HARDWARE LAB 4: VENDING MACHINE CONTROLLER DESIGN

Prerequisites: Before beginning this laboratory experiment you must be able to:

- Use a prototype board, voltmeter, and logic probe.
- Describe the truth tables that characterize the primitive logic gates (i.e., AND, OR, NOT, NAND, NOR.)
- Use DeMorgan's Law.
- Perform binary addition of multi-bit numbers.
- Convert between decimal and binary representations of numbers.
- Describe the function of a register.
- Use LogicWorks™.

Equipment: Digital Trainer Board, Voltmeter, Logic Probe.

Circuit Components: You will need the following circuits to complete the tasks in this lab:

- (1) 7402 (Quad 2-Input Nor Gates)
- (1) 7432 (Quad 2-Input Or Gates)
- (1) 74LS85 (4-Bit Magnitude Comparator) (NOT 74F85)
- (1) 74175 (Quad D Flip-Flop)
- (1) 74283 (4-Bit Binary Full Adder with Fast Carry)

Objective: In this laboratory exercise, you will gain some experience using medium-scale-integrated

(MSI) circuits to build a controller for a simple vending machine. You will gain an appreciation of the power of MSI circuits to simplify circuit designs. You will also gain an understanding of the design process, and the role of simulation in circuit design.

Outcomes: When you have completed this laboratory exercise, you will be able to:

- Describe the function of and use a 7485, 4-bit magnitude comparator.
- Describe the function of and use a 74283, 4-bit binary full adder.
- Describe the function of and use a 74175, quad D flip-flop.

- Interpret and use the function definition table of an MSI circuit.
- Simulate a controller for a simple vending machine.
- Build and debug a controller for a simple vending machine.
- Design and simulate an 8-bit controller for a vending machine.
- Appreciate the role of MSI circuits in the design of digital circuits.
- Describe the role of circuit simulation in prototype construction.
- Describe some aspects of the design process.

Lab Report Guidelines

Before you begin this laboratory exercise, make sure you understand the reporting requirements your instructor has specified. If you understand the reporting requirements then you will know the observations that are important to record in your lab notebook and the tests that need to be performed on the circuits you construct. For this lab, it is recommended that you use a top-down format when organizing your lab report. (If your instructor is using the report writing guidelines contained in this manual, you can review them by clicking on the blue text: Top-Down Report Writing Guidelines.)

Prologue

The previous laboratory exercises contained in this manual have largely used what is known as small-scale integrated (SSI) circuits. SSI circuits are so called because each IC is built using relatively few components. (SSI circuit packages typically contain fewer than 6 gates, or 1 (or at most 2) flip-flops.) Packaged circuits using many logic gates in their construction, such as registers and adders, are known as medium-scale integrated (MSI) circuit packages. (MSI circuit packages typically contain more than 6 but less than 100 gates.) In this laboratory exercise, you will gain some experience using MSI circuits to build a digital controller for a vending machine. It is hoped that upon the completion of this laboratory experiment, you will appreciate the power of MSI circuits to simplify circuit designs.

In the previous hardware laboratory experiments the circuits that you designed and build were rather simple; hence any design errors you (or we) made could be easily corrected in the laboratory with some minor re-wiring. For more complex design, this is not the case. With complex designs, errors in the design can affect many aspects of the design, causing the designer to completely disassemble their hardware prototype and start from scratch building a new design. This process can be time consuming, especially if significant errors are found in the second and third generation designs. In order to save time, all complex circuits are simulated before they are built. With simulations, errors can be corrected easily and exhaustive tests conducted quickly to validate all aspects of the designs performance. Once the design meets all of the design specification, a hardware prototype is constructed, and measurements made to

validate its performance. In this laboratory exercise you are going to go through this same process. First you are going to simulate the circuit to prove that it will indeed perform as desired and meet all design specifications. Once satisfied, you will then construct the circuit in the hardware laboratory. It is hoped that when you complete this laboratory experiment you will have your own ideas about the role that simulation plays in digital-circuit design.

Introduction

Depending upon how far along you are in your course material, you may (or may not) have studied the use of state diagrams, transition tables and Karnaugh maps to design synchronous sequential machines. These techniques are powerful and are used every day in the digital electronics industry to build both simple and complex synchronous machines. These techniques are used for two reasons: they are guaranteed to produce a design regardless of the complexity of the specification and they can be structured to yield a design that uses the minimum amount of digital logic. Designs that use the minimum number of logic gates are important in many contexts. For example, if you are building a widget that will be manufactured in the thousands or millions and you have to compete with other manufacturers, who are building similar widgets, minimizing the number of gates, and hence cost, may be of paramount importance. In this situation, the cost of production, rather than the cost of the design process, dominates the cost of your product.

There exist many situations where the resources consumed in designing, prototyping, and building a device are the dominant costs, not the costs of the logic gates used to build the device. These situations occur when you need a one-of-a-kind device or a device that will be replicated few times. In these situations you will be building these devices from IC's rather than creating a single IC to perform the task. You will be concerned less about the number of gates used and more about the number of IC's you need and the concomitant wiring demands. In these cases, you may be looking for a quick way to build a device with few MSI IC's rather than a truly minimal design that may require many SSI IC's. If we are so constrained, the design procedure used is much less formal. It reduces to:

- clearly understanding the given design specifications and discovering any additional requirements needed to complete the design specification,
- recognizing (from experience with digital IC's) which IC's provide the capabilities that can be used to realize the functions demanded of the design,
- forming a paper design that meets the design specification (usually with MSI circuits) using cut-and-try methods, then
- prototyping, debugging, and producing the design.

In this laboratory exercise, you will be taken through the design procedure that uses this approach to design a vending machine controller. Do not be concerned that you may have learned the state diagram and transition table techniques for naught; you will have ample opportunity in Hardware Lab 5 to use the formal design procedures you have learn.

Vending Machine Controller Design

The function of a vending machine controller is to tally the money deposited into a vending machine, to return money when requested from the user and to allow the vending mechanism to operate once sufficient money has been deposited. The outputs of such a vending machine controller depend on its current inputs (such as the value of the inserted coin, or a request for return of money) as well as its past inputs (such as the tally of the previous coin deposits). A digital circuit whose output depends on its past as well as present inputs is called a sequential circuit or finite state machine (FSM). When the change of state of the FSM is synchronized with a clocking event, we give the machine a special name: synchronous FSM or synchronous machine for short. In this laboratory exercise, we will design a synchronous machine that will function as a controller for a vending machine.

The first step in any synchronous machine design is to obtain a functional specification. The purpose of the *functional specification* is to define all of the functions that the device must be cable of performing. This specification is informal many times and (more importantly) incomplete. By incomplete, we mean that it contains insufficient information to completely define our design.

For example, the functional specification of a vending machine controller might be given to you verbally as follows:

> *"I'd like you to design a controller for a vending machine that dispenses only one type of product. The product vended will cost $0.30 but we need to have the capability of raising the price. Your controller should allow users to request their money back if they change their minds. You should do this in your design by providing a digital output line that produces a +5 V. signal to enable the money return mechanism when a user presses the coin return lever on the front of the machine.*
>
> *We'll be using these controllers to replace old controllers in existing vending machines on the campuses of educational institutions. The existing vending machines already have an electronic circuit that produces a +5 V. signal on one line when the product is requested and a +5 V. signal on another line when the coin return lever is depressed. Your controller will have to produce a digital output signal that is high when $0.30 has been deposited to enable the vending mechanism of the existing vending machine.*

Any questions?"

This initial functional specification statement, as we shall see, is indeed incomplete[1]. This means that as we proceed with the design, we will encounter many questions whose answers cannot be found in the informal functional specification quoted above. The sum total of the way we answer these questions will determine the complete *design specification* of the product. The *design specification* specifies in detail the values (in Volts, Amps, logic values, etc.) of all inputs to and outputs from our device and the operations that the device will perform (including any limit on propagation delay time) for every input condition.

To move from a functional specification to a design specification, we will need to answer many questions. In practice, when we encounter questions, we would ask them to the people responsible for using our design so that the end product we produce does everything the way they would want it had they thought through the design thoroughly. If there is no one to get answers from, we could examine one of the existing controllers our controller is meant to replace; chances are, our new design will have to perform many of the same functions - our controller will certainly have to use the same inputs and produce the same outputs. If we can't find an old controller and if there is no one to answer our questions (such as in this exercise) we need to make reasonable engineering assumptions using our own experience of the way vending machines operate and proceed based on those assumptions. Let's pose and answer some questions that we'll need to address to complete the vending machine controller design specification.

Assumptions

In the verbal functional specification, we are told that our design will need to accommodate a price change. Will the price change be given in increments of $0.01, $0.05, $0.10, etc.?

Assumption: It will probably work like all vending machines and have $0.05 increments.

What might the price rise to ultimately?

Assumption: If the price is $.30 cents today, and *assuming* a 5% inflation rate, the price will hit $1.00 if we *assume* a life span of 25 years. Since the useful life of this product is probably less than 25 years, *assuming* an ultimate price of $1.00 is reasonable.

How will the price be made available to our controller?

Assumption: Since our price will be a multiple of $0.05, we will save on hardware if we *assume* that the price will be a binary value representing the number of $0.05 increments

[1] The author has never worked on a project where the initial functional specification provided enough information to completely specify all aspects of the final design.

needed before we will vend our product. This means that if the price is 30 cents, the price input to our machine will be 30D/5D = 6D = 0110B. Since the price might eventually increase to 100 cents, our binary price inputs must be capable of representing numbers at least as big as 100D/5D = 20D = 10100B. This means that there must be at least five 1-bit price input lines to our controller.

What should we do if someone puts in too much money?

Assumption: This is a machine at an educational institution so we won't have to give change.

When we plug in our vending machine and power is first supplied to the controller, how can we guarantee that the controller is initialized to a state consistent with zero money deposited?

Assumption: Our design will need to contain a means to reset the controller so that when power is applied, it is in a state corresponding to a deposit of no money. The reset may need to be performed manually or automatically. Let's assume that an active-high external reset control signal is currently available in the existing vending machines for this reset function and that it will be triggered manually.

How can we detect whether a nickel, dime or quarter has been inserted?

Assumption: The existing vending machine must already have a mechanism for detecting the denomination of the coins inserted and providing this information to the existing controller. We will assume that digital signals from the vending machine coin mechanism are available as inputs to our controller and will take on the following values:

- A *quarter* input line momentarily equals 1 if a quarter is inserted, 0 if no quarter is inserted.
- A *dime* input line momentarily equals 1 if a dime is inserted, 0 otherwise.
- A *nicke*l input line momentarily equals 1 if a nickel is inserted, 0 otherwise.
- A *sample-sum* input line signal contains one $0 \rightarrow 1 \rightarrow 0$ digital-valued pulse that occurs only during the time any coin input line (*quarter*, *nickel* or *dime*) is momentarily high. (This signal will be used to trigger a circuit that will sample the coin input lines when a coin is inserted.)

We'll also assume that only one of the coin-sensing lines (*quarter*, *nickel* or *dime*) is high at any one time.

By asking and answering questions about the design, we have obtained all of the information you will need to complete your design specification. (Including all of the inputs needed by and outputs needed

from our controller as is shown in Figure 4-1.) Ideally the complete design specification should be completed before the design is begun, as you will do in Task 4-1. In practice, however, there are often things that are overlooked, so that additional specifications are added and inconsistencies in the design specification are uncovered and corrected as the design unfolds.

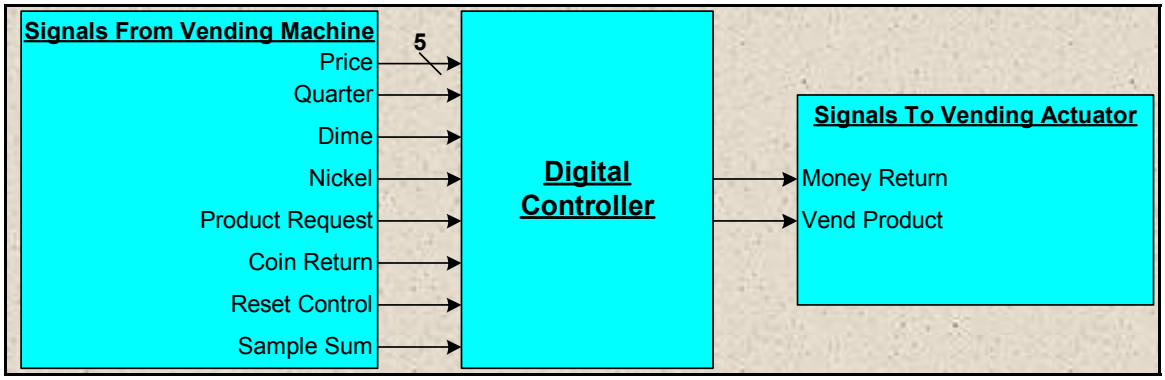


Figure 4-1. Input/output specification of a digital controller.

Task 4-1. Create Your Design Specification

Using the functional specification and the answers to the questions above, create a design specification for the design we're creating. This specification should list all inputs and outputs and should specify that each signal is a logic value[2]. Also describe in a concise but thorough way, how this controller should respond to each of it's inputs.

The Design Process

The design process consists of producing a detailed schematic diagram, which, when realized, satisfies the complete design specification. When the product to be designed is complex, this schematic diagram is usually constructed using a combination of what is known as a top-down process and a bottom-up process.

In a top-down approach, the design process starts with a high-level view of the completed product. The product to be designed is represented first as an interconnection of high-level interacting subsystems. The signals used to allow these subsystems to interact and specifications detailing the way each subsystem must respond to its inputs are precisely defined at this stage. Figure 4-1 along with the design specification you created in Task 4-1 is the first step in the top down design process.

[2] In real applications, some signals may not be logic values. In such cases we would need to design analog circuitry to convert these to logic signals.

Next, each subsystem is broken down into smaller interacting subsystems, a design specification for each of these subsystems is defined, and so on, until we reach a stage where the circuitry needed to construct a subsystem is obvious to the designer.

By contrast, in a bottom-up approach, existing IC's and gates are assembled to build subsystems. These subsystems are assembled to form bigger subsystems, and so on, until the requirements of the specification are satisfied.

The design we are embarking on is simple enough that the top-down and bottom-up approaches are almost the same. As we proceed, you will see aspects of both in the design process.

A Top-Down and Bottom-Up Design

Two of the functions that the controller needs to perform are:

- Tally the amount of money deposited.
- Compare the amount deposited with the price.

There are MSI circuits that can implement these two functions: one is an adder, and the other is a comparator. Using these two MSI circuits, and a register for storage, a signal flow diagram of one possible partial design of our controller is shown in Figure 4-2.

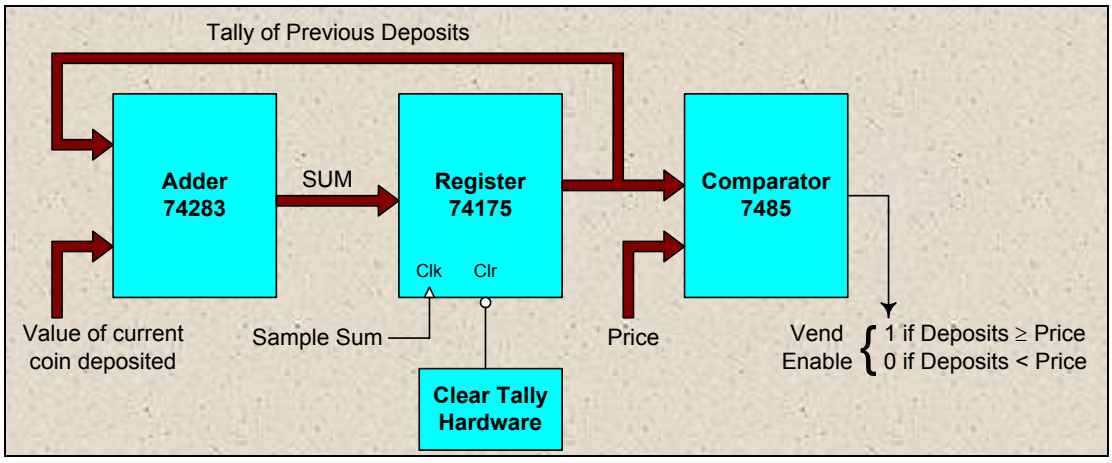


Figure 4-2. Signal flow diagram of vending machine controller.

In the design of Figure 4-2, the adder adds the value of the current coin deposited to the tally of the coins deposited since the last product was vended. This sum, stored in the 74175 register, represents the state of our finite-state machine. When the *sample-sum* (synchronizing) input has a positive edge, the 74175 register is loaded with the sum, thereby updating the state of the synchronous machine. (Recall our assumumption that a *sample-sum* signal is available to sample the value of the adder output at the time a coin is inserted.) The output of the register is fed back to the 'A' inputs of the adder so that it may be

added with subsequent coin deposits. The output of the register is also fed to a comparator. The comparator is a digital circuit that compares its two inputs and produces several output signals. The output we'll need is the one that is active (high) when the tally input (sum) is greater than or equal to the price input. The *clear-tally* hardware is a circuit that we'll design to clear the tally under the appropriate conditions – such as once the product has been vended or when a customer wants their money back.

Our design approach has aspects of both a top-down and bottom-up design. It looks like a top-down approach because we have broken the controller of Figure 4-1 into the four interacting subsystems. By immediately recognizing that three of the subsystems in Figure 4-2 can be realized with available MSI IC's, the design process also has aspects of a bottom-up approach. Because the *clear-tally* hardware cannot apparently be realized using only one IC, it is left defined only as a functional block that will require further design work. Let's use a bottom-up design procedure to fill in the details of the *clear-tally* functional block.

Bottom-Up Clear-Tally Hardware Design

From our own experience with vending machines, we will *assume* that the tally should be cleared when one of the following happens:

- Someone activates the coin return, OR
- We vend the product.

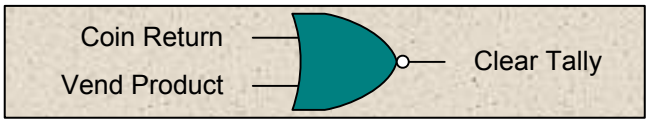


Figure 4-3. Partial clear-tally hardware design.

Since our *clear-tally* line is an active low signal, a simple NOR gate will implement this function (shown in Figure 4-3) provided the *coin-return* and *vend-product* signals are active high – which we know to be true from our original problem specification.

We must design the *vend-product* signal to be high (and actuate the vending mechanism) when the following is the case:

- The *product-select* line is high (i.e., the product has been selected), **AND**
- The *vend-enable* line (see Figure 4-2) is high, (i.e., the tally equals or exceeds the price.)

Using this understanding, the *vend-product* signal can be represented as (*product-select*)•(*vend-enable*). The complete design of the *clear-tally* hardware that results is shown in Figure 4-4.

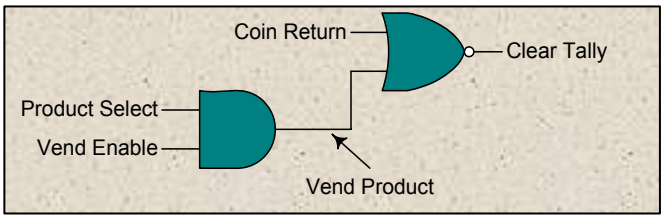


Figure 4-4. Clear-tally hardware design.

If we could look ahead at the completed design, we would see that the *only* AND and NOR gates we will need in the completed design will be

those used in the *clear-tally* hardware of Figure 4-4. If we use the design of Figure 4-4, we will have to use two IC's for the *clear-tally* hardware: one for the AND gate and one for the NOR gate. We can minimize the cost of our design, if we build the *clear-tally* hardware using only one IC. We can do this using only a 7402 (quad 2-input NOR gates) and the laws of Boolean algebra.

The *clear-tally* hardware realizes the Boolean algebraic function:

$$\text{ClearTally} = \overline{\text{ProductSelect} \bullet \text{VendEnable} + \text{CoinReturn}}\ .$$

If we double complement the product term we get,

$$\text{ClearTally} = \overline{\overline{\overline{\text{ProductSelect} \bullet \text{VendEnable}} + \text{CoinReturn}}}\ .$$

Then applying DeMorgan's law to this term we get,

$$\text{ClearTally} = \overline{\overline{\overline{\text{ProductSelect}} + \overline{\text{VendEnable}}} + \text{CoinReturn}}\ .$$

This equivalent form of the *clear-tally* function can be built using two NOR gates and two inverters. If we connect two of the four NOR gates of the 7402 to function as inverters we can build the circuit of Figure 4-4 using only one 7402 IC. Prove to yourself that the design Figure 4-5 is equivalent to that of Figure 4-4.

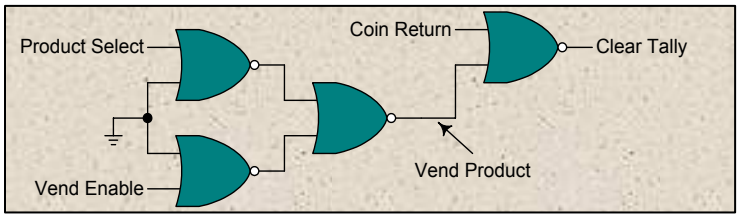


Figure 4-5. Tally clear hardware using one 7402 IC.

Adder Hardware Design

Let's design the adder hardware next. The first question we need to answer is how many input bits our adder will need and how many output bits it must produce. Using our assumption that the smallest coin accepted by the coin mechanism is a nickel, (and that all other coins are multiples of 5 cents) we can minimize the number of adder input (and output) lines if each adder increment represents 5 cents. This means that our adder will function by adding the effective value of each deposit (5 for a quarter, 2 for a dime and 1 for a nickel) to the existing sum. This also means that largest input to the adder from the coins sensing mechanism will be 0101, the binary equivalent of 5 for a quarter. The other input to the adder, the tally, must be capable of representing the ultimate price of our product, $1.00, which, in 5-cent increments, is represented by the binary string, 10100; hence our adder must be capable of at least 5-bit arithmetic.

Realizing that MSI circuits allow us to do 4-bit arithmetic easily, but require a fair amount of extra work to do 5-bit arithmetic, let's revisit our assumption that the ultimate price of the product may increase to $1.00. Recall that this assumption was based on a 25-year life time with a 5% inflation rate. If we assume either a 20 year life span or a 3.7% inflation rate, the ultimate price of our product will be $0.75, which is represented in our adder by 75D/5D = 15D = 1111B – which only requires 4-bit arithmetic! Let's assume that our revised assumption is reasonable. This means that we can design our adder to work using 4-bit arithmetic.

An MSI circuit which does 4-bit binary arithmetic is the 74283, 4-bit binary full adder (with fast carry). The connection (pin-out) diagram and logic symbol for the 74283 are shown in Figure 4-6[3]. The 74283 adder is capable of adding two 4-bit operands (supplied to the A and B input lines) to a 1-bit incoming carry (supplied to the C0 input line) and producing a 5-bit result on lines C4, S4, S3, S2, S1, and S0. Using base-10 arithmetic, the input output relationship of the 74283 is given by,

$$C0 + (A1 + B1) + 2*(A2 + B2) + 4*(A3 + B3) + 8*(A4 + B4) = S1 + 2*S2 + 4*S3 + 8*S4 + 16*C4$$

where, for this equation, + represents plus and * represents multiplication. An example of the addition operation is shown in Table 4-1.

Table 4-1. Example Operation of the 74283 4-Bit Binary Adder.

	A4	A3	A2	A1	B4	B3	B2	B1	C0	C4	S4	S3	S2	S1
Logic Level	1	0	1	0	1	0	0	1	0	1	0	0	1	1

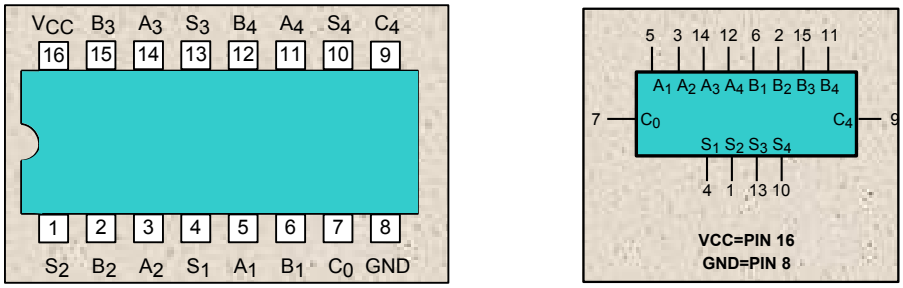


Figure 4-6. PPin-out diagram and logic symbol for the 74283 4-bit binary full adder.

Let's use the B inputs, B4 – B1 input lines, to provide the present tally of previous deposits to the adder as shown in Figure 4-7. (In the schematic diagram of Figure 4-7, the pin-label postscripts 4 and 1 indicate the most and least significant bits, respectively.). Let's use the A inputs, A4 – A1 input lines, to provide the value (divided by 5) of the present coin input (for quarters and dimes) and the carry input, C0, to act as the nickel input as shown in Figure 4-7. Using the definition of the coin sensing input lines, prove to yourself that connecting these lines to the A inputs as shown in Figure 4-7 will provide the appropriate input bit patterns given in Table 4-2.

[3] The pin labeling convention used for this IC is different in the data books supplied by different manufacturers.

Also, prove to yourself that the schematic of Figure 4-7 will function to add the present deposit (nickel, dime or quarter), to the tally input, which has yet to be designed.

Table 4-2. B and C Input Values as a Function of Coin Value

Coin Deposit	A4	A3	A2	A1	C0
Quarter	0	1	0	1	0
Dime	0	0	1	0	0
Nickel	0	0	0	0	1

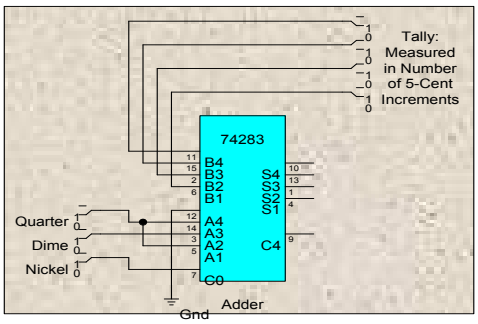


Figure 4-7. Adder Design using 74283.

Task 4-2. Simulate the Adder Hardware Circuit

Using LogicWorks™, construct a simulation of the adder design shown in Figure 4-7. (The pin numbers in the diagram are provided to help you when testing this circuit in the hardware laboratory as described in Task 4-5.) You will find LogicWorks™ parts for the 7400 series IC's in the 7400.clf library. You get access to this library by using the library selection area of the parts palette. The pin locations on the LogicWorks™ 7400 series IC's are sometimes different from those shown in this manual, so be careful when you construct your simulation; however the pin numbers on these drawings represent the identical input/output signals, respectively, as the LogicWorks™ pin numbers.

To test the circuit of Figure 4-7, connect binary switches to the tally input lines and the coin-sensor input lines and verify that the circuit performs the correct addition operations. You do not need to test all 128 possible input combinations. For example, do you need to test that the sum output works correctly when two or three of the coin-sensor input lines are active simultaneously? (Explain your answer in your report.) Test a sufficient number of combinations so that you are convinced that the correct sum is calculated when a nickel, dime or quarter is input into your machine. What is the maximum amount of money that your adder's output can indicate? Will the adder's output be sufficient to represent our revised ultimate price ($0.75) plus some overage due to excess deposits? Remember to record in your lab notebook all of the measurements that you make. You will be using these measurements to validate the performance of your hardware realization when you perform Task 4-5.

Table 4-3. Pin Definition for 74175 Register.

Pin Names	Definition
D0-D3	Data Inputs
CP	Clock Input (Pos.-Edge Triggered)
$\overline{MR}$	Reset (Clear) Input (Active Low)
Q0 - Q3	State Outputs
$\overline{Q0}$ - $\overline{Q3}$	Complement of State Outputs

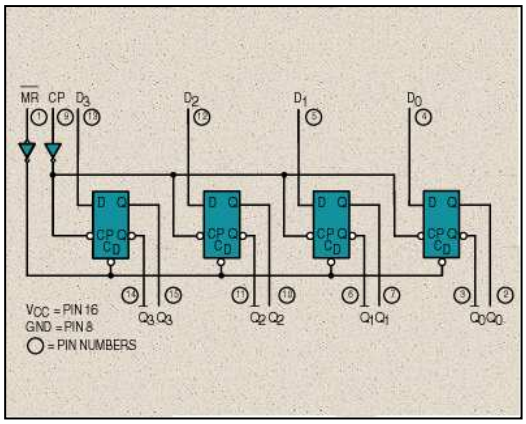


Figure 4-8. Internal logic diagram of a 74175, quadad D fD flip-flops.

Register Hardware Design

In our controller design, we will use the 74175, a positive-edge triggered, quad D flip-flop register with active-low reset (clear) control, to store the controller's state, (i.e., sum output of the adder). The internal logic diagram of the 74175 is shown in Figure 4-8 and its pin-out (connection) diagram and logic symbol are shown in Figure 4-9. The definition of each of these inputs is shown Table 4-3. (Note that in this table and in Figure 4-8 and Figure 4-9 we are using the data book [rather than textbook] abbreviation for the flip-flops' clock input, CP, [as described in Figure 3-4.])[4]

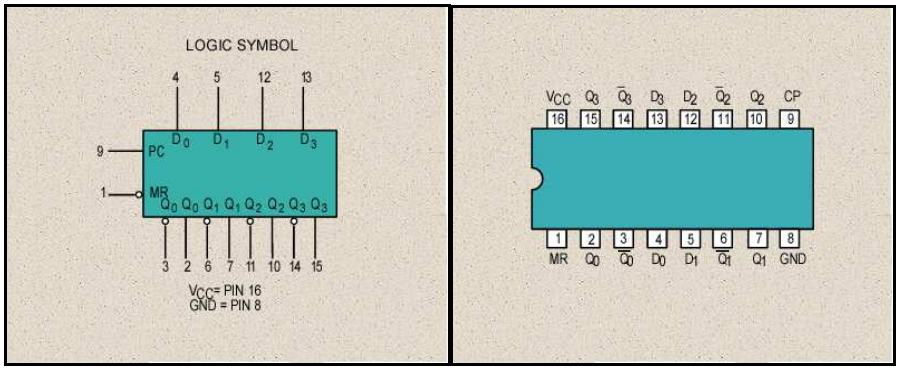


Figure 4-9. Logic symbol and connection diagram (pin-out) for a 74175, quad D flip flops.

Task 4-3. Simulate the Adder with Output Register & Clear-Tally Hardware

Using the 74175 register IC in the 7400.clf library of LogicWorks™, simulate the adder with register circuit shown in Figure 4-10. (Be aware that the 74175 IC schematic symbol shown in Figure 4-10 does not show the complement of the state variables as outputs – while the schematic symbol you get from the LogicWorks library will.) Test this circuit and record the results of your tests in your notebook.

[4] Earlier in this manual we used the typical textbook notation. Here we purposely use the data manual notation to help you familiarize yourself with it. The ability to use this notation will be important as you begin to design your own systems.

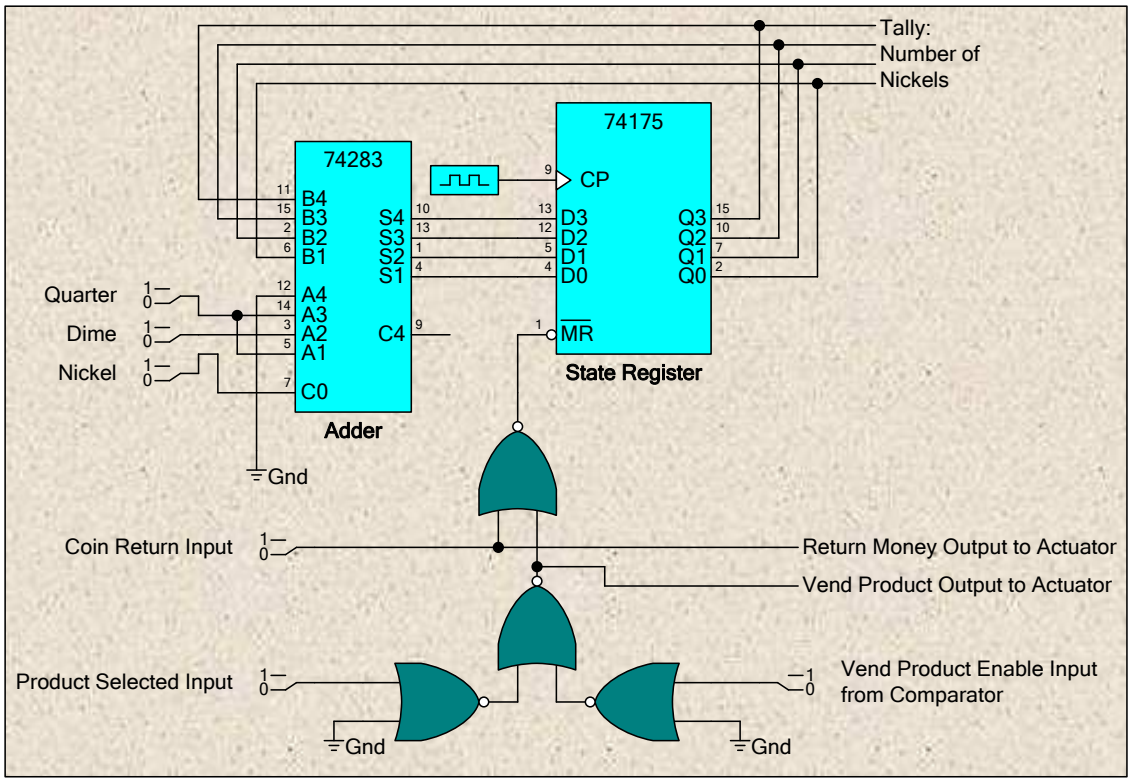


Figure 4-10. Adder with register and clear tally hardware.

(When building this circuit in the hardware laboratory (Task 4-5) be sure to disconnect the switches you used in Task 4-2 that drove the tally lines. Also, in the hardware laboratory, be sure to drive the clock (CP) input with a de-bounced switch. When testing this circuit in the hardware laboratory, you will also want to connect the Q3-Q0 state signals to LED's so that you can easily monitor the addition and verify that it is proceeding correctly. Use the records of the test you completed on your simulation to verify that your hardware implementation is working correctly.)

Next use a 7402 IC in from the 7400.clf to build the clear-tally hardware circuit. Test that it works correctly and record the results of these test in your notebook.

Next, add the clear-tally hardware circuit to your adder/register circuit, test the resultant circuit and record the results in your notebook. Be sure to use sufficient tests (which should include testing the coin-return, product-select and vend-product inputs) to insure that your circuit functions correctly beyond a reasonable doubt. Record in your notebook the results of these tests. (When you build this circuit in the hardware laboratory, you will use the results of the tests you performed on you simulation to verify that your hardware implementation is working correctly.)

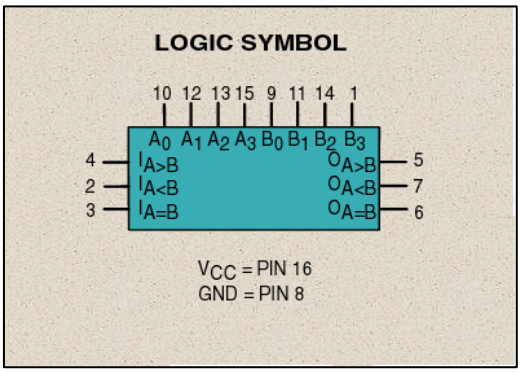


Figure 4-11. Logic symbol for a 7485, 4-bit magnitude comparator.

Table 4-4. Pin Definitions for a 7485.

Pin Names	Definitions
A0-A3, B0-B3	Input Ports
$O_{A=B}$	A Equal to B Output
$O_{A>B}$	A Greater than B Output
$O_{A<B}$	A Less than B Output
$I_{A=B}$,$I_{A>B}$,$I_{A<B}$	Expander Inputs

Adding the Comparator

A magnitude comparator is a device that compares the magnitudes of its binary inputs and provides outputs indicating whether the inputs are equal or not. The logic symbol and pin definitions are contained in Figure 4-11 and Table 4-4, respectively. The function definition table for the 7485 is shown in Table 4-5.

Table 4-5. Function Definition Table for 7485, 4-Bit Magnitude Comparator.

Comparing Inputs				Cascading Inputs[5]			Outputs		
A_3,B_3	A_2,B_2	A_1,B_1	A_0,B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	$O_{A>B}$	$O_{A<B}$	$O_{A=B}$
$A_3>B_3$	X	X	X	X	X	X	H	L	L
$A_3<B_3$	X	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2>B_2$	X	X	X	X	X	H	L	L
$A_3=B_3$	$A_2<B_2$	X	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1>B_1$	X	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1<B_1$	X	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0>B_0$	X	X	X	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0<B_0$	X	X	X	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	L	L	H	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	H	L	L	H	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	X	X	H	L	L	H
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	H	H	L	L	L	L
$A_3=B_3$	$A_2=B_2$	$A_1=B_1$	$A_0=B_0$	L	L	L	H	H	L

[5] These inputs are provided to allow several comparators to work together to compare numbers with more than 4-bits.

Task 4-4. Finish the Vending Machine Controller Design

Using the logic symbol and truth table definition, prove to yourself that the circuit of Figure 4-12 will compare the price with the tally and produce the appropriate outputs. Be sure that you can explain why the connections to the $I_{A=B}$, $I_{A>B}$, and $I_{A<B}$ inputs are as shown in Figure 4-12. Next simulate the circuit of Figure 4-12 and test it. (Record the results of your tests in your notebook so that they can be used to validate the performance of your hardware implementation in Task 4-5.) Set the price to a value of 6 (for $.30) and verify that your circuit performs the functions consistent with the functional specification. Be careful that the tests you design to verify your controller's operation are sufficient to convince the reader of your report that you have indeed built a controller that meets all design specifications. Are there any design specifications that our design does not meet?

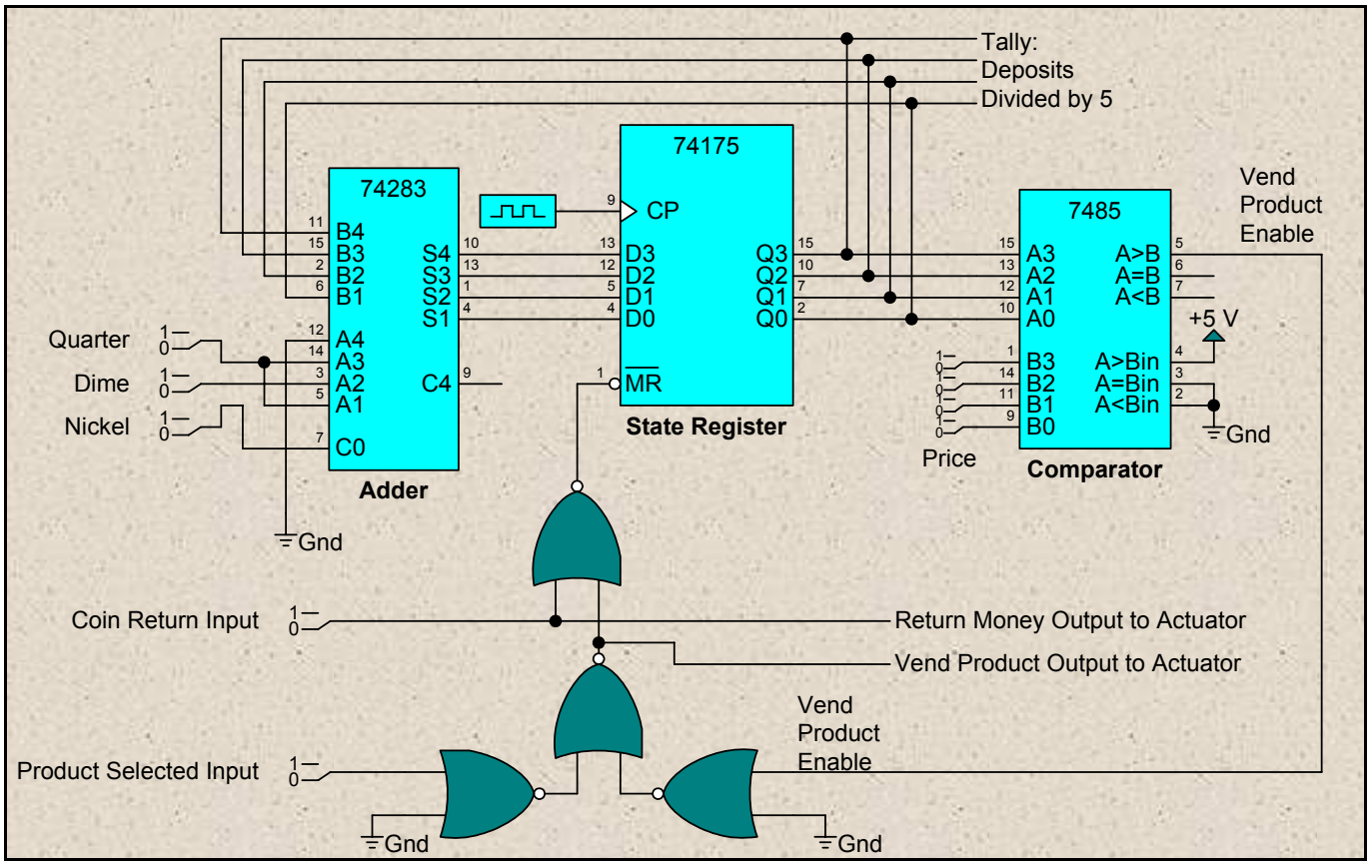


Figure 4-12. Complete vending machine controller design.

One of our functional specifications dealt with resetting the controller after power is applied so that it is in a state corresponding to a deposit of no money. How can this be accomplished with the design of Figure 4-12? What operations would you have the installer perform after plugging in the vending machine so that your controller does not allow product to be sold for less that $0.30?

Another design specification is that the *vend-product-output-to-actuator* signal be active when the tally is greater than or equal to the price (provided the *product-select* signal is high). Will the vending machine controller vend product under the equality condition or just when the tally is greater than the price; is our controller being fair, or is it gouging our customers? (You will need to carefully compare the connections made to the $I_{A=B}$, $I_{A>B}$, and $I_{A<B}$ inputs of Figure 4-12 with the information in Table 4-5 to answer this question.)

We have not used the traditional state diagram/transition table/Karnaugh map approach in this design process. Even so, the results of our process can be classified as either a Mealy or a Moore machine. Which is it? Be sure to justify your answer in your report.

Task 4-5. Implement the Complete Controller Using Hardware

You have built a simulation of the controller using LogicWorks™. The purpose of this simulation was to verify that the design of Figure 4-12 meets the design specification. Once you are satisfied that it does, you will build this circuit in the hardware lab using the results of the tests you performed on the simulation to validate the performance of your hardware implementation.

Tip: Before you begin to build your hardware circuit, recall the way you built the simulation: you built one piece (the adder) then tested it; you added on a second piece (the state register) and tested the combination; you built a third circuit (the *clear-tally*), tested it independently, then added it to the partially completed circuit and tested that circuit, etc. We had you complete the circuit in this fashion because this is the way complex circuits are built; pieces are constructed and tested independently, then added to the partially completed design and then that partial design is tested.

There is a reason circuit construction is handled this way: if anything does not perform correctly, the bug is most likely to be found in the piece we most recently added to our partial design, or is caused by the way the piece we added interacts with our partial design. This limits where we need to search for an error and makes debugging relatively easy. If, by contrast, we construct the entire circuit before we begin debugging, we often have little clue as to where to begin our search for errors if the circuit does not perform as expected. When you construct this circuit in the hardware lab, you are likely to have the easiest time debugging if you build and test your circuit in the same way you built your simulation, by sequentially following the instructions of Task 4-2, Task 4-3, and Task 4-5, respectively.

In your report, compare the difficulty of building a hardware realization vis-à-vis a simulation of the circuit shown in Figure 4-12. Based on your observations, what role do you believe circuit simulation should play in the design of new digital circuits?

Excess Money Deposits

In Task 4-3 we selected the 74283, 4-bit binary adder to add the tally of previous deposits to the current coin deposit. This adder produces a 5-bit output that is capable of representing a decimal value of 31, which is equivalent to $1.55 using our nickel-per-increment scheme. This is sufficient to meet the expected eventual price rise of $0.75; however, there are two problems. First, because only the least significant 4 bits of this value are stored in the state register and fed back to the input to the adder, our addition/register circuit will only perform correctly for tallies less than or equal to $0.75. That means that there is not room for excess deposits once the price reaches $0.75. The second problem also deals with this 4-bit limitation of the state register: only the 4 least-significant bits of this stored sum can be supplied to the comparator. This means that the largest tally that the comparator can detect is $0.75. Once $0.80 is deposited, the state of the state register becomes 0000B, even though the output from the adder is 10000B.

There are several ways to handle excess deposits. One way is to return all deposited money once the tally exceeds $0.75. Another way is to make the assumption that users are required to make their selection before they deposit money; then you could automatically vend the product whenever enough money was deposited. Another way might be to use a 5-bit state register (instead of a 4-bit state register) and store the most significant bit (C4) from the adder. To make this work correctly, you would need to redesign your adder and comparator circuits to accommodate 5-bit tally values. The advantage to this approach is that it would allow us to accommodate our initial estimate of the ultimate product price. Other solutions are certainly possible.

Task 4-6. Account for Excess Money Deposits

Using the schematic of Figure 4-12 as a starting point, produce the schematic of a circuit that will cause the *return-money-output-to-actuator* signal to become high once the tally exceeds $0.75. Describe another way that you might handle excess deposits and draw a schematic diagram of a circuit that would implement this approach. You are free to use one of the suggestions above or create your own idea. You are also free to make any reasonable assumptions about what inputs the vending machine might provide to you or what mechanisms exist in the vending machine that can be digitally controlled to effect your excess deposit scheme. Be sure to list in your report any assumption you make.

Task 4-7. Simulate Your Design of Task 4-6

Simulate one of the designs you created in Task 4-6 using LogicWorks™. Test your circuit to insure that it works and record the results of your test. Describe in words in your report how your design accounts for excess money deposits.

8-Bit Arithmetic

If you review your design, you will see that it functions according to our functional specification, provided we can accept the revised assumption of a smaller inflation rate or a shorter life span to get an ultimate price of $0.75. If this was a real application, we would discuss our revised assumption with people knowledgeable about the vending business and decide whether a $0.75 deposit limit was acceptable; however, even if this revised assumption is acceptable to experts in the vending business, we see that there are some undesirable limitations to our design; namely, we cannot accurately handle deposits over $0.75. Also, if we are held to the $1.00 eventual price rise, (and allowing for some overage in the buyer's deposit), we will need to enhance the design of Figure 4-12 to account for larger deposits, including the eventual acceptance of dollar-bills. Assuming we need to account for dollar-bills, and allowing for a certain amount of overage, we'll need to re-design our circuit to account for deposits somewhat greater than $1.00, or 20X$0.05. This means our adder must be able to yield valid sums for tallies somewhat great than 20D = 10100B and our comparator must be able to make comparisons of at least 5-bit numbers as well; hence our controller will need to be redesigned to account for greater precision. Because the 74283 is a 4-bit adder, any adder we build using multiple 74283 IC's will be capable of precision that is a multiple of 4 bits. The next tasks step you through the process of building a controller simulation that uses 8-bit arithmetic.

Task 4-8. Redesign the Adder to Perform 8-Bit Arithmetic

Using multiple 74283 adder IC's, re-design the adder and state register circuits to perform 8-bit arithmetic and to allow dollar-bills to be input. The information contained in Table 4-1 will help you understand the input/output relationships of the 74283. You may make the same assumptions regarding the dollar bill input as we made about the coin inputs. Show your schematic in your report. Do not build your design.

Task 4-9. Complete the Controller Design for 8-Bit Precision

Using the results of Task 4-8 and the logic symbols and truth table definitions of the 7485, (the 4-bit magnitude comparator), design then draw the schematic circuit of a controller that is capable of a full 8-bit price/tally comparison. To complete this task you will need to understand how the cascading inputs, $I_{A=B}$, $I_{A>B}$, and $I_{A<B}$, of the 7485 affect its output values. (See Table 4-5.) You will also need to use the $O_{A=B}$, $O_{A>B}$, and/or $O_{A<B}$ outputs of one 7485 (which compares the least significant bits of your 8-bit value) to control the cascading inputs of another 7485, which will compare the most significant bits. Do not build this circuit. You may wish to use LogicWorks™ to help in constructing the schematic of your design. (Using LogicWorks™ here will also help you when you begin the next task.)

The 8-bit arithmetic capability of your controller allows you to accurately account for deposits up to $12.75. (Your 8 bit adder can accurately account for FFH = 255D nickels, which corresponds to $0.05*255 = $12.75.) Given that the ultimate price of your product is $1.00, you do not need to account for deposits in excess of $12.75 in your design[6].

Task 4-10. Simulate and Verify the Design of Task 4-8 and Task 4-9

Using the 7400 series IC's available in LogicWorks™, build a simulation of the circuit you designed in Task 4-8 and Task 4-9 and verify that it works correctly. With the benefit of having completed this simulation, assess the relative difficulty of building, debugging, and modifying a simulation versus a hardware implementation. Comment in your report on any advantages you believe exist when you simulate a design before you build it using hardware.

Report Writing Tips

When you write your report, break it into two major sections: one to deal with the 4-bit controller and one for the 8-bit controller. When discussing the 8-bit controller, you can avoid repeating the same information by referring the reader to the subsections in the 4-bit controller description that discuss the corresponding topics. For example, when discussing the design specification for the 8-bit controller, refer the reader to the design specification of the 4-bit controller, then describe only the differences between these two design specifications.

[6] You could try to use the C4 output of the most significant 74283 adder in your design as an input to a circuit that would return money once the deposits exceed $12.75, but you will find that C4 has a glitch when the number of nickels goes from EFH to F0H. This glitch causes deposits in excess of $11.95 to be returned – which is an acceptable solution, even if it is not the one anticipated. There are ways of returning excess deposits without encountering this glitch. You are free to explore them if you are so inclined.

HARDWARE LAB 4: LAB REPORT GRADE SHEET

Name: _____

Instructor Assessment: Top-Down Oriented

Grading Criteria	Max Points	Points Lost
Report Writing		
Complete Title Page		
Organization		
Neatness, Clarity, and Concision		
Statement of Learning Objectives and Outcomes		
Top-Down Description of Work Performed		
The 4-Bit Controller Circuit		
Function and Testing of the Controller Circuit (Incl. Design Spec.)		
Function and Testing of the Adder Circuit		
Function and Testing of the State Register Circuit		
Function and Testing of the Comparator Circuit		
Function and Testing of the Clear-Tally Circuit		
Description of the Excess Deposits Modification		
The 8-Bit Controller Circuit		
Function and Testing of the Controller Circuit (Incl. Design Spec.)		
Function and Testing of the Adder Circuit		
Function and Testing of the State Register Circuit		
Function and Testing of the Comparator Circuit		
Function and Testing of the Clear-Tally Circuit		
Function and Testing of the Excess Deposits Function		
What Was Learned		
Lab Data Sheets		
Self-Assessment Worksheet (The content of the self-assessment worksheet will not be graded. Full credit is given for including the completed worksheet.)		
Lab Score	**Points Lost**	
	Late Lab	
	Lab Score	

Report Writing Reminders:

Caveat emptor: Before writing your report, check with your instructor to see if grading guidelines different from those stated here are to be used

• Title Page: Include **Your Name, ID, Class Time, Course Number, Your Instructors Name, Laboratory Experiment Number and Title**, and **Date Submitted** in all reports.
• Include lab data sheets for all labs. Use lab data sheets to document the lab as you perform it. It saves a lot of time when writing the report.
• Include **Truth Tables** or **Function Tables** as required to explain how circuits work and as proof of circuit tests.
• **Label** all figures or circuits. (E.g. Figure 1. Schematic of . . . , Figure 2. . . , etc.)
• Refer in the text of your report to all circuits or figures that you include in the body of your report.
• Use LogicWorks™ to draw your circuit schematics and cut/paste them into your word-processed document.

HARDWARE LAB 4: LAB REPORT GRADE SHEET

Name: _____

Instructor Assessment: Task Oriented

Grading Criteria	Max Points	Points Lost
Report Writing		
Complete Title Page		
Organization		
Neatness, Clarity, and Concision		
Statement of Learning Objectives and Outcomes		
Description of Assigned Tasks, Work Performed & Outcomes Met		
Task 4-1. Create Your Design Specification		
Task 4-2. Simulate the Adder Hardware Circuit		
Task 4-3. Simulate the Adder with Output Register & Clear-Tally		
Task 4-4. Finish the Vending Machine Controller Design		
Task 4-5. Implement the Complete Controller		
Task 4-6. Account for Excess Money		
Task 4-7. Simulate Your Design of Task 4-6		
Task 4-8. Redesign the Adder to Perform 8-Bit Arithmetic		
Task 4-9. Complete the Controller Design for 8-Bit Precision		
Task 4-10. Simulate and Verify the Design of Task 4-8 and Task 4-9		
What I Learned		
Lab Data Sheets		
Self-Assessment Worksheet (The content of the self-assessment worksheet will not be graded. Full credit is given for including the completed worksheet.)		
Lab Score	**Points Lost**	
	Late Lab	
	Lab Score	

Report Writing Reminders:

Caveat emptor: Before writing your report, check with your instructor to see if grading guidelines different from those stated here are to be used.
• Title Page: Include **Your Name, ID, Class Time, Course Number, Your Instructors Name, Laboratory Experiment Number and Title**, and **Date Submitted** in all reports.
• Include lab data sheets for all labs. Use lab data sheets to document the lab as you perform it. It saves a lot of time when writing the report.
• Include **Truth Tables** or **Function Tables** as required to explain how circuits work and as proof of circuit tests.

• **Label** all figures or circuits. (E.g. Figure 1. Schematic of . . . , Figure 2. . . , etc.)
• Refer in the text of your report to all circuits or figures that you include in the body of your report.
• Use LogicWorks™ to draw your circuit schematics and cut/paste them into your word-processed document.

SELF-ASSESSMENT WORKSHEET

Put an 'X' in the table below indicating how strongly you agree or disagree that the outcomes of the assigned tasks were achieved. Use '5' to indicate that you 'strongly agree', '3' to indicate that you are 'neutral', and '1' to indicate that you 'strongly disagree'. Use 'NA', 'Not Applicable', when the tasks you performed did not elicit this outcome. Credit will be given for including this worksheet with your lab report; however, your **responses will not be graded**. They are for your instructor's information only.

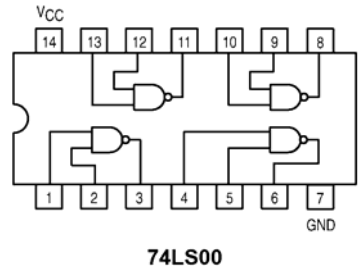
Table __: Self Assessment of Outcomes for Hardware Lab 4: Vending Machine Controller Design

After completing the assigned tasks and report I am able to:	5	4	3	2	1	NA
Describe the function of and use a 7485, 4-bit magnitude comparator.						
Describe the function of and use a 74283, 4-bit binary full adder.						
Describe the function of and use a 74175, quad D flip-flop.						
Interpret and use the function definition table of an MSI circuit.						
Simulate a controller for a simple vending machine.						
Simulate a controller for a simple vending machine.						
Design and simulate an 8-bit controller for a vending machine.						
Appreciate the role of MSI circuits in the design of digital circuits.						
Describe the role of circuit simulation in prototype construction.						
Describe some aspects of the design process.						

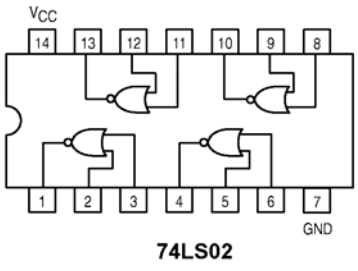
Write below any suggestions you have for improving this laboratory exercise so that the stated learning outcomes are achieved.

PIN-OUT DIAGRAMS

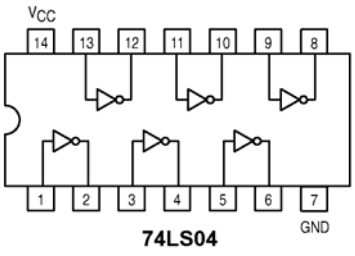
The pin-outs of several TTL devices are given below.



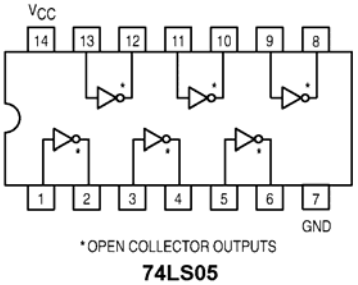
74LS00



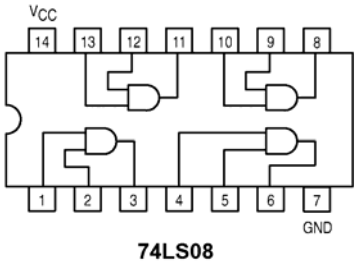
74LS02



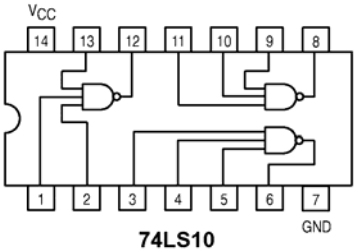
74LS04



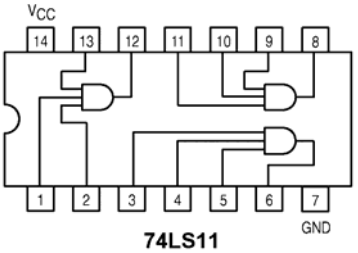
74LS05



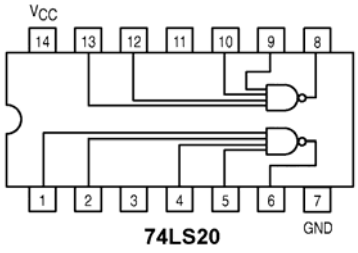
74LS08



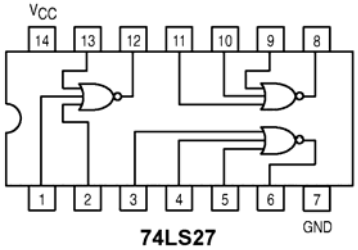
74LS10



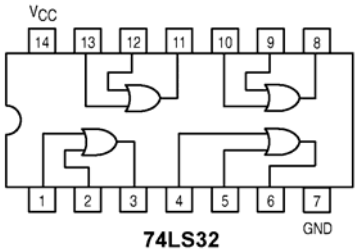
74LS11



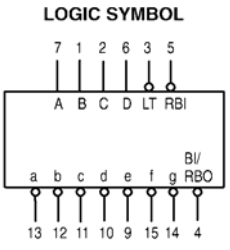
74LS20



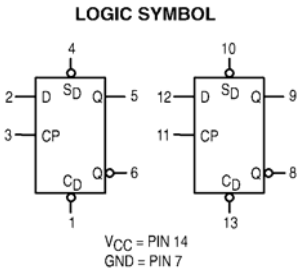
74LS27



74LS32

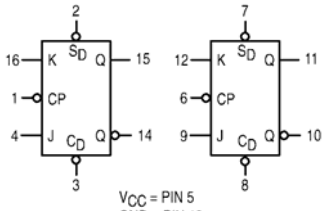


74LS47



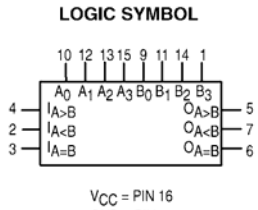
74LS74

LOGIC SYMBOL

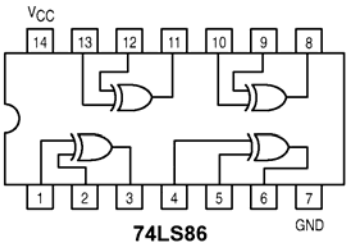


74LS76

LOGIC SYMBOL

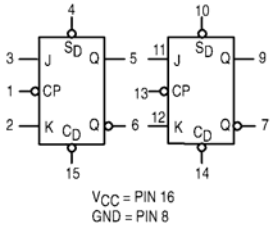


74LS85

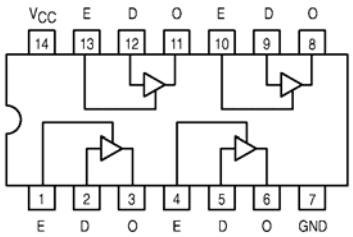


74LS86

LOGIC SYMBOL

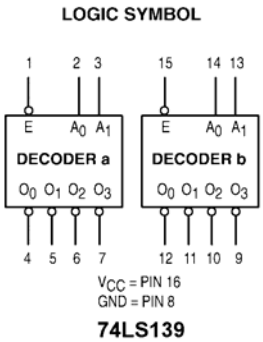


74LS112



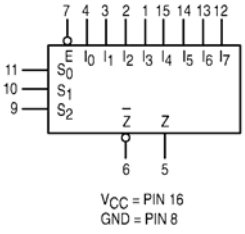
74LS126

LOGIC SYMBOL



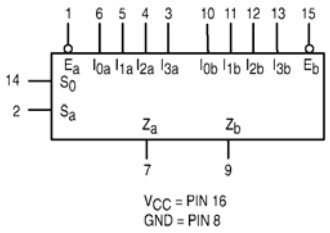
74LS139

LOGIC SYMBOL



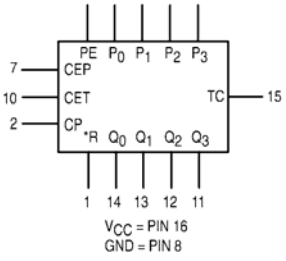
74LS151

LOGIC SYMBOL



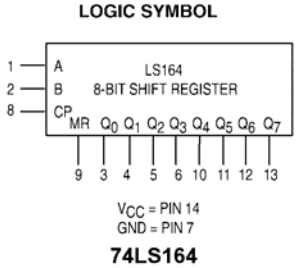
74LS153

LOGIC SYMBOL



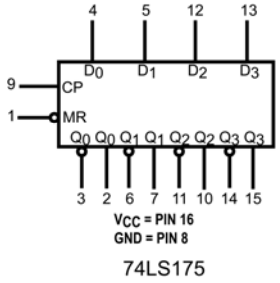
74LS163

LOGIC SYMBOL



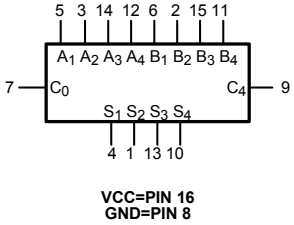
74LS164

LOGIC SYMBOL



74LS175

LOGIC SYMBOL



74LS283