Exp. 1: FAMILIARIZATION WITH THE DIGITAL TRAINER

"A journey of a thousand miles begins with a single step."
-a Chinese proverb

I. OBJECTIVES

1. To become familiar with the "digital trainer" unit.
2. To be acquainted with general circuit assembly techniques.

II. BASIC ELECTRICITY AND SAFETY

Digital circuits are made of electronic components and need electrical energy to operate. Two basic quantities of electricity are current and voltage. Current exists when there are moving charged particles (electrons). It is the equivalent of the water flow rate, in a water pipe system. Current is measured by how many charged particles move past a given point in one second. This flow rate is measured in amperes (or simply A). An ampere is a relatively large quantity for microelectronic circuits; we generally talk about currents in milliamperes (mA, one thousandth of an ampere) or microamperes (uA, one millionth of an ampere).

Voltage, on the other hand, represents the potential difference between two points. It is like the pressure difference in a water pipe system. The voltage is what makes the current flow, just as the pressure difference is makes the water flow. The unit for voltage is volt (or simply. V). +5 V has been the most widely used voltage source in digital circuits although the trend now is to use lower values such as +3.3 V or +1.5 V in order to conserve energy. The product of voltage and current is the power and is measured in watts (W). In order for a circuit to do anything, it must have a power supply (such as a battery) to serve as the source for both current and voltage. Electrical energy is measured in watt-hours (Wh), which is power consumed or supplied over a time period. A 6 V battery with a rating of 2 Ah(ampere-hour) has 12 Wh of stored energy (i.e., it can keep a 12W light bulb on for one hour).

Since voltage is the potential difference between two points in a circuit, it is easier if we have a common reference point for the entire circuit. This common reference point is known as the "ground," Thus, when we say a certain part of a circuit is at +4 V, we know that it means the potential difference between this point and the ground is +4 V. A power supply always provides two terminals: one for the "voltage" and the other for the "ground," For most applications, the "voltage" side is marked "+" while the "ground" side is marked "-" or "GND." For digital circuits, the voltage source is often referred to as V_{CC} (or V_{DD}).

Wires are used to connect electrical components into circuits. There are many different types of wires; the ones you'll use in this lab are composed of a single, solid conductor inside a plastic insulation tube. The insulation prevents the conductor from being exposed but must be removed at each end when making connections. Wires can conduct current (allow current to flow) with negligible difference between the two ends; hence wires are used to bring two circuit points together to the same electrical potential. When two points are connected by a piece of wire they have essentially the same voltage (with respect to the ground); these two points are said to be shorted. Never "short" any two circuit points that you don't intend to because it can damage circuit components and/or equipment that is connected to the circuit.

This laboratory uses a low voltage source so the chances of getting an electric shock are very remote. However, it only takes a very small amount of current to pass through a body to cause injury. Always exercise caution when working with electrical apparatus - turn the power off when making or changing circuit connections, keep hands dry, use insulated wires, don't touch bare, "live" wires with bare hands, never bypass or tamper with safety features of any equipment, etc. Use hand tools, such as a wire cutter, carefully so as not to cause injury to yourself or to others. It is important to develop good, safe laboratory habits.

III. THE DIGITAL TRAINER

The basic apparatus needed for hands-on digital hardware work is a "digital trainer." Some of the essential elements of the unit are outlined below.

- +5 V Power Supply. The unit should contain a +5V, 1 A regulated DC power supply. It should have a power on-off switch and a power-on indicator. To protect the power supply from an accidental short, the unit should also include a short-circuit protection feature and a short-circuit indicator. When short-circuit indicator is on, turn off the power immediately there is a direct short between the +5 V and the ground somewhere in the circuit.

- +5 V and ground distribution connections. Since both +5 V and ground are needed in many circuit components, these connection points make V_{CC} (+5 V) and GND (ground) more conveniently available for connection to the circuit board. The graphical symbols for V_{CC} and GND can be found in Fig. 1-1.

- Logic state indicators. Eight to twelve independent indicator lights with access pins are needed for visual indication of signal levels. If the signal applied to an access pin is high (greater than or equal to 1.5 V), the corresponding indicator light ON; if low (less than 1.5V), the light is OFF. It is customary to consider an ON light as indicating a logic-high signal and an OFF light as indicating a logic-low signal. The indicator lights will be referred to as LTs. Fig. 1-1 shows the graphical symbol for the indicator light.

- Clock source. A variable-frequency clock source is needed to provide automatic, continuous alternating high-low signals referred to as pulses. The clock frequency should be slow enough to allow visual observation -about 0.5 to 5 pulses per second. The four knobs in the top left corner select the waveform, frequency, and amplitude.

- Toggle switches. Eight to ten independent toggle switches are needed to provide continuous logic signals. A switch circuit may have either a single output (single-rail) or two outputs (double-rail). SPST (single-pole single-throw) and SPDT (single-pole double-throw) are terms electrical engineers use for single-rail and double-rail output switches, respectively. If two outputs are provided, one output is always the "opposite" of the other. For example, when the switch lever is UP, the upper access pin provides a "high" (+5V) signal while the lower access pin provides a "low" (GND). The outputs are reversed if the switch lever is DOWN. Thus, a double-rail switch can provide both the signal and its complement. The toggle switches on the digital trainer you will be using for this course only provide a single output. In order to also have the signal's compliment you will need to use an inverter. It is highly desirable that a switch output signal be "debounced". When a mechanical switch is "flipped" the spring of the switch always bounces up and down many times before settling on its final position. These bounces, lasting for about a few milliseconds, can cause the output to oscillate between high and low values. A "debouncer" eliminates these undesirable oscillating signals, thus providing a single, clean transition of the output for each switch action. The toggle switches will be referred to as TGs. The graphical symbols for toggle switches are shown in Fig. 1-1.

- Pushbutton switches. Two independent pushbutton switches are needed to provide "momentary" logic signals. These switches may also have either one or two outputs. When two outputs are provided, they furnish two "opposite" signals. When the button is NOT pushed (i.e., in its "normal" state), the "'NORMALLY ON" terminal provides a signal while the "NORMALLY OFF" terminal gives a low signal. The outputs are reversed when the button is pushed (and held). Like toggle switches mentioned above, the pushbutton switches provided on the digital trainer will not provide the complementary outputs. Again, the use of an inverter is needed. Pushbutton switches will be referred to as PBs. The graphical symbol for the pushbutton switch is also given in Fig. 1-1.

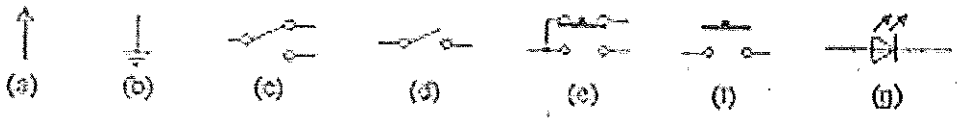


Fig. 1-1. Graphical symbols for (a) power supply, V_{CC}. (b) ground, GND. (c) double-rail toggle switch (SPDT). (d) single-rail toggle switch (SPST). (e) double-rail pushbutton switch. (f) single-rail pushbutton switch. (g) indicator light.

Note that the effect of a pushbutton action is "temporary" (outputs return to their "normal" state when the "push" action is removed) while the effect a toggle switch action is "permanent." Thus, toggle switches are used to provide "level" signals while pushbutton switches are used to generate "pulse" signals. A pulse is generated when the button is pushed and then released. A "positive" pulse (low-high-low) is generated at the NORMALLY OFF output for each time the button is pushed and released and, at the same time, "negative" pulse (high-low-high) is generated at the NORMALLY ON terminal. Since each pushbutton action is intended to generate one clean pulse, these outputs must be debounced.

In addition to the digital trainer unit, a "protoboard" on which to assemble the circuits is needed. The protoboard is a rectangular plastic board with hundreds of holes in it. Each of these holes has metal spring contacts buried inside to permit rapid insertion or removal of the circuit components. A sample layout of the board is shown in Fig. 1-2. The middle part of the board consists of two "bays," one on each side of the center gutter. Depending on the size of the board, there should be about 64 columns in each bay. The 5 holes in each column are all connected together underneath the board, so each column represents just one electrical potential point. This permits multiple access to the same circuit point. When building a circuit, each lead of a component must occupy a different electrical point. Integrated circuits (ICs) with dual-in-line packaging (DIP) have two rows of leads extending from the packaging, which must be inserted over the center gutter, one row of leads in each bay, in order to have a different electrical point for each lead.

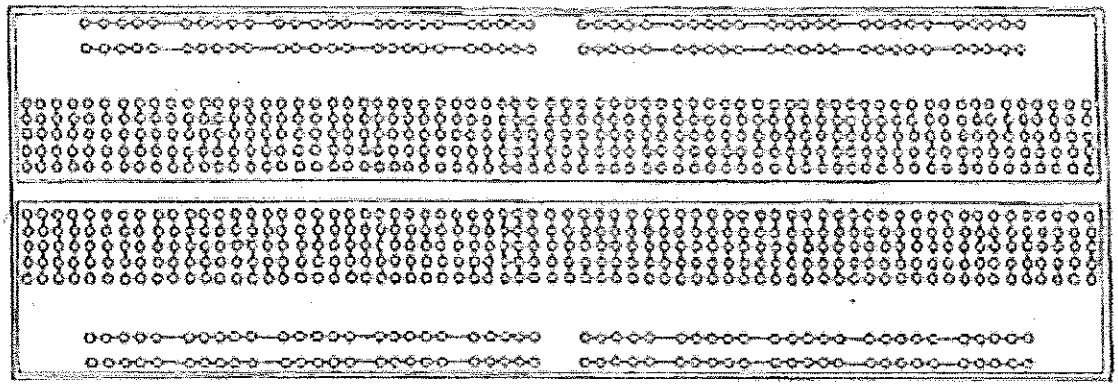


Fig. 1-2. Protoboard Internal connections

The protoboard also has two rows of contacts on both top and bottom edges of the board. The contacts of each row are all electrically connected (thus these four rows represent four electrical points). These rows of contacts are generally used to carry V_{CC} or ground to circuit components distributed throughout the board, and will be referred to as the V_{CC} bus or the ground bus, respectively. Note that some of the boards divide each row into two non-connecting halves (thus yielding 8 electrical points as in Fig. 1-2). A "jumper" wire must be used to connect the two halves if you want the same signal to appear across the entire row.

IV. INTEGRATED CIRCUITS

Besides the protoboard, the lab kit (see Appendix A for a complete list of parts needed to do experiments described in this manual) contains many integrated circuits (ICs). An IC is an electronic component fabricated from semiconductor material such as silicon. These tiny "chips" are enclosed in ceramic or plastic cases for protection and mechanical support. Access to the internal circuit is made available through conducting leads (also called pins) extending from the package. One of the packaging arrangements is called the dual-in-line package (DIP). Most of the ICs you'll be using in this course are 14-pin or 16-pin DIPs. The pins are numbered 1 through n/2 on one side and (n/2)+1 to n on the other side for an n-pin package as shown in Fig. 1-3.

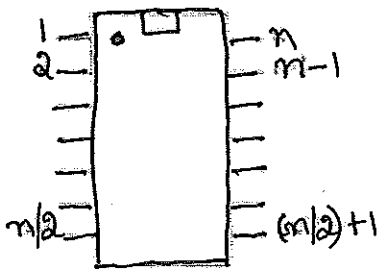


Fig. 1-3. IC pin numbering system.

Note that when viewed from the top, with the IC standing on its leads, the notch or the dot indicates the orientation of the IC (some foreign manufacturers may put the dot near pin n instead of pin 1). Pins are always numbered in the counter-clockwise direction. Normally, in an n-pin package, pin n/2 is the ground (GND) pin and pin n is the V_{CC} (+5V) pin. However, exceptions do exist. Always check IC pin assignments (see Appendix B), known as "pinouts", before making connections. V_{CC} and GND must always be connected for the IC to function even though they are often omitted in logic diagrams.

Three commonly used technologies for making SSI (small-scale-integration) and MSI (medium-scale-integration) devices are TTL (transistor-transistor logic), ECL (emitter-coupled logic), and (complementary metal-oxide-semiconductor). ECL circuits are known for their high speed and CMOS for their low power consumption and high packing density. TTL circuits are faster than CMOS circuits but consume more power. As the CMOS technology advances, its disadvantages are diminishing. At the present, CMOS is the dominant technology in digital circuits.

Logic circuits are grouped into "families." ICs of the same family share certain common characteristics. One of the most popular commercial-grade logic families is the "74" family. The military-grade "54" family is functionally equivalent to the "74" family but is more "rugged" (and more expensive). The "74" family offers a complete line of logic devices ranging from gates to memories. Within the "74" family, there are several different subfamilies. The "standard" (or "regular") series has two or three digits immediately following the number "74." A letter A or B may follow these digits. Together, they specify the logic functions of the device (i.e., function code). A letter "L" between the "74" and the function code stands for the low power series, "S" for Schottky, "LS" for low-power Schottky, "F" for fast, "AS" for advanced Schottky, and "ALS" for advanced low-power Schottky. The "74" series was originally defined for the TTL technology but has expanded to cover CMOS circuits as well. A "C" between the "74" and the function code stands for CMOS, "HC" for high-speed CMOS, "HCT" for high speed CMOS TTL compatible, "AHC" for advanced high-speed CMOS, "AC" for advanced CMOS, and "ACT" for advanced CMOS TTL compatible.

For example, 74LS00 means

- 74 -commercial grade
- LS -low-power Schottky series
- 00--function code for quadruple 2-input NAND gates

Letters preceding "74" represent manufacturer's code (such as MC for Motorola). Another letter or two may appear at the end to indicate the packaging (such as P for plastic). Logic functions are specified according to the positive logic definition, i.e., high voltage for logic-1 and low voltage for logic-0. Generally, positive logic is assumed unless stated otherwise. Thus, "TRUE", "1", and "HIGH" all have the same meaning; and "FALSE", "0", and "LOW" are all equivalent.

Note that each subfamily has its electrical characteristics and may not work correctly when used in a mixed fashion.

After obtaining the lab kit it is a good idea to sort the ICs according to their numerical order (as shown in Appendix A) and insert them onto a piece of flat, conducting foam for easy access as well as protection of the device.

Other components in the kit include resistors and capacitors. These are two-terminal devices as shown in Fig. 1-4.

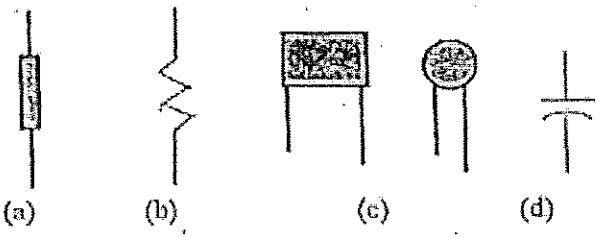


Fig. 1-4. (a) resistor physical outline, (b) resistor symbol, . (c) capacitor physical outlines, and (d) capacitor symbol.

V. LOGIC, SCHEMATIC, AND BLOCK DIAGRAMS

A *logic diagram* is used to show the "logical thinking" behind the design; it shows only the "logic" of the design without giving the exact device or the pin numbers. For example, we may use a 3-input XOR gate to illustrate the "logic" even though there is no device available. A *schematic diagram* is used to show how the circuit is to be built; hence IC device numbers and pin numbers, etc. are all included. Thus, the aforementioned XOR circuit is drawn as two commercially available 2-input XOR gates. Use IEEE standard logic symbols (some of them are shown in Fig. 1-5) for logic or schematic drawings. Each component is composed of an "outline" and input and output leads. Note that small circles (known as bubbles) associated with logic gates always mean inversion.

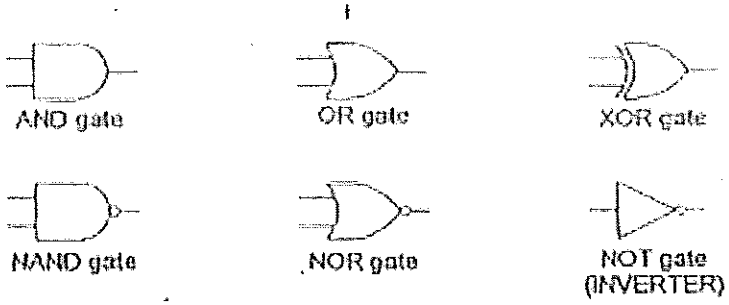


Fig. 1-5. IEEE standard symbols for logic gates.

Even though an SSI device may contain several gates, they do not have to be drawn together inside one box; rather; they are drawn wherever they are needed. For MSI and LSI logic devices, however, they are usually drawn as a single outline showing, generally, input signals on left-hand side of the outline and the output signals on the right-hand side. Signal names should be written inside the outline, and pin numbers, when given, should appear just above the connecting leads outside the outline. Pins should be arranged for convenience of logic connections, not in numerical order. Power and ground lines are customarily omitted in logic diagrams. The device type or number may be written either inside the outline or just above or below the outline, but try to be consistent. Signals in a logic or schematic diagram should generally flow only in one direction (i.e., from left to right.) Crossing lines are not considered to be connected unless a dot, ".", is drawn at the crossing. If it takes more than one page to show the complete logic diagram, signals coming from or going to other pages should be clearly identified. Fig. 1-6 (a) shows a sample logic diagram and Fig. 1-6 (b) a sample schematic diagram.

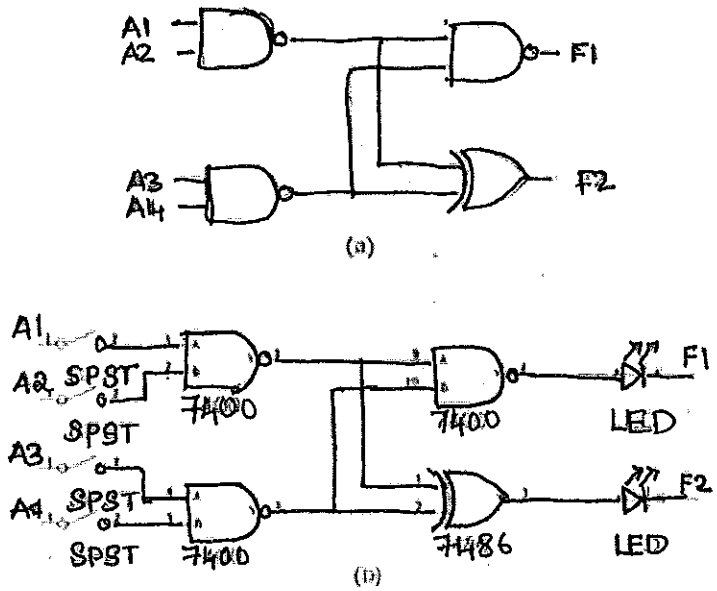


Fig. 1-6. (a).A sample logic diagram, (b) A sample schematic diagram.

A *block diagram* is used to show the general structure of a complex circuit and the relationships among its subunits; no details are given. It is intended to convey a plan or an idea at a higher level without worrying about the specific details of implementation. The subunits are simply drawn as boxes (blocks) and only "key" interconnecting signals are shown. Fig. 1-7 shows a sample block diagram.

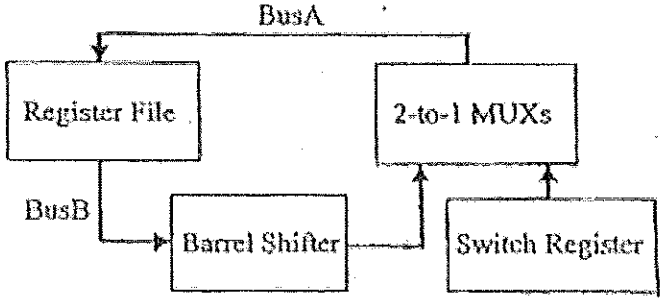


Fig. 1-7. A sample block diagram.

VI. CIRCUIT ASSEMBLY

> **CAUTION:** When building or changing circuit connections, always make sure that the power is OFF.

ICs are very and easily damaged. When inserting ICs into the protoboard, make sure that all its leads are perpendicular to the body, and that all leads are properly aligned with the holes of the board. Apply force evenly across the IC package and gently but firmly push it in until the bottom of the IC package touches the board surface. When removing an IC, use an IC extractor so that both ends of the IC come out at the same time. Another way to remove an IC is to insert a thin-blade screwdriver through the center gutter alternately under each end of the IC and gently pry it loose. Never use fingers to pullout an IC or you will end up with bent or broken leads and bloody fingers.

Place all ICs in the same direction on a circuit board, preferably upright (pin 1 is up) if vertical, or left (pin 1 to left) if horizontal. Use 22-gauge (or smaller) insulated solid wires for circuit connections. To make a connection, simply strip approximately a quarter of an inch of insulation from each end of a piece of wire and insert the ends into the desired connection points. Make sure that the ends of the wire are straight and that the metal part is not cut while stripping the insulation off. Do NOT push the insulated part of the wire into the holes or you will have bad connections. A pair of long nose pliers may be helpful for inserting and removing wires in a crowded area. Use wires just long enough for

intended connections and try to make a neat layout with wires close to the surface of the board. Different color wires for different groups of signals can save time later when checking connections or troubleshooting. Wires laid directly over an IC will make the replacement of that IC more difficult. It must be stressed that messy connections can cause malfunctions and hard-to-find problems. Do the wiring carefully and you will avert a lot of headaches. If you are building the circuit from a schematic diagram, use red pen to trace over the connection on the diagram every time you complete a connection on the circuit board. This is a good way of tracking connections made.

ICs generate a considerable amount of electrical noise when they switch states. To prevent the noise from affecting the operation of the circuit, it is suggested that a 0.1 capacitor be placed between the V_{CC} and the ground for every three or four ICs. These capacitors should be placed close to ICs and distributed throughout the circuit. Other circuit components such as resistors can also be inserted into the breadboard, but make sure that the leads of such components are comparable to that of the 22-gauge wire. Fig. 1-8 shows a simple circuit assembly on a protoboard.

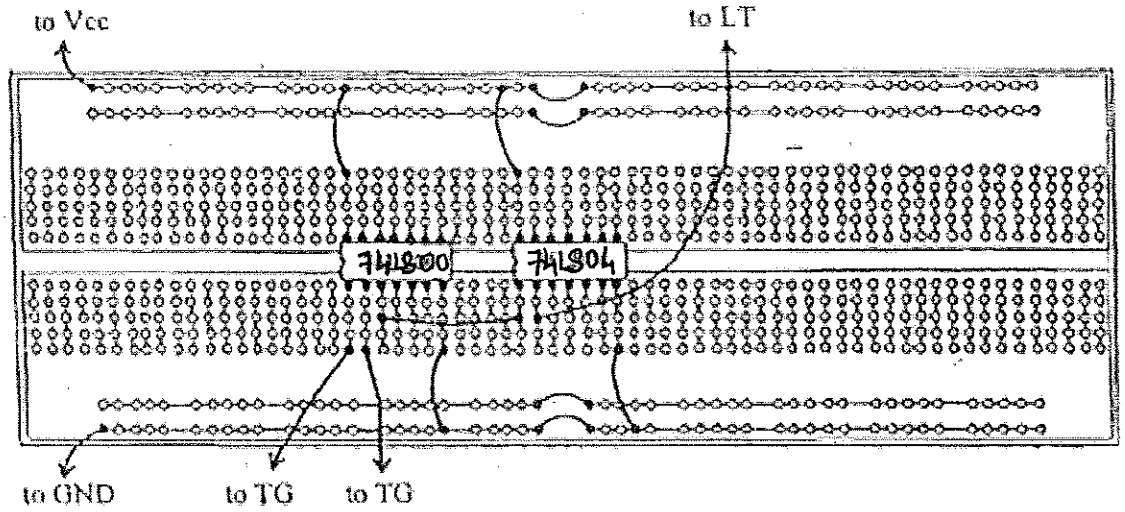


Fig. 1-8. Sample circuit assembly.

VII. CIRCUIT DEBUGGING

Troubleshooting a circuit is an art that is developed with experience. When a logic circuit does not perform according to expectations, there are generally three sources of errors: design error, wiring error, and component failure. In addition to being careful in design and wiring, always double-check the design and wiring. If the circuit is somewhat complex, build and test one section at a time. Make sure that power and ground pins are properly connected even though they are usually not shown in the logic diagram. Note that an unconnected input is "floating" and is not the same as "0". If a permanent logic-1 (0) is intended, it should be connected to a logic-1 (0) source. All "unneeded" inputs in any logic component (a gate, a flip-flop. a decoder. etc.) must always be connected to appropriate logic sources.

Component failures are rare (unless the component was mistreated). If you find a bad component, however, throw it away. Otherwise you may find yourself troubleshooting the same component later. One sure way to "blow" an IC is to connect a low output to a high source.

A logic probe is a very simple but useful debugging tool. It detects the presence or absence of a voltage level. The signal level (high, low, or floating) at the point touched by the tip of the probe is displayed by the LEDs (light-emitting diodes) of the probe. For TTL circuit, a voltage of 0.8 V or lower is considered to be low and a voltage of 2.0 V or higher considered to be high. For general CMOS circuits with 5V V_{CC} (or V_{DD}), a voltage of 1.5 V or lower is low and 3.5 V or higher is high. Most logic probes have a TTL/CMOS switch so that logic voltage levels can be properly selected. Most probes also have a latch to capture signals that are too short for our eyes to see. When using a logic probe, make sure that the tip of the probe touches only one signal point at a time. If you touch two points at the same time, you short-circuit those two points, which may damage the component. Any indicator light (LT) on the digital trainer unit can serve as a simple logic state indicator although it cannot differentiate open circuit from a logic-0. A simple pre-assembled logic probe circuit is included in the trainer. The voltage levels for this probe are set for TTL circuits but should work for most of the CMOS circuits as well. When this circuit receives a high input, the green LED is on; if the input is low the red LED is on; and if the input is open (or floating), neither LED is on.

With experience, you will develop better design and debugging skills. You will learn how to narrow down the problem to a small area by performing appropriate tests, and then performing further tests on that small area. Again, it must be

stressed that the best policy is to do everything carefully and systematically from the very beginning. Debugging is always frustrating and extremely time consuming.

III. LABORATORY WORK

1. Check the indicator lights on the digital trainer to see if they are all working. Describe the outcome and the procedures used to accomplish this task. Note that a digital circuit typically may encounter three different types of signals: high, low, and floating (no connection). If an indicator light is off, what can you say about the signal connected to it?

2. With a logic probe, check if the toggle switches on the digital trainer are operational. Describe the outcome and the procedures used to accomplish this task. How does your logic represent a high signal? A low signal? A floating signal?

3. With a logic probe, check if the pushbutton switches on the digital trainer are operational. Describe the outcome and the procedures used to accomplish this task.

4. Set up a V_{CC} bus and a ground bus on the protoboard. The top inner row and the bottom outer row (some boards may have marked these with red lines) should carry V_{CC} and the top outer row and the bottom inner row (some boards may have marked these with blue lines) should be the ground. Draw a diagram to show your connections. How did you verify that you indeed have a single V_{CC} bus and a single ground bus?

5. Connect an indicator light to the output of the function generator. Set the function knob to the far right position to output a square wave. Set the top FREQ knob to 10. Experiment by adjusting the lower FREQ knob from 0.1 to 1 and the AMP knob from 1 to 10. Observe the indicator light and estimate-the clock frequency. How does adjusting these two knobs affect the output?

IV. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Make sure to include answers to all questions raised in the experiment. See Appendix C about lab reports.

Exp. 2: BASIC LOGIC OPERATIONS

"If you accommodate others,
you will be accommodating yourself."
- a Chinese proverb

I. OBJECTIVES

1. To become familiar with digital circuit construction techniques.
2. To learn the properties of basic logic gates such as AND, OR, NAND, NOR, XOR, XNOR, and NOT (inverter).

II. COMPONENTS

1	74LS00	quadruple 2-input NAND gates
1	74LS04	hex Inverters (NOT gates)
1	74LS08	quadruple 2-input AND gates
1	74LS32	quadruple 2-input OR gates
1	74LS86	quadruple 2-input Exclusive-OR (XOR) gates

III. INTRODUCTION AND PRE-LAB

This is your first real experiment. You should review topics on basic logic operations, logic functions, and logic gates. Remember the difference between logic value and voltage. When we use "0" and "1" we are using logic values; when we use "HIGH" and "LOW" we are using voltages. In our case, voltage levels are actually represented by indicator lights, "ON" for "HIGH" and "OFF" for "LOW". There are two ways to convert the physical voltage levels to the abstract logic values: positive logic and negative logic. In the positive logic system, a "HIGH" (or "ON") means logic-1 and a "LOW" (or "OFF") means logic-0. The definition is reversed for negative logic. Thus, for a given circuit, what you call it depends on the logic assignments used. Normally we name a device based on the positive logic definition. To prepare you for the experiment, complete the truth table for the following two-variable functions:

x y	AND x•y	OR x+y	NAND (x•y)'	NOR (x+y)'	XOR x⊕y	XNOR (x⊕y)'
0 0						
0 1						
1 0						
1 1						

IV. LABORATORY WORK

1. Carefully insert all five ICs listed in the COMPONENTS section into the protoboard, making sure to orient all ICs in the same direction. Remember that all ICs must be inserted over the center gutter with one row of pins on each side of the gutter. Designate one of the top bus rows as the Vcc bus and one of the bottom bus rows as the ground bus. Refer to Appendix B for the pinouts of the five ICs used here and connect all Vcc pins to the Vcc bus and all ground pins to the ground bus.

2. Except for the '04 (inverter), the other four ICs are all "quad" devices. That means they each have four independent but identical gates. Note that they also have similar pinouts with each gate having two inputs and one output. For example, pins 1 & 2 are inputs and pin 3 is the output in all four cases. Make all connections shown in the following circuit diagram except the connections to the toggle switches and indicator lights.

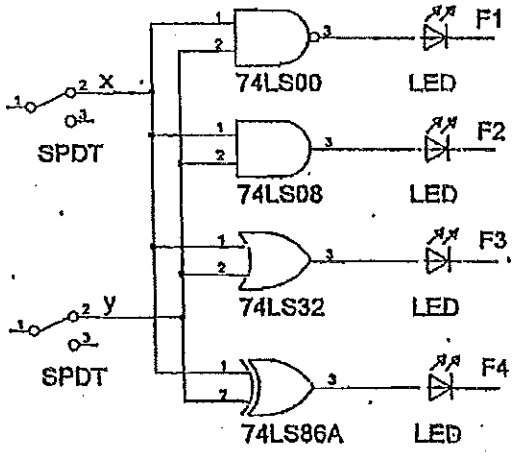


Fig. 2-1. Basic two-input logic gates.

3. Now move the breadboard to a place close to the digital trainer unit. With the power OFF, connect the Vcc bus of the breadboard to a +5 V terminal on the trainer; connect the ground

bus of the board to a ground terminal on the trainer. Complete the connections to the toggle switches and the indicator lights as shown in Fig. 2-1.

4. Double check your circuit connections before you turn the power on. If all connections are correct, turn the power switch ON and experimentally (i.e., based on your experimental observations) complete the following table by recording whether the light corresponding to that variable is on or off.

x y	F1	F2	F3	F4
off off				
off on				
on off				
on on				

5. With power turned off, gently remove connections made based on Fig. 2-1 (do not remove Vcc and ground wires to the ICs). Make all necessary connections to implement the circuit shown in Fig. 2-2.

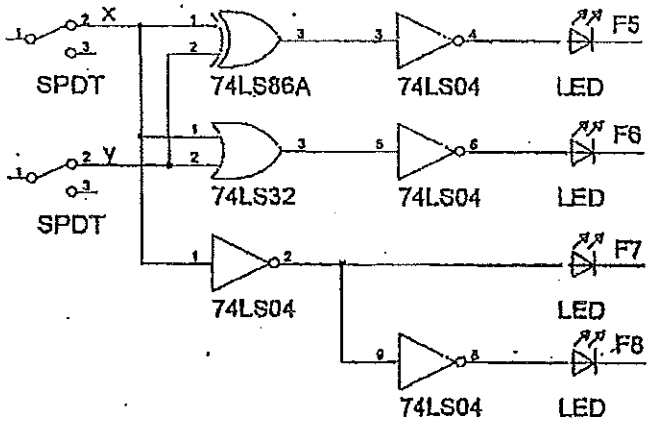


Fig. 2-2. Circuit with inverters

6. Turn the power back on and experimentally complete the following table:

x y	F5	F6	F7	F8
off off				
off on				
on off				
on on				

7. With the power off, remove all connections except those connected to the Vcc and the ground. Construct the circuit shown in Fig. 2-3.

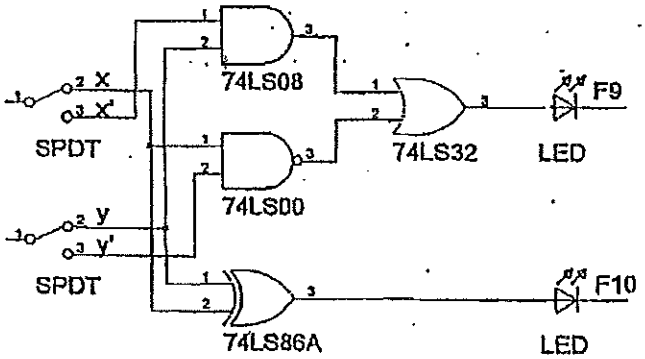


Fig. 2-3. XOR circuits.

8. Turn the power back on and complete the table based on your experimental observations.

x y	F9	F10
off off		
off on		
on off		
on on		

Demonstrate the operations of this circuit to your lab instructor and obtain his/her signature.

Instructor's signature:_____. Date:_____

V. POST LAB

1. Convert the tables obtained in steps 4, 6, and 8 of Section IV to logic tables by replacing "on" with "1" and "off" with "0".

X y	F1	F2	F3	F4
0 0				
0 1				
1 0				
1 1				

x y	F5	F6	F7	F8
0 0				
0 1				
1 0				
1 1				

x y	F9	F10
0 0		
0 1		
1 0		
1 1		

2. Express functions F1 to F10 as logic functions of x and y.

F1 =

F2 =

F3 =

F4 =

F5 =

F6 =

F7 =

F8 =

F9 =

F10 =

3. Which of the 10 functions in step 2 are identical to each other?

4. If the input x is kept at logic-1 (i.e., use only the bottom two rows of the table in step 1 of Section V), what functions are being implemented?

F1 =

F2 =

F3 =

F4 =

F5 =

F6 =

F7 =

F8 =

F9 =

F10 =

5. What logic functions are performed by these same circuits if one of their inputs is kept at 0?

F1 =

F2 =

F3 =

F4 =

F5 =

F6 =

F7 =

F8 =

F9 =

F10 =

6. What logic functions are performed when the same signal is applied to both inputs?

   F1 =

   F2 =

   F3 =

   F4 =

   F5 =

   F6 =

7. Based on your observations from steps 4, 5 and 6, complete the right-hand side of the following equations (A is a logic variable):

$A + 0 =$	$A + 1 =$	$A + A =$
$A \cdot 0 =$	$A \cdot 1 =$	$A \cdot A =$
$A \oplus 0 =$	$A \oplus 1 =$	$A \oplus A =$
$(A \cdot 0)' =$	$(A \cdot 1)' =$	$(A \cdot A)' =$
$(A')' =$		

8. There are a total of 16 logical functions of two variables, x and y. List all 16 functions as AND-OR-NOT combinations of x and y.

9. Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Attach all completed tables, equations, etc. to this report. See Appendix C for lab report requirements.

Exp. 3: FAMILIARIZATION WITH EDA TOOLS

"To do work well, first sharpen tools."
- a Chinese proverb

I. OBJECTIVES

1. To become familiar with EDA tools for schematic entry and simulation.
2. To verify logically equivalent operations by simulation.

II. COMPONENTS

1 nand2	quad 2-input NAND gates.
1 inv	hex inverte.
2 and2	quad 2-input AND gates.
1 nand3	triple 3-input NAND gates.
1 or2	quad 2-input OR gates

III. INTRODUCTION AND PRE-LAB

Modem EDA (Electronic Design Automation) tools are used extensively in digital design. Two major tools for design entry are schematic capture and HDL (hardware description language). Schematic capture is a graphics-based entry tool while HDL is a text-based entry tool. We will leave HDL for another course and deal only with schematic capture here. Another· essential EDA tool is a good functional simulator - to functionally verify the design without having to actually construct it These tools allow engineers to manage the design of complex systems reduce the design time, explore alternative solutions, reuse parts or all of a previous design. Some of these tools also translate designs into actual circuits and parts lists. There are also tools to perform timing simulation, optimize performance, enhance testability, reduce power consumption, etc.

Study any material (handout, manual, help file, etc.) that you may have for the EDA software to be used. If you have a copy of the software, install it on your personal computer and get acquaint with its usage. .The software m!!St e able to perform at least two functions mentioned above: schematic capture and functional simulation.

III .LABORATORY WORK

The instructor will demonstrate the basic capabilities of the software before you perform the

experiment. Specifically, the instructor will show you how to:

- set up a project, select the libraries to be included, select the paper size and orientation, construct a title block.
- select and place circuit component, make interconnections, place input and output terminals, enter text material.
- edit circuit diagrams (add, delete, or move components and connections).
- select signals to be simulated, generate input signals and observe output signals.
- print the circuit diagrams and simulation results.

1. Most EDA software tools have a "project management" program. This permits you to create a new project or reopen an existing project. You can also add or delete library files associated with the project. A library file defines a set of standard components, a custom designed part, or a previously designed circuit. Before you can use any component, its library file must be accessible to the project.

   Create a new project titled "lab3" and include the library file that has the components listed in Section II above. For now, the exact subfamily (LS, ALS, AC, HC, etc.) is not important but try to use components within the same subfamily. Also make sure that the paper size is set to A (8½ x 11) and the title box contains your name, project title, date, etc.

2. Using the EDA tool, enter the schematic diagram shown in Fig. 3-1. Note that your particular EDA tool may use different symbols for some of the components. The routing of the wires may also differ but try to make professional quality diagrams. Make sure to enter the text part of the diagram (x, y, F1, F2, F3, and F4) as well.

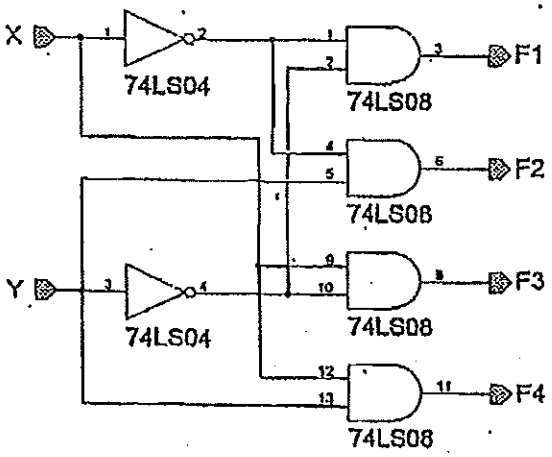


Fig. 3-1. An example schematic diagram.

3. Once you have the schematic diagram entered, identify the signals you wish to simulate. Some tools allow you to select these signals on the schematic diagram while others require you to go to the simulation mode and identify signals to be simulated there. For the circuit of Fig. 3-1, identify the two switch signals (x and y) and the four output functions (F1 to F4) as simulation signals and switch the program to the simulation mode.

4. Depending on the software, there are different ways to generate simulation inputs. For this experiment, it is easier to assign keys on the computer keyboard as input signals. You will be able to change the logic value of a signal by hitting the designated key. Systematically (i.e., following the binary sequence) go through all four input combinations of x and y and observe all four outputs as each new combination is applied. Obtain a printout of the simulation result.

5. Create a new project. Draw the schematic diagrams for the following pairs of logic functions, all on the same schematic sheet. By simulation, show whether f1=f2, f3=f4 and f5=f6. Demonstrate the simulation to your lab instructor and obtain her/his signature of approval.

   a) $f1 = ((x \cdot y)' \cdot z)'$ -- two 2-input NAND gates and
      $f2 = (x \cdot y \cdot z)'$ -- one 3-input NAND gate

   b) $f3 = x\,y + x'\,z + y\,z$
      $f4 = x\,y + x'\,z$ -- note that $f3 = f4 + y\,z$

   c) $f5 = w\,y + w\,z + x\,y + x\,z$
      $f6 = (w + x)\,(y + z)$

Instructor's signature:_____. Date:_____

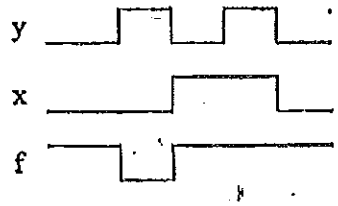
V. POST-LAB

1. If your program provides only waveforms for step 4 in Section IV, translate the waveform into a truth table; if your program generates only tables, draw the waveforms represented by the table. You must include both the table and the waveforms in your report. The example below shows the relationships between the two.

Truth Table:

x	y	f
0	0	1
0	1	0
1	0	1
1	1	1

Waveform:



2. Describe what the circuit of Fig. 3-1 does.

3. Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Your report must also include a) all printed schematic diagrams and simulation results, and b) answers to questions raised in this experiment.

Exp. 4: COMBINATIONAL CIRCUIT DESIGN

"The value of knowledge is not in its accumulation,
but in its application."
- a Chinese proverb

I. OBJECTIVES

1. To practice simple combinational circuit design.
2. To develop some proficiency in using XOR and NAND gates.

II. COMPONENTS

1	74LS00	quadruple 2-input NAND gates
1	74LS04	hex inverter
1	74LS08	quadruple 2-input AND gates
1	74LS10	triple three-input NAND gates
1	74LS32	quadruple 2-input OR gates
1	74LS86	quadruple 2-input Exclusive-OR (XOR) gates

III. INTRODUCTION

The digital computer is only one of the countless applications where digital circuits are used. In this experiment, you will see some simple applications of the digital circuits. As in any engineering design work, digital design is the process of conceiving, planning, and laying out a solution to the problem. In a combinational circuit design, the first step is always to analyze the problem. We must understand the problem clearly enough to be able to restate the problem completely and unambiguously. When the problem is thoroughly understood, we should be able to identify the inputs and outputs of the circuit that we are trying to design. Once inputs and outputs are established, problem requirements can be translated into input-output relationships. These input-output relationships are represented in mathematical models as truth tables or Boolean equations. Finally, Boolean equations are converted into logic diagrams. During the entire design process, we should always check to see if the original requirements are being met. Sometimes we may have to reinterpret the problem or to modify the problem statements, if that is permitted. Of course, we must also keep in mind that not all problems have solutions. If conflicting conditions arise, then the problem has no logical solution.

Two methods are commonly used for translating a combinational logic problem into a mathematical model. The first and generally recommended method for a small circuit is to construct a truth table in which all possible input combinations are listed. In logic circuits, each

input and output has only two possible values, so the table has a finite number of combinations. We then analyze each input combination and determine the outputs based on the given specifications. Once a truth table is obtained, the remaining steps are quite straightforward. Here is an example:

> "A three-person panel is convened to test and compare two products, A and B. Each panel member will vote for either product A or product B. Design a circuit that will indicate the preference of the panel."

After carefully analyzing the problem statement, we need to make the following clarifying assumptions in order to make the problem statement complete and unambiguous:

- Panel members must vote for one of the products - voting for both or none is not allowed.
- A simple majority (in this case, the one that receives 2 or more votes) determines the winner.

We then proceed to identify the inputs and outputs. We have three inputs, one from each panel member, and we will use variables x, y, and z to represent them. We have two outputs, products A and B. The second step is to show the input-output relationships by using a table of combinations (truth table). We assume that for each input, a 0 is a vote for product A and a 1 is a vote for product B; and for each output, a 1 signifies that it has the majority vote.

Inputs			outputs	
x	y	z	A	B
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

Note that A and B are never the same (they cannot both be winners or losers). As a matter of fact, they are complements to each other. Therefore, it is possible to have just one output, say A. If A is 1, then A is the winner; if A is 0, then B must be the winner. Now we can write the Boolean expressions to represent the above input-output relationships:

$$A = x' y' z' + x' y' z + x' y z' + x y' z'$$
$$= x' y' + x' z' + y' z'$$
$$B = A'$$

or,

$$B = x' y z + x y' z + x y z' + x y z$$
$$= x y + x z + y z$$
$$A = B'$$

Finally, Fig. 4-1 shows how the second alternative can be implemented using only NAND gates.

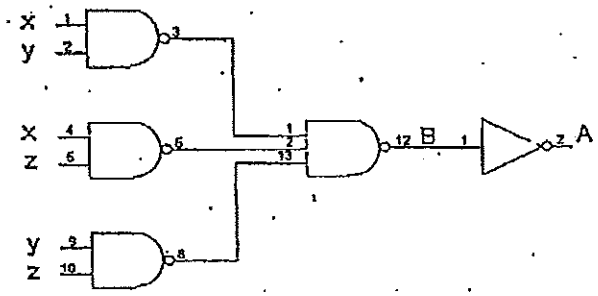


Fig. 4-1. An all NAND implementation of the voting machine.

The second method of translating problem statements into mathematical models is to express input-output relationships directly in Boolean algebraic forms without going through the truth table. Some times this is an easier or more natural interpretation of the problem requirements. For instance, if the panel is expanded to 5 people in the above voting machine example, the truth table would now contain 32 rows (still doable, but think of the number of rows in the truth table if the panel is expanded to, say, 11 members). However, we know that the product is a winner if 3 or more people vote for it. This can be easily expressed in the Boolean form as (using v, w, x, y, and z as the input variables):

$$B = v w x + v w y + v w z + w x y + w x z + x y z$$
$$A = B'$$

As another example, consider the following problem statement:

"The sprinkler should be turned on if activated by a person or if the soil moisture sensor indicates that the soil is dry and the rain sensor says it is not raining."

One important first step about a problem statement that involves several "and" and "or" connectives is to clarify the phrases to which these connectives apply. In this case, the statement can be made clearer (by using the more logical interpretation) if we put a comma after the word "person." Here is the revised statement:

> "The sprinkler should be turned on if activated by a person, or if the soil moisture sensor indicates that the soil is dry and the rain sensor says it is not raining."

We then assign the following variables to the inputs and the output:

Inputs:

P = activated by a person,
D = soil is dry, and
R = raining.

Output:

S = sprinkler on,

Next, from the revised problem statement, it is quite obvious that the Boolean expression representing the statement is $S = P + (D \cdot R')$. Note that without the added comma in the problem statement, some people may interpret it as $S = (P + D) \cdot R'$.

An implicit requirement in any logic design is to reduce, as much as possible, the total cost. Since the total cost is influenced by many different factors, the "best" design must be judged based on the design objectives and constraints. Generally, we try to reduce the total number of logic gates and/or IC packages.

IV. PRE-LAB

Design the following combinational circuits using only components listed in Section II above. Unless specific requirements are indicated, try to use a minimum number of ICs to implement each circuit. You may assume two-rail inputs (i.e., both uncomplemented and complemented variables are available as inputs). Attach detailed design work as an appendix to your lab report.

1. Binary to Gray code converter. The table below shows a four-bit binary number representation and its equivalent Gray code. Design a binary-to-Gray code converter and show your logic circuit next to the truth table. Hint: think of an XOR implementation.

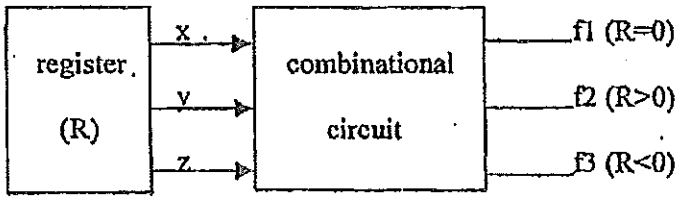
binary code b3 b2 b1 b0	Gray code g3 g2 g1 g0
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 0
0 1 0 0	0 1 1 0
0 1 0 1	0 1 1 1
0 1 1 0	0 1 0 1
0 1 1 1	0 1 0 0
1 0 0 0	1 1 0 0
1 0 0 1	1 1 0 1
1 0 1 0	1 1 1 1
1 0 1 1	1 1 1 0
1 1 0 0	1 0 1 0
1 1 0 1	1 0 1 1
1 1 1 0	1 0 0 1
1 1 1 1	1 0 0 0

2. Three-way light control. Design a switching circuit which can turn a light on or off from three different switches. You may assume that only one switch will be acted on at any given time. (Hint: Start with a truth table in which the light is off when all three switches are at logic-0; then determine what the outputs should be for other input combinations knowing that each switch must be able to turn the light on and off.) Show your logic circuit below.

3. Voting machine. A committee of five members, A - E, each controls a toggle switch. When a vote is to be taken, each member simply sets his/her toggle switch according to his/her wish: up (logic-1) for yes and down (logic-0) for no. No abstentions are permitted. A motion passes when the chairperson and two or more other members vote yes, in which case the decision light is automatically turned on. Note that the chairperson (the one who controls

switch A) has veto power. That is, a motion is defeated as long as the chairperson votes no. Design a logic circuit which can implement this decision making task. Hint: obtain a Boolean expression directly from the problem statement and then simplify it by using Boolean algebraic manipulations. Show your logic circuit below.

4. Status flags. Sometimes it is necessary for a digital computer to know whether a number stored in a register is zero, positive non-zero, or negative. A positive number would have its most significant bit equal to 0 and the remaining bits not all 0s. A zero would have all bits equal to 0. A negative number would have its most significant bit equal to 1. In order to make the problem manageable at this level, you may assume that registers are only three bits long. Design a circuit, using only NAND gates, to detect whether the number in the register is zero, positive, or negative. A block diagram of the system is shown below. Assume that x is the most significant bit. Try to use as few IC packages as possible. (Hint: think of sharing logic gates among the three functions.)



Show your circuit to generate the status flags, f1, f2 and f3, here:

V. LABORATORY WORK

Construct all four circuits you designed in the PRE-LAB Section, one at a time, and verify the operations of these circuits using the digital trainer. You must demonstrate the operations of one or more (as specified by your lab instructor) of these circuits to your instructor.

Instructor's signature:_____. Date:_____

VI. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Also attach all design work (truth tables, K-maps, logic simplification steps, etc.).

Exp. 5: INTRODUCTION TO TROUBLESHOOTING

"The cautious seldom err. "
-a Chinese proverb

I. OBJECTIVE

To practice systematic hardware construction and troubleshooting techniques.

II. COMPQNENTS

74LS00 quadruple 2-input NAND gates
74LS08 quadruple 2-input AND gates
74LS86 quadruple 2-input XOR gates

III. INTRODUCUTION

A. Hardware Errors

By now you probably have experienced .some frustration' with non-functional circuits. Even experienced designers encounter these kinds of problems. This is a good time to develop a more systematic way of trouble-shooting your circuits. We will assume that the only tool you have is a logic probe. There are three major types of errors that you may encounter indigital circuits:

- Logic error - This is a design error. To reduce this type of error, you must know the subject well and do the design work systematically and carefully.

- Component failure - Modem ICs are fairly reliable and "'student-tolerant," so this type of error does not occur very often. Even so, it is important that you do not abuse the parts. For example, never connect an output to +5 V, ground, or another output.
  If you are certain that a part is defective, throw it away; don't-keep it with your working parts; otherwise, you will probably use it again.

- Connection error -This type of error accounts for most of the problems. It includes missed, misplaced, and "bad" connections. 'To minimize this type of error, study the circuit construction procedures outlined below and practice them faithfully.

B. Circuit Construction Techniques

1. Assign each IC and logic gate in the logic diagram an ID number. For example, the lower AND gate in Fig. 5-1 is labeled U1B, meaning IC #1, gate #B. Also write pin #s for each input and output on the logic diagram.

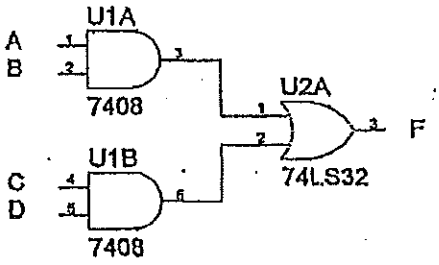


Fig. 5-1. A logic diagram with device IDs and pin assignments.

2. Carefully insert all necessary ICs on the protoboard in the same orientation (e.g., pin 1 to the left.)

3. Connect the Vcc and the ground pins of each IC to the +5 V bus and the ground bus, respectively. These connections are easy to forget since they are not included in the logic diagram. Also remember that the two halves of the buses on the protoboard may not be connected internally (i.e., you may need to have jumper wires connecting the two halves).

4. Strip the ends of interconnection wires so that about 1/4" of the wire is exposed at each end. If the exposed part is too long, you'll either insert too much of the wire into the board and cause an unexpected electrical short with an adjacent pin underneath the board, or leave too much exposed wire above the board and cause an accidental short with another exposed conductor. This is important - never let two exposed wires come into contact with each other. It is also important that you do not push the insulated part of the wire into the hole as it will force the spring contacts in the board too far apart, thereby preventing them from making a good contact with the exposed part of the wire.

5. A wire can be broken inside the plastic insulation. To check whether a wire is good or not, touch one end of the wire to the Vcc and the other to an indicator light. If the light comes on, the wire is probably unbroken.

6. As you make a circuit connection, use a red pen to retrace the connection on the logic diagram. Do this one wire at a time. This will reduce the chance of neglecting to make some connections. Count the pin numbers carefully.

7. Construct and test the circuit one section at a time. A smaller circuit is much easier to test than a larger one.

In summary, do the connections carefully and systematically. You will save time and minimize frustration. If the circuit does not function properly and you are confident that your design is good, take a close look at your circuit assembly and check the following items first before you start tracing the signals.

- Is the power supply turned on?
- Are the correct components used?
- Are the ICs plugged in properly (no bent leads underneath the IC) and in the orientation you intended (not backwards)?
- Do all ICs have power and ground connections?
- Are the power and ground buses properly connected to the power supply?
- Are the correct IC pinouts used?
- Did you count the pins correctly?
- Are there bare wires touching each other?
- Did you insert the wires too deeply into the holes or not deep enough?
- Are the tops of ICs too hot to touch? If so, turn off the power immediately. The IC is drawing too much current due to some improper connections. Recheck the wiring carefully, particularly the output connections.

C. Introduction to Troubleshooting

A logic circuit that has the possibility of generating an erroneous signal is said to contain a fault. Most of the faults that you may encounter in this course are time-independent (or static) faults. There are two possible types of static faults:

- Solid fault -- A solid fault does not disappear; it is always there. A solid fault may be "stuck-at" or "non-stuck-at." For the stuck-at type of fault, the signal value at the fault origin stays unchanged under different circuit operating conditions. The signal may be stuck-at-1 (s-a-1) or stuck-at-0 (s-a-0). For the non-stuck-at type of fault, the signal value at the fault origin may change when circuit operating conditions, such as input combinations, are changed. A solid fault may exist due to any of the three types of errors described above.

- Intermittent fault -- This type of fault occurs intermittently and is usually caused by environmental factors (temperature, humidity, vibration, etc.) In this laboratory

course, intermittent faults are most likely due to poor joint contacts (contacts are made sometimes but not other times. i.e., bad connections.)

It is important to realize that a fault may not always produce erroneous results. For example, a fault at point "A" of Fig. 5-1 will be propagated to point "X" only if the signal at "B" is 1. If B=0, the fault at "A" is actually "masked out" by the AND gate since the output of the AND gate is 0 regardless of the value at "A". For a circuit with a large number of inputs, it is impossible to test operations under all possible input combinations. The selection of "test vectors" (test inputs) thus becomes very important. This is a more advanced topic in testing and we will not get into it here. Simply remember that a working circuit is not necessarily fault-free. Effects of faults sometimes can be "masked out" as they propagate down the line. This is the whole idea behind the "fault-tolerant" design.

From your knowledge of logic gates, it is easy to derive the observations summarized in the table below. Remembering these simple rules will help you speed up the debugging process.

Gate type	expected output	observed output	where to check for faults
AND	0	1	inputs that should be at 0
	1	0	all inputs should be at 1
OR	0	1	all inputs should be at 0
	1	0	inputs that should be at 1
NAND	0	1	all inputs should be at 1
	1	0	inputs that should be at 0
NOR	0	1	inputs that should be at 1
	1	0	all inputs should be at 0

For small combinational circuits such as the ones you will encounter, it is always possible to test all possible input combinations and observe the outputs to see if the circuit is performing correctly. Since all interconnection points are available for observation, troubleshooting is a relatively easy task compare to that of a circuit where only inputs and the final outputs are available. If a certain input combination produces an incorrect output, you can either trace the signal "backward" or "forward" as will be illustrated later.

Use a logic probe for signal tracing. A logic probe has two indicator lights, "high" and "low". It can be used to show whether a signal is "high" (the high indicator is on), "low" (the low indicator is on), "floating" (neither indicator is on), or "oscillating" between 1 and 0 (both

indicators are on but somewhat dim.) Always check signals at the IC pins. This isolates device faults from connection faults.

Example 1. The intended circuit is shown in Fig. 5-2. If pin 4 of the OR gate U2B, is incorrectly connected to pin 3 instead of pin 2 of the hex inverter (due to miscounting the pins), the resulting circuit is shown in Fig. 5-3.

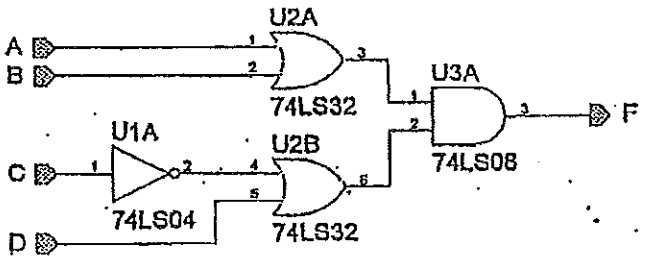


Fig. 5-2. Example 1 circuit.

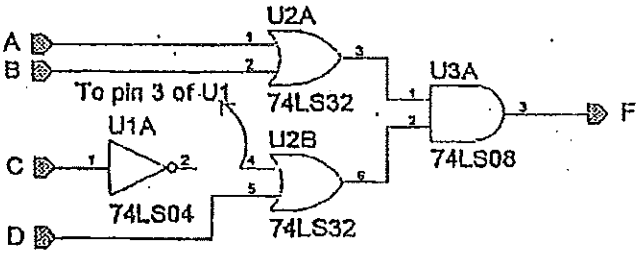


Fig. 5-3. Example 1 circuit with an error.

When we test the circuit, we discover that under the input combination ABCD = 0110, the output is 1 instead of 0. We know that something is wrong. A debugging process may be composed of the following steps.

1. Find all expected signal values on the logic diagram for the selected test vector (in this case, ABCD = 0110) as shown in Fig. 5-4. From the expected signal values, we immediately suspect that the wrong output is probably caused by the lower input of the AND gate U3A since an output of 1 can only be produced by 1s at both inputs.

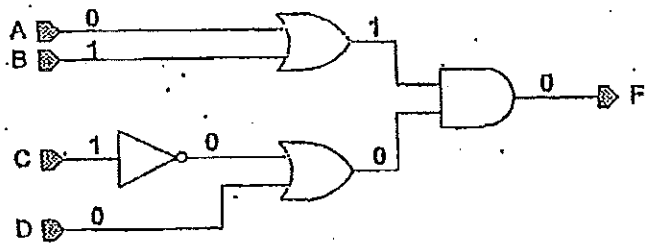


Fig. 5-4. Example 1 circuit with the expected input and output values indicated.

2. Use a logic probe to trace the signal back from the output. First, check the primary suspect, pin 2 of the AND gate U3A. It reads 1, which is wrong (our suspicion is confirmed.) The incorrect output of the AND gate is due to this erroneous input signal.

3. Tracing the faulty signal back towards the input we find that the signal at pin 6 of the OR gate U2B is also at logic-1. So the error is probably not caused by the interconnection between the output of the OR gate U2B and the input of the AND gate U3A. At this point, we suspect that one of the inputs for the OR gate U2B may be at logic-1 instead of 0.

4. Trace the signal further back and we come to pin 4 of the OR gate U2B. It indicates that the signal is floating. We therefore suspect that the problem is probably due to the wiring of this input. A quick check at pin 2 of the inverter shows a logic value of 0. This confirms our suspicion that there is a bad connection, since this 0 is not getting to the input of the next gate. A close examination of the connection uncovers the wiring error.

The overall testing process and outcome are summarized below.

pin #	gate #	expected	observed	remark
3	U3A	0	1	Wrong; suspect input pin 2.
2	U3A	0	1	Wrong; trace the connection to its source - U2B pin 6.
6	U2B	0	1	Wrong; so the wire seems to be ok; need to check inputs of U2B.
4	U2B	0	-	Aha, there is the problem; is it due to the gate before it? Check U1A pin 2.
2	U1A	0	0	This is ok; must be the wiring.
Check the wiring				Eureka! Found the miss-connected wire.

Example 2. If the forward trace method is used to check out the circuit of Example 1, we may obtain the information summarized in the following table. In this case, we start from the first signal, pin 1 of U1A.

pin #	gate #	expected	observed	remark
1	U1A	1	1	Ok.
2	U1A	0	0	Ok; gate U1A seems to be ok.
1	U2A	0	0	Ok.
2	U2A	1	1	Ok.
3	U2A	1	1	Ok; gate U2A seems to be ok
4	U2B	0	-	Problem! must be the connection
Check the connection				Yes! Found the problem.

It is difficult to say whether the forward trace or the backward trace is better. Depending on where the fault lies, one method may be faster in locating the fault than the other, but there is no way to predict which one beforehand. On the average, the backward trace may work slightly better because one can narrow down the faulty branches of the circuit a little faster.

IV. LABORATORY WORK

The "correct" circuit is shown in Fig. 5-5. Obtain the truth table (i.e., find f) for the "correct" circuit.

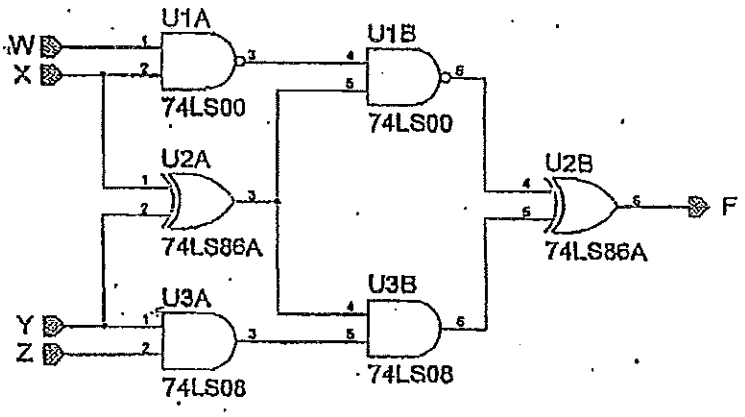


Fig. 5-5. The fault-free circuit.

W	x	y	z	f	g
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

In order to introduce a fault, connect the circuit as shown in Fig. 5-6 (i.e., pin 5 of the NAND gate U1B is connected to pin 3 of a wrong IC). Construct the faulty circuit of Fig. 5-6. List the observed outputs of this circuit, g, and record them next to f in the table shown on the previous page.

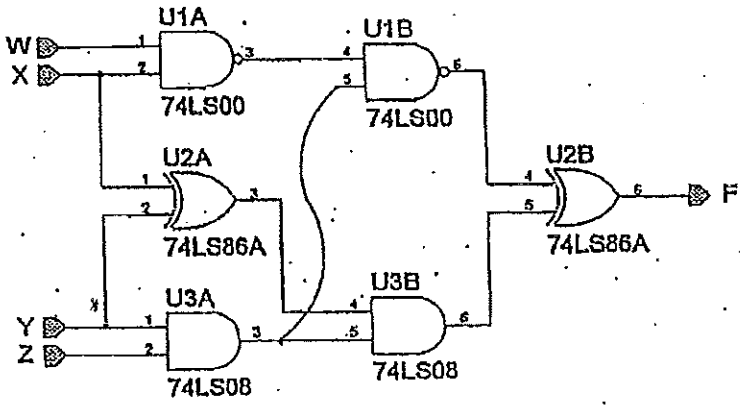


Fig. 5-6. Circuit with an intentionally introduced fault.

Choose one of the input combinations that produced an erroneous output (where f and g differ). Perform both backward and forward trace tests on the circuit and report your debugging procedures and results. Each lab partner should practice this separately, using a different input combination as the test vector.

Correct the error so that you have the original circuit of Fig. 5-5. Have your partner introduce a single error (read the caution below) without your knowing what it is. Find the error and describe your test procedure and observed results in your report. Each partner should have a different faulty circuit to work with. Note: even if you spot the error immediately by visual inspection, you must go through the signal tracing process just to gain some additional experience.

> **Caution:** Do not connect an output to Vcc, ground or another output. Also don't connect an output to an input that is already connected to another output. The safest way to introduce an error is to remove a wire or change an input to come from a different source.

V. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Make sure to include both lab partners' troubleshooting exercises.

Exp. 6: DECODERS AND MULTIPLEXERS

"If we do not change our direction,
we are likely to end up where we are headed."
- a Chinese proverb

I. OBJECTIVES

1. To learn the operations of basic decoders and multiplexers.
2. To learn the applications of decoders and multiplexers.

II. COMPONENTS

1	74LS04	hex inverters
1	74LS10	triple three-input NAND gates
1	74LS138	3-to 8-line decoder/demultiplexer
1	74LS157	quad 2-to 1-line data selectors/multiplexers

III. INTRODUCTION

Note: This introductory material includes the description of the design work required for the experiment. Pay particular attention to the material where the word *Design* appears.

Some logic circuits are so widely used in digital systems that semiconductor manufacturers have packaged them as ICs. Examples of these functional units are decoders, multiplexers, counters, and registers. The first two are combinational circuits and are the subjects of this experiment, while the other two are sequential circuits and will be the topics of future experiments. Since these functional units typically contain more than a dozen logic gates, they belong to the MSI group.

A. Decoder and Demultiplexer

A decoder is a combinational circuit that converts information from one coded form to another form, generally with more outputs than inputs. The most often used decoder converts an n-line binary code to 2^n-line outputs such as the 3-line-to-8-line decoder (74LS138). Another example of a decoding circuit is the BCD-to-seven-segment decoder (74LS247) which converts a BCD digit (4 lines) to seven outputs so that the decimal value represented by the four inputs can be displayed on a 7-segment display unit. The opposite of decoding is encoding. An encoder converts many independent input lines into a binary coded form. Some examples are the 8-line-to-3-line encoder (74LS148) and the decimal-to-BCD (10-to-4) encoder (74LS147). The logic

circuits needed to implement these functional units are rather straightforward and can be found in most introductory logic design books.

The 74LS138 is a 3-line-to-8-line (simply referred to as 3-to-8) decoder. The data sheet for the '138 indicates that it has 3 inputs (A, B, and C), 8 outputs (Y0, ..., Y7), and 3 enable lines (G1, G2A, and G2B). G1 is an active-high enable input while G2A and G2B are active-low enable inputs. These enable inputs provide control of the outputs, and also facilitate interconnection of many such units to form a larger decoding network. The outputs are all active-low. Study the data sheets of '138 and make sure that you understand its operations completely before you perform the design work.

A major application area for decoders is address decoding. Larger decoders can be built from smaller decoders. Consider a simple memory unit that has 64 words (0 - 63) of storage. To address each word (location) in this unit, 6 address lines are needed, and therefore eight '138s are required for decoding (8X8=64). Assume that address lines are A5, ..., A0 and they appear only in uncomplemented form. ***Design***, using a '138, the part of a decoding network that selects words 40-47 (in binary, 101000-101111). The three enable inputs can be effectively used here to perform partial selection (decoding).

Some specific information can be "extracted" from encoded bits using decoders. For example, a certain microprocessor generates three status signals, S1, S2, and S3, for external use. They represent the status of the processor as shown in the table below.

S1 S2 S3	Processor States
0 0 0	bus idle
0 0 1	op-code fetch
0 1 0	memory write
0 1 1	memory read
1 0 0	output
1 0 1	input
1 1 0	interrupt acknowledge
1 1 1	reserved (unused)

A decoder such as the '138 can obviously generate all eight processor states from the three status signals. Furthermore, if we are also interested in knowing when the bus is in the output mode (F1), or when the memory is involved (F2), all we have to do is to combine the appropriate decoder outputs. F1 would include both "memory write" and "output" states. F2 would encompass "memory read", "memory write", and "op-code fetch". ***Design*** a logic circuit

to implement F1 and F2 by logically combining the appropriate decoder outputs. Note that decoder outputs are active-low while the desired F1 and F2 signal are active-high.

Since a decoder can also be used as a demultiplexer, the data books typically refer to these devices as decoders/demultiplexers. When the device is used as a demultiplexer, the "enable" input of the decoder actually becomes the "data" input of the demultiplexer; and the "data" input of the decoder becomes the "selection" inputs of the demultiplexer. Figure 6-1 shows how '138 can be used as (a) a decoder and (b) a demultiplexer. Note that we could have chosen any one of the three enable signals in the '138 as the "enable" or "data" input.

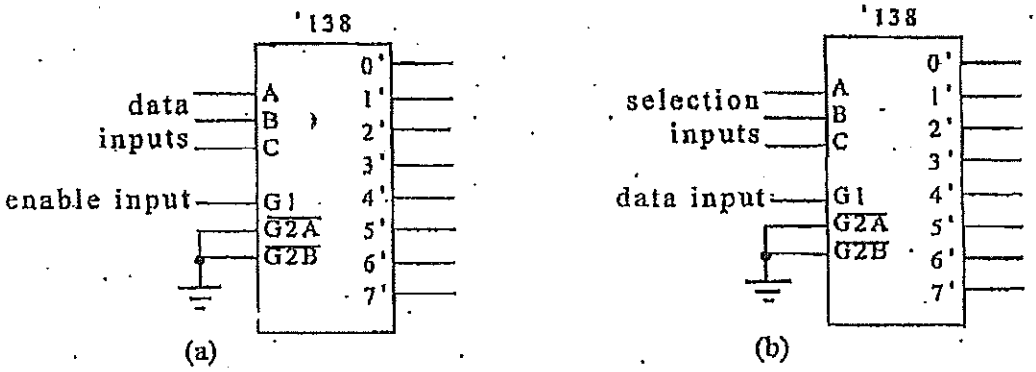


Fig. 6-1. '138 used as (a) a decoder and (b) a demultiplexer.

B. Multiplexer

A data selector or multiplexer (nicknamed MUX) selects one of many input lines to appear at its single output line. Which of the input lines to select is dependent on the value of the control (select) lines. Examples of MUXs are the 74LS157 (2 input lines) and the 74LS153 (4 input lines). The inverse function of multiplexing is demultiplexing, which transmits information from a single source to one of many possible destinations.

The 74LS157 is a quad 2-line-to-1-line (or simply 2-to-1) data selector. It is like a two-position rotary switch that can connect one of the inputs to the output depending on whether the selection input is high or low. There are 4 identical 2-to-1 MUXs in the '157. 1A and 1B are inputs and 1Y is the output for the first MUX, 2A, 2B, and 2Y for the second, and so on. The selection input, S, however, is common to all four MUXs. So the four MUXs must either all select A or all select B. The "strobe" (STB) is like an enable signal. The outputs are valid only if the STB input is "true" (since it is an active-low input, so "true" means low). Study the data sheets '157 carefully before you proceed with the design task.

Multiplexers can also be used as logic elements for combinational circuit design. ***Design*** a logic circuit to implement the following four functions simultaneously using one '157.

$$G1 = x'y,\ G2 = x \oplus y,\ G3 = x',\ G4 = y'$$

Please remember that the '157 has a common select input for all four MUXs.

Multiplexers and demultiplexers can be combined to provide any kind of data path for switching or routing. ***Design*** a data routing circuit which can send a single bit information from any one of two inputs to any one of eight outputs by cascading a 2-to-1 multiplexer and a 1-to-8 demultiplexer. The selected output should copy exactly the data applied at the selected input. A block diagram illustrating the process is shown in Fig. 6-2.

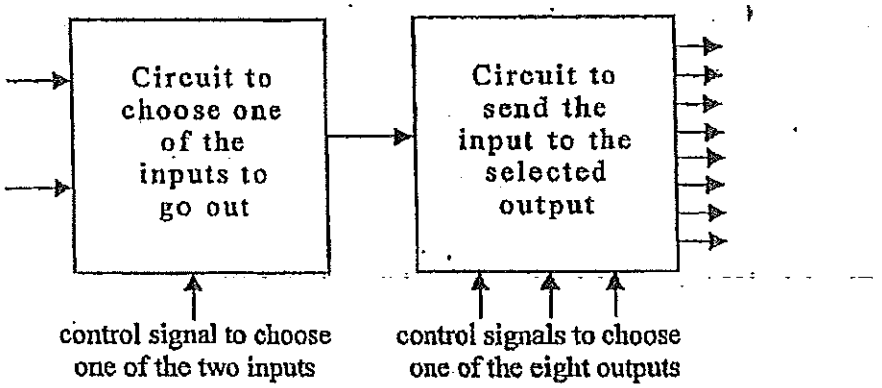


Fig. 6-2. Block diagram for the data routing circuit.

IV. PRE-LAB

Here is a brief summary of the pre-lab work that was described in the previous section. Please read Section III, if you haven't done so already, before you attempt to do the design work.

1. Design, using a '138, a decoding network that will recognize decimal addresses 40-47 (in binary, 101000-101111).
2. Design a logic circuit to implement two functions, F1 and F2 described in the previous section, by making use of a decoder.

3. Design a logic circuit to implement the four functions, G1 - G4 described in this section, simultaneously using one '157.

4. Design a data routing circuit which can send a single bit information from any one of two inputs to any one of eight outputs by cascading a 2-to-1 multiplexer and a 1-to-8 demultiplexer. The selected output must follow exactly the signal applied at the selected input.

V. LABORATORY WORK

Construct the four circuits that you have designed in the PRE-LAB Section and verify their operations using the digital trainer. Demonstrate the data routing circuit to your lab instructor and obtain his/her signature of approval.

Instructor's signature:_____. Date:_____

VI. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Make sure to include all design work and discuss the experimental outcome.

Exp. 7: ITERATIVE CIRCUITS- ADDERS AND SUBTRACTORS

"One-tree does not make a forest.."
-a Chinese proverb

I. OBJECTIVES

1. To learn iterative design techniques for large repetitive circuits.
2. To learn the design of basic binary add/subtract circuits.

II. COMPONENTS

3 nand2 quad 2-input NAND
3 xor2 quad XOR
Other devices as needed.

III. INTRODUCTION

A very attractive feature of digital computer arithmetic-logic circuits is their "regularity." Data in computer systems are usually represented in multiple bits known as words. Typical word lengths are 8, 16, 32, or 64 bits. Operations on all bits of a word are usually identical. Thus, one needs only to design the circuit for one bit and imply interconnect multiple copies of the same circuit to form a complete circuit for the word. This is known as the iterative design approach. Making copies of the same circuit is especially easy when CAD tools are used.

Full Adder:

Fig. 7-1 shows the block diagram representation of a one-bit adder. This adder receives one bit each (x and y) from the corresponding bits of the two operands and the "carry" bit (c) from the previous (less significant) stage. It must generate the sum (s) for the present bit position and a carry (k) to the next stage. This circuit is known as a full adder (FA). A FA, therefore, has three inputs and two outputs. *Design* this circuit Hint: a FA can implemented using two XOR gates and three 2-input NAND gates. Since in arithmetic operation, we work from the least significant bit towards the most significant bit, the carry signal will propagate from the right to the left. We'll draw these circuits, against our regular convention of signal going from left to right, with the carry signal going from right to left. Alternatively, we can draw the circuit with carry going from the top to the bottom.

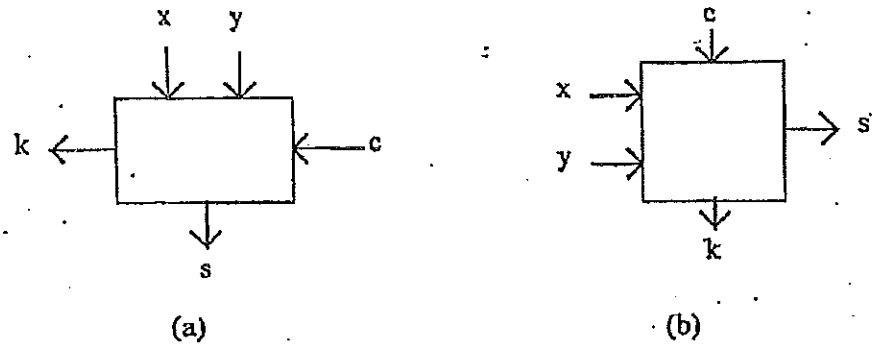


Fig. 7-1. Block diagrams for a full adder with the carry chain propagates
(a) horizontally from right to left and
(b) vertically from top to bottom.

Ripple Adder

When several stages of FAs are interconnected as shown in Fig. 7-2, it forms a ripple adder. This is the simplest kind of adder circuit. Since the full adder circuit is duplicated many times here, it is important that the full adder is designed using as few logic gates as possible. Note that in a ripple adder, the carry signal must propagate from one end to the other. It takes as many FA stages of delays as there are number of bits in the operand before the result becomes valid. Thus, if there are n stages with m seconds of delay in each stage, it takes (n x m) seconds to complete the addition. Many "fast adder" design techniques exist. But, in general, the faster adders require more hardware -- a classic speed-hardware tradeoff.

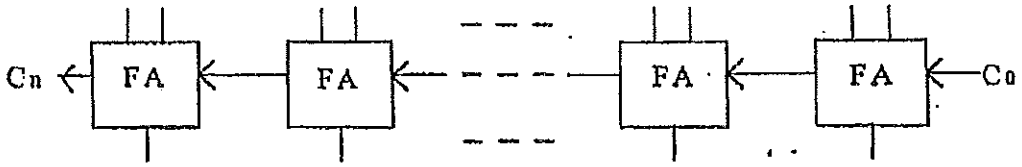


Fig. 7-2. A ripple-carry adder

IV. PRE-LAB

Design a full adder, as described in the previous section, using as few logic gates as possible.

V. LABORATORY WORK

1. Using the schematic capture software construct a full adder as designed in the PRE-LAB. Simulate the circuit to make sure that it is performing all desired functions.

2. Make four copies of the full adder and interconnect them into a 4-stage ripple adder. Experimentally complete the table below. The result is formed by the carry output of the most significant stage and the four sum outputs of the FAs (so there are a total of 5 bits). What should be applied to the carry input (C_0) of the least significant FA?

operand 1	operand 2	result
1010	0101	
1111	0001	
1111	0000	
0000	0000	
1100	0111	
0011	0101	

3. The adder circuit can be made into an adder/subtractor by inclusion of an add/sub control signal and a 1's complementer (XOR gates). A simplified block diagram of this circuit is shown below:

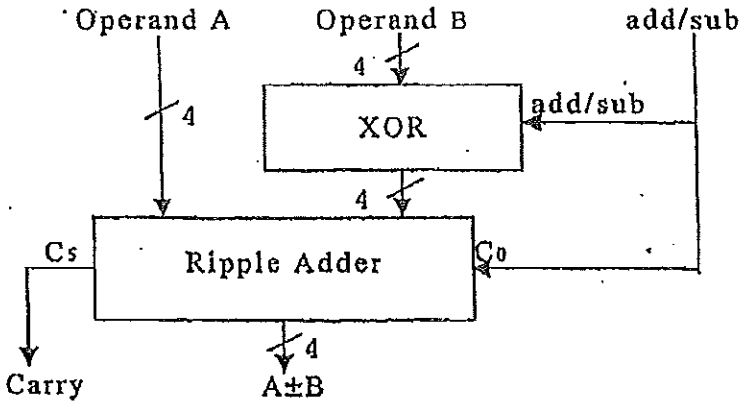


Fig. 7-3. A simplified block diagram of an adder/subtractor

4. Experimentally complete the table shown below. Make sure to include the final carry output (C_5) in the result.

operation	operand 1	operand 2	result
sub	1010	0101	
sub	0000	0001	
sub	0000	1111	
sub	1111	0000	
add	1101	0111	
sub	1011	0111	
add	1111	0101	

Demonstrate the operations of your adder/subtractor circuit to the lab instructor.

Instructor's signature:_____. Date:_____

VI. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Make sure to include the originals of all schematic and simulation printouts.

Exp. 8: LATCHES AND FLIP-FLOPS

"The beginning of wisdom is to call things by their right names"
-a Chinese Proverb

I. OBJECTIVE

To become familiar with latch and flip-flop circuits and their operational characteristics.

II. COMPONENTS

2 74LS00	quadruple 2-input NAND gates
1 74LS04	hex inverters
1 74LS08	quadruple 2-input AND gates
1 74LS74A	dual D type positive-edge-triggered flip-flops
1 74LS112A	dual JK negative-edge-triggered flip-flops

III. INTRODUCTION

If a specific output value is to be maintained in a combinational circuit, its required input combination must also stay the same. A latch or a flip-flop (FF), on the other hand, is a bistable device. That is, it has two stable states and stays in one of these two states unchanged even after inputs that caused the circuit to go into that state are "removed" (that is, become inactive). A latch or a FF, therefore, is a memory element. Each latch or FF is capable of storing one bit of information and, when properly interconnected, can form functional units such as registers, counters, memory, etc. These functional units are essential building blocks in digital systems.

Let's also review the pushbutton switch operation since one of these switches will be serving as the clock signal. Each sequence of push-then-release action produces a positive pulse at the NORM OFF terminal and a negative pulse at the NORM ON terminal. Remember that you need to use an inverter in order to get the complement of the signal. If a positive pulse is desired, then the NORM OFF terminal should be used. At the instant when the clock button is pushed, a low-to-high transition occurs at its NORM OFF terminal. This is known as the positive-edge of the clock pulse. When the button is released a high-to-low transition takes place and the NORM OFF terminal produces the negative-edge of the pulse. Generally, we use pushbutton switches to produce pulse signals (such as the clock) and toggle switches for level signals (such as data).

IV. PRE-LAB

Review the operations of latches and FFs. It is essential that you have a good grasp of their operational characteristics in order for you to understand the operations of the circuits contained in this experiment. Study the sections on latches and FFs in your textbook before coming to the lab.

V. LAB WORK

For the first few circuits of the experiment, each succeeding circuit will add a few more logic gates to that of the preceding one. So, do not disassemble a circuit until you are sure that it is no longer needed.

A. Basic SR Latch

Latches can be constructed from basic logic gates such as NAND, NOR, or Inverters. Fig. 8-1 shows two cross-coupled NAND gates. It is called an SR latch. More properly, it is an S'R' latch since it takes a logic-0 at these inputs to change the states of the latch. When a LOW signal represents an "active" signal, it is referred to as "active-low" or "low-true".

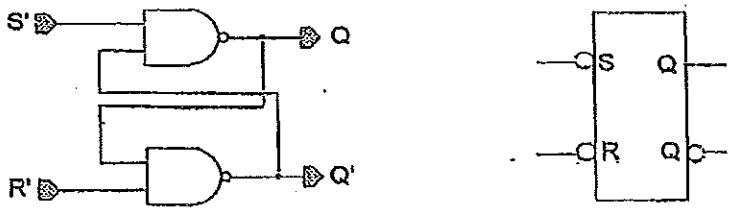


Fig. 8-1. Basic S'R' latch and its graphical representation.

The letter "S" stands for SET and "R" for RESET. A latch is ON or SET if its Q output is HIGH (hence Q' is LOW) and is OFF or RESET if Q is LOW (hence Q' is HIGH). Under proper operating conditions, Q and Q' are always complements of each other (that is why they are so labeled.) A logic-0 at the S' input causes the latch to be ON (the latch is SET) while a logic-0 at the R' input turns the latch OFF (the latch is RESET). When S' and R' are both HIGH (inactive), the latch is in its "memory" state. That is, the latch stays unchanged. The combination S'=R'=0 is not allowed since it makes both Q and Q' outputs high. The logic symbol representing the latch circuit is also given in Fig. 8-1. This latch circuit is a key part to all other latches and FFs. Construct the basic S'R' latch and verify its operations.

B. Clocked (Strobed) SR Latch

The outputs of a basic latch change "instantaneously" (ignoring the circuit delay) as the inputs are changed. In order to control the timing of the output change, Fig. 8-2 shows a clocked SR latch constructed by adding two NAND gates to the circuit of Fig. 8-1. In Fig. 8-2, the basic latch is represented by its logic symbol. In a clocked circuit, outputs will not change unless the clock is also present. The clock signal can be designed to come only if inputs have settled thus preventing unwanted changes due to different signal delays at the S and R inputs. The logic symbol for the new clocked SR latch is also shown in Fig. 8-2. Use the two remaining NAND gates in the same 74LS00 package to complete the circuit. The CK (clock) signal comes from the NORM OFF output of a pushbutton switch. A clock "pulse" is generated when the button is pushed and released. These switches are "debounced" to provide "clean" pulses. The circuit used to debounce a mechanical switch is exactly the S'R' latch you have just studied.

A clocked latch is best described by a table showing its "before clock" and "after clock" behaviors. The before clock values of S, R, and Q (the present state of the latch) will determine the value of Q+ and Q'+ after the clock pulse (the next state of the latch).

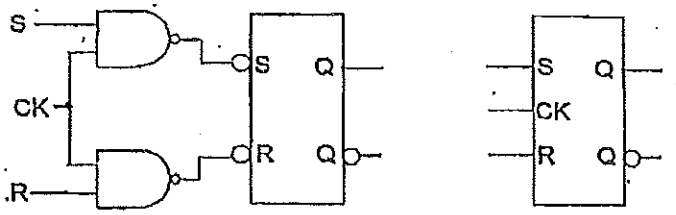


Fig. 8.2. Clocked SR latch and its graphical representation.

before CK			after CK	
S	R	Q	Q+	Q'+
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Experimentally complete the characteristic table to the right of Fig. 8-2. It is not necessary to follow any particular order when completing the table. Note that even under the "not allowed" (S=R=1) condition, the "after clock" state of the latch is almost always determined for any given circuit. This is because the two halves of the latch will "race" to a stable state as soon as the clock is removed. Since the two sides are never perfectly symmetrical, one side always wins the race. However, the designer cannot predict which one is going to win the race and, quite often, replacing an IC may change the outcome. Note also that any changes in the output occur as soon as the CK button is pushed. The clocked SR latch of Fig. 8-2 is not a true

synchronous flip-flop because when the clock is HIGH, a change in S or R produces immediate (asynchronous) changes in its outputs as prescribed by the characteristic table.

In summary, when the CK is absent (LOW), the S and R inputs have no effect on the latch and the circuit is said to be in the "blocked" mode. When the CK is HIGH, any change in S or R causes an immediate output response and the circuit is said to be in the "transparent" mode. When using this kind of latch, changes in S and R are usually made before the clock goes high. When the clock is high, S and R are held unchanged.

C. Clocked (Strobed) D Latch

A clocked SR latch can be converted to a clocked D latch by simply adding an inverter between the S and R inputs as shown in Fig. 8-3. The logic symbol for the D latch is also given in the same figure. The clocked D latch also has two modes of operations: transparent (or open) and blocked (or closed). When the Q output follows the changes at the D input instantaneously, the latch is in the transparent mode of operation. When the latch output does not react to changes at the D input, it is in the blocked mode of operation.

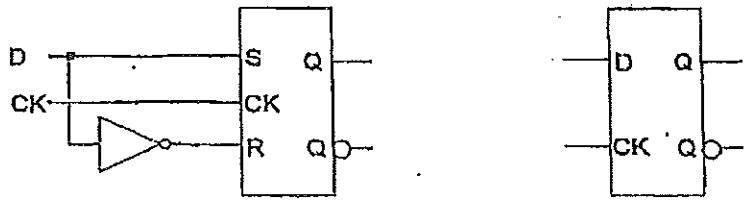
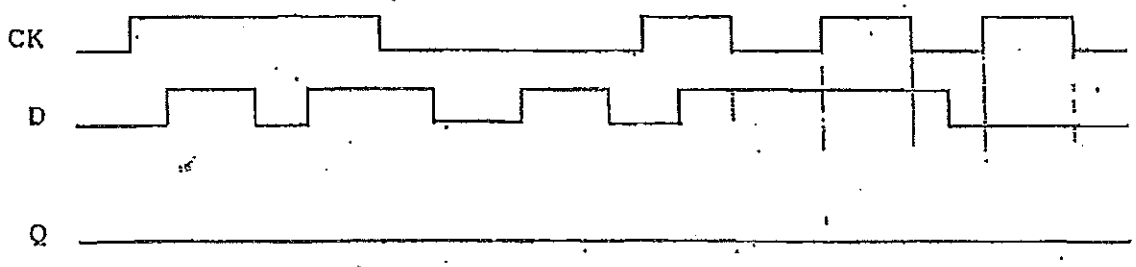


Fig. 8-3. Clocked D latch and its graphical representation.

Convert your clocked SR latch to a clocked D latch. The CK input is sometimes referred to as the "strobe" (STB) input since, when active, it "strobes" the input at D into the latch. Experimentally complete the Q output waveform based on the given CK and D inputs shown next.



D. D Flip-Flop

If we connect a clocked D latch to another clocked D latch in series as shown in Fig. 8-4, we have a D FF. The difference between a latch and a FF is that the state of a FF is changed only on a clock edge while the state of a latch can change as long as the clock is asserted. The first (left) latch is called the master while the second (right) latch is known as the slave. Note that the CK input for the slave is obtained through an inverter. This is to ensure that the master and the slave sections will not be in the transparent mode at the same time. Proper operation of this FF requires that the D input be applied a certain amount of time before the clock makes a high-to-low transition so that the master can catch the input. The master, in turn, applies its newly acquired output to the slave latch when the clock input changes to low. The output of the FF, therefore, changes at the negative-edge of the clock. The minimum time that the D input must be valid before the triggering clock edge is referred to as the setup time. The minimum time that the D input must stay valid after the triggering clock edge is called the hold time. All digital circuits that involve FFs must be designed to satisfy these timing requirements.

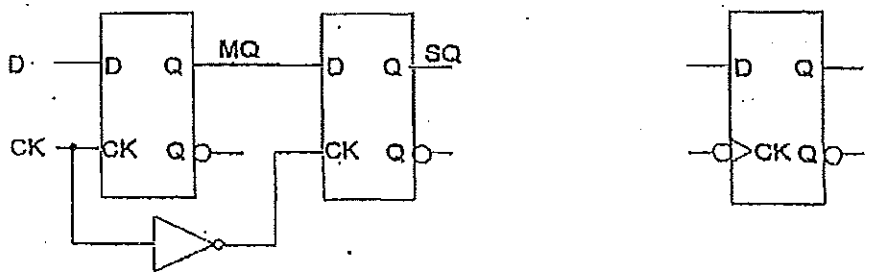
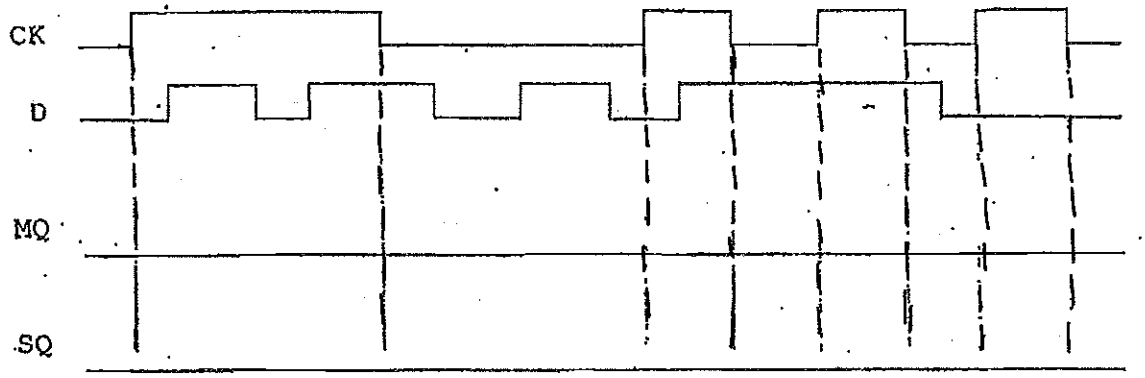


Fig. 8-4. Negative-edge triggered D FF and its graphical representation.

The graphic symbol ">" associated with the CK input indicates edge sensitivity. The small circle in front signifies that the CK is negative-edge sensitive.

Construct the D FF circuit as shown in Fig. 8-4. You already have one D latch; use another 74LS00 package for the slave section. Connect an indicator light to each of the latch Q output so that the behaviors of both the master and the slave can be monitored. Experimentally complete the MQ and SQ waveforms for the given CK and D inputs.



Demonstrate the operations of this circuit to the lab instructor.

Instructor's signature:_____. Date:_____

E. The 74LS74A

The full name for the 74LS74A is "dual D-type positive-edge-triggered flip-flop with preset and clear." Each D FF also has two direct inputs - PRESET' and CLEAR'. The inputs PRESET' and CLEAR' work just like S' and R', respectively, of the basic latch. The PRESET' and CLEAR' inputs can cause the FF to change without the activation of the clock and are called direct or asynchronous inputs. On the other hand, the D input can cause the FF to change only if the clock makes a low-to-high transition and is referred to as the clocked or synchronous input. Connect one D FF in the 74LS74A as shown in Fig. 8-5. We use the abbreviations PRE for PRESET and CLR for CLEAR.

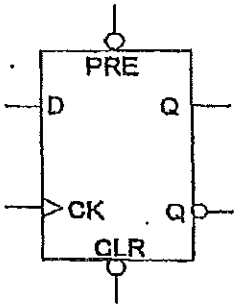


Fig. 8-5. ½ of the 74LS74A.

Answer the following questions by experimenting with the D FF in the 74LS74A.

- What is the active level for the asynchronous inputs? Do these inputs need the presence of a clock in order to change the output?
- What kind of signals must be applied to the asynchronous inputs for the FF to be ON? OFF?
- What should be done with the PRE and CLR inputs if you just want a positive-edge triggered D FF?
- The manufacturer's data sheet for the 74LS74A uses the following symbols in the function table. What do they mean?

"x" : _____.

"↑" : _____.

"Q_0" : _____.

F. The 74LS112A

The 74LS112A, contains two negative-edge-triggered JK FFs with direct PRESET and CLEAR. Connect one JK FF in the 74LS112A package as shown in Fig. 8-6. Check to see if your circuit operations agree with those given in the manufacturer's data sheet. Pay particular attention to direct inputs vs. clocked inputs. What should you do with PRE and CLR inputs if you just want to use it as a simple JK FF?

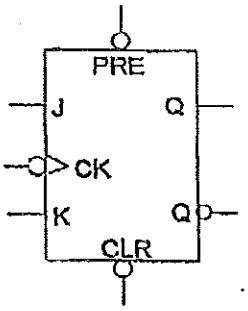


Fig. 8-6. ½ of 74LS112A.

G. Using JK Flip-Flops

A JK flip-flop with preset and clear is perhaps the most popular FF because of its versatility. A JK FF such as the one in 74LS112A can be easily modified to perform the functions of other FFs. If only PRESET and CLEAR inputs are used, we have the basic latch. Recall that in an SR FF, S and R cannot both be 1 at the same time. If we observe the same restrictions in a JK FF, it, in effect, is an SR FF with S=J and R=K. If an inverter is used between the J and the K inputs, we have a D FF. If J and K are tied together, we have a T FF. This last circuit is shown in Fig. 8-7. Construct the T FF and fill in the characteristic table.

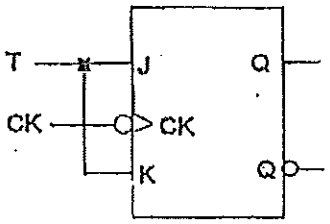
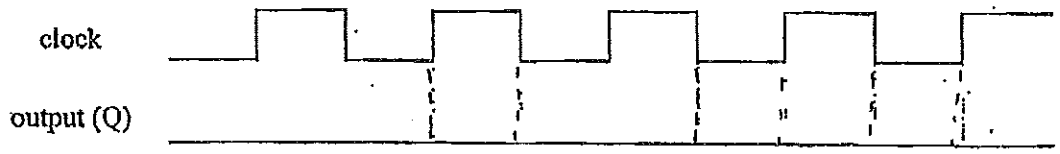


Fig. 8-7. T FF from JK FF.

before CK		after CK
T	Q	Q+
0	0	
0	1	
1	0	
1	1	

For the T FF, if the T input is kept at logic-1, the output is "toggled" to an opposite state every time the clock makes a negative transition. If the clock waveform shown below is applied to the T FF with T=1, complete the output waveform.



VI. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments that you may have. Make sure to include the answers to specific questions raised in the experiment.

Exp. 9: SEQUENTIAL CRICUIT DESIGN

"Reduce a major problem to a minor one;
Reduce a minor problem to none."
-a Chinese proverb

I. OBJECTIVES

1. To practice sequential circuit design.
2. To learn systematic construction and testing of sequential circuits.

II. COMPONENTS

1 74LS112A dual JK negative-edge-triggered flip-flops.
1 74LS08 quadruple 2-input AND gates.
1 74LS04 hex Inverters.
1 74LS32 quadruple 2-input OR gates
Other logic gates as needed (limited to SSI circuits).

III. INTRODUCTION

A digital computer performs its tasks by first entering a "fetch cycle" to obtain a machine instruction from memory and then going through an "execution cycle" to perform operations by the instruction. When the instruction is completely executed the machine go into a fetch cycle again and the process is repeated. Each machine instruction contains a field known as "op-code". After an instruction is fetched from memory into the processing unit, the op-code is stored in a register called the Instruction Register (IR). This op-code stays in the IR until it is replaced by an op-code from another fetch operation.

The HU-356 computer takes one clock period for the fetch operation but may take from one to *three* clock periods for execution depending on whether the instruction is in group 1, 2, or 3. The first group of instructions takes only one clock period to execute while the second group takes two clock periods and the third, three. In this machine, the last two bits of an op-code (IR1 and IR0) actually indicate the group to which the instruction belongs:

IR1	IR0	Instruction group
0	0	1
0	1	2
1	1	2
1	0	3

Note that both combinations (0 1) and (1 1) represent group 2 instructions. In other words, if IR0 is 1, it is a group 2 instruction regardless of IR1. If IR0 is 0, then IR1 determines whether the instruction belongs to group 1 or group 3.

If we let

F represent the fetch cycle,
S1 represent the first clock period of an execution cycle,
S2 represent the second clock period of an execution cycle, and
S3 represent the third clock period of an execution cycle,

then we may rephrase the problem as the following (X= either 0 or1. i.e., don't care):

If (IR1 IR0) = (0 0), the sequence is F, S1, F, ...
If (IR1 IR0) = (X 1), the sequence is F, S1, S2, F, ...
If (IR1 IR0) = (1 0), the sequence is F, S1, S2, S3, F, ...

The problem, therefore, is one of a sequencer design. The sequence, however, varies depending on the inputs, IR1 and IR0. F, S1, S2, and S3 are the "states" of the sequencer where F alone represents the fetch cycle and S1, S2 and S3 are all parts of the execution cycle. You can assume that IR1 and IR0 will only be changed in the state F (i.e., when a new instruction is loaded into the IR).

IV. PRE-LAB

Design a sequential circuit that will implement the state sequencer for the HU-356 computer as described in the INTRODUCTION Section. You must provide means for initializing (see item 1 in Section V) the circuit into a proper starting state (a digital computer always starts with a fetch cycle) and a visual indication of the current state. Use JK FFs in your design. The following steps are suggested design procedures.

1. Construct a state diagram showing the desired state transitions. The states are F, S1, S2, and S3. The inputs are IR1 and IR0. Do not worry about the outputs at this time.

2. Convert the state diagram into a state table.

3. Assign a unique two-bit code for each state and obtain a state transition table with encoded states.

4. Obtain excitation tables using JK FFs as the memory elements.

5. Obtain reduced logic expressions for J and K inputs with the help of K-maps.

6. Obtain four Boolean expressions, one for each indicator light. Each light represents a distinct state of the sequencer. You may recognize this as a 2-to-4 decoding circuit.

7. From logic expressions obtained in steps 5 and 6, draw a complete logic diagram with pin numbers indicated to facilitate its construction.

8. Add circuits needed for initialization. You may use any logic gates that are included in your kit but try to reduce the total number of IC packages in your design.

V. LABORATORY WORK

Because of the complexity of the circuit, it should be built and verified in stages. Try to follow the steps outlined below.

1. Prepare the two JK FFs. Connect the Vcc and GND lines. Connect both CLR inputs to a PB so that the FFs can be cleared by simply activating that button. Connect each PRE input to a separate PB so that each FF can be set individually. Use the fourth PB for clock and connect it to the CK inputs of both FFs. Make sure to use the correct PB outputs. Connect the Q output of each FF to a separate LT to monitor the state of each FF. Note that some of these connections are not required for implementing the design, but they will greatly facilitate testing and debugging later. Make sure that PREs, CLRs and LTs are all working properly before proceeding to the next step.

2. Construct circuits for the J and K inputs. You can get IR1 and IR0 from two TGs. Check to see if the FFs are following the desired transition paths. If so, go and buy a lottery ticket. This is your lucky day. If not, you are among the majority and will have to go through the debugging process. You can put the FFs in any state you wish with the CLR and PRE buttons. Then, check the values of J and K inputs for various possible conditions. If an FF input is not getting its expected values, you should check your design and the combinational circuit implementing that particular function. Alternately, you may wish to construct the circuit for one FF input at a time and check each circuit before building the next one.

3. Construct the state decoding circuit to drive four LTs, one for each state.

4. Verify the circuit operations using the IR1 and IR0 toggle switches and the CK pushbutton. If satisfied, you can replace the CK button with the clock generator on the power supply unit. This provides a continuous train of pulses. Use a frequency that gives you a comfortable visual observation (about one pulse per second is a good clock rate).

5. Demonstrate the operations of the circuit to your lab instructor when you are satisfied that it is working properly.

Instructor's signature:_____. Date:_____

VI. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. In particular, your lab report should include the following items:

1. The entire sequential circuit design process including state diagram, state table, excitation tables, K-maps, etc.

2. The final schematic diagram including pin numbers.

3. A discussion of the experiment.

Exp. 10: REGISTERS

"What cannot be cured, must be endured"
-a Chinese proverb

I. OBJECTIVES

1. To practice the design of data storage and shift registers.
2. To master the operations of a universal shift register.
3. To learn some of the applications of shift registers.

II. COMPONENTS

6 x74_174 D Flip-Flops
2 x74_153 dual 4-to-1 multiplexers
2 x74_194 4-bit universal shift register
Other IC also be needed.

III. INTRODUCTION

Registers lire simply groups of FFs that are uses collectively to perform functions such as data storage, shifting (either to the right or left) rotation. Serial-parallel data conversion, counting, and waveform generation. These FFs are usually interconnected except in simple data storage applications. Since registers are sequential circuits, they can be designed by following the normal sequential circuit design procedures. However, most of the register circuits can be designed "intuitively" or "heuristically" by using multiplexers.

The essential part of a register is the FF. To implement a shift register, the FF must be of the edge-triggered type. An edge-triggered FF that you are already familiar with is the 74LS74A. Each 74LS74A has two positive edge-triggered D FFs with PRESET and CLEAR. For simplicity, all PRE and CLR connections are omitted from logic diagrams but they should be properly connected in your experiment.

Data Storage Registers

A data storage register can be as simple as a group of FFs sharing a common clock and, quite often, a common clear signal. Data bits appearing at the inputs of these FFs are loaded into the FFs at the same time when the clock arrives. Fig. 10-1 shows four D FFs in parallel each data must appear at the

inputs of these FFs for every clock pulse. That is, you cannot keep the data in the register for more than one clock period. This is a very serious limitation for D FFs since clock pulses occur regularly in a digital system.

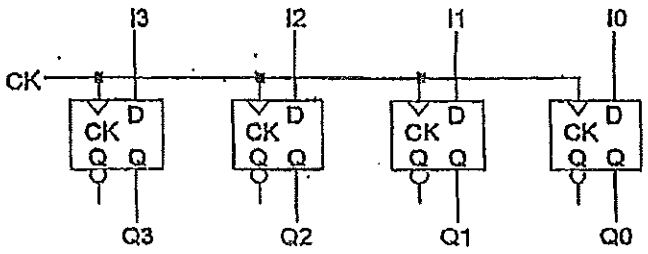


Fig.10-1. Four D FFs in parallel.

The above problem can be overcome by feeding the FF outputs back to their respective inputs and adding a control signal to select either the current data or new data as the input to the FFs. This is shown in Fig. 10-2.

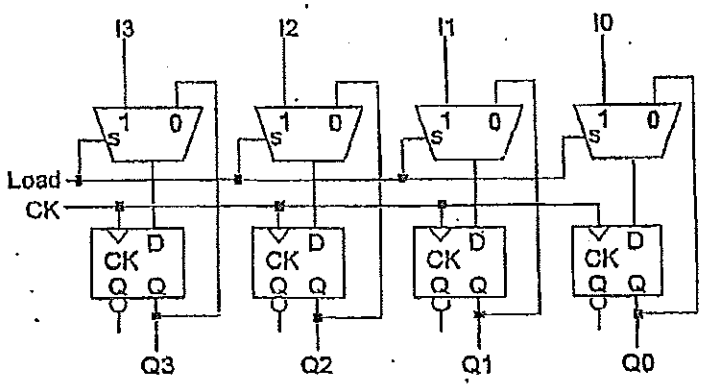


Fig. 10-2. A four-bit parallel load storage register.

Multiplexers can be used before the FF inputs to expand the number of data paths to the register. Each potential source represents an input to the MUXs. The selection of a specific source for loading at any given time is determined by the combination of the control (selection) signals applied to the MUXs at the time. If the input source is from the output of a neighboring FF of the same register, we have a shift register.

Shift Registers

A simple shift register looks very much like the circuit given in Fig. 10-2 except that the inputs, I0, I1, I2, and I3, are connected to the outputs of adjacent FFs. Fig. 10-3 shows a left-shift register. A LIN (left input) signal is included to provide serial inputs to the right-most FF. The connections of the i^{th} FF output to the $(i+1)^{th}$ FF input would be reversed (i.e., the $(i+1)^{th}$ FF output to the i^{th} FF input) for a shift right register. Again, MUXs can be used for selecting appropriate inputs if a bidirectional shift register is desired. This is shown in Fig. 10-4. RIN is the right serial input when shifting to the left. The control, S, determines whether a right shift or a left shift operation is to be performed. If the output of the last FF is connected to the input of the first FF, the shift is circular, also referred to as rotation. In an arithmetic right-shift, the sign-bit (the most significant bit) must be "extended." That is, do a normal right-shift operation except that "nothing" is shifted into the sign bit (so the sign-bit stays the same). Note: there are registers with built-in MUXs (e.g., 74LS298).

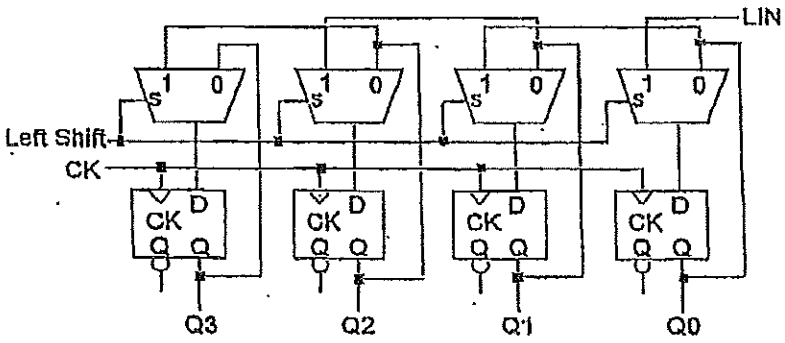


Fig. 10-3. A left-shift register.

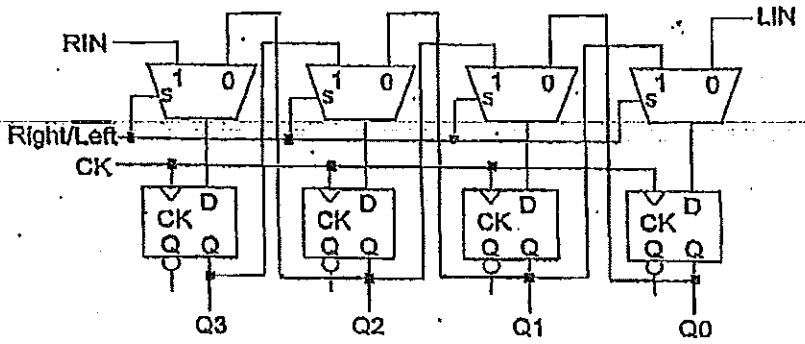


Fig. 10-4. A bidirectional shift register.

Notice that, in Fig. 10-4, all four bits of the register are available in parallel even though input bits are provided to the register serially. This means that a shift register can be used as a "serial-to-parallel converter." If we had "larger" MUXs (MUXs with more inputs), we could have also included external input lines to the MUX inputs. We would then be able to load data into the register in parallel and then shift them out serially one at a time, thus having the "parallel-to-serial conversion" capability. The 74LS194A has all these capabilities packaged in one IC and will be introduced later.

Ring Counters

A ring counter is really not a counter but a circular shift register with restricted operations. The restriction is that it can only have a single 1 circulating in the register. That is, one and only one FF of the register is on at any given time. In effect, it is a counter with a built-in decoder. The outputs of a 4-bit ring counter are the same as the outputs of a 2-bit binary counter followed by a 2-to-4 decoder. It is sometimes referred to as a pulse distributor since it parcels the input pulses (the clock) to separate outputs. A simple ring counter, however, is very susceptible to noise -- once the pattern is changed, the wrong pattern will continue to circulate. A "self-correcting" ring counter is shown in Fig. 10-5. The counter will always correct itself in no more than 4 clock periods. Note that a "self-correcting" ring counter is also "self-starting." That is, no matter what the starting pattern is, after no more than 4 clock periods, the ring counter will operate properly.

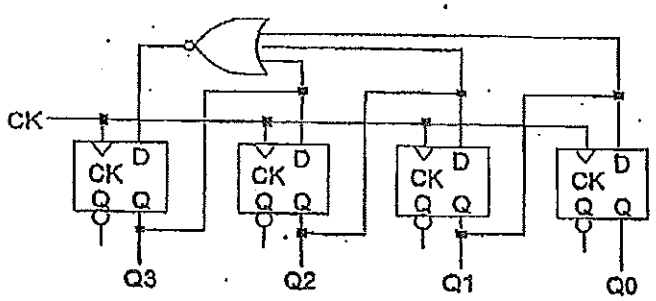


Fig. 10-5. A 4-stage self-correcting ring counter.

Linear Feedback Shift Register

A circular shift register with some XOR gates in its data path is referred to as a linear shift register. It is "linear" because the XOR operation is the same as mod-2 addition, which is a linear operator. In addition, a linear feedback shift register (LFSR) has, no surprise, a feedback path. Linear shift registers are used in many applications such as error-correcting codes, pseudo-random number generation, signature analysis (a form of testing), etc. For a 4-bit register there are 15 non-zero patterns. If the LFSR can generate all 15 patterns, in "random" order, before it repeats the sequence, it is a maximum length LFSR. For example, the circuit shown in Fig. 10-6

is a 4-bit maximum length pseudo-random number generator. Note that a LFSR with a starting content of all zeroes will continue to stay at zero. Depending on the placement of the XOR gate, other "random" patterns may be generated.

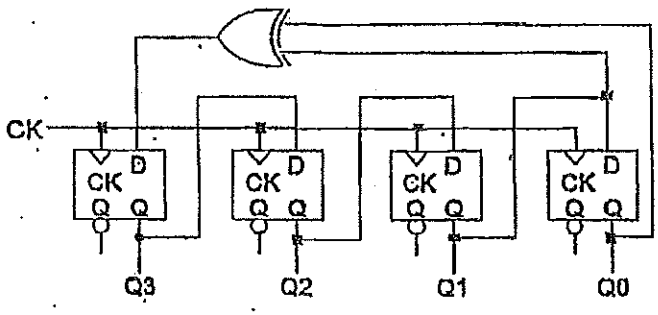


Fig. 10-6. A 4-stage linear feedback shift register.

The 74LS194A

The graphic symbol for the 74LS194A is shown in Fig. 10-7. It is a 4-bit bidirectional universal shift register. Internally, it consists of four D FFs each with a 4-to-1 MUX to select one out of four inputs. The four inputs to the MUX are the output from the left, the output from the right, external input, and its own FF output. Thus, it can perform right shift, left shift, parallel load, or remain unchanged. With a simple external connection (e.g., QD output to SHIFT RIGHT SERIAL INPUT), circular shift can be accomplished. Since there are four inputs to the MUXs, two selection signals (S1, S0) are required. The inputs at S1 and S0 determine the operating mode of the circuit:

S1	S0	mode	operation
0	0	0	no change
0	1	1	shift right (or shift down; from QA down towards QD)
1	0	2	shift left (or shift up; from QD up towards QA)
1	1	3	parallel load

Parallel load and shift are positive-edge-triggered synchronous operations while clear (CLR) is an active-low asynchronous operation.

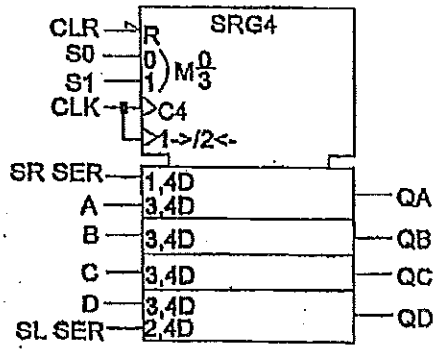


Fig. 10-7. 74LS194A.

IV. PRE-LAB

Study the data sheets of the devices listed in Section II. Do some preliminary design of the experiments to be performed (see next section).

V. LABORATORY WORK

This experiment is to be performed using the EDA software. Plan your test inputs carefully so that the verification process is compact yet complete.

1. Using 74LS74As and 74LS153s, design and verify the operations of a 4-bit register with the following capabilities:

S1	S0	mode	operation
0	0	0	no change
0	1	1	1's complement
1	0	2	shift right
1	1	3	parallel load

2. Make a copy of the circuit constructed in step 1 (as your instructor may want you to demonstrate that circuit) and, with minimal changes to the circuit, make the shift right operation into a 4-stage linear feedback shift register (still maintain all other functions). Starting with the pattern 1000, what is the sequence being generated by the LFSR? Make sure to show the sequence in the form of a timing diagram as one of your simulation outputs.

3. Using two 74LS194As, design and verify the operations of an 8-bit register with the following capabilities:

S1	S0	mode	operation
0	0	0	no change
0	1	1	arithmetic right-shift
1	0	2	rotate (circular shift) left
1	1	3	parallel load

4. Make a copy of the 8-bit register done in step 3. With minimal changes to the circuit, change the rotation operation into a self-correcting ring counter. Verify its operations, especially the self-correcting property. Make sure to include a timing diagram showing the pulse distribution feature. Demonstrate the operations of this circuit to your lab instructor.

Instructor's signature:_____, Date:_____

VI. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Also attach all schematic diagrams and simulation results.

Exp. 11: COUNTERS

"Learning without thought is labor lost."
-a Chinese proverb

I. OBJECTIVES

1. To practice the design of various counters.
2. To become familiar with the operation of a commercially available counter.

II. COMPONENTS

1	74LS00	quad two-input NAND gates
1	74LS04	hex inverters
1	74LS08	quad two-input AND gates
1	74LS10	triple three-input NAND gates
2	74LS112A	dual negative-edge-triggered JK flip-flops
1	74LS157	quad 2-to-1 multiplexers
1	74LS161A	synchronous 4-bit counter

III. INTRODUCTION

There are many different types of counters -- binary up-counter, binary down-counter, decade up-counter, decade down-counter, to name just a few. All counters are made of interconnected FFs. Since a counter is a sequential circuit which goes through a well defined sequence of states, it can be designed by following the formal sequential circuit design approach as you will do for the Pre-Lab. However, a counter usually has a large number of states and the formal sequential circuit design procedure becomes quite tedious. Since we have learned about FF operational characteristics, we can design most counters "intuitively" or "heuristically" without too much difficulty. For example, we know that the output of a T FF divides its input CK frequency by 2. If the output is used as the CK input to a second T FF, then the output of the second FF is the original CK frequency divided by 4, the next one would divide the input CK frequency by 8, and so on. The factors 1, 2, 4, 8, etc. are exactly the weights (multiplication factors) associated with binary numbers as we move from the least significant bit toward more significant bits. Thus, the circuit shown in Fig. 11-1 represents a 4-bit binary up-counter. JK FFs are used but they are configured as T FFs. Note that if COUNT=0, no FF can change its states and the counter is stopped. If COUNT=1, the falling edge of the CK pulses toggle the low bit of the counter. Every

time the low bit goes from high to low, it toggles the next higher bit of the counter, and so on. The CK input represents the pulses to be counted.

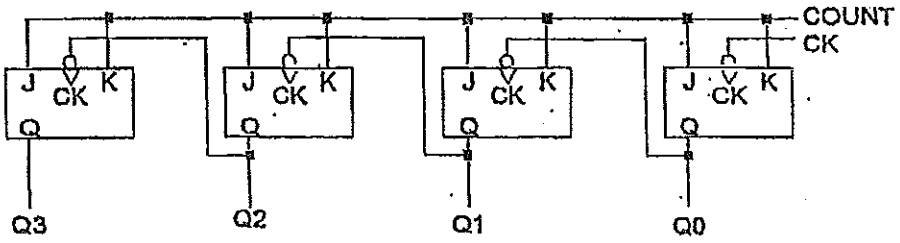


Fig. 11-1. A simple ripple counter.

The counter circuit shown in Fig. 11-1 is an extremely simple counter. It requires no additional logic circuits other than FFs. The problem with this counter is that the outputs change one after the other instead of simultaneously. It is also known as a "ripple counter" since the signal "ripples" from one FF to the next. This causes "glitches" in logic circuits and very long delays in large counters. For these reasons, we will concentrate on synchronous counters in which all FF changes take place at the same time.

In order to avoid cluttering logic diagrams, the following connections are not shown in diagrams but must be made for all counters except that of 74LS161A: Note: TGs and PBs may be replaced by input ports, and LTs replaced by output ports .

- All FF PRE inputs are connected to a single TG and with the TG normally set to 1.
- All FF CLR inputs are connected to the NORM ON output of a single PB.
- All FF Q outputs are connected to LTs in the same left to right sequence (DCBA) as given in the logic diagrams.
- The CARRY OUT, if it exists, is connected to the LT immediately to the left-most LT of the counter.

Binary Counters

By carefully observing the counting sequence of a binary up-counter, we find a specific pattern of changes:

> The least significant bit is toggled every count. All other FFs are toggled if and only if all of its lesser significant bits are 1s.

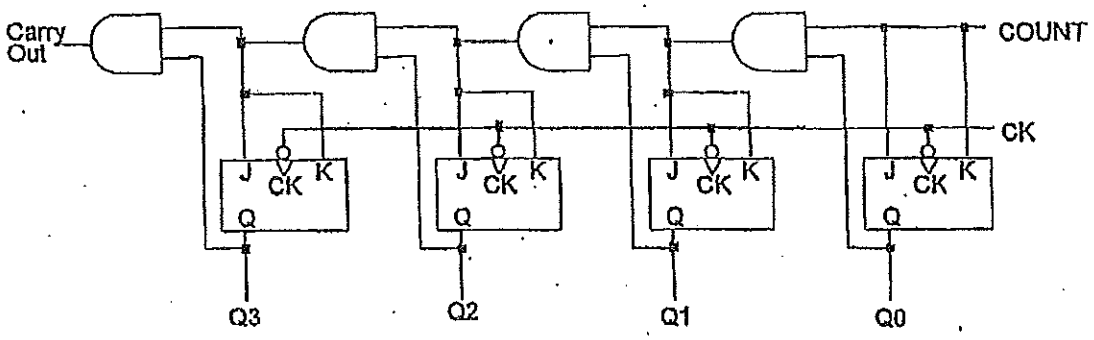


Fig. 11-2. A synchronous binary up-counter.

Therefore, for each FF, except the least significant one, we need a circuit to detect if all of its lesser significant FFs are at 1. The circuit shown in Fig. 11-2 does exactly that. The "COUNT" switch controls the counting. If COUNT=0, the counter is stopped since all J and K inputs are now 0. Note that the CARRY OUT is 1 if and only if all stages of the counters are at 1. If there is a COUNT ENABLE signal, it should be a part of the CARRY OUT signal. That is, if the COUNT ENABLE is deactivated, then the CARRY OUT signal should be deactivated as well. The CARRY OUT is generally used as the COUNT ENABLE signal for the next stage in multiple stage counters.

If we observe the binary down-counting sequence, we find that the least significant bit is again toggled every count but the more significant bits are toggled if and only if all of its lesser significant bits are 0s. Hence, all we have to do to convert the binary up-counter into a binary down-counter is to connect Q' outputs instead of Q outputs to the inputs of the AND gates. This circuit is shown in Fig. 11-3.

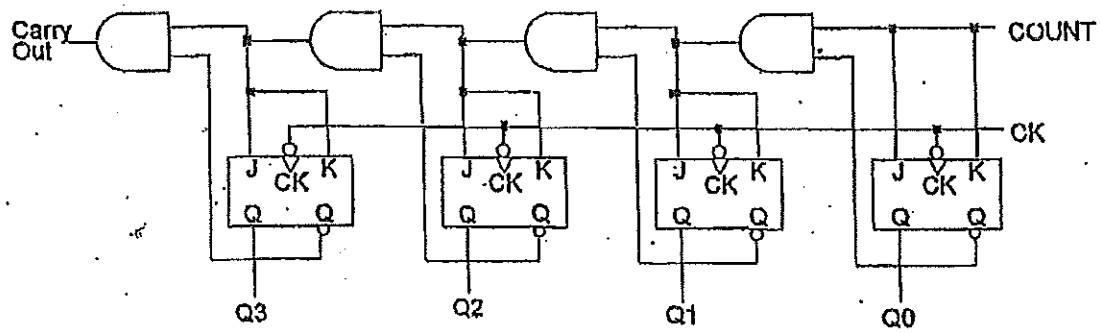


Fig. 11-3. A synchronous binary down-counter.

By adding four 2-to-1 multiplexers to choose either Q or Q' outputs, we have an up-/down-counter as shown in Fig. 11-4. Note that the UP/DOWN switch controls the direction of counting, low for UP and high for DOWN.

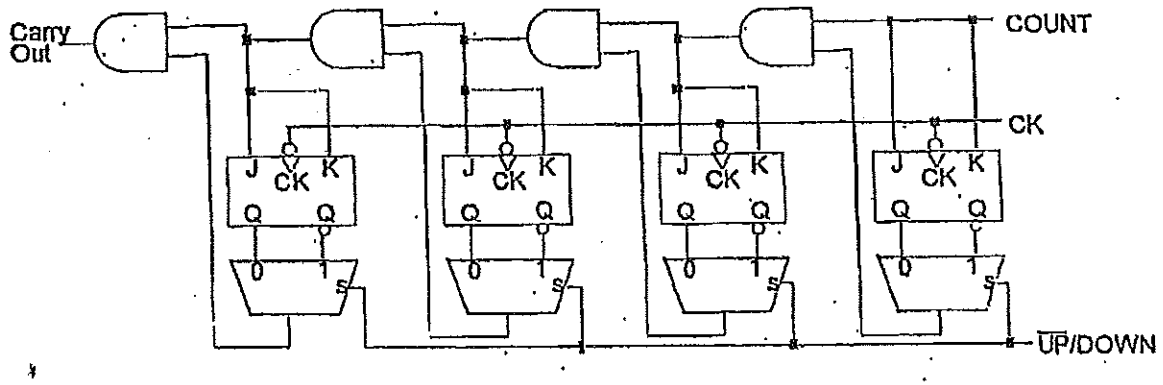


Fig. 11-4. A synchronous binary up-/down-counter.

Modulo-Counters

As you have observed, an n-bit binary counter goes through all 2^n possible combinations. There are times when counters that do not have exactly 2^n states are needed; these are modulo counters. For example, you will be asked to design a modulo-10 (decade) up-counter in the PRE-LAB. The CARRY OUT of a modulo-counter is generated when it reaches the final count. As with the binary counter, the COUNT ENABLE signal should be a part of the CARRY OUT signal in modulo counters. That is, if the COUNT ENABLE is deactivated, the CARRY OUT should be deactivated as well.

The 74LS161A Counter

The 74LS161A is a "synchronous 4-bit binary counter." The counter is "programmable" in the sense that it can be set to any desirable starting count. There are 4 parallel inputs, one for each counter stage. If the LOAD' input is low, the counter is in the load mode and the signals appearing at the parallel inputs are loaded into the counter at the positive-edge of the clock. If the LOAD' input is high, the circuit is in the counting mode. The circuit also has two enable signals, ENABLE P (EN-P) and ENABLE T (EN-T), and they must both be active if the counter is to advance. Thus, the circuit will count up by one at each positive-edge of the clock input if it is in the count mode and both enable inputs are active. The difference between EN-P and EN-T is that EN-T is used to enable the RCO (ripple carry output) while EN-P is not. Therefore, EN-P can be considered as the general enable signal and EN-T is the carry propagation enable signal. Note that RCO is generated only when the count is 15. The circuit also has an asynchronous, active-low CLR input which can be used to clear the counter to 0.

IV. PRE-LAB

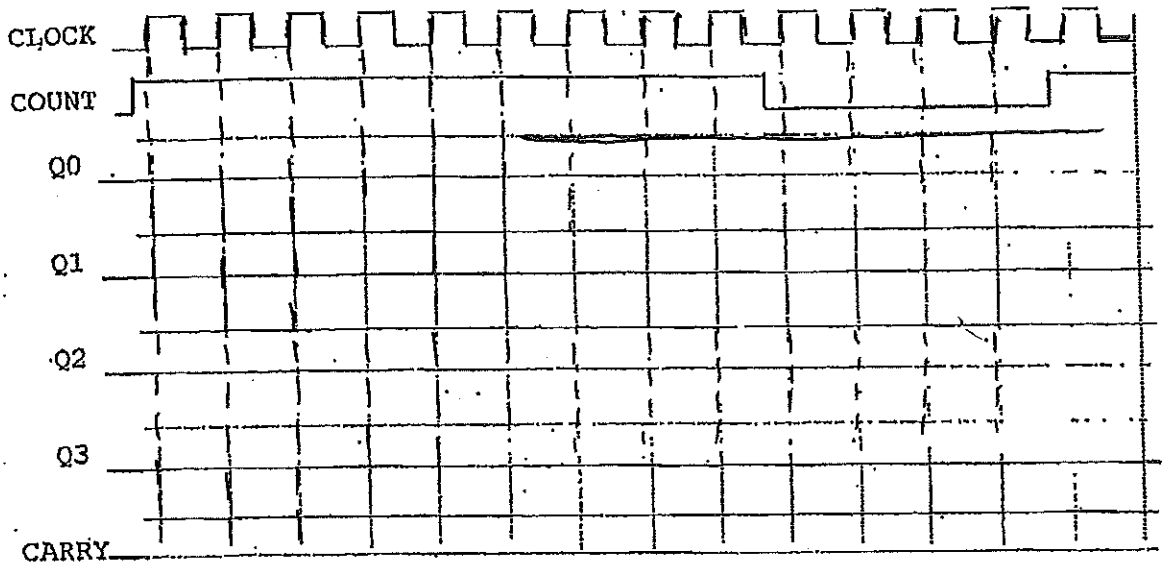
1. Design a decade up-counter using JK FFs. The counter should count from 0 to 9 and back to 0, etc.. Include a "COUNT" input to control the counter. The counter will advance only if COUNT=1. Also include a "CARRY OUT" output. This output is high when the counter is enabled and has reached the count 9. The CARRY OUT signal is used as the COUNT input for the next stage in a multi-stage counter. The CARRY OUT should be 0 if the COUNT input is 0. This will assure that all stages are disabled from counting. Attach your detailed design work as an appendix to this report.

2. The 74LS161A can be easily converted to a modulo-n counter, where n is between 2 and 15. A modulo-6 up-counter can be obtained by simply let it count from 0000 to 0101 and then load 0000 into the counter at the 6th clock pulse. Using the 74LS161A, design a mod-6 counter. Make sure that the counter does not advance (even from 5 to 0) when it is not enabled.

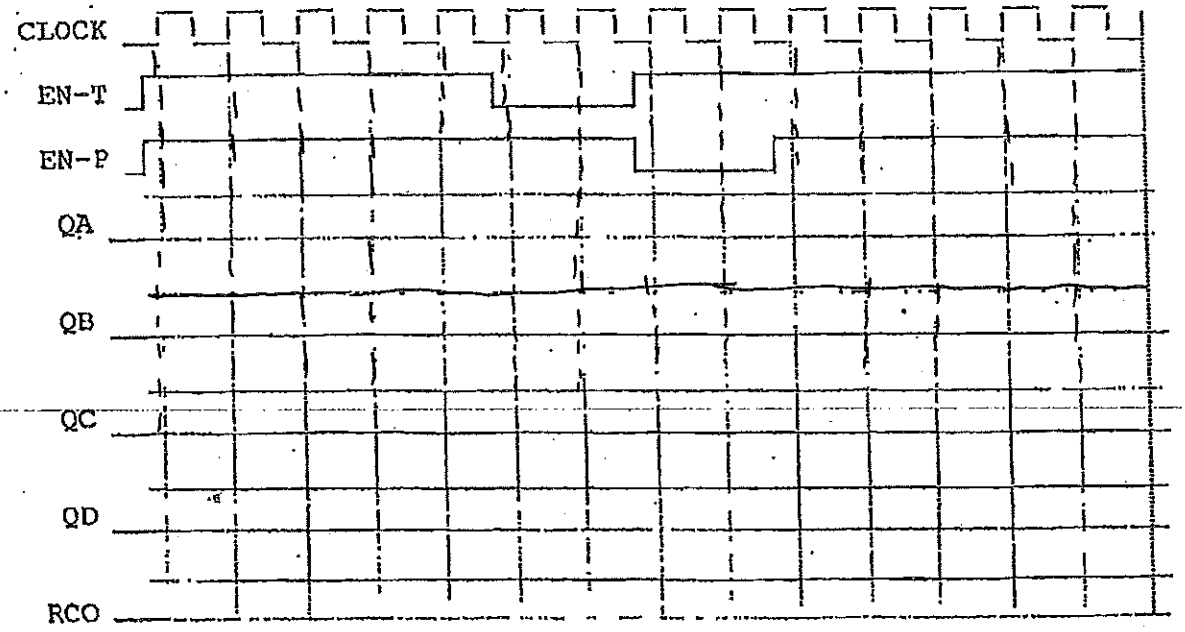
V. LABORATORY WORK

This experiment is to be implemented and simulated using the EDA tools. Include all schematic diagrams and simulation results in your report. Choose simulation inputs carefully so that the simulation is concise yet complete.

1. Enter the mod-10 counter (designed in step 1 of PRE-LAB) using the schematic entry software and simulate its operations. Make sure that when the COUNT input is disabled, the counter does not advance. (Try this test: advance the counter to 9 then disable the COUNT input; the counter should not advance when the counter is clocked a few times.) Also, when you disable the COUNT, the CARRY OUT should change from 1 to 0. Apply the "CLOCK" and "COUNT" inputs exactly as the waveforms shown on the next page and obtain the counter outputs. Do NOT delete this counter until the end of the entire experiment.



2. On the same schematic sheet as the mod-10 counter above, enter the modulo-6 counter circuit (designed in the step 2 of the PRE-LAB). Clock it many times and check the operations of the circuit. Apply the following "CLOCK", "EN-T", and "EN-P" signals to the simulator and obtain the output waveforms of the counter.



3. When you are satisfied that both the mod-10 and mod-6 counters are working properly, you are ready to combine them into a mod-60 decimal counter. The combined counter should count from 00 to 59 and back to 00 in BCD.

4. Modify the mod-60 counter of step 3 so that it stops at 59. That is, the counter counts from 00 to 59 and stops at 59 even if the clock continues to tick. Since counters have "COUNT" control inputs, if is quite easy to make these counters "self-stopping." All one needs to do is to recognize the stopping count and use it to disable the counter. Show the design of the self-stopping counter and demonstrate the workings of this counter to your lab instructor.

Instructor's signature:_____, Date:_____

V. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Make sure to include all schematics and simulation printouts.

Exp. 12: SEMICONDUCTOR MEMORY

"Give a man a fish and you feed him for one day;
teach a man to fish and you feed him for a lifetime."
- a Chinese proverb

I. OBJECTIVES

1. To learn the operations of a read-write memory.
2. To review the operations of three-state devices, counters, and registers.

II. COMPONENTS

1	74LS04	hex inverters
1	74LS161	synchronous 4-bit counter
1	74LS194A	4-bit universal shift register
1	74LS241	octal three-state buffers
1	2114	1024 x 4-bit static RAM

III. INTRODUCTION

The processor of a modern digital computer can process a lot of instructions and data in a very short amount of time. To satisfy the high capacity, high-speed storage function, the main memory becomes an integral part of a computing system. In addition to high capacity and high speed, the main memory must also be inexpensive, small in physical size and reliable. Semiconductor memories satisfy all these requirements.

The process of storing information in the memory is known as the "write" operation while the process of retrieving information from the memory is referred to as the "read" operation. A RAM (short for Random Access Memory but really means random access read/write memory) can perform both read and write operations at high speed while a ROM (short for Read Only Memory) can only perform the read operation. Programs and/or data are put into the ROM by some other means, generally a much slower process. A RAM is volatile (it loses its stored information when power is removed), but a ROM is non-volatile. ROMs are used to store programs or data that are not to be changed, which include the basic program to get the computer started.

Most of the semiconductor RAMs used in the main storage nowadays are of the MOS type. There are two basic types of semiconductor RAMs – static and dynamic. In a dynamic

RAM, memory cells are composed of switch transistors and charge-storing capacitors. The presence or absence of charge in a capacitor determines whether a 1 or a 0 is stored. Since charge stored in a capacitor tends to "leak" out, periodic refreshment of stored information is required. In a static RAM, memory cells are FFs that can maintain its stored information indefinitely. Static RAMs do not need refreshing but require more components per memory cell hence they have a much lower packing density.

RAMs come in many different sizes and organizations. A typical semiconductor RAM has (2^n x m) storage cells (one cell per bit) where 2^n is the number of words the memory can hold and m is the number of bits that can be can be moved in or out in a single memory operation. It also means that the memory needs n address lines to uniquely address each word and m input and m output data lines (or m bi-directional data lines). m is referred to as the memory word length. There are also control lines CS (or CE) and WE (or R/W'). CS (short for chip select) enables or disables the IC. WE (short for write enable) signifies writing when active and reading when not active. Both CS and WE are generally active-low signals.

Three-State Outputs

In digital circuits, it is absolutely forbidden to connect two or more regular logic circuit outputs together. However, in order to facilitate logic devices sharing common signal lines, some logic circuits are designed with three possible output states -- high, low and high-impedance (Hi-Z, or third state). A three-state (or tri-state) device requires an extra output enable signal. If the device is enabled, its output provides regular logic levels of high or low. When a three-state circuit output is disabled, its output is at the Hi-Z state. In the Hi-Z state, the output is, in effect, disconnected from any other circuit. This permits two or more three-state circuit outputs to share a common line as long as only one three-state output is enabled at any given time. Three-state devices are widely used as transmitters in "bus" oriented systems.

The 74LS241 has eight three-state buffers arranged in two banks of four. All four buffers in the same bank are enabled or disabled by a common control signal. One bank requires an active-high enable signal (2G) while the other bank requires an active-low enable signal (1G').

The 2114 RAM

The 2114 is a 4096-bit MOS static RAM organized as a (1024 x 4) storage device. It has 10 address lines (2^{10} = 1024), 4 bi-directional input/output (I/O) data lines, a CS line and a WE line. A manufacturer's data sheet for the device is included in Appendix B. Notice that the four data lines are bi-directional. If the device is not selected (CS is high), its data lines automatically assume the Hi-Z state. If CS is low and WE is high (read mode), the output three-state buffers in the memory are enabled to permit data from the memory to appear on the I/O lines. If CS is low and WE is low; the memory is in the write mode and will accept data from I/O lines and store

them in the memory. Since the data lines are used for both input and output transfers, be very careful when connecting circuits to these pins. All circuits that drive these data lines must go through three-state buffers. When the memory is in the read mode, all other drivers of the data lines must be in the Hi-Z state.

The Hexadecimal Display Unit

There are two identical hex display units on the digital trainer. Each display unit can provide the visual patterns of the hex digits from 0 to F. Each hex display unit has 4 inputs marked "A", "B", "C", and "D" for the 4 inputs required to give one hex digit. The unit also contains the decoding circuit that converts BCD to the LED display signals. "D" is the most significant bit and "A" the least. Thus, when DCBA = 0011, the display shows 3. The unit also contains 4 latches to capture and hold input values for display after the inputs are changed. The inputs are latched when the unit receives the "STB" signal.

IV. PRE-LAB

The overall block diagram of a general memory unit is shown in Fig. 12-1. In addition to the memory itself, two registers are needed: one to provide the address and one to provide/receive data.

Fig. 12-1. A general memory block diagram.

To simplify the design and reduce potential errors, the design of the memory system is divided into several steps.

The Address Register

Use the 74LS161 as the address register. When the counter is set to the counting mode, it can generate successive memory locations. Since the '161 is a 4-bit counter, it can only access 16 different locations. Describe how the following inputs of the '161 should be connected in order for it to function as a binary counter.

LOAD' :
CLEAR' :
ENT :
ENP :
A,B,C,D :

The Memory

Use the 2114 as the storage unit. This is a 1024 x 4 static RAM with 10 address inputs and 4 bidirectional data lines. Since the counter can provide only 4 address inputs, the remaining address lines can be connected to some fixed values. If the counter outputs are connected to the least significant 4 address lines, what would be the address range when A4-A9 are connected to the ground? To Vcc?

- Address range when A4-A9 are connected to the ground:

- Address range when A4-A9 are connected to Vcc:

The Output Data Register

Use the 74LS194A as the data register. However, the '194 is not a bidirectional register so it will be used only as a register to receive data from the memory. Show how the following '194 inputs should be connected in order to set it to the parallel load mode.

CLR' :
S1 :
S0 :
SR SER :
SL SER :

The Input Data Register

A 4-bit "switch register" (i.e., 4 toggle switches) will be used to provide data to the memory. Since these signals are applied to the bidirectional data lines, they must go through three-state

buffers. Use one half of the 74LS241 to drive the memory data lines as shown in the diagram. Note that these buffers must NOT be enabled when the memory is in the READ mode. Thus, it is a good idea to derive the enable signal (1G) from the same source as the WE signal to the memory. That is, the three-state buffers are enabled only when the memory is in the WRITE mode. Since WE is active-low, it is better to use bank 1 of the '241, which also uses an active-low enable signal, 1G.

The Memory Assembly

Fig. 12-2 shows the general structure of the memory assembly with some connections yet to be completed. Briefly, the '161 serves as the address register, the '194 serves as the output data register, and the combination of the switch register and the '241 serves as the input data register. Note that the '194 should only be loading when the memory is in the read mode. Complete the connections in the diagram so that data from the switch register can be written into different memory locations addressed by the '161 and data in memory can also be read back and displayed via the '194. After completing the design, answer the following questions.

- What are the address locations addressable by the counter?

- What is the step-by-step procedure for transferring data from the switch register to the memory (i.e., write operation)?

- What is the step-by-step procedure for transferring data from the memory to the '194 register (i.e., read operation)?

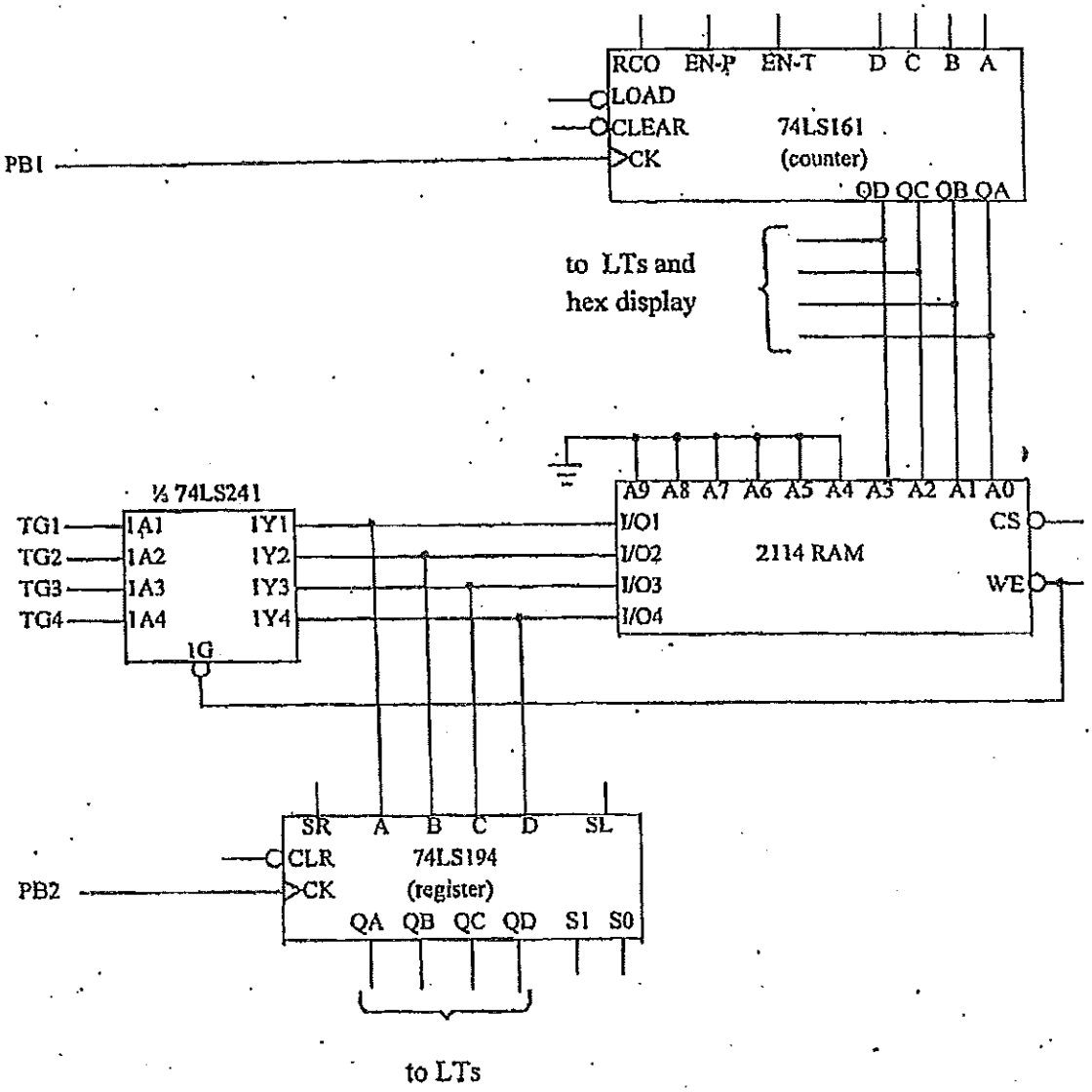


Fig. 12-2. The memory assembly.

- Memories (particularly ROMs) are also used to implement logic functions. This is referred to as table lookup method of implementing logic functions. It can be done by simply storing the outputs of the truth table in the memory storage cells. Show below how a random-access memory can be used to realize a 4-bit binary to Gray code converter by answer the following questions.

a) How many memory words are required?

b) How many bits are required in each word?

c) Give, in binary, memory locations and their contents to implement a 4-bit binary to Gray code converter.

memory location	content

V. LABORATORY WORK

Make sure that the logic diagram in Fig. 12-2 is complete with pin numbers indicated before you start the circuit construction.

1. Construct the circuit shown in Fig. 12-2. Since there are quite a few wires to connect, do it carefully and systematically.

2. Construct the two-phase, free-running clock circuit as shown in Fig. 12-3 in a separate area of the protoboard. Note that the outputs of the two inverters are named CK1 and CK2. Do not connect CK1 and CK2 to anything yet. These two signals will be used later but build them now when the power is off.

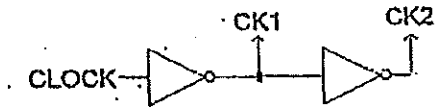


Fig. 12-3. A two-phase, free-running clock circuit.

3. Following the memory write procedure, try to write the bit pattern 1010 into a memory location and then read it back. If the display does not show 1010, your system is not working. If you do get 1010, try writing 0101 and the read it back. If this is also successful, the system is probably working properly. You are now ready to write more than one word into the memory.

4. Write the 4-bit binary to Gray code converter table into the memory unit.

5. Follow the memory read procedure to see if all data patterns are correctly stored. The data read from the memory should be the Gray code equivalent of the binary number in the address counter. If things are working properly, you can now switch over to using the free-running clocks instead of the pushbutton clocks. First, make sure that the memory is enabled and in the read mode. Without turning off the power, very carefully move the ends of two wires: one from PB1 to CK1 and the other from PB2 to CK2. You should see that the address is being incremented by the free-running clock and the contents of the consecutive memory locations are being displayed. Since CK1 and CK2 are 180° out of phase (i.e., they are the inverse of each other), the data coming out of the memory will be lagging the address by one half of a clock period. Demonstrate this table lookup implementation of the binary to Gray code converter to your lab instructor.

Instructor's signature_____, Date_____.

VI. POST-LAB

Write a report summarizing your experience with this experiment, lessons learned, and any other comments you may have. Make sure to attach the complete design and include answers to all questions raised in the experiment.

Exp. 13: TRANSFER AND SHIFT MICROOPERATIONS

"He who asks is a fool for five minutes,
but he who does not ask remains a fool forever."
- a Chinese proverb

I. OBJECTIVES

1. To become familiar with the register transfer notations.
2. To learn the hardware implementation of transfer and shift microoperations.

II. COMPONENTS

1	74LS139	dual 2-to-4 decoders
6	74LS157	quad 2-to-1 MUXs
4	74LS374	octal D FFs with three-state outputs

Other ICs as needed

III. INTRODUCTION

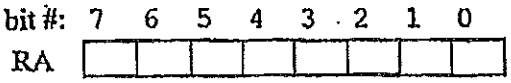
A digital computer accomplishes data manipulations simply by moving data from registers to registers and, during the transfer process, the data bits pass through some logic circuits where the desired operations are performed. Because of this, the language developed to describe hardware operations is often referred to as the register transfer language (RTL). Unfortunately, notations used in RTL will vary somewhat from user to user. The notations we will use in this and the next experiment are introduced in this section; it is by no means a complete set of RTL.

In this and the subsequent experiment, various basic hardware operations performed by simple digital computers will be introduced. These basic operations are called microoperations and they are directly implemented in hardware. A microoperation is generally executed in one clock time. One or more microoperations form what is called machine instructions. Symbolic notations representing machine instructions form the foundation of an assembly programming language. Here are some basic notations for our RTL:

- All registers will be represented by a capital letter followed by either more capital letters or numbers. Acronyms and abbreviations are usually employed for special function registers (e.g., PC for program counter, SP for stack pointer, IR for instruction register, MDR for memory data register, MAR for memory address register, ACC for accumulator register, XR

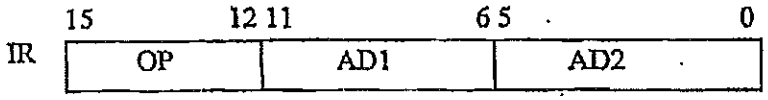
for index register, etc.) while RA, RB, RC, ... or R1, R2, R3, ... are used for general purpose registers.

- Register bits are numbered from right to left starting with 0 as illustrated below.



- If it is necessary to identify a part of a register, the register name with the bit numbers inside a pair of parentheses will be used. Thus, RA(0) means the right-most bit (bit-0) of the register RA and RA(3:0) refers to the right-most 4 bits (bit-3 to bit-0) of the RA register.

- Registers may be subdivided into "fields." For example, register IR may be subdivided into three fields as shown. In such cases, we can also refer to the bits of each field by using the field name inside a pair of parentheses instead of using the bit numbers.



Therefore, IR(15:12) = IR(OP), IR(11:6) = IR(AD1), and IR(5:0) = IR(AD2).

- Two or more registers or parts of registers may be concatenated by including the symbol "//" between the parts. For example, RA//RB means that RA and RB are considered as one entity with RB occupying the right part (and RA the left part) of the cascaded register.

- A left-pointing arrow, "←—", will be used to indicate data moving from the source (the tail side of the arrow) to the destination (the head side of the arrow). Thus, the destination register is written to the left of the arrow and the source register appears on the right side of the arrow. When performing data transfers, the source and the destination must have exactly the same number of bits. Examples:

MAR ←— PC	Transfer data from register PC to register MAR, bit-0 to bit-0, bit-1 to bit-1, etc. The content of PC is unchanged. So after the transfer, PC and MAR will have exactly the same contents in them.
IR ←— MDR(15:11)	Transfer bits 15 through 11 of the MDR register to the IR

register, in the same bit order. So the IR must be a 5-bit register and IR(0) copies MBR (11), ... IR(4) copies MBR(15).

RA ←— RA(14:0)//0 The least significant 15 bits of RA concatenated with a 0 (for a total of 16 bits) are transferred to RA. This means that RA must be a 16-bit register (bits 0 - 15) and the transfer is actually doing a logical left shift. So the operation will insert a 0 to bit-0 of RA, the old bit-0 becomes the new bit-1, ..., the old bit-14 becomes the new bit-15.

- Arithmetic/logic/shift operations may be included in register transfer notations. Standard operational symbols will be used with "+" and " · " reserved for arithmetic operations addition and multiplication, respectively, and "&" and "|" for logic operations AND and OR, respectively. Also, we will use "sln" for logic shift left n bits, "srn" for logic shift right n bits, "rrn" for rotate right n bits, and "rln" for rotate left n bits. Examples:

PC ←— PC + 1	PC is incremented.
ACC ←— RB&RC	ACC gets the result of bit-by-bit AND operation of RB with RC. (RB and RC stay unchanged.)
RA ←— RA + XR	RA is modified by adding the content of the XR register to it.
ACC ←— ACC' + 1	The number in ACC is replaced by its 2's complement.
RA ←— sl3 RA	Shift RA to the left 3 positions.

Point-to-Point (Direct) Transfer

One way to include data transfer capability is to have a dedicated set of wires for each possible transfer path. In a system such as a digital computer, the number of interconnecting wires would be too large to be practical using the point-to-point transfer method. For example, in a system with thirty-two 64-bit registers, there would be 32X32X64 = 65,536 wires if we want the registers to be completely interconnected. In addition, these registers are formed by 32X64 = 2048 FFs (or latches) so we need 2048 32-input MUXs in order to "combine" wires coming into the FF (latch) inputs.

Bus Transfer

In order to reduce the number of interconnecting wires and MUXs, many digital systems use one or more buses for data transfers. A bus is a common set of wires that allows different sources to send signals through the same wires at different times. This requires difference sources to be connected to the same set of wires. Since regular TTL (or CMOS) circuit outputs cannot be connected to each other, circuits with three-state or open-collector outputs must be used to "drive" the bus. Of course, the system must be designed such that only one device is driving the bus at any given time. A bus-oriented system requires fewer wires hence lower cost (or silicon area). A point-to-point system, on the other hand, has higher performance since it can transfer many data simultaneously via different sets of wires. In real applications, multiple buses are used as a compromise of the two interconnection schemes. This is a classical example of the time-cost (space) tradeoff.

Barrel Shifter

Shift registers were introduced in the REGISTERS experiment. However, most computers perform shift operations by a common circuit known as the barrel shifter. The shifter makes registers simple -- just simple data storage registers will do. Shift operations are performed by routing the targeted register content through the barrel shifter. A barrel shifter is a combinational circuit so it is fast. A simple barrel shifter performs only a single bit shift while a more complex barrel shifter can be commanded to perform a specified number of shifts in one pass. Fig. 13-1 shows an 8-bit barrel shifter that is capable of rotating the input data to the left by 0, 1, 2, or 3 bit positions. The circuit is composed of 16 2-to-1 MUXs and the number of bits rotated is controlled by S1 and S0 as shown on top of the next page.

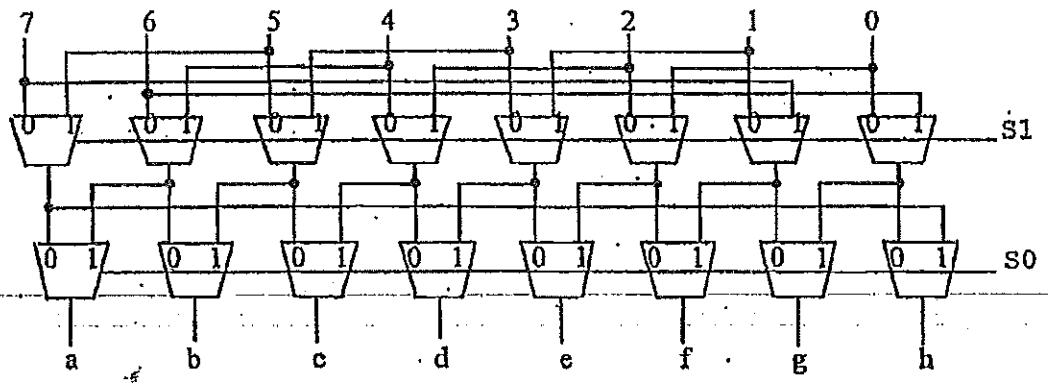


Fig. 13-1. An example barrel shifter.

S1	S0	operation
0	0	no rotation
0	1	rotate left by 1 bit
1	0	rotate left by 2 bits
1	1	rotate left by 3 bits

The 74LS374

The 73LS374 contains 8 D FFs with three-state outputs. The inputs of these 8 FFs are controlled by a common positive-edge triggering clock. The outputs of the register are controlled by a common active-low output control signal, OC'. That is, when OC' is low, the states of the FFs will appear at the output; when OC' is high, all outputs will be at their Hi-Z states.

Register File

When several storage registers with common functionality are grouped together, it is referred to as a register file. Generally, these registers will share common input and output buses. Since these registers share a common set of output lines, they must have three-state or open collector outputs. Fig. 13-2 shows the block diagram of a register file with two registers (R1 and R2), two output bus (BusB and BusC) and one input bus (BusA). It is assumed that R1 and R2 are simple data storage registers with regular outputs. Two sets of three-state buffers are needed for each register, one for each output bus. Different sets of buffers are controlled by different enable signals (*R1B, R1C, R2B,* and *R2C*) so that each register is capable of transferring data through either BusB or BusC. Each register is also capable of getting data from the BusA using the normal register input control (*R1A* and *R2A*). Buses A, B and C are generally connected to some other units such as an ALU (arithmetic-logic unit) or a barrel shifter.

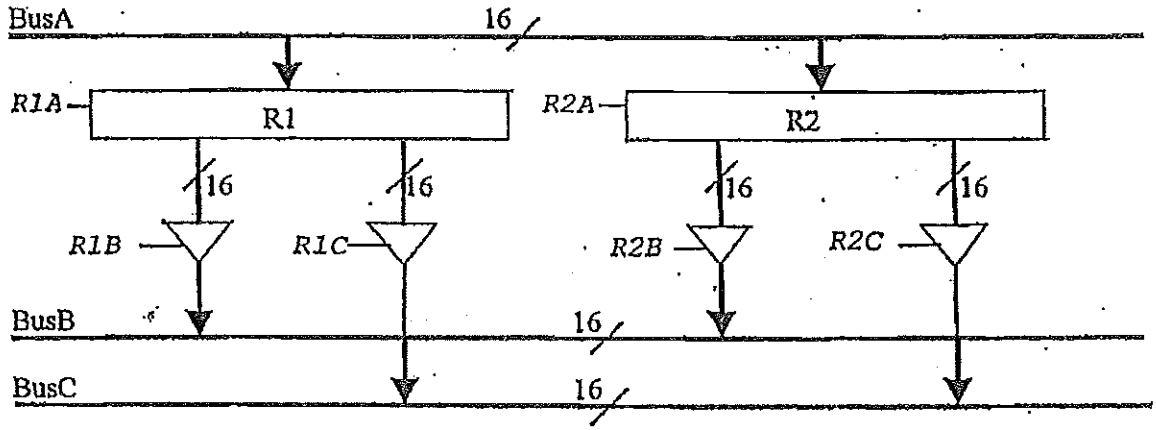


Fig. 13-2. Block diagram of a register file in a three-bus system.

IV. PRE-LAB

Study all sections of this experiment carefully and do a detailed logic design of the register file and the barrel shifter described in the next section. Make sure to include control circuits that enable register inputs and outputs.

V. LABORATORY WORK

This experiment is to be performed using the EDA software. Make sure to use bus notations in your schematic diagrams. Save your design for the next experiment. You may want to read the instructions for Experiment 14 to better plan your layout so that additional circuits needed for that experiment can be easily added to your circuit diagram. Depending on your instructor's decision, you may be able to consider this and the next experiment together as one two-week experiment. If so, you need only to complete and demonstrate the circuit of Experiment 14 since it include all the circuits used in this experiment. Plan your test inputs carefully so that the verification process is concise yet complete.

1. Design and verify the operations of a register file with 4 registers of 8 bits each, named R0 - R3. The register file is connected to one 8-bit input bus, and one 8-bit output bus. The circuit should permit the selection of one of the registers to output its content to the output bus and one of the registers to receive data from the input bus. The selected input and output registers may or may not be the same. Use the 74LS374 registers plus a minimum number of other ICs as needed.

2. Design and verify the operations of an 8-bit barrel shifter capable of rotating right 0 to 3 bits in one pass. Use the 74LS157 quad 2-to-1 MUXs. Use 74LS257 if you don't have the '157 in the component library. Note that the 74LS257 is also a quad 2-to-1 MUX. The only difference is that the '257 has three-state outputs. If these three-state outputs are always enabled, then it works exactly like the '157.

3. Connect the outputs of the barrel shifter to one of the inputs of 2-to-1 data selectors. Connect the other input of the data selectors to the outputs of a switch register so that the data selector may choose to pass on to BusA either the signals from the switch register or the barrel shifter. Connect the register file output bus, BusB, to the inputs of the barrel shifter and the outputs of the data selectors to the register file input bus (BusA). A block diagram of the general structure is shown in Fig. 13-3.

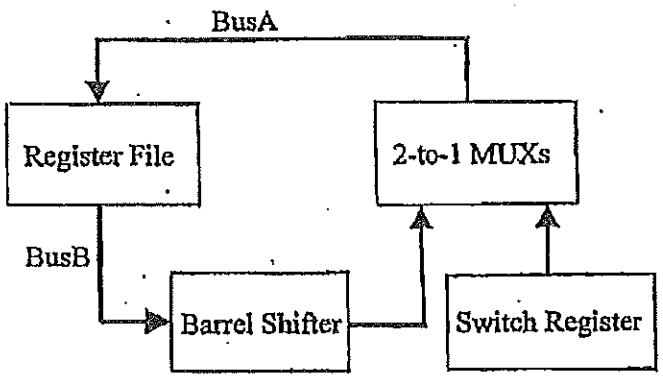


Fig. 13-3. Block diagram showing the interconnections.

4. Design other necessary circuits so that
   a. the contents of the switch register can be transferred to any of the registers in the register file.
   b. the contents of any register in the register file can be transferred to any other register in the register file.
   c. the contents of any register in the register file can be rotated to the right 0, 1, 2, or 3 positions with the result stored in any of the registers in the register file.

5. Simulate and demonstrate, one at a time, the following transfer and shift microoperations. Record the contents of all registers after each microoperation.

   R0 ← 10100011 (loading R0 with a constant from the switch register)
   R1 ← 11000101 (loading R1 with a constant from the switch register)
   R2 ← R0
   R3 ← R1
   R3 ← rr2 R0
   R2 ← rr3 R1

VI. POST-LAB

Write a report summarizing your design and testing experience, lessons learned, and any other comments you may have. Make sure to attach all schematic diagrams and simulation results.

Exp. 14: A SIMPLE ARITHMETIC-LOGIC UNIT

"With great doubts comes great understanding;
with little doubts comes little understanding."
- a Chinese proverb

I. OBJECTIVES

1. To become familiar with the structure of a simple dataflow unit.
2. To learn the basic operations of the arithmetic-logic unit.

II. COMPONENTS

2	74LS139	dual 2-to-4 decoders
6	74LS157	quad 2-to1 MUXs
2	74LS181	4-bit ALU
1	74LS241	octal three-state line drivers
4	74LS374	octal D FFs with three-state outputs

III. INTRODUCTION

In a digital computer, the elementary operations performed by the hardware are referred to as microoperations. A microoperation is typically executed in one clock time. Implementations of transfer and shift microoperations were studied in an earlier experiment. We will expand that into a simple arithmetic-logic unit (ALU).

The ALU is a combinational circuit that performs most of the microoperations, including transfer, arithmetic, logic, and shift operations. It is the unit where all data manipulations are done. We will experiment with some of the basic ALU functions. In more complex ALUs, the unit may include fast adder, multiplier, and divider circuits. In more advanced architecture, the dataflow structure may employ pipelining to improve the throughput. Even higher performance can be realized with multiple pipelines involving multiple computational circuits to execute multiple operations at the same time.

General Block Diagram

Fig. 14-1 shows the dataflow structure of a simple processing unit. There are a total of 5 registers: 3 in the register file named R0 to R2, one switch register, RS, and one temp register, RT. The register file is comprised of three 74LS374s and is similar to the one you used in the

previous experiment. The fourth 74LS374 can be used as the temp register. The switch register is consisted of toggle switches just like the one you used in the previous experiment. Notice that the switch register is connected to the BusB via a set of three-state buffers instead of the 2-to-1 MUXs of the previous experiment. The ALU is composed of two 4-bit arithmetic-logic units, 74LS181s. This is really the only new unit you will add in this experiment. The barrel shifter can perform the right rotation for 0, 1, 2 or 3 bits depending on the command given and is the same one that you designed in the previous experiment.

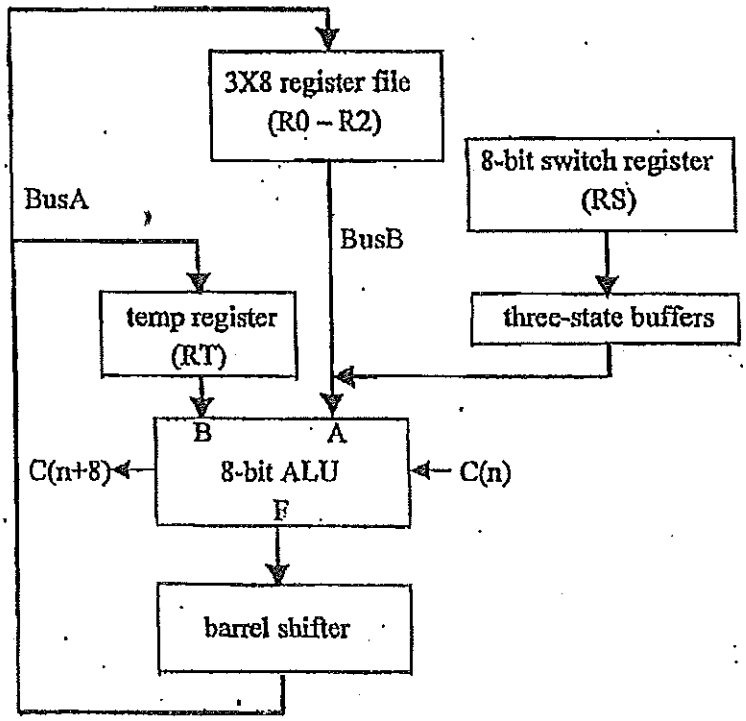


Fig. 14-1. Dataflow structure of a simple processing unit

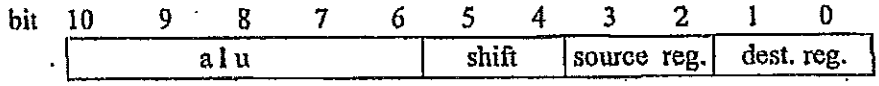
THE 74LS181

The 74LS181 is a 4-bit parallel ALU. It performs all kinds of arithmetic and logic operations of two 4-bit operands. It can be easily cascaded to form ALUs of 8-bit, 12-bit, 16-bit, etc. If you examine closely the internal logic structure of the '181, you will find that it employs the carry-lookahead circuit so that more significant bits do not have to wait for the carry from the less significant positions to "ripple" through many levels of logic gates. The data sheet lists the typical 4-bit addition time for the 74LS181 to be 24 ns.

The data sheets of the 74LS181 describe its operations fully. Briefly, the '181 accepts two 4-bit operands, A3, A2, A1, A0, and B3, B2, B1, B0 as inputs. It generates a 4-bit output F3, F2, F1, F0. C(n) is the carry input to the least significant bit and C(n+4) is the carry output from the most significant bit. M, S3, S2, S1, and S0 are selection signals that determine the particular arithmetic-logic operation to be performed. These operations include addition, subtraction, increment, decrement, comparison, AND, OR, XOR, straight transfer, etc. Although some are redundant and some are not very useful, a total of 16 arithmetic operations (when M is low) and 16 logic operations (when M is high) can be performed. Notice that when performing arithmetic operations, the carry input, C(n), must be set to an appropriate value. Notice further that the active level of the carry signals, C(n) and C(n+4), is the opposite of the active level of the data signals. For example, if data signals are active-high, the carry signals are active-low.

Control Instruction Format

The operation of the ALU is dictated by the control instruction in effect. Each control instruction selects the source operands and the destination for the result. It also commands what microoperations to perform. Since we are only going to perform a relatively simple set of operations, we will keep the control instruction simple and easy to use. The control instruction format that we will use is shown below:



The code assignments of the subfields of the instruction are described below.

alu:

Bits 10 - 6 are the same as the function code used in '181 (refer to the data sheets of '181). Specifically,

bit 10 = M
bit 9 = S3
bit 8 = S2
bit 7 = S1
bit 6 = S0

shift:

bit 5 4	operation
0 0	no rotation
0 1	rotate right 1 bit
1 0	rotate right 2 bits
1 1	rotate right 3 bits

source register:		destination register:	
bit 3 2	register	bit 1 0	register
0 0	R0	0 0	R0
0 1	R1	0 1	R1
1 0	R2	1 0	R2
1 1	RS	1 1	RT

Note that with the above assignments, we are always selecting one of the registers. If we wish to have some operations that do not involve any of the registers, we must have an additional combinations reserved for "no register."

Here are some examples on how to interpret instructions (assuming active-high data signals):

Instruction 1: 11111,00,11,10
Interpretation: 11111 → send the content of the source register straight through
00 → no rotation
11 → source register is RS
10 → destination register is R2
In register transfer notation: R2 ←— RS

Instruction 2: 11111,00,11,11
Interpretation: 11111 → send the content of the source register straight through
00 → no rotation
11 → source register is RS
11 → destination register is RT
In register transfer notation: RT ←— RS
(same as instruction 1 except that RT is now the destination)

Instruction 3: 01001,00,10,00 with C(n) input low
Interpretation: 01001 → add the contents of the source register and the temp register
00 → no rotation
10 → source register is R2
00 → destination register is R0
In register transfer notation: R0 ←— R2 + RT
(the combination of instructions 2 and 3 would result in R0 ←— R2 + RS)

Instruction 4: 11011,00,10,01
Interpretation: 11011 → AND the contents of the source register and the temp register
00 → no rotation
10 → source register is R2
01 → destination register is R1
In register transfer notation: R1 ←— R2 & RT

Instruction 5: 11111,10,01,01
Interpretation: 11111 → send the contents of the source register straight through
10 → rotate to the right 2 bits
10 → source register is R1
01 → destination register is R1
In register transfer notation: R1 ←— rr2 R1

In order to perform the operation R2 ← R1 - R0, we need the following two instructions:

11111,00,00,11	RT ←— R0	
00110,00,01,10	R2 ←— R1 - RT	note: with C(n) input low

IV. PRE-LAB

1. This is a rather complex experiment so you must come well prepared. You should study the data sheets for the '181 carefully and then experiment with the '181 alone to make sure that you fully understand the operations of the circuit. It is extremely important that you understand the operations of the '181 clearly and correctly as the rest of this experiment will dependent on that understanding.

2. Check the design of your previous experiment on TRANSFER AND SHIFT MICRO-OPERATIONS and make sure that it is still working. You will modify and add new circuits to that design.

V. LABORATORY WORK

This experiment is to be performed using the EDA software. Make sure to use bus notations in your schematic diagrams. You should have most of the system already completed in the previous experiment. Plan your test inputs carefully so that the verification process is concise yet complete.

1. Starting with the design from the previous experiment, TRANSFER AND SHIFT MICRO-OPERATIONS, remove R3 from the register file and make it the register RT. Note that this can be done simply by disconnecting its outputs from the register file output bus (BusB) – no need to change the input connections. Also connect the switch register outputs to BusB via three-state buffers instead of 2-to-1 MUXs. Next, insert the ALU circuit (two '181s) such that BusB and RT are at its input side and the barrel shifter at its output side. Finally, complete all remaining connections following the general structure of the ALU shown in Fig. 14-1. Don't forget to connect the two '181s into one 8-bit ALU. Use switches or input ports for the switch register. Even though Fig. 14-1 does not show any control signals, you must include them in your circuit diagram. Use switches or external inputs to generate control instructions. Use the components listed in Section II but feel free to add other components if needed. Keep your design as simple as possible.

2. Enter the data and perform the operations shown below. In your report, clearly outline the steps taken to complete each of the tasks, including all switch settings and sequence of instructions used. Note that tasks a) to d) involve the same set of operands and so do tasks e) to h). That is, there are only four different operands. If you plan the registers carefully, you can complete all eight tasks by loading each operand from the switch register only once.

a)	01110101 + 00101011	(add) and store the result in R2
b)	01110101 ⊕ 00101011	(XOR) and store the result in R2
c)	r3 01110101	(rotate right 3 bits) and store the result in R1
d)	00101011 + 1	(increment) and store the result in R0
e)	10001100 – 00011011	(subtract) and store the result in R0
f)	10001100 | 00011011	(OR) and store the result in R0
g)	r2 00011011	(rotate right 2 bits) and store the result in R1
h)	10001100 – 1	(decrement) and store the result in R2

Demonstrate the operations of the simple ALU to the lab instructor.

Instructor's signature:_____. Date:_____

VI. POST-LAB

Write a report summarizing your design and testing experience, lessons learned, and any other comments you may have. Make sure to attach all schematic diagrams and simulation results.

Appendix A: LIST OF COMPONENTS

Quantity	Part Number	Description
3	74LS00	quadruple 2-input NAND gates
2	74LS04	hex inverters
2	74LS08	quadruple 2-input AND gates
1	74LS10	triple 3-input NAND gates
1	74LS32	quadruple 2-input OR gates
2	74LS74A	dual positive-edge-triggered D flip-flops
2	74LS86	quadruple exclusive-OR gates
2	74LS112A	dual negative-edge-triggered JK flip-flops
1	74LS138	3-to-8 line decoder
1	74LS157	quadruple 2-to-1 multiplers
1	74LS161	synchronous 4-bit counter
1	74LS181	4-bit arithmetic-logic unit
2	74LS194A	4-bit bidirectional universal shift register
1	74LS241	octal three-state buffers
1	2114	static 1024x4 memory
1		IC puller
1		strips of protoboard
1		4-position DIP switches
3		0.01 μF capacitors
6		1 KΩ resistors (1/4 W, 10%)

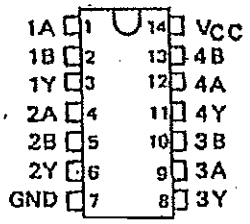
The component library of the schematic capture and simulation software must include all ICs listed above. In addition, the following parts are also needed.

74LS153	dual 4-line to 1-line data selectors/multiplexers
74LS374	8 D-type FFs with 3-state outputs.

Appendix B: IC PINOUTS AND FUNCTION TABLES

TEXAS INSTRUMENTS

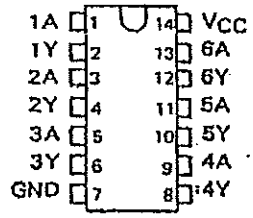
SN74LS00, SN74S00 . . . D, J OR N PACKAGE
(TOP VIEW)



FUNCTION TABLE (each gate)

INPUTS		OUTPUT
A	B	Y
H	H	L
L	X	H
X	L	H

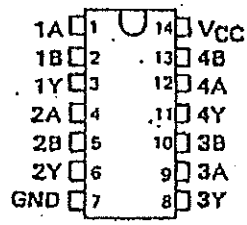
SN74LS04, SN74S04 . . . D, J OR N PACKAGE
(TOP VIEW)



FUNCTION TABLE (each inverter)

INPUTS A	OUTPUT Y
H	L
L	H

SN74LS08, SN74S08 . . . D, J OR N PACKAGE
(TOP VIEW)

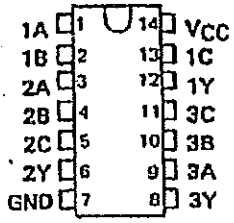


FUNCTION TABLE (each gate)

INPUTS		OUTPUT
A	B	Y
H	H	H
L	X	L
X	L	L

All 74 series data are reprinted with permission of Texas Instruments, copyright holder of original.

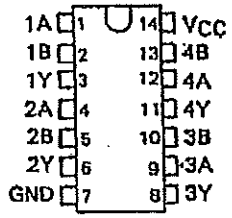
SN7410, SN74H10 . . . J OR N PACKAGE
SN74LS10, SN74S10 . . . D, J OR N PACKAGE
(TOP VIEW)



FUNCTION TABLE (each gate)

INPUTS			OUTPUT
A	B	C	Y
H	H	H	L
L	X	X	H
X	L	X	H
X	X	L	H

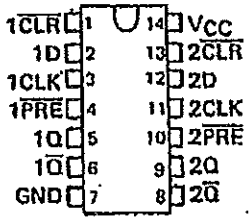
SN7432 . . . J OR N PACKAGE
SN74LS32, SN74S32 . . . D, J or N PACKAGE
(TOP VIEW)



FUNCTION TABLE (each gate)

INPUTS		OUTPUT
A	B	Y
H	X	H
X	H	H
L	L	L

SN7474, SN74H74 . . . J OR N PACKAGE
SN74LS74A, SN74S74 . . . D, J OR N PACKAGE
(TOP VIEW)

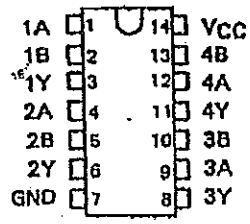


FUNCTION TABLE

INPUTS				OUTPUTS	
PRE	CLR	CLK	D	Q	Q̄
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H†	H†
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\bar{Q}_0$

† The output levels in this configuration are not guaranteed to meet the minimum levels for V_{OH} if the lows at preset and clear are near V_{IL} maximum. Furthermore, this configuration is nonstable; that is, it will not persist when either preset or clear returns to its inactive (high) level.

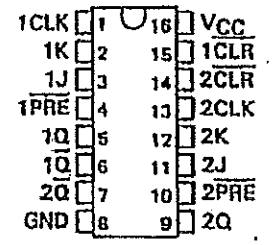
SN7486 . . . J OR N PACKAGE
SN74LS86A, SN74S86 . . . D, J OR N PACKAGE
(TOP VIEW)



FUNCTION TABLES

INPUTS		OUTPUT
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

SN74LS112A, SN74S112 . . . D, J OR N PACKAGE
(TOP VIEW)

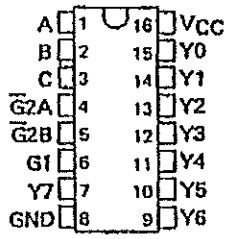


FUNCTION TABLE (each flip-flop)

INPUTS					OUTPUTS	
PRE	CLR	CLK	J	K	Q	Q̄
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H†	H†
H	H	↓	L	L	Q_0	$\bar{Q}_0$
H	H	↓	H	L	H	L
H	H	↓	L	H	L	H
H	H	↓	H	H	TOGGLE	
H	H	H	X	X	Q_0	$\bar{Q}_0$

† The output levels in this configuration are not guaranteed to meet the minimum levels for V_{OH} if the lows at preset and clear are near V_{IL} maximum. Furthermore, this configuration is nonstable; that is, it will not persist when either preset or clear returns to its inactive (high) level.

SN74LS138, SN74S138 . . . D, J OR N PACKAGE
(TOP VIEW)



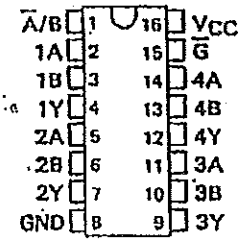
FUNCTION TABLE

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	G̅2̅*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

*$\bar{G}2 = \bar{G}2A + \bar{G}2B$

H = high level, L = low level, X = irrelevant

SN74157 . . . J OR N PACKAGE
SN74LS157, SN74S157,
SN74LS158, SN74S158 . . . D, J OR N PACKAGE
(TOP VIEW)

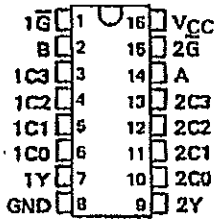


FUNCTION TABLE

INPUTS				OUTPUT Y	
STROBE G̅	SELECT A̅/B̅	A	B	'157, 'L157, 'LS157, 'S157	'LS158 'S158
H	X	X	X	L	H
L	L	L	X	L	H
L	L	H	X	H	L
L	H	X	L	L	H
L	H	X	H	H	L

H = high level, L = low level, X = irrelevant

SN54153, SN54LS153, SN54S153 . . . J OR W PACKAGE
SN74153 . . . N PACKAGE
SN74LS153, SN74S153 . . . D OR N PACKAGE
(TOP VIEW)

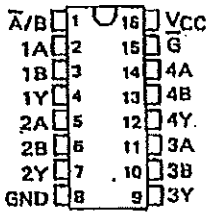


FUNCTION TABLE

SELECT INPUTS		DATA INPUTS				STROBE	OUTPUT
B	A	C0	C1	C2	C3	G̅	Y
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
L	H	X	L	X	X	L	L
L	H	X	H	X	X	L	H
H	L	X	X	L	X	L	L
H	L	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

Select inputs A and B are common to both sections.
H = high level, L = low level, X = irrelevant

SN54LS257B, SN54S257,
SN54LS258B, SN54S258 . . . FK PACKAGE
(TOP VIEW)

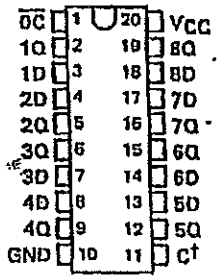


FUNCTION TABLE

INPUTS				OUTPUT Y	
OUTPUT CONTROL	SELECT	A	B	'LS257B 'S257	'LS258B 'S258
H	X	X	X	Z	Z
L	L	L	X	L	H
L	L	H	X	H	L
L	H	X	L	L	H
L	H	X	H	H	L

H = high level, L = low level, X = irrelevant,
Z = high impedance (off)

SN54LS373, SN54LS374, SN54S373,
SN54S374 . . . J OR W PACKAGE
SN74LS373, SN74LS374, SN74S373,
SN74S374 . . . DW OR N PACKAGE
(TOP VIEW)



'LS374, 'S374
FUNCTION TABLE

OUTPUT ENABLE	CLOCK	D	OUTPUT
L	↑	H	H
L	↑	L	L
L	L	X	Q_0
H	X	X	Z

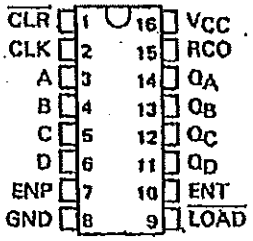
'161, 'LS161A, '163, 'LS163A, 'S163 BINARY COUNTERS

typical clear, preset, count, and inhibit sequences

Illustrated below is the following sequence:

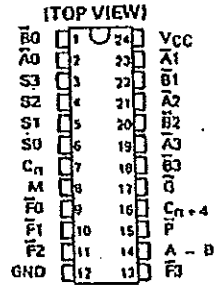
1. Clear outputs to zero ('161 and 'LS161A are asynchronous; '163, 'LS163A, and 'S163 are synchronous)
2. Preset to binary twelve
3. Count to thirteen, fourteen, fifteen, zero, one, and two
4. Inhibit

SERIES 74' . . . J OR N PACKAGE
SERIES 74LS', 74S' . . . D, J OR N PACKAGE
(TOP VIEW)

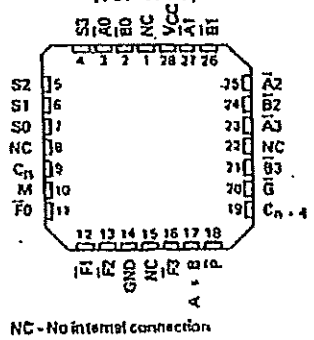


- Full Look-Ahead for High-Speed Operations on Long Words
- Input Clamping Diodes Minimize Transmission-Line Effects
- Darlington Outputs Reduce Turn-Off Time
- Arithmetic Operating Modes:
  - Addition
  - Subtraction
  - Shift Operand A One Position
  - Magnitude Comparison
  - Plus Twelve Other Arithmetic Operations
- Logic Function Modes:
  - Exclusive-OR
  - Comparator
  - AND, NAND, OR, NOR
  - Plus Ten Other Logic Operations

SN54181, SN54LS181, SN54S181 . . . J OR W PACKAGE
SN74181 . . . J OR N PACKAGE
SN74LS181, SN74S181 . . . DW, J OR N PACKAGE
(TOP VIEW)



SN54LS181, SN54S181 . . . FK PACKAGE
SN74LS181, SN74S181 . . . FN PACKAGE
(TOP VIEW)



NC – No internal connection

TYPICAL ADDITION TIMES

NUMBER OF BITS	ADDITION TIMES USING '181 AND '182	ADDITION TIMES USING 'LS181 AND '182	ADDITION TIMES USING 'S181 AND 'S182	PACKAGE COUNT ARITHMETIC/ LOGIC UNITS	PACKAGE COUNT LOOK-AHEAD CARRY GENERATORS	CARRY METHOD BETWEEN ALU's
1 to 4	24 ns	24 ns	11 ns	1		NONE
5 to 8	36 ns	40 ns	18 ns	2		RIPPLE
9 to 16	36 ns	44 ns	19 ns	3 or 4	1	FULL LOOK-AHEAD
17 to 64	60 ns	68 ns	28 ns	5 to 16	2 to 5	FULL LOOK-AHEAD

description

The '181, 'LS181, and 'S181 are arithmetic logic units (ALU)/function generators that have a complexity of 75 equivalent gates on a monolithic chip. These circuits perform 16 binary arithmetic operations on two 4-bit words as shown in Tables 1 and 2. These operations are selected by the four function-select lines (S0, S1, S2, S3) and include addition, subtraction, decrement, and straight transfer. When performing arithmetic manipulations, the internal carries must be enabled by applying a low-level voltage to the mode control input (M). A full carry look-ahead scheme is made available in these devices for fast, simultaneous carry generation by means of two cascade-outputs (pins 15 and 17) for the four bits in the package. When used in conjunction with the SN54182, SN54S182, SN74182, or SN74S182, full carry look-ahead circuits, high-speed arithmetic operations can be performed. The typical addition times shown above illustrate the little additional time required for addition of longer words when full carry look-ahead is employed. The method of cascading '182 or 'S182 circuits with these ALU's to provide multi-level full carry look-ahead is illustrated under typical applications data for the '182 and 'S182.

If high speed is not of importance, a ripple-carry input (C_n) and a ripple-carry output (C_{n+4}) are available. However, the ripple-carry delay has also been minimized so that arithmetic manipulations for small word lengths can be performed without external circuitry.

description (continued)

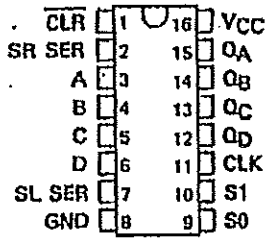
The '181, 'LS181, and 'S181 will accommodate active-high or active-low data if the pin designations are interpreted as follows:

PIN NUMBER	2	1	23	22	21	20	19	18	9	10	11	13	7	16	15	17
Active-low data (Table 1)	$\overline{A}_0$	$\overline{B}_0$	$\overline{A}_1$	$\overline{B}_1$	$\overline{A}_2$	$\overline{B}_2$	$\overline{A}_3$	$\overline{B}_3$	$\overline{F}_0$	$\overline{F}_1$	$\overline{F}_2$	$\overline{F}_3$	C_n	C_{n+4}	$\overline{P}$	$\overline{G}$
Active-high data (Table 2)	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3	F_0	F_1	F_2	F_3	$\overline{C}_n$	$\overline{C}_{n+4}$	X	Y

SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	$\overline{C}_n$ = H (no carry)	$\overline{C}_n$ = L (with carry)
L	L	L	L	$F = \overline{A}$	F = A	F = A PLUS 1
L	L	L	H	$F = \overline{A+B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	$F = \overline{A}B$	$F = A + \overline{B}$	F = (A + $\overline{B}$) PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	L	L	$F = \overline{AB}$	F = A PLUS A$\overline{B}$	F = A PLUS A$\overline{B}$ PLUS 1
L	H	L	H	$F = \overline{B}$	F = (A + B) PLUS A$\overline{B}$	F = (A + B) PLUS A$\overline{B}$ PLUS 1
L	H	H	L	$F = A \oplus B$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A\overline{B}$	F = A$\overline{B}$ MINUS 1	F = A$\overline{B}$
H	L	L	L	$F = \overline{A}+B$	F = A PLUS AB	F = A PLUS AB PLUS 1
H	L	L	H	$F = \overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	F = (A+$\overline{B}$) PLUS AB	F = (A + $\overline{B}$) PLUS AB PLUS 1
H	L	H	H	F = AB	F = AB MINUS 1	F = AB
H	H	L	L	F = 1	F = A	F = A PLUS A PLUS 1
H	H	L	H	$F = A+\overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	F = (A + $\overline{B}$) PLUS A	F = (A + $\overline{B}$) PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

SELECTION				ACTIVE-LOW DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	C_n = L (no carry)	C_n = H (with carry)
L	L	L	L	$F = \overline{A}$	F = A MINUS 1	F = A
L	L	L	H	$F = \overline{AB}$	F = AB MINUS 1	F = AB
L	L	H	L	$F = \overline{A}+B$	F = A$\overline{B}$ MINUS 1	F = A$\overline{B}$
L	L	H	H	F = 1	F = MINUS 1 (2's COMP)	F = ZERO
L	H	L	L	$F = \overline{A+B}$	F = A PLUS (A + $\overline{B}$)	F = A PLUS (A + $\overline{B}$) PLUS 1
L	H	L	H	$F = \overline{B}$	F = AB PLUS (A + $\overline{B}$)	F = AB PLUS (A + $\overline{B}$) PLUS 1
L	H	H	L	$F = \overline{A \oplus B}$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A+\overline{B}$	$F = A+\overline{B}$	F = (A + $\overline{B}$) PLUS 1
H	L	L	L	$F = \overline{A}B$	F = A PLUS (A + B)	F = A PLUS (A + B) PLUS 1
H	L	L	H	$F = A \oplus B$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	F = A$\overline{B}$ PLUS (A + B)	F = A$\overline{B}$ PLUS (A + B) PLUS 1
H	L	H	H	F = A + B	F = (A + B)	F = (A + B) PLUS 1
H	H	L	L	F = 0	F = A	F = A PLUS A PLUS 1
H	H	L	H	$F = A\overline{B}$	F = AB PLUS A	F = AB PLUS A PLUS 1
H	H	H	L	F = AB	F = A$\overline{B}$ PLUS A	F = A$\overline{B}$ PLUS A PLUS 1
H	H	H	H	F = A	F = A	F = A PLUS 1

SN74194 . . . J OR N PACKAGE
SN74LS194A, SN74S194 . . . D, J OR N PACKAGE
(TOP VIEW)

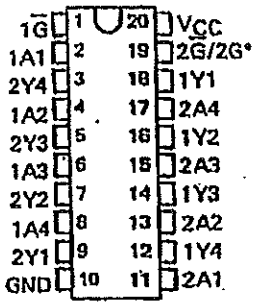


FUNCTION TABLE

INPUTS									OUTPUTS			
CLEAR	MODE		CLOCK	SERIAL		PARALLEL						
	S1	S0		LEFT	RIGHT	A	B	C	D	Q_A	Q_B	Q_C	Q_D
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	L	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	Q_{An}	Q_{Bn}	Q_{Cn}
H	L	H	↑	X	L	X	X	X	X	L	Q_{An}	Q_{Bn}	Q_{Cn}
H	H	L	↑	H	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	H
H	H	L	↑	L	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	L
H	L	L	X	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}

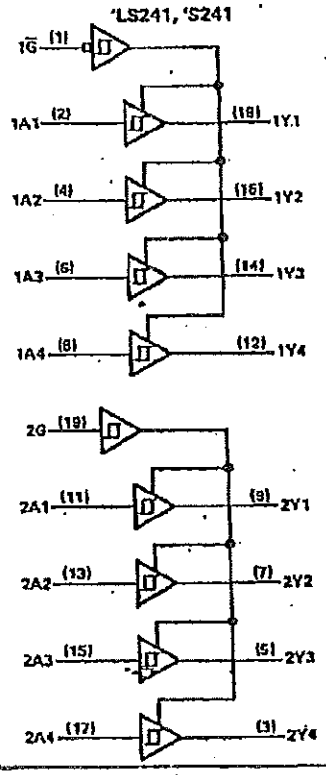
H = high level (steady state)
L = low level (steady state)
X = irrelevant (any input, including transitions)
↑ = transition from low to high level
a, b, c, d = the level of steady-state input at inputs A, B, C, or D, respectively.
Q_{A0}, Q_{B0}, Q_{C0}, Q_{D0} = the level of Q_A, Q_B, Q_C, or Q_D, respectively, before the indicated steady-state input conditions were established.
Q_{An}, Q_{Bn}, Q_{Cn}, Q_{Dn} = the level of Q_A, Q_B, Q_C, respectively, before the most recent ↑ transition of the clock.

SN74LS', SN74S' . . . DW, J OR N PACKAGE
(TOP VIEW)



*2G for 'LS241 and 'S241 or $\overline{2G}$ for all other drivers.

'LS241, 'S241



intel

2114A
1024 X 4 BIT STATIC RAM

	2114AL-1	2114AL-2	2114AL-3	2114AL-4	2114A-4	2114A-5
Max. Access Time (ns)	100	120	150	200	200	250
Max. Current (mA)	40	40	40	40	70	70

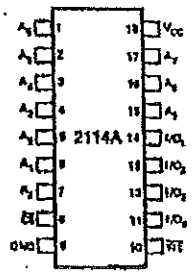
- HMOS Technology
- Low Power, High Speed
- Identical Cycle and Access Times
- Single +5V Supply ±10%
- High Density 18 Pin Package
- Completely Static Memory - No Clock or Timing Strobe Required
- Directly TTL Compatible: All Inputs and Outputs
- Common Data Input and Output Using Three-State Outputs
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel® 2114A is a 4096-bit static Random Access Memory organized as 1024 words by 4-bits using HMOS, a high performance MOS technology. It uses fully DC stable (static) circuitry throughout, in both the array and the decoding, therefore it requires no clocks or refreshing to operate. Data access is particularly simple since address setup times are not required. The data is read out nondestructively and has the same polarity as the input data. Common input/output pins are provided.

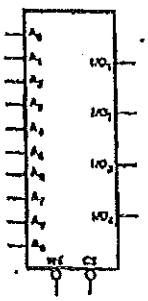
The 2114A is designed for memory applications where the high performance and high reliability of HMOS, low cost, large bit storage, and simple interfacing are important design objectives. The 2114A is placed in an 18-pin package for the highest possible density.

It is directly TTL compatible in all respects: inputs, outputs, and a single +5V supply. A separate Chip Select ($\overline{CS}$) lead allows easy selection of an individual package when outputs are or-tied.

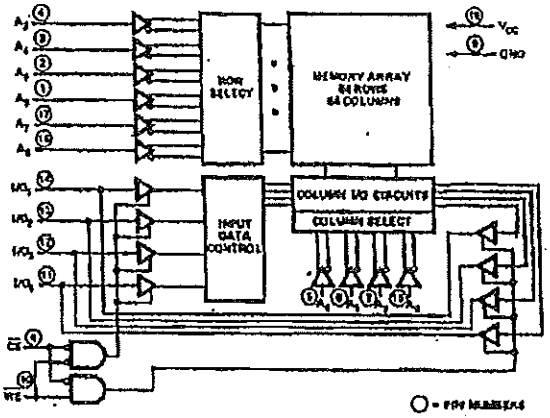
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

A_0–A_9	ADDRESS INPUTS	V_{CC}	POWER (+5V)
$\overline{WE}$	WRITE ENABLE	GND	GROUND
$\overline{CS}$	CHIP SELECT		
I/O_1–I/O_4	DATA INPUT/OUTPUT		

SAMPLE SAMPLE SAMPLE SAMPLE SAMPLE

Experiment 1: Familiarization with the Digital Trainer

ENGR 356
Section 2
Fall, 1998

Mohammed Smith
Guadalupe Wong

Date experiment performed: Sept. 2, 1998
Date report submitted: Sept. 9, 1998

Procedures and Observations

We read the lab manual before we went to the lab so we had some idea about the digital trainer. The instructor also briefly went through different parts of the trainer with us. We then followed the procedures given in the manual to learn first-hand the capabilities of the digital trainer.

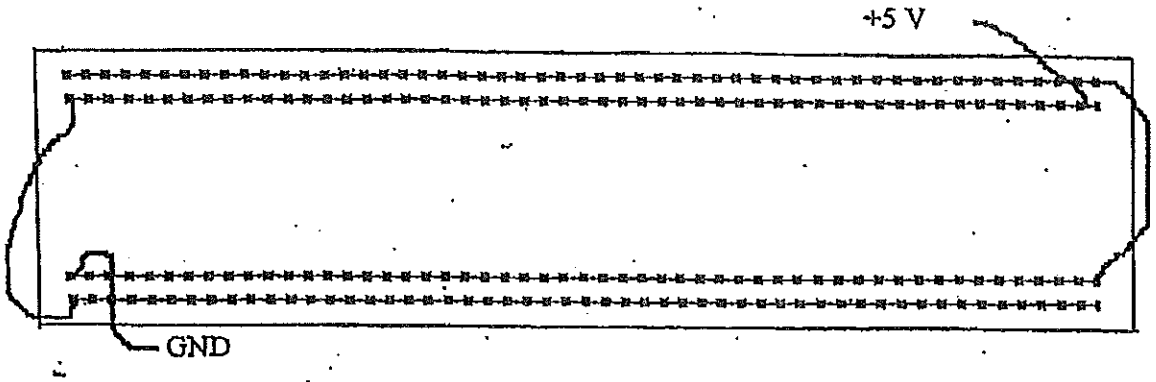
1. When we connected a wire between the +5 V source and an indicator light, the light came on. When we switched one end of the wire from the +5 V to ground, the light went off. The light was also off when there was nothing connected to it. Thus, the light is on when the signal is high and is off when the signal is either low or floating.

2. We connected a wire from the logic probe input to the upper output of a toggle switch. We also connected the same toggle switch output to an indicator light. With the switch lever up, the green light of the logic probe and the indicator light both came on. With the lever down, the red light of the logic probe came on and the indicator light went off. Since we known that when the indicator light is on, the signal is high, we conclude that the green light of the logic probe represents a high signal and the red light, a low signal. When no signal was applied to the logic probe, both lights were off. We then switched connections to test the lower output of the same toggle switch and found that the signal was the "opposite" of the upper output.

3. We performed the same kind of tests as in step 2 on a pushbutton switch and found that the upper output (NORM ON) is high when the button was <u>not</u> pushed and low when pushed. The lower output (NORM OFF) again gave the opposite signal to that of the upper output.

4. We connected the left-most bus pin to the +5 V output and we checked all the pins on that row using an indicator light. The light came on in each case. That means the entire row represents just one electrical point. We checked all four rows and they responded exactly the same. So the bus pins go the entire length on our protoboard, which gives us 4 electrical points.

5. We connected the board as shown on top of the next page to get the Vcc and GND buses. We used the logic probe to check at least one pin on each of the four rows and found that the top inside row and the bottom outside row were high and the other two rows were low.



6. We used the largest capacitor for our clock circuit and connected the clock output to an indicator light. The light was flashing on and off at about one second rate. When we replaced the capacitor with a smaller one, the light was flashing faster. Thus, a smaller capacitor provides a faster clock rate.

Summary of Results

We learned where to obtain +5V (Vcc) and the ground from the digital trainer and how to distribute them to various points on a protoboard by using on-board buses. Our board has four buses, two on each side, running across the entire board. We also learned how to acquire logic signals (by using toggle and pushbutton switches) and how to display logic signals (by using indicator lights). We found that all switches on the unit can provide both complemented and uncomplemented signals and that indicator lights cannot differentiate a low signal from a floating one (open connection). The built-in logic probe, however, can distinguish whether a circuit point is high, low, or floating. We also observed that smaller capacitors give us higher frequency signals at the clock output. We believe that this has something to do with changing the RC time constant of the clock circuit.

Discussion

Besides the work described above, we obtained our lab kit and confirmed that all components are present. Although we did not have to use any logic circuits in this experiment, we learned about integrated circuit (IC) handling and pin counting techniques from the lab manual. The manual also explained to us some hardware terms such as DIP, LED, TTL, etc. and the proper ways of assembling a circuit. The lab manual was easy to read and follow.

Overall, it was a simple experiment to familiarize us with the equipment. It was fun to see a blinking light. We expect that the logic probe will be a very useful debugging tool for future experiments since we can use it to check whether a signal is high or low. We noticed that the trainer also contains two hex display units but we did not do any experimentation involving them.

Appendix A: LIST OF COMPONENTS

Quantity	Part Number	Description
3	74LS00	quadruple 2-input NAND gates
2	74LS04	hex inverters
2	74LS08	quadruple 2-input AND gates
1	74LS10	triple 3-input NAND gates
1	74LS32	quadruple 2-input OR gates
2	74LS74A	dual positive-edge-triggered D flip-flops
2	74LS86	quadruple exclusive-OR gates
2	74LS112A	dual negative-edge-triggered JK flip-flops
1	74LS138	3-to-8 line decoder
1	74LS157	quadruple 2-to-1 multiplers
1	74LS161	synchronous 4-bit counter
1	74LS181	4-bit arithmetic-logic unit
2	74LS194A	4-bit bidirectional universal shift register
1	74LS241	octal three-state buffers
1	2114	static 1024x4 memory
1		IC puller
1		strips of protoboard
1		4-position DIP switches
3		0.01 μF capacitors
6		1 KΩ resistors (1/4 W, 10%)

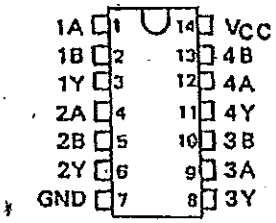
The component library of the schematic capture and simulation software must include all ICs listed above. In addition, the following parts are also needed.

74LS153	dual 4-line to 1-line data selectors/multiplexers
74LS374	8 D-type FFs with 3-state outputs.

Appendix B: IC PINOUTS AND FUNCTION TABLES

TEXAS INSTRUMENTS

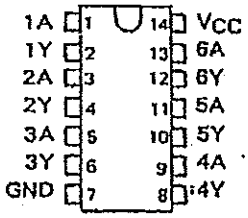
SN74LS00, SN74S00 . . . D, J OR N PACKAGE
(TOP VIEW)



FUNCTION TABLE (each gate)

INPUTS		OUTPUT
A	B	Y
H	H	L
L	X	H
X	L	H

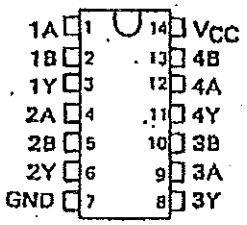
SN74LS04, SN74S04 . . . D, J OR N PACKAGE
(TOP VIEW)



FUNCTION TABLE (each inverter)

INPUTS A	OUTPUT Y
H	L
L	H

SN74LS08, SN74S08 . . . D, J OR N PACKAGE
(TOP VIEW)

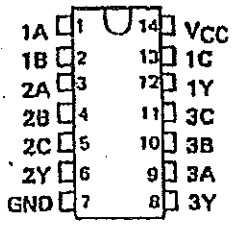


FUNCTION TABLE (each gate)

INPUTS		OUTPUT
A	B	Y
H	H	H
L	X	L
X	L	L

All 74 series data are reprinted with permission of Texas Instruments, copyright holder of original.

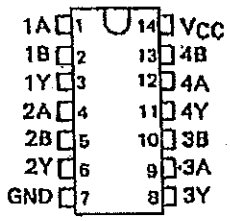
SN7410, SN74H10 . . . J OR N PACKAGE
SN74LS10, SN74S10 . . . D, J OR N PACKAGE
(TOP VIEW)



FUNCTION TABLE (each gate)

INPUTS			OUTPUT
A	B	C	Y
H	H	H	L
L	X	X	H
X	L	X	H
X	X	L	H

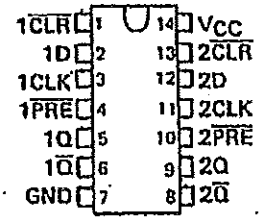
SN7432 . . . J OR N PACKAGE
SN74LS32, SN74S32 . . . D, J or N PACKAGE
(TOP VIEW)



FUNCTION TABLE (each gate)

INPUTS		OUTPUT
A	B	Y
H	X	H
X	H	H
L	L	L

SN7474, SN74H74 . . . J OR N PACKAGE
SN74LS74A, SN74S74 . . . D, J OR N PACKAGE
(TOP VIEW)

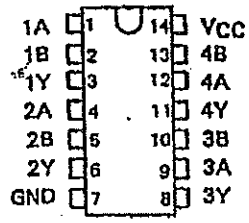


FUNCTION TABLE

INPUTS				OUTPUTS	
PRE	CLR	CLK	D	Q	Q̄
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H†	H†
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\bar{Q}_0$

† The output levels in this configuration are not guaranteed to meet the minimum levels in V_{OH} if the lows at preset and clear are near V_{IL} maximum. Furthermore, this configuration is nonstable; that is, it will not persist when either preset or clear returns to its inactive (high) level.

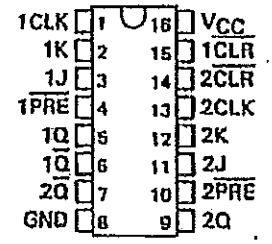
SN7486 . . . J OR N PACKAGE
SN74LS86A, SN74S86 . . . D, J OR N PACKAGE
(TOP VIEW)



FUNCTION TABLES

INPUTS		OUTPUT
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

SN74LS112A, SN74S112 . . . D, J OR N PACKAGE
(TOP VIEW)

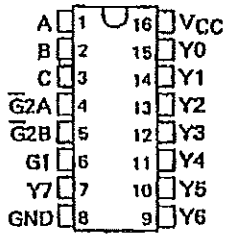


FUNCTION TABLE (each flip-flop)

INPUTS					OUTPUTS	
PRE	CLR	CLK	J	K	Q	Q̄
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H†	H†
H	H	↓	L	L	Q_0	$\bar{Q}_0$
H	H	↓	H	L	H	L
H	H	↓	L	H	L	H
H	H	↓	H	H	TOGGLE	
H	H	H	X	X	Q_0	$\bar{Q}_0$

† The output levels in this configuration are not guaranteed to meet the minimum levels for V_{OH} if the lows at preset and clear are near V_{IL} maximum. Furthermore, this configuration is nonstable; that is, it will not persist when either preset or clear returns to its inactive (high) level.

SN74LS138, SN74S138 . . . D, J OR N PACKAGE
(TOP VIEW)



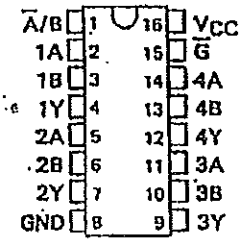
FUNCTION TABLE

INPUTS					OUTPUTS							
ENABLE		SELECT										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

*$\bar{G}2 = \bar{G}2A + \bar{G}2B$

H = high level, L = low level, X = irrelevant

SN74157 . . . J OR N PACKAGE
SN74LS157, SN74S157,
SN74LS158, SN74S158 . . . D, J OR N PACKAGE
(TOP VIEW)

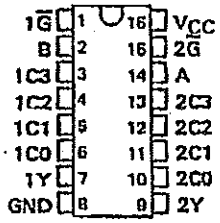


FUNCTION TABLE

INPUTS				OUTPUT Y	
STROBE G	SELECT A/B	A	B	'157, 'L157, 'LS157, 'S157	'LS158 'S158
H	X	X	X	L	H
L	L	L	X	L	H
L	L	H	X	H	L
L	H	X	L	L	H
L	H	X	H	H	L

H = high level, L = low level, X = irrelevant

SN54153, SN54LS153, SN54S153 . . . J OR W PACKAGE
SN74153 . . . N PACKAGE
SN74LS153, SN74S153 . . . D OR N PACKAGE
(TOP VIEW)

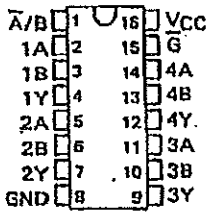


FUNCTION TABLE

SELECT INPUTS		DATA INPUTS				STROBE	OUTPUT
B	A	C0	C1	C2	C3	G̅	Y
X	X	X	X	X	X	H	L
L	L	L	X	X	X	L	L
L	L	H	X	X	X	L	H
L	H	X	L	X	X	L	L
L	H	X	H	X	X	L	H
H	L	X	X	L	X	L	L
H	L	X	X	H	X	L	H
H	H	X	X	X	L	L	L
H	H	X	X	X	H	L	H

Select inputs A and B are common to both sections.
H = high level, L = low level, X = irrelevant

SN54LS258, SN54S257,
SN54LS258, SN54S258 . . . FK PACKAGE
(TOP VIEW)

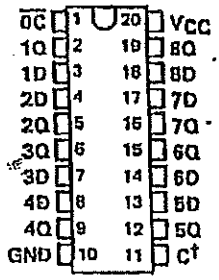


FUNCTION TABLE

INPUTS				OUTPUT Y	
OUTPUT CONTROL	SELECT	A	B	'LS257B 'S257	'LS258B 'S258
H	X	X	X	Z	Z
L	L	L	X	L	H
L	L	H	X	H	L
L	H	X	L	L	H
L	H	X	H	H	L

H = high level, L = low level, X = irrelevant,
Z = high impedance (off)

SN54LS373, SN54LS374, SN54S373,
SN54S374 . . . J OR W PACKAGE
SN74LS373, SN74LS374, SN74S373,
SN74S374 . . . DW OR N PACKAGE
(TOP VIEW)



'LS374, 'S374
FUNCTION TABLE

OUTPUT ENABLE	CLOCK	D	OUTPUT
L	↑	H	H
L	↑	L	L
L	L	X	Q_0
H	X	X	Z

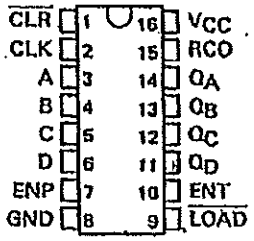
'161, 'LS161A, '163, 'LS163A, 'S163 BINARY COUNTERS

typical clear, preset, count, and inhibit sequences

Illustrated below is the following sequence:

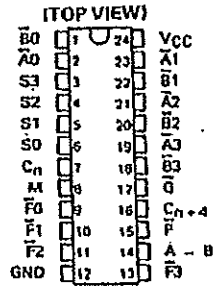
1. Clear outputs to zero ('161 and 'LS161A are asynchronous; '163, 'LS163A, and 'S163 are synchronous)
2. Preset to binary twelve
3. Count to thirteen, fourteen, fifteen, zero, one, and two
4. Inhibit

SERIES 74' . . . J OR N PACKAGE
SERIES 74LS', 74S' . . . D, J OR N PACKAGE
(TOP VIEW)

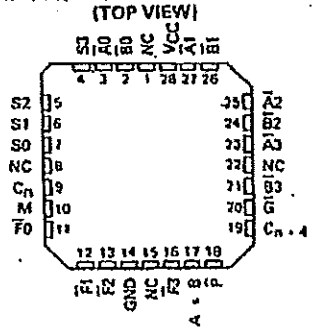


- Full Look-Ahead for High-Speed Operations on Long Words
- Input Clamping Diodes Minimize Transmission-Line Effects
- Darlington Outputs Reduce Turn-Off Time
- Arithmetic Operating Modes:
  - Addition
  - Subtraction
  - Shift Operand A One Position
  - Magnitude Comparison
  - Plus Twelve Other Arithmetic Operations
- Logic Function Modes:
  - Exclusive-OR
  - Comparator
  - AND, NAND, OR, NOR
  - Plus Ten Other Logic Operations

SN54181, SN54LS181, SN54S181 . . . J OR W PACKAGE
SN74181 . . . J OR N PACKAGE
SN74LS181, SN74S181 . . . DW, J OR N PACKAGE
(TOP VIEW)



SN54LS181, SN54S181 . . . FK PACKAGE
SN74LS181, SN74S181 . . . FN PACKAGE
(TOP VIEW)



NC – No internal connection

TYPICAL ADDITION TIMES

NUMBER OF BITS	ADDITION TIMES USING '181 AND '182	USING 'LS181 AND '182	USING 'S181 AND 'S182	PACKAGE COUNT ARITHMETIC/ LOGIC UNITS	LOOK-AHEAD CARRY GENERATORS	CARRY METHOD BETWEEN ALU's
1 to 4	24 ns	24 ns	11 ns	1		NONE
5 to 8	36 ns	40 ns	18 ns	2		RIPPLE
9 to 16	36 ns	44 ns	19 ns	3 or 4	1	FULL LOOK-AHEAD
17 to 64	60 ns	68 ns	28 ns	5 to 16	2 to 5	FULL LOOK-AHEAD

description

The '181, 'LS181, and 'S181 are arithmetic logic units (ALU)/function generators that have a complexity of 75 equivalent gates on a monolithic chip. These circuits perform 16 binary arithmetic operations on two 4-bit words as shown in Tables 1 and 2. These operations are selected by the four function-select lines (S0, S1, S2, S3) and include addition, subtraction, decrement, and straight transfer. When performing arithmetic manipulations, the internal carries must be enabled by applying a low-level voltage to the mode control input (M). A full carry look-ahead scheme is made available in these devices for fast, simultaneous carry generation by means of two cascade-outputs (pins 15 and 17) for the four bits in the package. When used in conjunction with the SN54182, SN54S182, SN74182, or SN74S182, full carry look-ahead circuits, high-speed arithmetic operations can be performed. The typical addition times shown above illustrate the little additional time required for addition of longer words when full carry look-ahead is employed. The method of cascading '182 or 'S182 circuits with these ALU's to provide multi-level full carry look-ahead is illustrated under typical applications data for the '182 and 'S182.

If high speed is not of importance, a ripple-carry input (C_n) and a ripple-carry output (C_{n+4}) are available. However, the ripple-carry delay has also been minimized so that arithmetic manipulations for small word lengths can be performed without external circuitry.

description (continued)

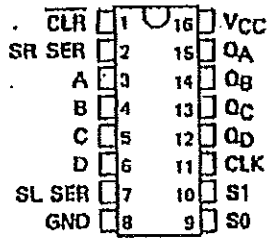
The '181, 'LS181, and 'S181 will accommodate active-high or active-low data if the pin designations are interpreted as follows:

PIN NUMBER	2	1	23	22	21	20	19	18	9	10	11	13	7	16	15	17
Active-low data (Table 1)	$\overline{A}_0$	$\overline{B}_0$	$\overline{A}_1$	$\overline{B}_1$	$\overline{A}_2$	$\overline{B}_2$	$\overline{A}_3$	$\overline{B}_3$	$\overline{F}_0$	$\overline{F}_1$	$\overline{F}_2$	$\overline{F}_3$	C_n	C_{n+4}	$\overline{P}$	$\overline{G}$
Active-high data (Table 2)	A_0	B_0	A_1	B_1	A_2	B_2	A_3	B_3	F_0	F_1	F_2	F_3	$\overline{C}_n$	$\overline{C}_{n+4}$	X	Y

SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	$\overline{C}_n$ = H (no carry)	$\overline{C}_n$ = L (with carry)
L	L	L	L	$F = \overline{A}$	F = A	F = A PLUS 1
L	L	L	H	$F = \overline{A+B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	$F = \overline{A}B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	L	L	$F = \overline{AB}$	$F = A$ PLUS $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS 1
L	H	L	H	$F = \overline{B}$	F = (A + B) PLUS $A\overline{B}$	F = (A + B) PLUS $A\overline{B}$ PLUS 1
L	H	H	L	$F = A \oplus B$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
H	L	L	L	$F = \overline{A} + B$	F = A PLUS AB	F = A PLUS AB PLUS 1
H	L	L	H	$F = \overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS 1
H	L	H	H	F = AB	F = AB MINUS 1	F = AB
H	H	L	L	F = 1	F = A	F = A PLUS A PLUS 1
H	H	L	H	$F = A + \overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

SELECTION				ACTIVE-LOW DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	C_n = L (no carry)	C_n = H (with carry)
L	L	L	L	$F = \overline{A}$	F = A MINUS 1	F = A
L	L	L	H	$F = \overline{AB}$	F = AB MINUS 1	F = AB
L	L	H	L	$F = \overline{A} + B$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
L	L	H	H	F = 1	F = MINUS 1 (2's COMP)	F = ZERO
L	H	L	L	$F = \overline{A + B}$	$F = A$ PLUS $(A + \overline{B})$	$F = A$ PLUS $(A + \overline{B})$ PLUS 1
L	H	L	H	$F = \overline{B}$	$F = AB$ PLUS $(A + \overline{B})$	$F = AB$ PLUS $(A + \overline{B})$ PLUS 1
L	H	H	L	$F = \overline{A \oplus B}$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A + \overline{B}$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
H	L	L	L	$F = \overline{AB}$	F = A PLUS (A + B)	F = A PLUS (A + B) PLUS 1
H	L	L	H	$F = A \oplus B$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	$F = A\overline{B}$ PLUS (A + B)	$F = A\overline{B}$ PLUS (A + B) PLUS 1
H	L	H	H	F = A + B	F = (A + B)	F = (A + B) PLUS 1
H	H	L	L	F = 0	F = A	F = A PLUS A PLUS 1
H	H	L	H	$F = A\overline{B}$	F = AB PLUS A	F = AB PLUS A PLUS 1
H	H	H	L	F = AB	$F = A\overline{B}$ PLUS A	$F = A\overline{B}$ PLUS A PLUS 1
H	H	H	H	F = A	F = A	F = A PLUS 1

SN74194 . . . J OR N PACKAGE
SN74LS194A, SN74S194 . . . D, J OR N PACKAGE
(TOP VIEW)

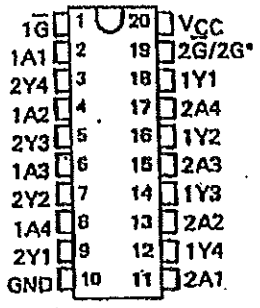


FUNCTION TABLE

INPUTS									OUTPUTS			
CLEAR	MODE		CLOCK	SERIAL		PARALLEL						
	S1	S0		LEFT	RIGHT	A	B	C	D	Q_A	Q_B	Q_C	Q_D
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	L	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	Q_{An}	Q_{Bn}	Q_{Cn}
H	L	H	↑	X	L	X	X	X	X	L	Q_{An}	Q_{Bn}	Q_{Cn}
H	H	L	↑	H	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	H
H	H	L	↑	L	X	X	X	X	X	Q_{Bn}	Q_{Cn}	Q_{Dn}	L
H	L	L	X	X	X	X	X	X	X	Q_{A0}	Q_{B0}	Q_{C0}	Q_{D0}

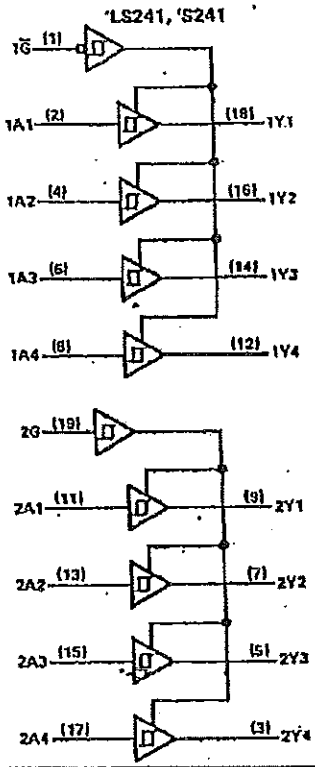
H = high level (steady state)
L = low level (steady state)
X = irrelevant (any input, including transitions)
↑ = transition from low to high level
a, b, c, d = the level of steady-state input at inputs A, B, C, or D, respectively.
Q_{A0}, Q_{B0}, Q_{C0}, Q_{D0} = the level of Q_A, Q_B, Q_C, or Q_D, respectively, before the indicated steady-state input conditions were established.
Q_{An}, Q_{Bn}, Q_{Cn}, Q_{Dn} = the level of Q_A, Q_B, Q_C, respectively, before the most-recent ↑ transition of the clock.

SN74LS', SN74S' . . . DW, J OR N PACKAGE
(TOP VIEW)



*2G for 'LS241 and 'S241 or $\overline{2G}$ for all other drivers.

'LS241, 'S241



intel

2114A
1024 X 4 BIT STATIC RAM

	2114AL-1	2114AL-2	2114AL-3	2114AL-4	2114A-4	2114A-5
Max. Access Time (ns)	100	120	150	200	200	250
Max. Current (mA)	40	40	40	40	70	70

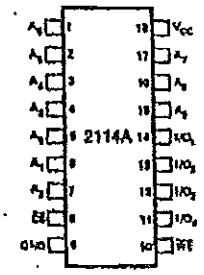
- HMOS Technology
- Low Power, High Speed
- Identical Cycle and Access Times
- Single +5V Supply ±10%
- High Density 18 Pin Package
- Completely Static Memory – No Clock or Timing Strobe Required
- Directly TTL Compatible: All Inputs and Outputs
- Common Data Input and Output Using Three-State Outputs
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel® 2114A is a 4096-bit static Random Access Memory organized as 1024 words by 4-bits using HMOS, a high performance MOS technology. It uses fully DC stable (static) circuitry throughout, in both the array and the decoding, therefore it requires no clocks or refreshing to operate. Data access is particularly simple since address setup times are not required. The data is read out nondestructively and has the same polarity as the input data. Common input/output pins are provided.

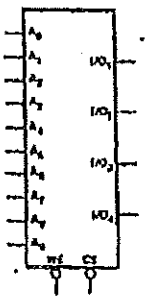
The 2114A is designed for memory applications where the high performance and high reliability of HMOS, low cost, large bit storage, and simple interfacing are important design objectives. The 2114A is placed in an 18-pin package for the highest possible density.

It is directly TTL compatible in all respects: inputs, outputs, and a single +5V supply. A separate Chip Select ($\overline{CS}$) lead allows easy selection of an individual package when outputs are or-tied.

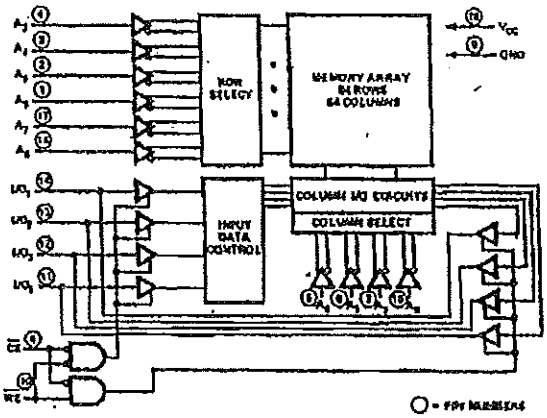
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

A_0–A_9	ADDRESS INPUTS	V_{CC}	POWER (+5V)
$\overline{WE}$	WRITE ENABLE	GND	GROUND
$\overline{CS}$	CHIP SELECT		
I/O_1–I/O_4	DATA INPUT/OUTPUT		

Experiment 1: Familiarization with the Digital Trainer

ENGR 356
Section 2
Fall, 1998

Mohammed Smith
Guadalupe Wong

Date experiment performed: Sept. 2, 1998
Date report submitted: Sept. 9, 1998

Procedures and Observations

We read the lab manual before we went to the lab so we had some idea about the digital trainer. The instructor also briefly went through different parts of the trainer with us. We then followed the procedures given in the manual to learn first-hand the capabilities of the digital trainer.

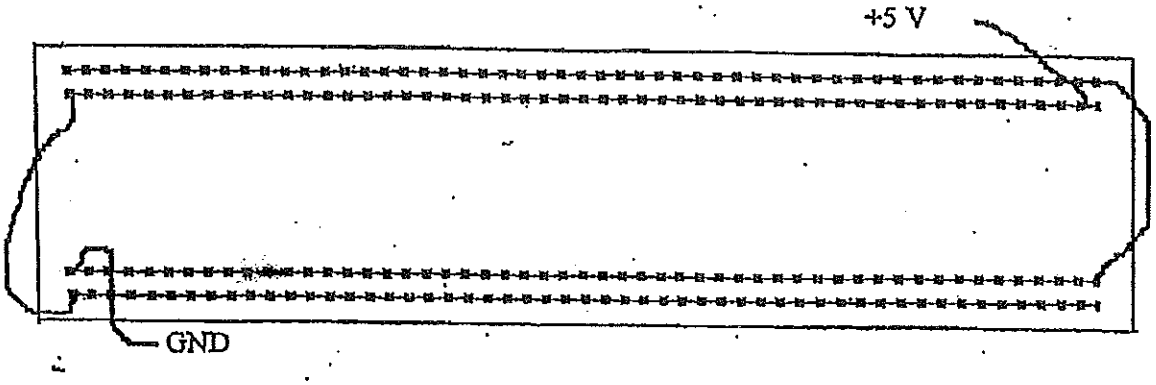
1. When we connected a wire between the +5 V source and an indicator light, the light came on. When we switched one end of the wire from the +5 V to ground, the light went off. The light was also off when there was nothing connected to it. Thus, the light is on when the signal is high and is off when the signal is either low or floating.

2. We connected a wire from the logic probe input to the upper output of a toggle switch. We also connected the same toggle switch output to an indicator light. With the switch lever up, the green light of the logic probe and the indicator light both came on. With the lever down, the red light of the logic probe came on and the indicator light went off. Since we known that when the indicator light is on, the signal is high, we conclude that the green light of the logic probe represents a high signal and the red light, a low signal. When no signal was applied to the logic probe, both lights were off. We then switched connections to test the lower output of the same toggle switch and found that the signal was the "opposite" of the upper output.

3. We performed the same kind of tests as in step 2 on a pushbutton switch and found that the upper output (NORM ON) is high when the button was <u>not</u> pushed and low when pushed. The lower output (NORM OFF) again gave the opposite signal to that of the upper output.

4. We connected the left-most bus pin to the +5 V output and we checked all the pins on that row using an indicator light. The light came on in each case. That means the entire row represents just one electrical point. We checked all four rows and they responded exactly the same. So the bus pins go the entire length on our protoboard, which gives us 4 electrical points.

5. We connected the board as shown on top of the next page to get the Vcc and GND buses. We used the logic probe to check at least one pin on each of the four rows and found that the top inside row and the bottom outside row were high and the other two rows were low.



6. We used the largest capacitor for our clock circuit and connected the clock output to an indicator light. The light was flashing on and off at about one second rate. When we replaced the capacitor with a smaller one, the light was flashing faster. Thus, a smaller capacitor provides a faster clock rate.

Summary of Results

We learned where to obtain +5V (Vcc) and the ground from the digital trainer and how to distribute them to various points on a protoboard by using on-board buses. Our board has four buses, two on each side, running across the entire board. We also learned how to acquire logic signals (by using toggle and pushbutton switches) and how to display logic signals (by using indicator lights). We found that all switches on the unit can provide both complemented and uncomplemented signals and that indicator lights cannot differentiate a low signal from a floating one (open connection). The built-in logic probe, however, can distinguish whether a circuit point is high, low, or floating. We also observed that smaller capacitors give us higher frequency signals at the clock output. We believe that this has something to do with changing the RC time constant of the clock circuit.

Discussion

Besides the work described above, we obtained our lab kit and confirmed that all components are present. Although we did not have to use any logic circuits in this experiment, we learned about integrated circuit (IC) handling and pin counting techniques from the lab manual. The manual also explained to us some hardware terms such as DIP, LED, TTL, etc. and the proper ways of assembling a circuit. The lab manual was easy to read and follow.

Overall, it was a simple experiment to familiarize us with the equipment. It was fun to see a blinking light. We expect that the logic probe will be a very useful debugging tool for future experiments since we can use it to check whether a signal is high or low. We noticed that the trainer also contains two hex display units but we did not do any experimentation involving them.