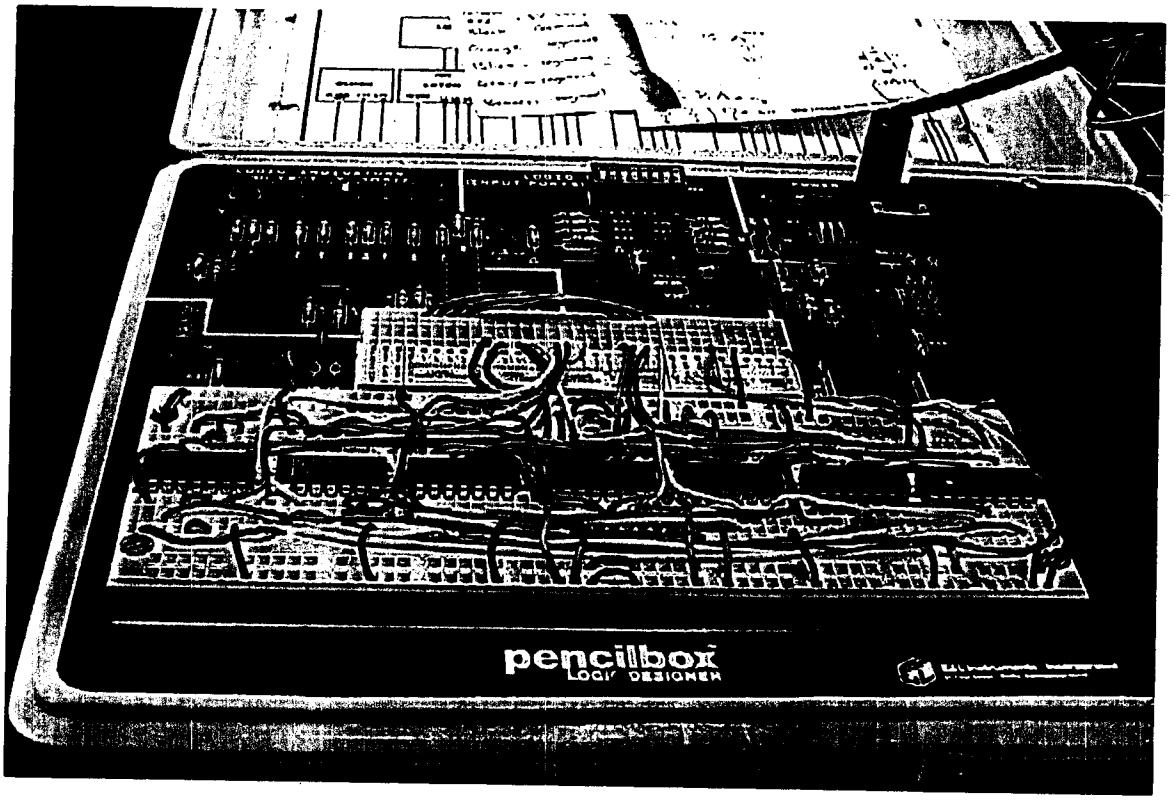
Project Report on
ECE 2504 Design Project 2:
Design and Implementation
of a Simple 4-bit "Computer"



Prepared by

[redacted]

CRN [redacted] MWF 09:05

Submitted to

[redacted]

Table of Contents

Introduction

Design Project 2 introduces the concept of building a complex digital circuit from a collection of simpler elementary logic devices. The project specification calls for a simple 4-bit Arithmetic Logical Unit (ALU) that is capable of performing six given arithmetic and logical operations clear, load, add, subtract, increment, and arithmetic right-shift. Available digital devices were limited to: four multiplexers, an adder, four D-Type flip-flops and gate ICs.

The ALU is to be built around a 4-bit register, and includes the necessary logic to perform six arithmetic operations (CLEAR, LOAD, ADD, SUB, INC and ASHR). The external 4-bit data and the control signal inputs are "keyed in" by the user via Dual-inline-packaged (DIP) switches. A sequential circuit design, the clock is manually pulsed in via a push-button switch. Each time the switch is pressed and released, the instruction as specified by the current control signal inputs is executed.

Background Information

The Concept of an Arithmetic Logic Unit

The ALU performs a variety of arithmetic and logic operations on data. These operations always include addition and subtraction, but may include bit-wise operations, shifting, increments and decrements. The more advanced ALUs have operations that can perform multiplication and division. During a microcomputer operation, the operations that an ALU has to perform are under the control of the timing and control signals, which depends on the instruction codes. Results of the operation can be transferred to memory storage or to an I/O unit.

74LS153 Dual 4-Line to 1-Line Multiplexers

The Multiplexer is a logic circuit that accepts several digital data inputs and selects one of them to pass on to the output. The routing of the desired data input to the output is controlled by SELECT inputs.

74LS175 quadruple D-Type Flip-Flops

The edge-triggered D flip-flop ensures that the output will responds to the D input *only* when the active transition of the clock occurs. The Q output will look exactly like D.

74LS283 4-bit Binary Full Adder

This full adder with look-ahead carry (a propagation delay reduction circuitry that utilizes logic gates to look at the lower-order bits of the augend and addend to see if a higher-order carry is to be generated) adds a 4-bit input with another 4-bit input as well as the carry-in bit. The sum of the addition is asserted on the output pins (typically labeled as Σ3-Σ0).

Subtraction with 2's Complement

When the 2's complement system is used, the number to be subtracted (the subtrahend) is changed to its 2's complement from (by inverting its bits and then added with 1) and then added to the number which the subtrahend is being subtracted from (the minuend). The 74283's Carry-in input then comes in useful when subtracting; it adds one to the already-complemented subtrahend. In other words, a subtraction essentially becomes an addition of a positive number (the minuend) and a negative number (the subtrahend).

Hardware Development

Design Approach

A bottom-up approach was taken for this project. It was explicitly specified ~~in the specifications~~ that the "Enable" pins of the multiplexers and the asynchronous-clear of the D-type flip-flops may not be used. This restricted all data manipulation to the following available input pins:

Table 1: List of Available ICs and Their Inputs

Available IC	**Input pins**
74LS153 4-to-1 Multiplexer	A, B, C3-C0
74LS283 Full Adder w/ carry-in	A3-A0, B3-B0, Cin
74LS175 D-Type Flip-Flops w/ clear	None—D3-D0 were tied to the adder's Σ3-Σ0
74LS00 NAND and 74LS04 NOT logic gates	A3-A0 (NAND); A5-A0 (NOT).

Since all six of the specified instructions were essentially the sum of a binary addition. The heart of the ALU is the Full Adder (FA); it simply adds its three inputs together—A, B and Cin (the "carry-in" bit). Thus, with the proper combination of: a proper input selection from a multiplexer; the correct logic level from Register Q; and a logic high/low fed to Cin, the desired output may be achieved.

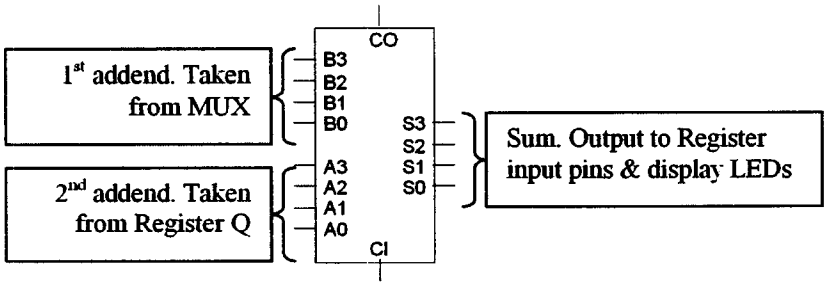


Fig. 1: 4-bit Full Adder Schematic Block Diagram

Table 2: List of Operational Instructions for the ALU

Instruction	**Mnemonic**	**RTL notation**	**F.A.'s "A + B + Cin"**
Add to *w* (store in *w*)	ADDLW	$w \leftarrow w + l$	$l + w + 0 \Rightarrow A + B + 0$
Subtract *l* from w (store in w)	SUBLW	$w \leftarrow w - l$	$l' + w + 1 \Rightarrow A + B + 1$
Increment *w* (store in w)	INCW	$w \leftarrow w + 1$	$0 + w + 1 \Rightarrow 0 + B + 1$
Arithmetic right shift *w* (store in *w*)	ASHRW	$w \leftarrow$ ashr of w	$Q + 0 + 0 \Rightarrow A + 0 + 0$
Clear *w*	CLRW	$w \leftarrow 0$	$0 + 0 + 0 \Rightarrow 0 + 0 + 0$
Load *l* into *w*	MOVL	$w \leftarrow l$	$l + 0 + 0 \Rightarrow A + 0 + 0$

The 4-to-1 multiplexer (MUX) was used as a "Selector switch" in this application. Depending on the logic levels fed into the its control pins A and B, the MUX selects which of its 4 input pins will "pass thru" and subsequently "fed" into the FA's A3-A0 inputs—therefore the MUX is responsible for the first half of the equation. By referring to the above table, it was thus determined that the following control signals would be fed into the MUX.

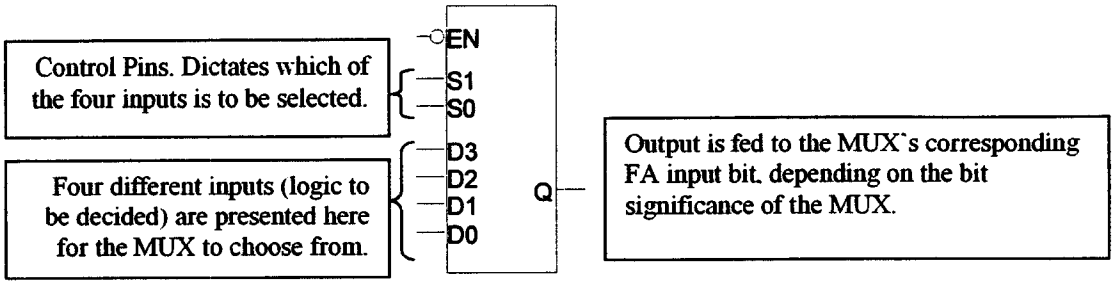


Fig. 2: 4-to-1 Multiplexer Schematic Block Diagram

Table 3: Multiplexer Inputs and Their Assignments

MUX input	**Control Signal Assignment**
C3	**Reserved for ASHRQ**: Tied to an output pin of register Q (Q3-Q1), depending on the actual bit significance of the MUX.
C2	**Literal**: Tied to a bit (S3-S0) from the DIP switches, depending on the actual bit significance of the MUX.
C1	**Complement of Literal**: Logic-wise, it is C2', implemented by first passing the control signal through a NOT gate.
C0	**Logic '0'**. Tied to Ground.

The second half of the Equation may be implemented by tying FA's B3-B0 inputs to its corresponding output of Register Q, like so:

Table 4: Full Adder Inputs and Their Assignments

FA input	**Control Signal Assignment**
B3	**Q3**
B2	**Q2**
B1	**Q1**
B0	**Q0**

However, according to the logic equations in table n, instructions ASHRQ, CLRW, and MOVL requires FA's B3 through B0 be at logic '0'. The FA's Cin bit demands a similar requirement. For instructions SUBLW and INCW, a logic '1' is to be asserted; logic '0' otherwise. As such, the logic levels of B3-B0 and Cin has to be determined by a combination circuit. By means of NAND and NOT gates, the intent of such a circuit determines if the output word from Register Q should be allowed to continue as-is to the FA's B inputs, or be forced logic 'high' or 'low'. In a similar fashion, the logic level of Cin will also be determined based on the control signal inputs.

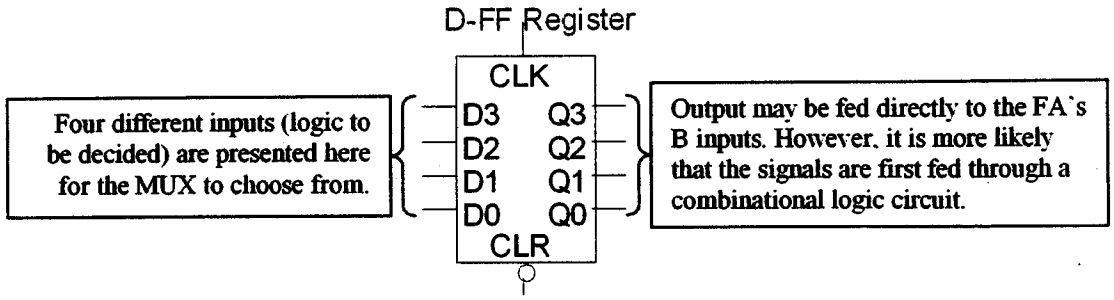


Fig. 3: D-FF Register Schematic Block Diagram

Therefore, the ALU architecture adopted a "select-add-store-feedback" structure as shown below:

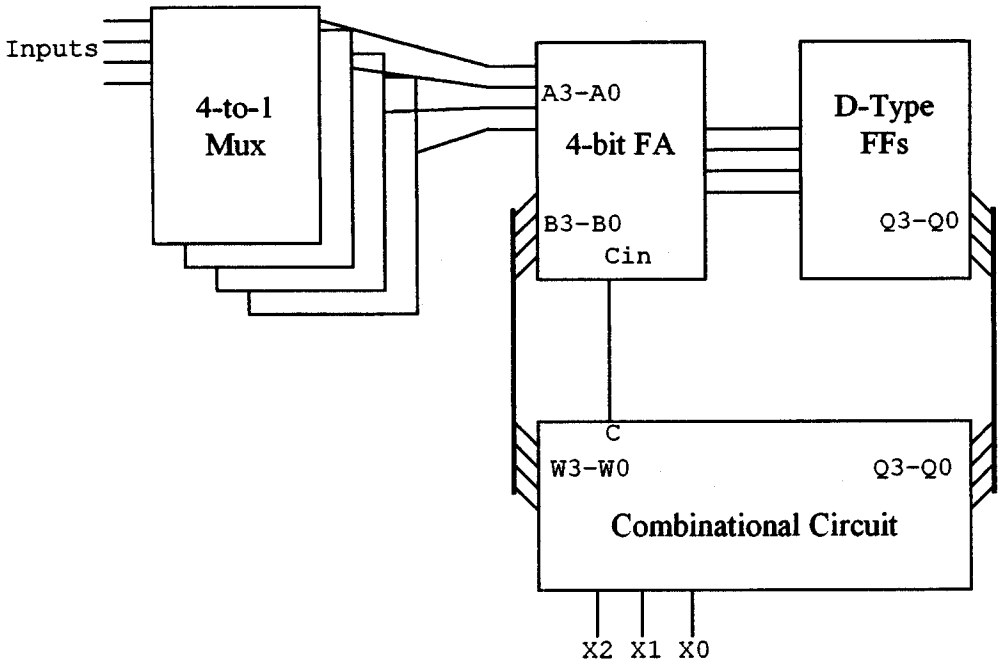


Fig. 4: 4-bit ALU Conceptual Block Diagram

Truth Table

Although the combinational circuit had seven inputs and five outputs, in essence it only had four inputs (Qn, X2, X1, X0) and 2 outputs (Wn, C). In addition, X1 and X0 was used for the MUX control signals, and the binary pattern had to be repeated twice for each logic level of Qn. Thus, the following truth table was obtained:

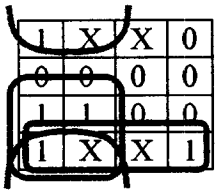
Table 5: Truth Table for Wn and C Outputs

Instruction	Qn	X2	X1	X0	Wn	C
CLRW	0	0	0	0	1	0
X	0	0	0	1	X	X
INCW	0	0	1	0	0	1
X	0	0	1	1	X	X
MOVL	0	1	0	0	0	0
SUBLW	0	1	0	1	0	1
ADDLW	0	1	1	0	0	0
ASHRW	0	1	1	1	0	0
CLRW	1	0	0	0	1	0
X	1	0	0	1	X	X
INCW	1	0	1	0	1	1
X	1	0	1	1	X	X
MOVL	1	1	0	0	1	0
SUBLW	1	1	0	1	1	1
ADDLW	1	1	1	0	0	0
ASHRW	1	1	1	1	0	0

Karnaugh Mapping

Two output columns on a truth table dictates that two Karnaugh maps have to be prepared—one for each output. When deciding as to loop the '0's or '1's, the choice that yields the least number of loops was chosen:

Wn



Wn = X2'X1' + QnX1' + QnX2'

However, as this stage it became apparent that Wn would take an unbearable number of logic gates to implement. Having to implement such an minterms for *four* outputs of W3-W0 (when equipped with only NAND and NOT gates) proved to be unfeasible.

Revision of Truth Table and Consequent Karnaugh Mapping

However, by ingeniously ensuring that if the logic '1's in the K-map could be rearranged into positions where the number of loops were minimized, *and* resulted in non-complemented minterms, gate-count could be dramatically reduced.

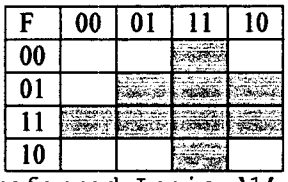


Fig. 5: Preferred Logic '1' Locations

The positions of the logic '1's were mainly dictated by the binary sequence of the MUX's control inputs X1 and X0. If instances where Wn had to be held at logic '1' were in the shaded areas as shown above, then the resulting Boolean expression would be tremendously simplified. Having optimized the positions of such '1's, the optimized values of X1 and X0 were examined noted and, as a direct consequence, the MUX's inputs D3-D1 was re-assigned. All these corrective measures resulted in a revised truth table:

Table 6: Revised Truth Table for Wn and C Outputs

Instruction	Qn	X2	X1	X0	Wn	C
ASHRW	0	0	0	0	0	0
MOVL	0	0	0	1	0	X
X	0	0	1	0	X	1
CLRW	0	0	1	1	0	X
X	0	1	0	0	X	0
ADDLW	0	1	0	1	0	1
SUBLW	0	1	1	0	0	0
INCW	0	1	1	1	0	0
ASHRW	1	0	0	0	0	0
MOVL	1	0	0	1	0	X
X	1	0	1	0	X	1
CLRW	1	0	1	1	0	X
X	1	1	0	0	X	0
ADDLW	1	1	0	1	0	1
SUBLW	1	1	1	0	0	0
INCW	1	1	1	1	0	0

The revised truth table (Table 6) resulted in a much elegant-looking Karnaugh map:

Wn

0	0	x	x
x	0	0	0
x	1	1	1
0	0	0	x

Wn = QW2
= (QW2)''

Cin was also re-arranged to suit this new assignment, and an equally elegant Karnaugh map was obtained:

Cin

0	0	0	x
x	0	1	1
x	0	1	1
0	0	0	x

Cin = X2X1
= (X2X1)''

Hardware Implementation & Simulation

By optimally assigning control signals and hence reducing the complexity of the minterms, I managed to use only a maximum of five NAND and nine NOT gates. When 3 NAND gates were wired as inverters, IC chip- count was reduced down from eight to seven. This allowed me the relative luxury of having more room to lay wires on the breadboard.

Simulation was conducted on LogicWorks™. Binary Switches was used to enter both the literal *l* word and the 3-bit control signals. LEDs served as an visual output indicator and proved useful during real-time simulation—the simulated LEDs turned on/off in accordance to the input instruction code, making it very intuitive when checking the correctness of the circuit's logic.

Next, after simulation has verified logic design, the actual implementation was carried out in various steps. First, power and ground were wired in and tested with a voltmeter—this not only ensures the correct voltage is being supplied, but also simplifies future trouble-shooting by virtually eliminating power-related problems. Secondly, with the aid of the circuit printout, each segment was wired in one-by-one and its connections were subsequently tested for continuity using an ohmmeter. Next, using the on-board logic switches, a test-run was then conducted to verify the functionality of each bit before moving on to the next bit.

This meticulous and methodical approach in wiring kept troubleshooting to a minimum. A logic probe was used to double-check the correct logic values were propagating through the system—starting from the Full Adder (FA). If the FA outputs were incorrect, there was usually a wiring error in the multiplexer ICs. If the LEDs were displaying erroneous outputs, then chances are there were mistakes between the D-type flop-flops and input B of the FA. Unstable LED output (the display changes each time push-button PB1 was depressed despite unchanged DIP switches input) usually denotes an error in the combinational circuit wiring. Typical mistakes were usually the wrong pins were tied, or an open-circuited wire.

The revised truth table (Table 6) resulted in a much

A final test-run was conducted upon completion. A sequence of instructions was implemented to evaluate the following math expression: (x+1)/2 – y + z (where x=2, y=4, and z=2).
The final results obtained (L1-4 = $-1_{10} = 1111_2$ after instruction ADDLW) was as expected:

Table 7: Test Inputs with Corresponding Outputs

Input	**Switch settings**		**LED Outputs**
Instruction Mnemonic	**External Input *I* S3 S2 S1 S0**	**Control Signals X2 X1 X0**	**Register *w* *w*3 *w*2 *w*1 *w*0 L3 L2 L1 L0**
CLRW	xxxx	011	0000
MOVL	0010	001	0010
INCW	xxxx	111	0011
ASHRW	xxxx	000	0001
SUBLW	0100	110	1101
ADDLW	0010	101	1111

Logic Devices Circuit Layout

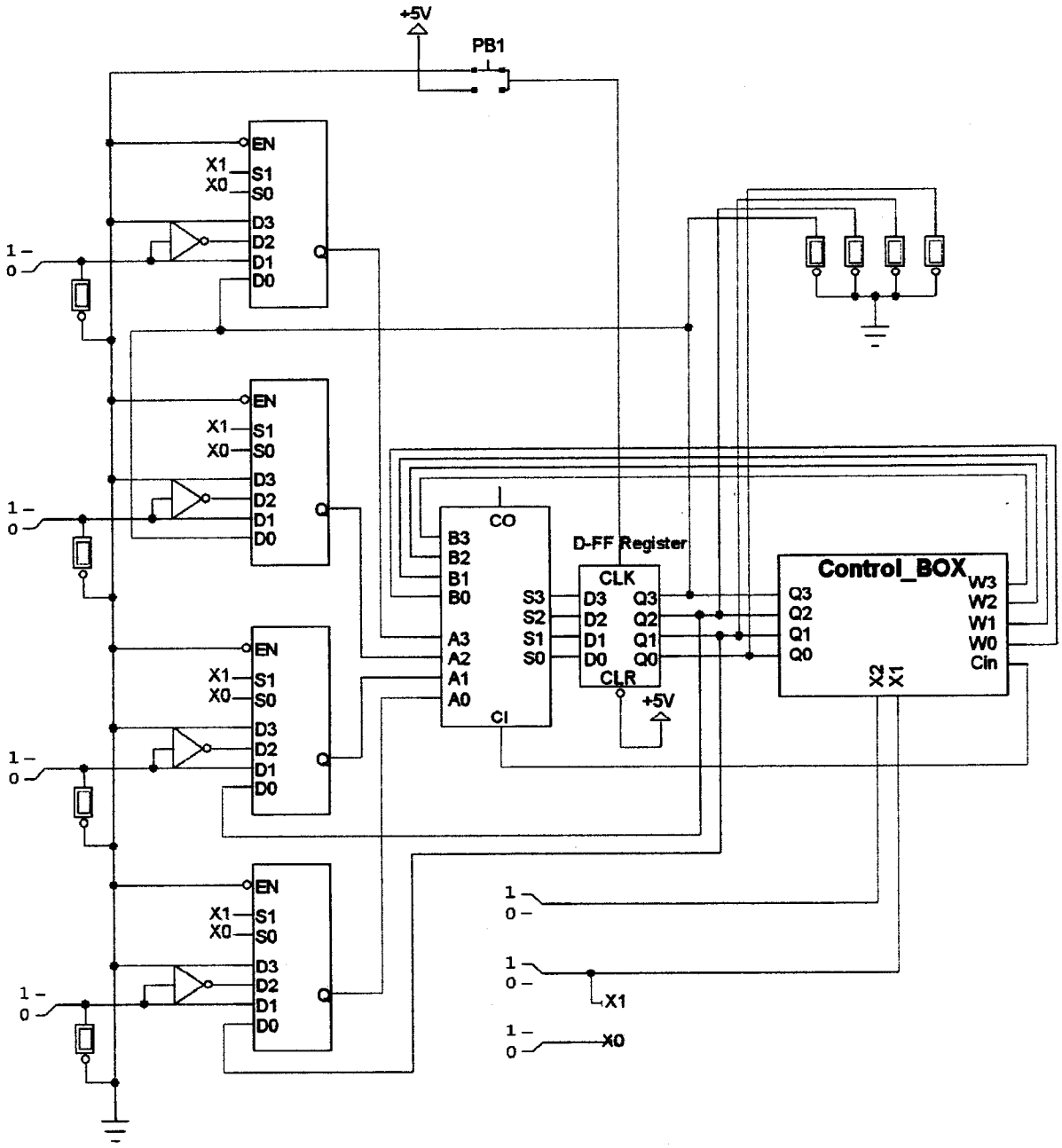


Fig. 6: Overall Schematic Diagram of 4-bit ALU

"Control_BOX" Combinational Circuit Layout (Gate ICs)

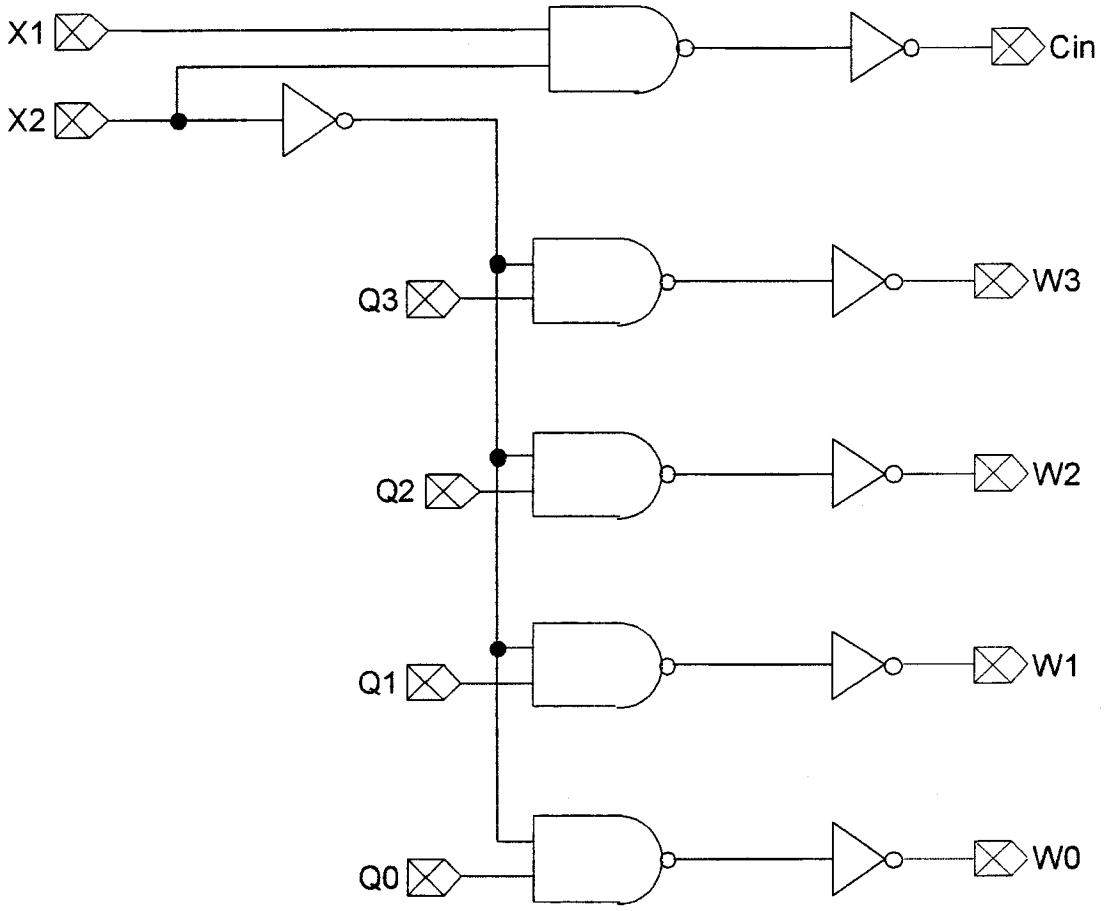


Fig. 7: Schematic Diagram of the "Control_BOX" Combinational Circuit

Combinational Circuit Layout (Actual Layout)

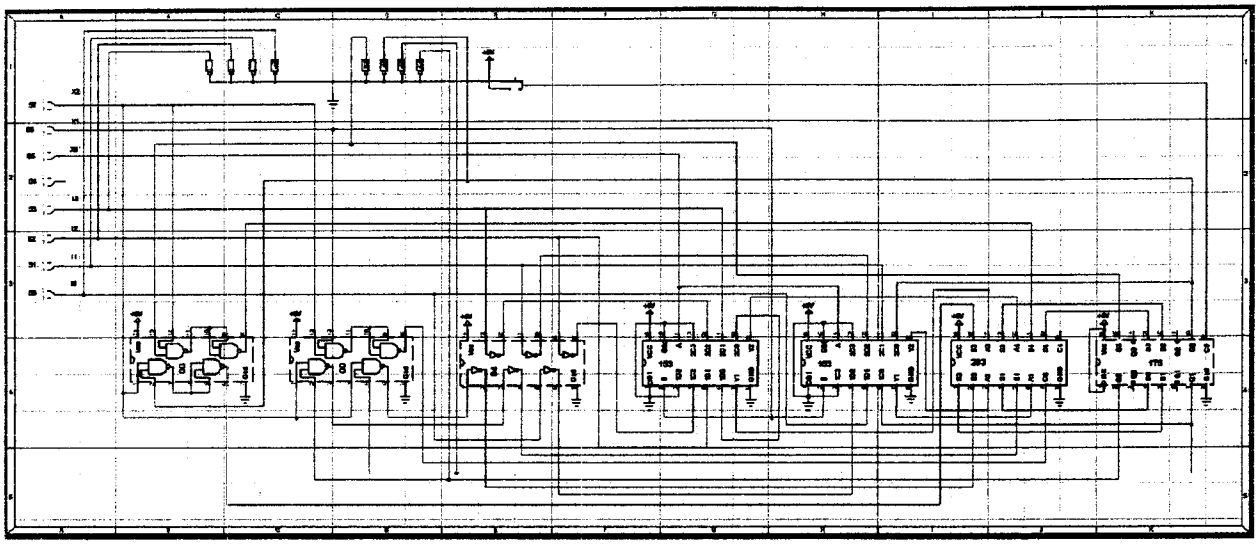


Fig. 8: Circuit Wiring Layout (Actual Pin Layout)

Timing Diagram

A timing file was also prepared and imported into the simulation. Its results confirmed the earlier real-time simulation results—circuit design is correct.

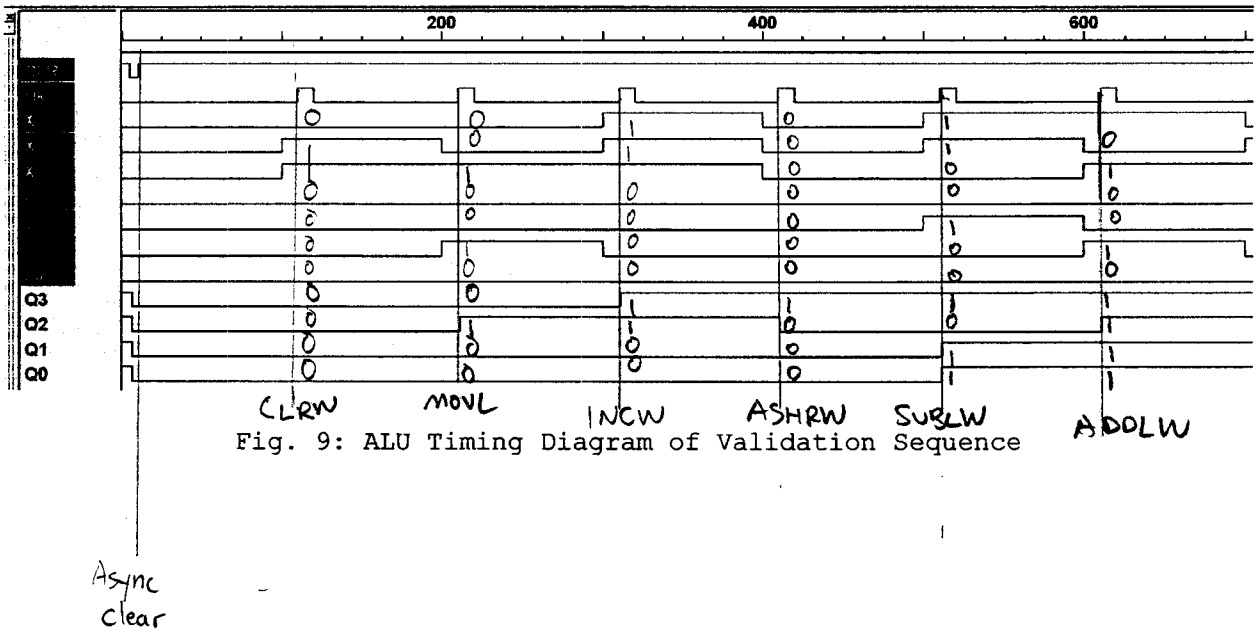


Fig. 9: ALU Timing Diagram of Validation Sequence

Timing Delay Calculation

The worse-case time-delay was simply the time needed for signals to propagate the "longest path" from input DIP switches to output LEDs. Assuming that the individual gate delays are as follows:

Table 8: Delay times of ICs

IC Type	Delay (ns)
7400	10
7404	10
74153	22
74175	15
75283	16

The "longest path" in the circuit design is for the instruction SUBLW, due to the fact that the L3-L0 signals have to propagate through a 7404 Inverter IC prior to 74153 MUX (path 1). Simultaneously, signals from the combination circuit are propagating through the system as well (path 2). All outputs of the "Control_BOX" combinational circuit have the same propagation delay—every signal passes though 2 logic gates.

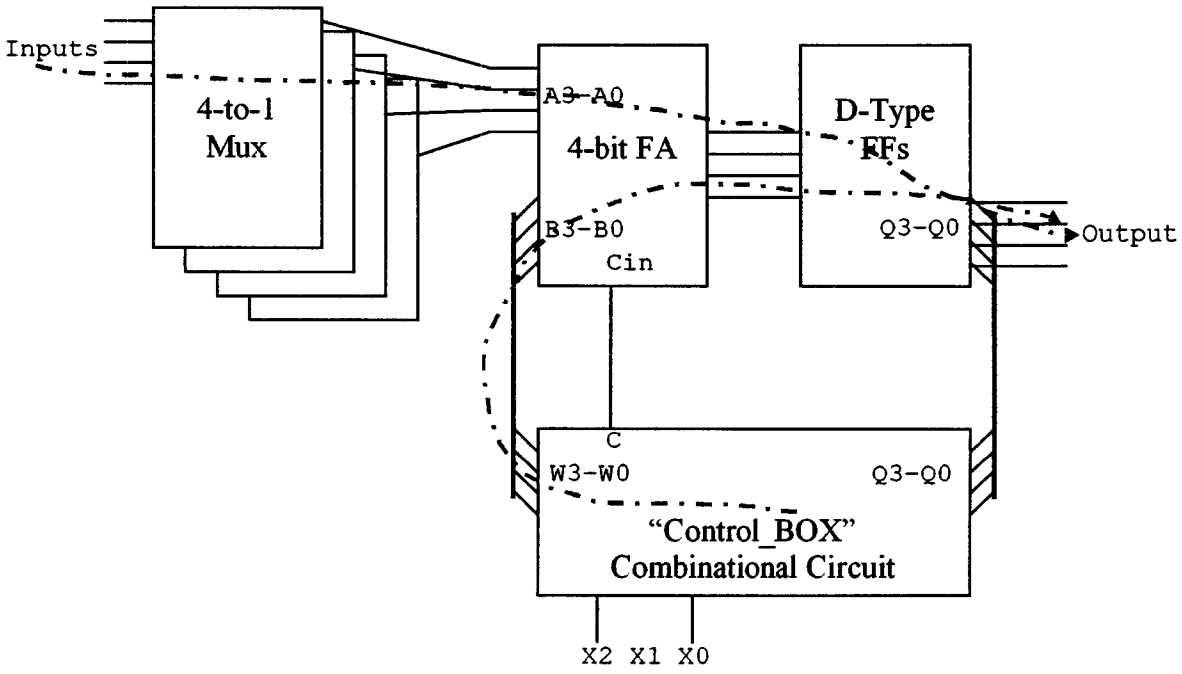


Fig. 10: ALU Timing Diagram of Validation Sequence

Path 1 = $Td_{7404} + Td_{74153} + Td_{74283} + Td_{74175}$
= (10 + 22 + 16 + 15) ns
= **63ns**

Path 2 = $Td_{7400} + Td_{7404} + Td_{74283} + Td_{74175}$
= (10 + 10 + 16 + 15) ns
= **51ns**

∴ The worse-case time-delay is **51 nanoseconds.**

IC and Wiring Layout

Chip layout on the Pencilbox™ logic designer is as shown below:

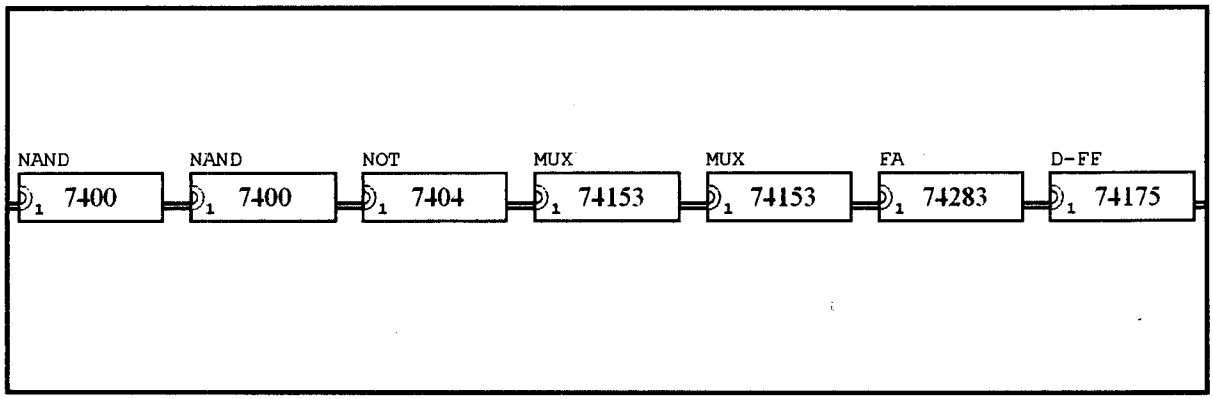


Fig. 11: Chip Layout on Breadboard

Circuit Breadboarding and Wiring

It was decided to place the ICs as similar as possible to the layout in Fig. 6. This is to minimize the amount of wire criss-crossing all over the breadboard. Care was also taken to ensure that ICs were connected to one another on the same side (for example, each input pin on the 74LS153 MUX was connected to a logic gate on the same side on the breadboard).

Color-coding was also utilized:

Table 9: Segment Wiring Color Key

Signal	Color
+5V	Red
Gnd	Black
Bit 3	Orange
Bit 2	Blue
Bit 1	Purple
Bit 0	White
Clock	Yellow
Input Pins tied to Ground	Green
Input Pins tied to High	Pink
Cin	Yellow

This is a photograph of the actual wiring on the Pencilbox™ logic designer:

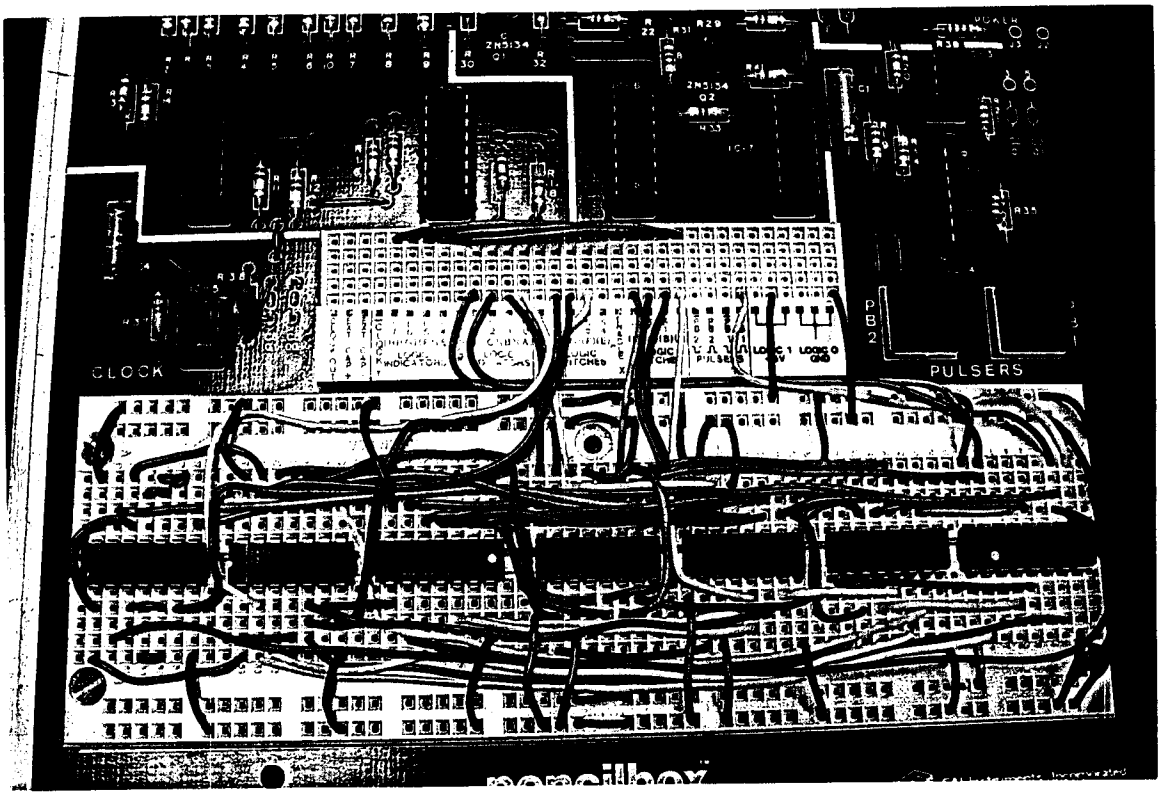


Fig. 12: Plan View Photograph

V. Nice wiring.

GTA Validation Results

The Pencilbox™ was validated on November 10th 2003. GTA Validation verified that the implemented logic circuit design was working correctly. The wiring was also inspected for neatness—a grade of "10" was awarded.

Table 10: GTA Validation Checklist

Instruction Mnemonic (Completed before validation)	Switch Settings (Completed before validation)		LED Outputs
	Ext. Input I I3 I2 I1 I0 S3 S2 S1 S0	Control Signals X2 X1 X0 S7 S6 S5	Register w w3 w2 w1 w0 L3 L2 L1 L0
CLRW	X	011	0000
MOVL	0010	001	0010
INCW	X	111	0011
ASHRW	X	000	0001
SUBW	0100	110	1101
ADDLW	0010	101	1111

Conclusion

Design Project 2 was successfully completed; a 4-bit ALU with six arithmetic/logical functions (as specified) was correctly implemented and simulated. The design goals were met as well, and this project was a good initiation in designing a fairly complex digital device. A problem was initially specified and a conceptual block diagram was developed in response, accompanied with a tentative truth table. Although the corresponding Karnaugh maps were drawn thereafter, the truth table still had to be revised in order to produce a more desirable and elegant K-map (which resulted in a minimal number of NOT gates). The obtained minterms were then simple enough to use a minimal number of gate ICs, which in turn lowered the chip count to only seven ICs. The circuit was first simulated on LogicWorks, before being implemented on the Pencilbox™.

This project—a true baptism-of-fire of digital design—also served to underscore how control codes may be optimally assigned to simplify Boolean logic equations. As detailed in the report, the reduction of just *one* NOT gate—as well as wiring NAND gates to implement NOT logic—reduced the IC chip-count to seven. This resulted in ample space to wire up the breadboard in a efficient yet neat fashion. In the fabrication world, board space is a prized commodity. If this were a commercial project, the elimination of just one IC chip in this design would considerably reduce PCB size and significantly reduce production cost.

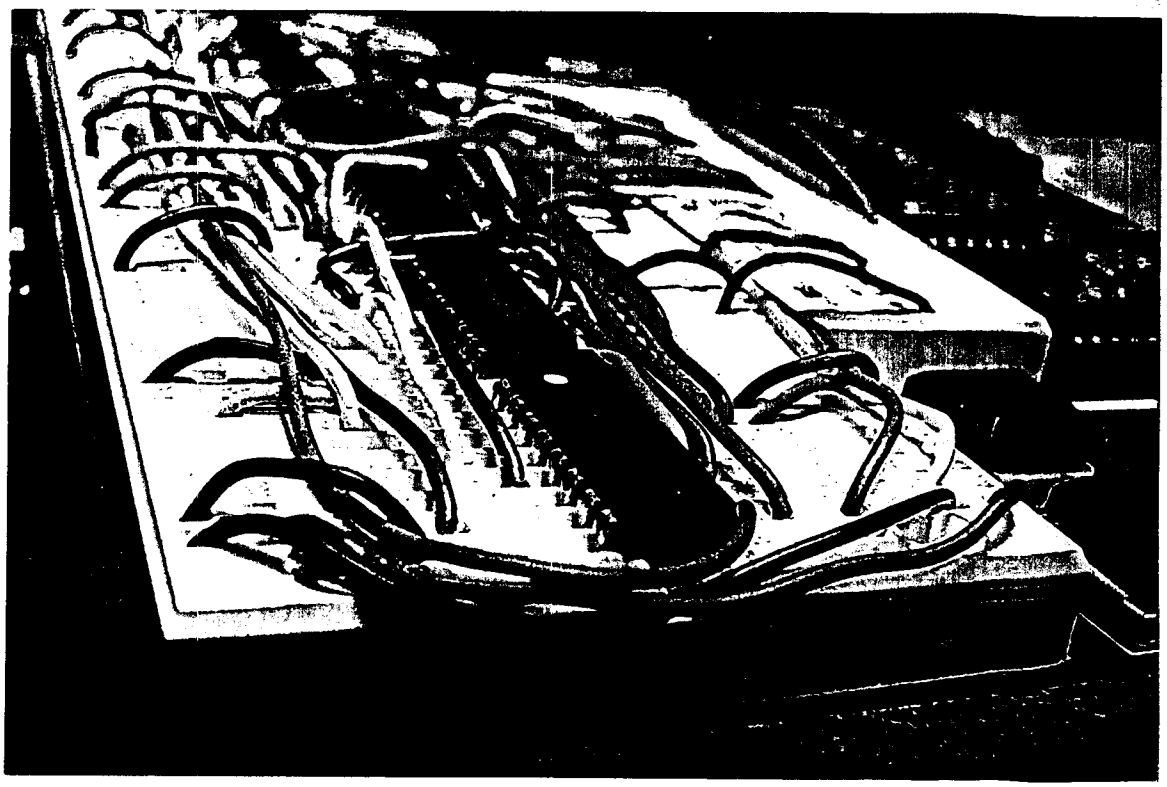


Fig. 13: Reduced Chip Count Brought About Neater Wiring