connecting the inputs of the NAND gate to two switches called, for example, S1 and S0, insert two input pin symbols, one for each input of the NAND gate. Do this by selecting "Edit | Insert Symbol. . ." and then in the "Libraries:" window drill down to "<drive:/filepath>/libraries/primitives/pin/input." Then click OK and place an input pin near the left side of the drawing window. Add a second such pin below the first before escaping the insertion process (<ESC> key). Each insertion will have some default "pin_name" label that you will need to edit to give the inputs recognizable names. Change the top "pin_name" to "S1" by slow-double-clicking on the label and then editing it. To end the edit, press <Enter> on the keyboard. Change the bottom "pin_name" to "S0." In a similar fashion, insert an output pin on the right side of the schematic and label it "L0." (In some rare cases the text is too small to slow-double-click with the mouse. In that case, right-click on the pin, choose "Properties," and edit the signal name in the dialog box that will open. Click "OK" to return to the schematic.)

Now all the parts of the schematic need to be connected together with lines that symbolize wires. In Quartus II these lines, or wires, are called *nodes*. An *Orthogonal Node* is one that will be drawn only with horizontal and vertical lines. When you see the word *node*, just think *wire*. By default, when you start dragging the mouse cursor from a connection on a symbol to any other point, Quartus II will start drawing an orthogonal node. Drag a wire from the output of the "S1" pin to a point that is about two dots to the left of the NAND gate and at the same vertical level as the top input to the NAND gate. Let up on the mouse button to anchor this point of the wire, and then without moving the mouse, click down and only then continue dragging the wire in a straight horizontal path to the input of the NAND gate. You may click anywhere in a blank area of the drawing window to deselect the last wire drawn. In a similar way, connect the S0 pin to the other input of the NAND gate and connect the output of the NAND gate to the L0 output pin. At this point your schematic should look similar to Figure 6. Click on the save icon to save the schematic in your project. (Or choose "File | Save" or using the keyboard, <ALT-F>, <S>.)

A Tip: In Quartus II, if there is an asterisk ("*") following a file name in a window title bar, it means there are unsaved edits in that window.

Generating a Netlist for Functional Simulation

Besides saving the schematic diagram (*.bdf file) you need to translate it into a form that can be directly used by the simulator. This is called "generating the netlist."

To generate the functional simulation netlist from the schematic, first make sure the schematic is open and selected as the active window. (The window is active if the window's title bar is highlighted. Click on a blank area of a window to make it active.) Then, from the Processing menu click on "Processing | Generate Functional Simulation Netlist." If you are asked if you want to save changes to your schematic, answer "Yes." This way the netlist is sure to match the schematic as shown on the screen. (If you select "No" the netlist will be generated from the schematic file—not necessarily the same as the schematic shown on the screen.) After the generation of the netlist is finished you should get a message to the effect that the generation of the functional simulation netlist was successful with no errors or warnings.

After you have generated a netlist, if you go back and edit the schematic it is not good enough just to save the edited schematic. You must also generate the netlist again. Quartus II requires all these steps so that it can accommodate alternatives to drawing a schematic, such as using a hardware design language—even if you are not using any alternatives in your specific project.

Specifying How the Inputs Behave

The simulator will need to know the simulated logical values of the inputs. This is done with a "Vector Waveform File." To create such a file, click on "File | New. . . | Other Files (tab) | Vector Waveform File." As soon as this window opens, it is wise to name it and associate it with the project. Do this with "File | Save As. . ." ("File | Save" will not properly name and associate the file until you have first done "File Save As. . .") Usually you will want to give the file the same name as the project and the schematic ("TUTORIAL1" in this example) and be sure that the "Add file to current project" box has a check mark. Then click "OK." In the following steps you will describe for the simulator the input signals and how to display the output. The result will look like Figure 7.

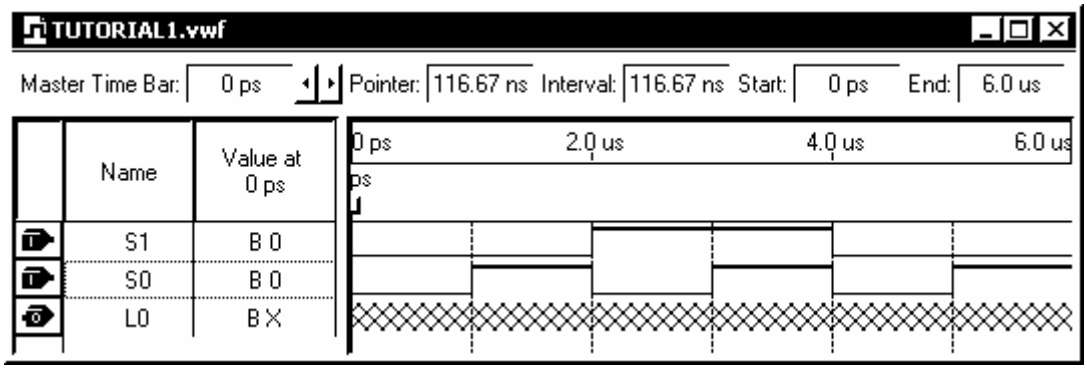

Figure 7. The finished vector waveform file used to describe inputs and outputs

Start building this file by specifying the end time and grid size for the time axis (shown along the top of the graph). Click on "Edit | End Time" and set it to 6 us. Then click on "Edit | Grid Size" and set it to 2 us with 50% duty. The result will be that the waveforms will be divided into 6 segments, each 1 us long. (Each segment is a

duty fraction of the grid size. For reasons having to do with clock signals, a duty fraction of 50% is convenient and conventional for most situations.) Now make the time axis fit the window. Click "View | Fit in window." If you later resize the window, you might want to repeat that.

Now add to the vector waveform window the names of the signals that are inputs on the schematic. It is important that the names you use in this waveform editor exactly match, in spelling and capitalization, the names of the inputs on your schematic. Also watch out for zeros that might look like capital letter "oh" (0 and O) and the number one which might be confused with a lower-case "ell" (1 and l). With the vector waveform editor window active, click on "Edit | Insert Node or Bus. . ." In the dialog box fill in the name of the input you want listed on top in your final vector waveform file. For example, type "S1" for the name. Make sure the "Type" is "INPUT." Take the defaults for all the other options and click "OK." You have inserted a "bus of width 1," which means the same thing as a "signal." Repeat for the "S0" signal.

You will also need to tell the simulator how to report the output. In this case, the output will be displayed as a waveform below the inputs. Click on "Edit | Insert Node or Bus. . ." again and fill "L0" in as the "Name," but this time change the type to "OUTPUT." Take all the other defaults and click "OK." The graph for L0 fills with "X" marks, which means that the logical value of the signal is unknown since the simulator has not yet been run to compute it.

Now you are ready to add the waveforms for the inputs. On the S1 signal's trace, drag the mouse from about 2 to 4 us. A segment from exactly 2 to 4 us should be selected (highlighted). Along the left side of the waveform window, look for the logic 1 icon. It has a small pulse on it and the number "1." Click that and the waveform in the selected area will become logic one (high). Repeat for the S0 segments from 1 to 2 us. Also set S0 high from 3 to 4 and 5 to 6 us. (Do not add a waveform for the L0 output. The simulator will compute that waveform.) You can change selected areas of high waveforms to low with the logic 0 icon, should there be that need.

When you are done editing the waveforms, save the file by clicking on the "Save File" icon.

The waveform editor has a timing cursor which you can drag around with the mouse. It looks like a vertical blue line through the waveforms with a small grey square *handle* at the top. Often, when you create a waveform file the timing cursor and its handle are only half-way visible because they are positioned up against either the left or right edge of the waveforms. In Figure 7 the timing cursor is jammed up against the left side. The handle is located just under a label "ps." Find it and drag it by its handle to about 1 us. Note that it snaps to the timing grid you specified earlier. Look in the "Value at. . ." column to the left of the waveforms and observe that the logic values shown there match the logic levels just to the right of the timing cursor. (The timing cursor is "forward looking.") Drag the timing cursor to some other positions and observe how it behaves and how the logic values listed in the "Values" column follow the timing cursor when you release the mouse button (not while you are dragging). Finally, observe that the waveforms shown in Figure 7, and hopefully in your waveform file, follow the rows of truth tables shown in Figure 2.

Running the Simulation

Before running the simulation it needs to be set up to do a *functional* simulation and it needs to be told where the waveforms are stored. (In more complicated designs it is common to have many vector waveform files, each of which is set up to drive the simulated logic through a different scenario. Thus, it is not practical to select a vector waveform file by default.) To make the settings, click on "Assignments | Settings" Then in the "Category" window click on "Simulator." Change the "Simulation mode" to "Functional." Use the browse feature (small icon with three dots on it) near the "Simulation input" box to select the vector waveform file (TUTORIAL1.vwf). Accept all the other defaults (Run until all vectors are used, Automatically add pins, Simulation Coverage Reporting) and select "OK."

Now you are ready to run the simulator. Click on "Processing | Start Simulation." If any "Save changes" dialog pops up, it is safest to answer "Yes." You should get a message indicating success with no errors or warnings. If any errors or warnings have anything to do with "timing," you forgot to change the simulator setting from "timing" to "functional."

When the simulator finishes, a new window opens showing a "Simulation Report." It includes a new vector waveform display with the output L0—formerly shown with X marks—now shown as logic 1 and logic 0 levels. Notice that by sliding the timing cursor along you can read rows of the truth table in the "Values" column.

CONCLUSION

In this lab exercise you have learned how to find a data sheet for a logic gate and wire the gate up. You used switches for control of the input signal logic levels and a LED or a logic probe to observe the output.

Two types of real logic gates were investigated: standard output and open collector output. The open collector type of logic gate cannot generate its own logic-1 value. A *pull-up* resistor needs to be used along with the gate to get normal functionality. Open collector gates are used in certain situations when more than one output might accidentally be driven at the same time. This can happen for example in a computer when there is an I/O address conflict due to an incorrect software or hardware installation.

The logic probe provides more information than a simple LED. A probe can indicate an open connection.

You also learned how to use the Quartus II logic simulator to predict the behavior of real logic gates. You have gone through all the steps of setting up a functional simulation and observing the results.

A trivially simple logic circuit (a single NAND gate) was studied so that you could better become familiar with the equipment used to wire, power up, control, monitor, and simulate logic circuits.