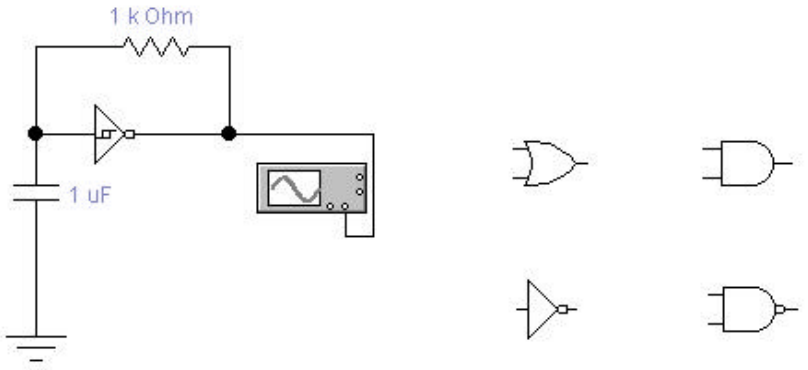
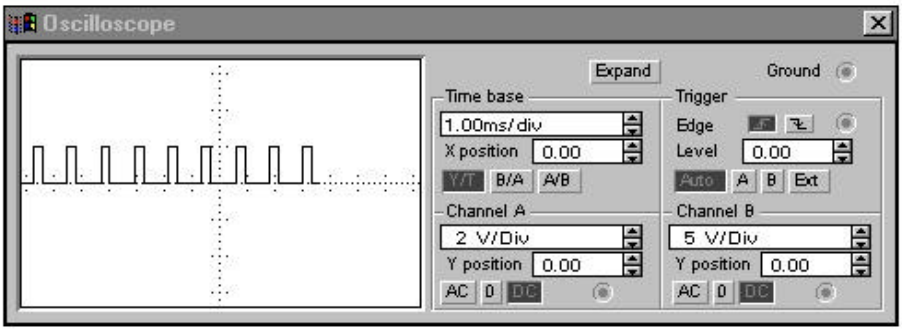
MARQUETTE UNIVERSITY

EECE 143

DIGITAL ELECTRONICS LABORATORY
CLASS NOTES





J. Christopher Perez

TABLE OF CONTENTS

INTRODUCTION TO DIGITAL DESIGN LABORATORY

COURSE EXPECTATIONS, GOALS AND DIGITAL DESIGN LABORATORY BASICS

There are several goals of EECE 143 Digital Electronics Laboratory. First, the student will gain experience in digital design and assembly. Secondly, The student will learn to test and troubleshoot digital circuits. The student will learn to design digital circuits using discrete integrated circuits in combinational as well as sequential circuits. The third goal of this course is use programmable logic devices as an alternative to discrete logic. The fourth goal is to learn how to use microprocessors for control and other applications. A fifth goal is to give the students experience in communicating their designs through presentations and technical report writings.

The laboratory is divided into four main sections. The first four laboratory experiments involve learning to use the equipment in the laboratory to troubleshoot discrete combinational and sequential circuits. Students will use the theory of combinational and sequential design that they learned in Digital Electronics (EECE112) and design, build and troubleshoot with actual components.

The second section of this course will allow students to design circuits using programmable logic devices as alternatives to discrete integrated circuits. The students will learn to write source code to program the PLDs and compile the code using a software application called CUPL. Students will learn to write code using Boolean equations, truth tables and state machines.

The third portion of the course involves the use of microprocessors. Students will learn to use the Motorola 68HC11EVB microcontroller. The 68HC11 is programmed using assembly language of the 6800 instruction set with extensions. The student will learn to program the 68HC11 in assembly language to perform several applications.

The final portion of Digital Design Laboratory is a design project in which students will be able to use the knowledge they have acquired throughout the course to design a circuit that will perform some function. The students are expected to present their design and complete a written technical report.

The remainder of this chapter explains the particular rules for setting up the laboratory notebooks. Chapter 2 contains a brief tutorial of the two main equipment used in digital laboratory, the Agilent 54622D Mixed Signal Oscilloscope and the CADET training board. The class notes go further to present discrete logic components and the different types of logic families. Chapter 3 discusses the various specifications and troubleshooting techniques. Chapter 4 contains information for inputting and outputting data for digital circuits, particularly push-buttons and DIP switches with pull-up resistors. A main portion of the chapter covers 7-segment displays as output devices. Chapter 5 presents various

designs of digital clock circuits including one-shots, crystal oscillators and other circuits. Chapter 6 covers counters, flip-flops, and shift registers. Programmable Logic Devices and techniques for programming them with CUPL are covered in Chapters 7 and 8. Chapter 9 is used as an additional reference for the Motorola 68HC11 microcontroller. More information may be found on the class website. http://www.eng.mu.edu/~perezjc/eece143/index.html

EECE 143 LABORATORY NOTEBOOKS FORMAT

Each students is required to maintain his/her own laboratory notebook. The notebooks should be Engineering & Science Notebooks available at the campus bookstore.

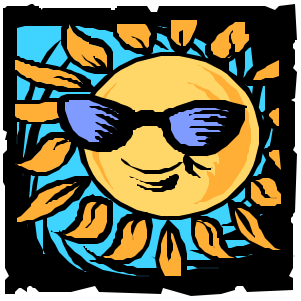
Notebooks will be inspected for pre-laboratory work at the beginning of the laboratory period.

The following image shows how each notebook should be labeled.

EECE 143
Digital Electronics
Laboratory Notebook

Student Name

Semester and Year
Lab Section 200X



The first page should have a table of contents that will be filled out as each student completes the lab.

The remaining pages will contain the Experiments section of the notebook. Students should number all pages at the start of the class.

Students should write on the back sides of the pages only when necessary for making corrections or attaching code.

Table of Contents
Experiment #, Title, Date, Page #s

Experiments section should follow the following format:
Title of Experiment
La**b Group Names**
Date Laboratory is performed
General Pre-Lab Questions And Problems.
Schematic Diagram Or Circuit Diagram With Parts List
Data Tables And Results should include Empty Columns For Measured Data
Most data tables are included in the Laboratory Manual. Students should complete the Theoretical or Expected Data columns. If no data table is provided, students should provide their own data table.
Troubleshooting Summary
If the students experience any problems while building and testing the circuits, they should log any troubleshooting they performed and how they solved their problems.

LABORATORY NOTEBOOK RULES

The following are some rules to be observed regarding the laboratory notebooks.

1. Use Engineering Notebook (National 33-610, MU bookstore)
2. Do not remove any pages.
3. Number all pages right away.
4. Use pen only NOT pencil. Blue or black only. Not red or green.
5. Use 2 cm minimum right margin.
6. Print neatly.
7. Cross out errors with a single line or X a whole section.
8. Tape any additions to an original page in the notebook.
   (PSpice plots, computer printouts, Data sheets, photocopies of other pertinent data, figures, ...)
9. All original hand-done work must be done on original notebook pages.
10. Label figures with "Figure 1-1", for Figure 1 in Experiment 1, ...
11. Label tables with "Table 1-1", ...
12. Put all data in tables.
13. Have TA initial data when taken after each experiment section.
14. Give notebook to TA or place in the drop-box by the designated time on the day before your lab period. Notebooks turned in should contain the completed post-lab work for previous lab as well as a new entry for this week's pre-lab
15. Notebooks might not be returned at the end of the semester.

SAMPLE PRE-LAB

The following is a sample of how Pre-Lab should be entered in notebooks. Data tables can be placed at the end of the Lab entry if you are using the pre-made data tables included in the Laboratory Manual.

Combinational Boolean Implementations

Experiment #1

Bruce Hoeppner & J. Christopher Perez

8 JAN 2002

Part 1:

Part 1A. AND-OR Implementation A = w'x + wx'

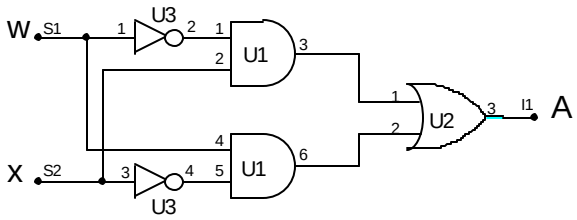


Figure 1-1A: Schematic for A = w'x + wx'

Function	Part#	IC#	Vcc	GND
AND	74HC08	U1	14	7
OR	74HC32	U2	14	7
NOT	74HC04	U3	14	7

Part 1A Data

		Theoretical	Measured
W	x	A	A
0	0	0	
0	1	1	
1	0	1	
1	1	0	

Part 1A Comments

(parts 1B, and C here!)

Part 1D Decoder*-NAND Implementation: D(w,x,y,z) = S (0,3,8,11)

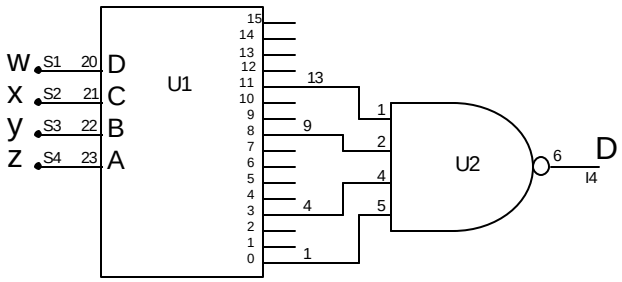


Figure 1-1D: Decoder*-NAND Implementation D(w,x,y,z) = S (0,3,8,11)

Function	Part#	IC#	Vcc	GND
Decoder	74HC154	U1	24	12
NAND	74HC20	U2	14	7

(remainder of Part D)

(rest of Experiment 1 Parts)

Troubleshooting Summary

1. Several ICs were not connected to power rail +5V. (Part1, 2)

2. Check for connections to IC enables. 74HC154's G1* and G2* were left off of pre-lab schematics. ICs don't work when not enabled.

CHAPTER 2 DIGITAL EQUIPMENT AND COMPONENTS

A BRIEF TUTORIAL ON EQUIPMENT AND COMPONENTS

AGILENT 54622D MIXED SIGNAL OSCILLOSCOPE TUTORIAL

The Agilent 54622D Mixed Signal Oscilloscope Tutorial is one of the main equipment to be used in the digital electronics laboratory. This tutorial is a brief introduction to the Oscilloscope and several of its features.

The 54622D MSO has 2 Analog Channels + 16 Digital Channels which can be used simultaneously. It has a100MHz Bandwidth with a sample Rate of 200MSa/s and a Memory Depth of 200MB/ch.

More information is available in the Agilent 54622D Mixed Signal Oscilloscope User's Guide which is available at the course website.

CONTROL LAYOUT

The control layout of the front panel is shown in Figure 2-1. The user interface consists of the display, disk drive, Softkeys, Intensity Control, Analog and Digital inputs and controls, Horizontal Controls, Run controls, Measure keys, Waveform keys, Trigger Controls and Utility key.

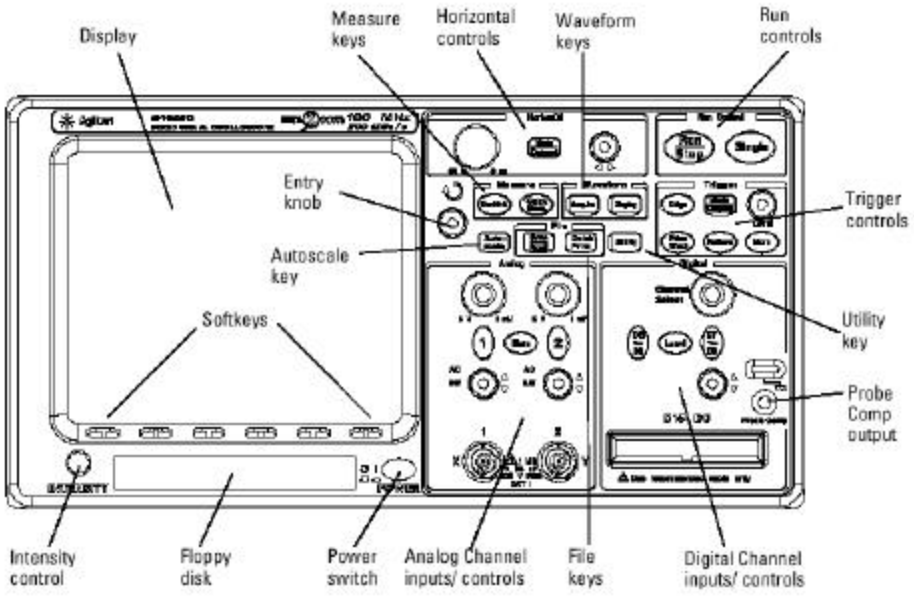


Figure 2-1 Agilent 54622D mixed Signal Oscilloscope Front Panel

1. **Display** – Shows the onscreen menus and waveforms
2. **Softkeys** – allow for different options that change depending on use
3. **Intensity Control** – allows user to modify the intensity of the display
4. **Floppy Disk Drive** - allows user to save data and screen dumps to diskette
5. **Power Switch** – push to turn on and off
6. **Horizontal Controls** – Use to change the horizontal (time) settings of the display. The left scroll dial is used for changing the scale, the smaller right scroll dial is used to shift the waveform left or right
7. **Run Controls** – Press Run/Stop to start and stop repetitive mode. Press Single to use single mode acquisition.
8. **Measure keys** – Press Cursors to use X Y markers to measure manually. Press Quickmeas to have the MSO perform quick calculations. These buttons bring up a soft menu for use with softkeys.
9. **Waveform** – Acquire and Display buttons bring up additional softmenus
10. **Trigger Controls** – Edge button sets the positive or negative edge trigger setting. The scroll dial is used to change the trigger level.
11. **Autoscale key** – Allows the MSO to automatically scale the display
12. **File keys** – Save/Recall and QuickPrint buttons allow you to change settings for saving to diskette
13. **Utility key** – displays softmenus for various utilities
14. **Analog Channel inputs/Controls** – used to change the vertical (voltage) settings of the two analog channels. The math key allows the user to perform several mathematical functions on the waveforms
15. **Digital Channel inputs/Controls**- used to change settings for the 16 digital channels

Refer to the Agilent 54600-Series Oscilloscopes User's Guide for a more in depth look at the controls of the 54622D MSO.

C.A.D.E.T. II TUTORIAL

The CADET II training board is one of the main equipment to be used in the digital electronics laboratory. The various features of the CADET include a +5V power supply, a 1.3V-15V variable power supply, a –1.3V- -15V variable power supply, 12.6 VAC power supply, 8 channel Logic Monitors (LED indicators), Logic Probe, BCD – 7-Segment LED Display, 8Ohm Speaker, two SPDT switches, 1KΩ potentiomenter, 10KΩ potentiometer, eight logic switches, one BNC connector, two debounced pushbutton switches, and a variable function generator.

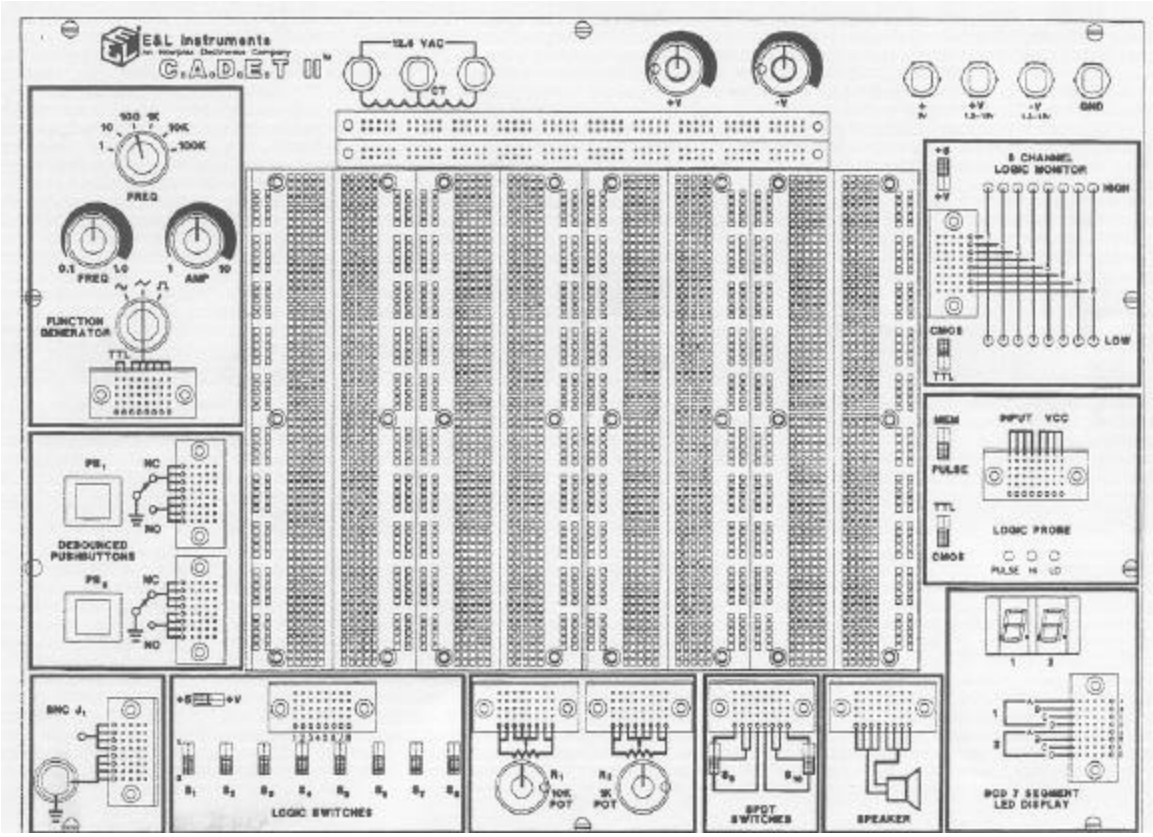


Figure 2-2CADET II Board

The power switch is located on the left rear side of the CADET board. The rocker switch lights up to show power is supplied to the board.

POWER SUPPLIES

The Power Supply Connections are located at the upper right corner of the CADET board. Connections can be made using banana plugs or by connecting wires from the banana jacks to the terminal strips located at the top center of the CADET board. The terminal strips are connected horizontally and can be used and power and ground rails for circuits

At the top of the C.A.D.E.T. board above the power supply connections are two voltage adjustment knobs. One is marked +V and controls the 1.3V-15V power supply. The other is marked –V and controls the -1.3V- -15V power supply. Most experiments will be using 5V so you will not have to make any adjustments.

NOTE: Always double check your power connections. Applying the wrong voltage may damage your components.

You may also connect your circuits using banana plug test leads with the banana jacks located at the upper right corner of the C.A.D.E.T. board. Each is labeled according to the power supply which it is connected.

LOGIC INDICATORS

The eight logic indicators on the right side of the C.A.D.E.T. board allow you a convenient way to test the logic level of certain signals in your circuits. The LED-Is consist of two horizontal rows of LEDs. The top row of red LEDs indicate a logic level of 1. The bottom row of green LEDs indicate a logic level of 0. To the left of the LED-Is is a block of connectors consisting of 2 columns of eight tie points. Each row of tie points is connected to a pair of red and green LED-I. The top row is connected to the first LED-I on the left.
On top of the LED-I connections is a toggle switch marked +5 and +V. The voltage supply for the LED-Is is changed by toggling this switch. Since most experiments use +5V, keep this switch at the +5 setting.

Note: Always check which voltage setting the LED-I switch is set, an incorrect setting may cause unwanted results.

Below the LED-I connections is another toggle switch marked TTL and CMOS. Due to different output characteristics of TTL and CMOS ICs, this toggle switch must be set accordingly to the type of chips being used.

LOGIC PROBE

Below the LED-I, connections to Logic Probe is found. Connect the Vcc to 5 Volts and connect a wire to the INPUT. You can use the wire to probe your circuit to determine if a signal is pulsing, logic level 0 or logic level 1.

BCD 7 SEGMENT LED DISPLAY

Below the Logic Probe is a BCD – 7 Segment LED Display. You can connect 2 4-bit BCD numbers and display them on the 7 segment displays.

8Ω SPEAKER

Below the LED-I, connections to an 8Ω Speaker is found. The connections to the speaker are found in a connection block to the left. The top two rows are tied together. Similarly the bottom two rows are internally connected. To use the speaker connect a top tie point to your source and a bottom tie point to ground (GND).

BNC CONNECTOR

The BNC connector is located on the lower left corner of the C.A.D.E.T. board. The center is already connected to ground. All rows of the connection block are tied to the BNC connector.

SPDT SWITCHES

On the bottom of the C.A.D.E.T. board can be found the two SPDT switches. A connection diagram on the C.A.D.E.T. board shows the three leads of each SPDT switch. When the switch is in the UP position the top and the middle leads are connected. When the switch is in the DOWN position the middle and the bottom leads are connected.

POTENTIOMETERS

There are two potentiometers 1KΩ and 10KΩ located at the bottom of the C.A.D.E.T. board. The center adjustable lead on each pot is connected to 4 tie points. The two fixed leads of each pot has 2 connections each. As you turn the knob in either direction, the resistance between the center lead and the one you are moving towards will decrease. The resistance between the center lead and the lead you are moving away from will increase.

LOGIC SWITCHES

Located at the bottom of the C.A.D.E.T. board are eight toggle switches. The voltage level of the logic switches are controlled by a separate toggle switch labeled +5 and +V. By setting to +5, the voltage level is set to 5V or TTL mode. By setting to +V the voltage level is set to +V/CMOS mode.

CAUTION: Always check the value of the +V voltage supply before setting to CMOS mode.

The logic switches will output a logic 0 or 0 volts when in the down position. They will output a logic level 1 or +5V when in TTL mode or +V when in CMOS mode.

DEBOUNCED PUSHBUTTON SWITCHES

The debounced switches are called “debounced” because they consist of the physical mechanical switch as well as additional circuitry to eliminate the multiple switch closure normally found when operating mechanical switches (called bouncing). Each switch has eight tie points of two different types. Four of the points are marked “NC” which means normally closed. These points are normally closed to ground and become open when the button is pressed. The other four points are marked “NO” which means normally open. These points are normally open and become connected to ground when the button is pushed.

FUNCTION GENERATOR

The function generator can produce sine, triangular, TTL and square waves from 0.1 Hz to 100kHz. The type of signal is changed by selecting sine, triangular or square wave on a slide switch. Connections are made through a set of connection points. The first column of connection points are TTL output only. The remaining three columns output the selected signal. The amplitude is adjustable (except when using the TTL function) by a vertical slide switch marked AMP. The frequency is varied by using a vertical slide switch marked FREQ. This sets the frequency between 0.1 and 1.0. The frequency can further be set by adjusting two other switches, a multiplier from Hz to kHz, and a multiplier from 1, 10, and 100.

NOTE: Always check power and ground connections when troubleshooting.

NOTE: Always turn off power when making connections to your circuits.

LOGIC FAMILY SUMMARY

Various families of Logic Integrated Circuits (ICs) exist on the market however the families mainly used in Digital Electronics Laboratory are the TTL and the High-Speed CMOS families. Some spec of the other families are included for comparison.

Transistor-Transistor Logic (TTL)

+ wide variety of functions and capabilities
+ good availability
+ low cost
+ easy to use
+ positive logic (theoretically simple)
+ high speed

- uses 5 Volt power supply
- consumes more power than other families
- typical active-low inputs and outputs

High-Speed CMOS (HC)

High Speed CMOS is not in the TTL family. However, it is designed to be functionally similar. Generally HC can be used in place of LS with a fanout restriction of 1.

+ very low power (HC μW vs TTL mW)
  one CMOS transistor of the pair is always off
  zero gate current
  no internal resistors
+ variable supply voltage
  74HC 2.0 to 6.0 Volts (use three, or four, 1.5 V batteries)
  74C 3.0 to 15.0 Volts
  CD4000 series 2.0 to 15.0 Volts
+ TTL replacements
+ high noise margin
+ can use pull-down or/and pull-up resistors
- handling
- speed

CMOS 4000 SERIES

CMOS stands for Complementary Metal Oxide Semiconductor. Gates are made with pairs of MOS transistors (one N-channel, one P-channel). Typically, one of the two transistors is "off". This accounts for extremely low power consumption. Another advantages of 4000 series CMOS is a high noise margin. CMOS gates have high input impedance. Fanout is limited more by capacitive rather than by DC loading.

EMITTER-COUPLED LOGIC (ECL)

ECL gates have lower propagation delays (higher speeds) than TTL. Gates are designed so transistors do not saturate when they turn on. Logic 1 (High) is -0.8 V. Logic 0 (Low) is -1.8 V. Typically the circuit is powered with Vcc = GND, and Vee = -5.2V. A modern ECL NOR gate is Motorola's M10KH100. ECL noise immunity (0.25 V) is lower than TTL, or CMOS.

Table 2-1 Logic Family Speed/Power Comparison

Device	7400	74LS00	74ALS00	74HC00	74C00	CD4011	M10KH100
Speed	10 ns	9.5 ns	5 ns	9 ns	50 ns	65 ns	1 ns
Power	10 mW	2 mW	1 mW	25 μW	10 nW	10 nW	25 mW

speed = tpd (typical) power = Vcc * Icc (per gate)

LOGIC IC NAMING

mm74xxxnnnrp

Mm	Manufacturer
74 or 54	Temperature Range
Xxx	Technology Type
Nnn	Logic Function
R	Revision
Pp	Package Type

Manufacturer -- mm

SN	Texas Instruments, Motorola
DM	National Semiconductor
None	Signetics

Pinouts will be the same for different manufacturers. Specifications may be slightly different.

Temperature Range -- 74 or 54

74	Standard (Commercial)	0 to 70 °C
54	Military	-55 to 125 °C

Pinouts may be different for the same function, and technology type, but different temperature range.

Technology Type -- xxx

TTL includes different types of integrated circuits with the same logic function. These differences are based on the type and size of transistors and diodes, and resistor values. These variations primarily affect the power and speed of the device. The following table summarizes speed and power using standard TTL as the base. High Speed CMOS is not in the TTL family. However, it is designed to be functionally similar. Generally HC can be used in place of LS with a fanout restriction of 1 LS device. Others: AC, ACT, BCT

Table 2-2 IC Technology Types

Type	Speed	Power	Name
	std.	std.	Standard
H	high	high	High Power
L	low	low	Low Power
LS	std.	low	Low Power Schottky
S	high	high	Schottky
ALS	high	low	Advanced Low Power Schottky
AS	v. high	std.	Advanced Schottky
F	v. high	high	Fast TTL
HC	std.	v. low	High Speed CMOS
HCT	std.	v. low	High Speed CMOS with TTL Inputs
C	low	v. v. low	CMOS -- TTL Pinouts

Logic Function -- nnn

Two to four digits identifies the logic function performed by the IC.

Table 2-3 Example TTL Parts

Part Number	Description
7400	Quad 2-Input NAND Gate
74LS00	Quad 2-Input NAND Gate
74LS01	Quad 2-Input NAND with Open-Collector Output
74LS32	Quad 2-Input OR Gate
74LS74A	Dual D-Type Positive-Edge-Triggered Flip-Flop with Preset and Clear
74LS138A	3:8 Decoder/Demultiplexer
74LS161A	4-Bit Synchronous Counter with Direct Clear
74LS636	8-Bit Parallel Error Detection and Correction Circuit with 3-State Output

Revision -- r

Improvements to an IC that correct slight errors or glitches have a letter suffix. The basic function of the circuit has not changed. The previous device becomes obsolete. Possible example: 74LS161 vs 74LS161A

Packaging -- pp

Table 2-4 Texas Instruments TTL Packaging

Pp	Type	Package Name	Comments
J	DIP	Ceramic Dual-In-line Package	14 to 20 pins, 0.3" centers
JW	DIP	Ceramic Dual-In-line Package	24 pins, 0.6" centers
JT	DIP	Ceramic Dual-In-Line Package	24 pins, 0.3" centers
N	DIP	Plastic Dual-In-Line Package	14 to 40 pins, 0.3" or 0.6"
W	FP	Ceramic Dual Flat Package	14 to 24 pins, surf. mount
D	SOP	Small Outline Package	0.244" wide
DW	SOP	Wide Small Outline Package	0.410" wide, 16 or more pins
FK	LCC	Leadless Chip Carrier	square, surface mount only

Other TI packages: JD, JG, P

CHAPTER 3 DISCRETE LOGIC COMPONENTS

INTEGRATED CIRCUIT COMPONENTS

CMOS OUTPUTS

The same logic function may be found in one of three output types.

Standard Outputs. standard CMOS output

+ direct connect of logic stages
+ fast

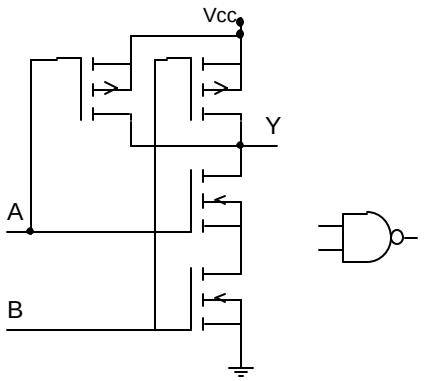


Figure 3-1High Speed CMOS NAND

Open Drain

Logic 0 outputs are the same as totem-pole.
Logic 1 outputs are "passive".
+ wired logic
+ change Logic 1 output voltage
+ drive large loads
- requires pull-up resistor
- slower than totem-pole

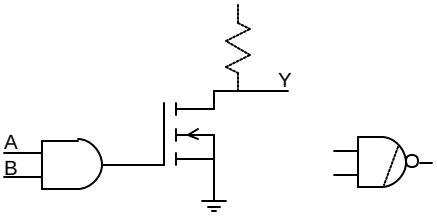


Figure 3-2 High Speed CMOS NAND

Three-State Outputs

Output can be logic 1, logic 0, or 3-stated. 3-stated is a high impedance state, open circuit, no connection.

+ multiplexing
+ disconnect an output from the circuit
- more complex, extra input needed

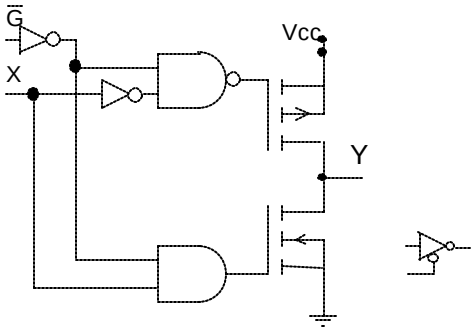


Figure 3-3 HC 3-State Inverter

DC ELECTRICAL SPECIFICATIONS

Data sheets give "worst case" values.
"Worst case" is the manufacturers guarantee of performance.
The worst case can be a minimum or maximum depending on which would be less desirable.

Typical values are sometimes given. These should be used for comparisons only. Worst case values should be used when designing circuits.

VOLTAGES

A complete understanding of Boolean Algebra, and Digital Theory, can be achieved with the ideal inputs of logic 1 = 5 V, and logic 0 = ground. Real circuits provide a range of input and output voltages to allow for loss and noise.

High = most positive voltage in a binary system

Low = most negative voltage in a binary system

Positive Logic assigns a logic 1 to the most positive voltage.

Input Voltages

Vil low level input voltage
maximum voltage guaranteed to be accepted as a logic 0 at an input (min)

Vih high level input voltage
minimum voltage guaranteed to be accepted as a logic 1 at an input (max)

Output Voltages

Vol low level output voltage
maximum output voltage with input conditions applied that, according to the product specifications, will establish a logic 0 at the output. (max)

Voh high level output voltage
minimum output voltage with input conditions applied that, according to the product specifications, will establish a logic 1 at the output. (min)

Table 3-1 Voltage Comparisons

V	74HC00A*			74HCT00**			74LS00		
spec	min	typ	max	min	typ	max	min	typ	max
Vih	3.15			2.0			2.0		
Vil			1.35			0.8			0.7
Voh	4.4			4.0			2.7	3.4	
Vol			0.1			0.4		0.3	0.5

* @ Vcc = 4.5V, Iout $\leq$ 20 uA** @ Vcc = 4.5V, Iout = 4.0 mA

NOISE MARGIN

Voltage ranges for inputs and outputs are a desirable characteristic in digital circuits. Assume device A outputs a logic 1 as 5.0 V, and device B recognizes 5.0 V as logic 1. If noise causes the 5.0 V to be reduced to 4.8 V, what logic level does B see? Having ranges for inputs and outputs allow for fluctuations without causing errors.

Noise margin is defines as the worst case difference between the low level input and output voltage, or between the high level input and output voltage. A large noise margin is desirable.

NM = smaller of: { [Voh - Vih], [Vil - Vol] } for a specific device.

For a 74HC00, NM = smaller of { 0.4 V, 0.4V } == 1.25V

For a 74LS00, NM = 0.2V

LOGIC GATE CURRENTS

Ideal devices require zero current to operate, yet, can supply infinite current to a load. Real devices deviate from the ideal. TTL devices are made with NPN transistors ($I_c = \beta I_b$). TTL inputs will have some required input drive current. Also, a real device cannot supply infinite current. There are some limits. IC currents are always specified as being into the device. Negative signs indicate current is leaving the device.

Input Currents

Iil low-level input sink current
maximum current into an input when a low-level voltage is applied to that input.

Iih high-level input drive current
maximum current into an input when a high-level voltage is applied to that input.

Output Currents

Iol low-level output sink current
maximum (manufacturer guaranteed) current into an output when input conditions indicate the output should be low (logic 0).

Ioh high-level output source current
maximum current into an output when input conditions indicate the output should be high (logic 1).

Note: Although specified as a maximum, a specific devices may sink or source more current and still be within the correct voltage range. (i.e. Iout = 1 mA, for Vout = 3.0 V)

Table 3-2 Current Comparisons

MA	MC74HC00A*			74HCT00A*			74LS00		
Spec	min	typ	max	min	typ	max	min	typ	max
Iih			.0001			.0001			0.02
Iil			-.0001			-.0001			-0.4
Ioh			-4.0*			-4.0*			-0.4
Iol			4.0*			4.0*			8.0

* @Vcc = 4.5 V, Voh @ 4.0 V, Vol @ 0.26 V, Temp < 25 °C
Actual maximum output current is determined by desired output voltage.

OTHER DIGITAL IC SPECIFICATIONS

Propagation Delay

In idea logic devices, an input change, results in an immediate output change. In real devices, the output change is delayed. This delay is called propagation delay (tpd). Propagation delay is due to transistor switching, and circuit capacitance. tpd is a measure of the speed of a device. It is measured as two different values: propagation delay, high-to-low output, (tphl), and propagation delay, low-to-high output, (tplh). Measured with respect to the output irregardless of device function. Other propagation delay specifications related to switching from a logic value to or from a 3-state condition.

Other timing specs: rise time (tr), setup time (tsu), hold time (th), and minimum clock pulse width (tpw or tw).

Maximum Clock Frequency fMAX

Flip-flops and other real clocked devices has a maximum clock frequency. fMAX is the highest clock speed at which the manufacturer guarantees the device will operate correctly. Note: Like many specs, an actual device may work at higher frequencies in a prototype, but, you should not exceed specs when designing for mass production.

Higher CMOS, and HC clock frequency increases power consumption.

POWER REQUIREMENTS

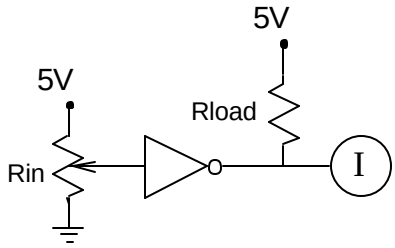
Real ICs consume energy to operate. This energy is not used for external useful work. It is wasted (?) as heat. Typically, we would like this to be as small as possible. Power requirements vary the most between logic families. IC power consumption is measured as Icc * Vcc with outputs open. A 5 W power source can supply ≈100 74LS ICs at 50 mW each. Allowing for a margin of error, a limit of 20 is reasonable. Note: Each LED in a circuit requires about the same amount of current as an IC. Decrease the number of ICs by one for each LED.

Vcc Ranges

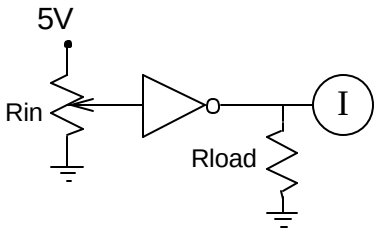
TTL ICs are typically powered from a 5.00 V, +/- 5% source. The specified range of operation for a 74LS00 is 4.75 V to 5.25 V. Correct operation is not guaranteed outside of this range. HC devices can operate with a large range of supply voltages. Typiclly: 2.0 V < Vcc < 6.0 V. Two, 3, or 4 AA batteries can supply several ICs.

Localized, momentary brown-outs can occur on circuits during high-speed operation, or high-current switching. Decoupling capacitors help to relieve this problem. One 10 - 100 μF capacitor per board, and one 0.1 μF per IC is sufficient.

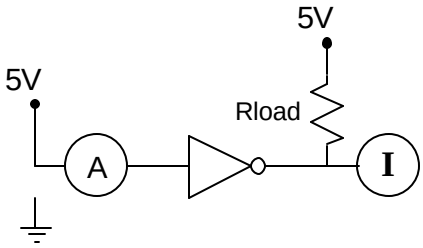
DC SPEC TEST CIRCUITS



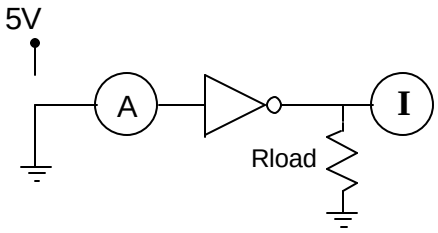
Vih Test Circuit



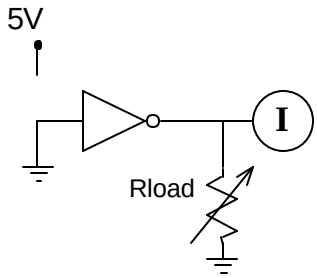
Vil Test Circuit



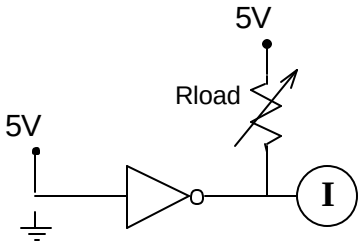
Iih Test Circuit



Iil Test Circuit

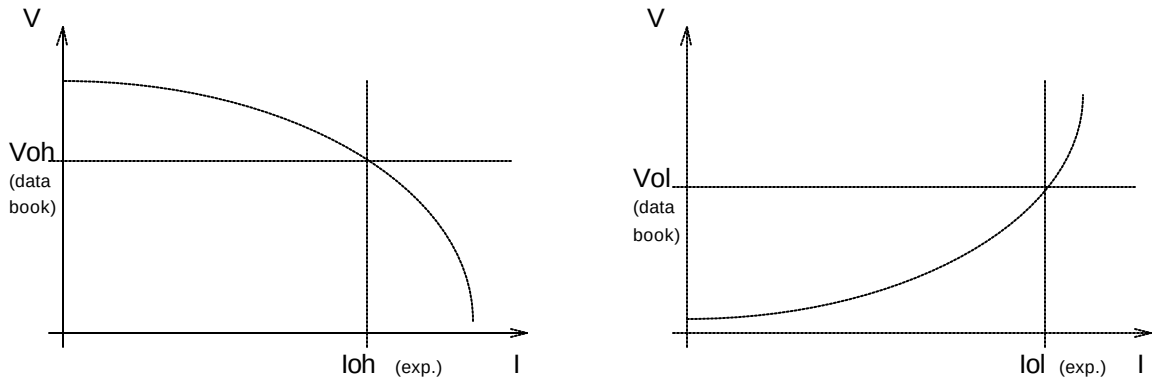


Voh, Ioh Test Circuit



Vol, Iol Test Circuit

Figure 3-4 Digital IC Test Circuits



Determine Experimental Ioh Determine Experimental Iol

Figure 3-5Testing Curve Waveforms

BREADBOARDING DIGITAL CIRCUITS

SUGGESTIONS FOR SUCCESSFUL BREADBOARDING

1. Use a separate node rail for power and ground. Use the top line for +5 V, and the bottom for ground.
2. Anytime an IC is inserted, **immediately** connect it to power and ground rails.
3. Insert ICs an logical order. Insert all ICs in with the same orientation. Good placement reduces wiring complexity.
4. Cut wires to smallest usable size. Keep your wires short.
5. Change wire colors often. Try to use Red for +5 V, and Black for ground.
6. Highlight or somehow record on the schematics as connections are made.
7. Avoid alligator clips.
8. Label inputs and outputs using tape.
9. Connect discrete components directly to IC/nodes.
10. Do not twist resistor or capacitor leads to get exact values. Use a node on the breadboard. (Typically, digital circuits do not need exact values.)
11. Break each circuit into simple blocks. A block consists of a single IC or logic level. Assemble and test one block at a time.
12. Have one partner wire the circuit while the other records data. If something goes wrong, switch places. Sometimes it is easier to find someone else's mistake.
13. Alternate assembler and recorder positions after each experiment section so both partners get experience.
14. Split up very large circuits so that each partner assembles a section.
15. Keep power and clock lines short. Especially between boards.
16. Use decoupling capacitors in large circuits, or at high clock rates.

LOGIC PROBES

If access to neither a logic probe nor a CADET board with Logic Indicators are available, the student may want to build a logic probe using the following design.

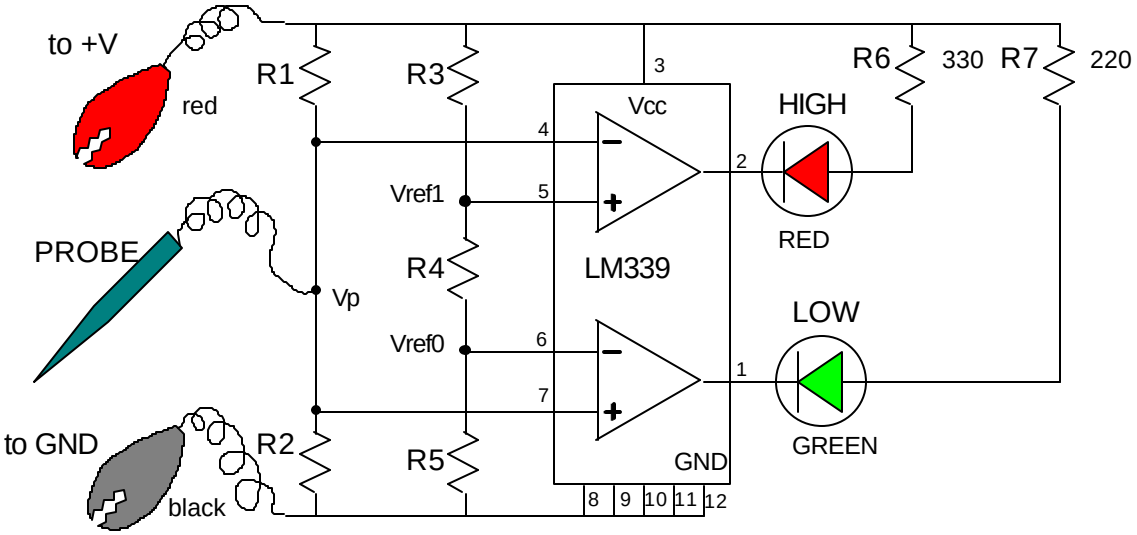


Figure ***Error! No text of specified style in document.****-6Simple Logic Probe*

Operation

Comparator (LM339)
When +V > -V ==> Vout = Vcc
When +V < -V ==> Vout = GND

Resistor Selection

Design R1, R2 to bias probe between Vil and Vih when not connected.
Design R3, R4, R5 to provide reference voltages to comparator so that pin 5 is at Vih and pin 6 is at Vil.
Design R6, R7 to maximum LED brightness for given comparator.

DIGITAL TROUBLESHOOTING CHECK LIST

Use this check list to find the problem with your digital circuit.

DEFINITIONS

Block Diagram: A diagram in which the essential units of any system are drawn in the form of blocks. Their relationships to each other are indicated by appropriately connected lines. The path of the signal may be indicated by arrows.

Schematic: A complete circuit diagram including component identification, wiring connections, and approximate component locations.

Identification:

ICs: part number, chip number, pin descriptions, pin numbers.

Resistors: value, tolerance, power rating if needed.

Capacitors: value, tolerance, max voltage.

Other: as required.

When a problem occurs with your circuit, try these steps one at a time.

POWER ON

1. Check power supply with DMM or Oscilloscope.
   Measure DC value and ripple on Vcc and ground.

2. Check power and ground to each IC. Use DMM, scope, or logic probe.

3. Check all constants. Constants include: Chip enables, Presets, and Clears, and any other IC inputs that do not change during the operation of the circuit.

4. Check clock. Measure magnitude and frequency. Check with scope to detect any defects.

5. Make sure all IC inputs are connected somewhere. See #3. All unused inputs should be connected to power or ground (use pull-up resistors if needed). Make sure unused IC outputs are left open-circuit. (Inputs of unused gates of an IC do not need to be connected.)

POWER OFF

6. Check schematic with actual circuit. Compare the schematic component by component with the circuit. Are correct pinouts used? Not all digital ICs have Vcc, and GND at opposite corners!

7. Check continuity between components. Use an Ohmmeter to measure the resistance from pin to pin.

R = 0 (0 - 0.9 ohms) is OK
R = infinity (>1 ohm) is bad

Measure resistance with the DMM set at the lowest ohm range. Do not use on "AUTO". Put the ohmmeter probes directly on the IC pins, do not push the probes into breadboards. If you have a DMM with audio continuity, use it!

8. Check components one at a time. Remove component from circuit.
Start with the "warmest" components first, then check:

Resistors: Measure value.
Capacitors: Measure value and check polarity.
Diodes: Is it in backward? Check with curve tracer or DMM.
Transistors: Check with curve tracer or DMM.
ICs: Check with appropriate IC tester, or try it in a "verified" circuit, or build a simple test circuit.

If a component is bad, check component specs with circuit specs. Was the component bad to start with, or was it damaged in the circuit?

DO NOT REPLACE ASSUMED BAD COMPONENTS UNTIL VERIFIED BAD!

DO NOT REPLACE BAD COMPONENTS UNTIL CIRCUIT IS DETERMINED TO NOT HAVE CAUSED THE DAMAGE!

9. Check schematics with block diagram. Verify pinouts with a data book. Check for correct bottom or top view of pinouts. Verify that specific ICs perform the desired block function.

10. Check design process. Verify design and concepts. Question and review all assumptions.

POWER ON

11. Check the functional blocks. Isolate the circuit into functional blocks as described by block model. Connect the block's inputs to logic switches, and the block's outputs to LED indicators. Perform a truth table verification of the block. Repeat for all blocks in the design.

CHAPTER 4 INPUT AND OUTPUT

INPUT AND DISPLAY DESIGN FOR DIGITAL CIRCUITS

SWITCH INTERFACING: SPST, SPDT WITH PULL-UP RESISTORS

Interfacing an SPST to TTL Inputs

Use a pull-up resistor configuration.

0) **Logic 0**: Close Switch

Vx is directly grounded.
∴ **Logic 0**

1) **Logic 1:** Open Switch

To be a logic 1 at the input Vx ≥ Vih.

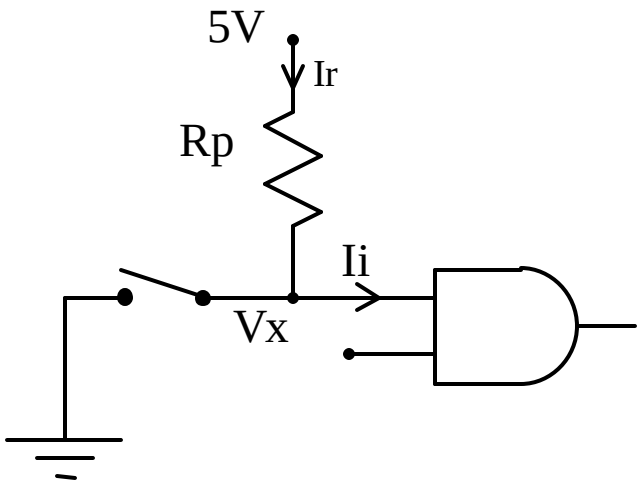


Figure 4-1 Switch with Pull-up Resistor

∴ assume	Vx	≥	Vih	
	5.0 V - VR	≥	Vih	KVL: Vx = 5 V - VR
	5.0 V - Ir*Rp	≥	Vih	Ohm's Law: VR = Ir * Rp
	5.0 V - Iih*Rp	≥	Vih	Worst Case: Ir = Iih, (Ir ≤ Iih)
	5 V - 40 μA*Rp	≥	2.0 V	Iih = 40 μA, Vih = 2.0 V
	5.0 V	≥	2.0 V + 40 μA*Rp	
	5.0 V - 2.0V	≥	40 μA*Rp	
	3.0 V	≥	40 μA*Rp	
	3.0 V ÷ 40 μA	≥	Rp	
	75,000Ω	≥	Rp	
∴	Rp	≤	75 kΩ	

Rp ≤ 75 kΩ is the maximum value. Can Rp be 0Ω? ... 100 Ω.
↑ Rp ↓ power wasted in Rp when the switch is closed.
↓ Rp ↑ the noise margin for logic 1.
Example: Select Rp = 10 kΩ Vx = 5.0 V - (40 μA * 10 kΩ) = 4.6 V

SPDT Switches
SPDT Switches and buttons are easier to use as logic inputs. Bouncing may also be a problem.

SEVEN-SEGMENT DISPLAYS

Seven-segment displays are used for decimal numeric displays. Seven light-emitting diodes are arranged so that all ten digits (0 - 9) can roughly be displayed.

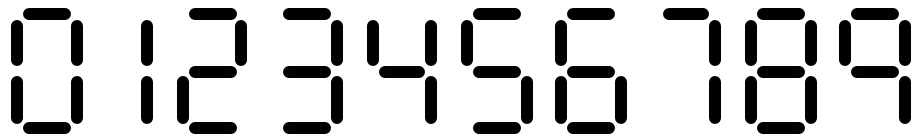
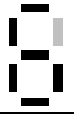


Table 4-1 7-Segment Table

Decimal	4-bit binary w x y z		seven-segment a b c d e f g
0	0 0 0 0		1 1 1 1 1 1 0
1	0 0 0 1		0 1 1 0 0 0 0
2	0 0 1 0		1 1 0 1 1 0 1
3	0 0 1 1		1 1 1 1 0 0 1
4	0 1 0 0		0 1 1 0 0 1 1
5	0 1 0 1		1 0 1 1 0 1 1
6	0 1 1 0		1 0 1 1 1 1 1
7	0 1 1 1		1 1 1 0 0 0 0
8	1 0 0 0		1 1 1 1 1 1 1
9	1 0 0 1		1 1 1 0 0 1 1

Single digit 7-segment displays come in two main IC packages

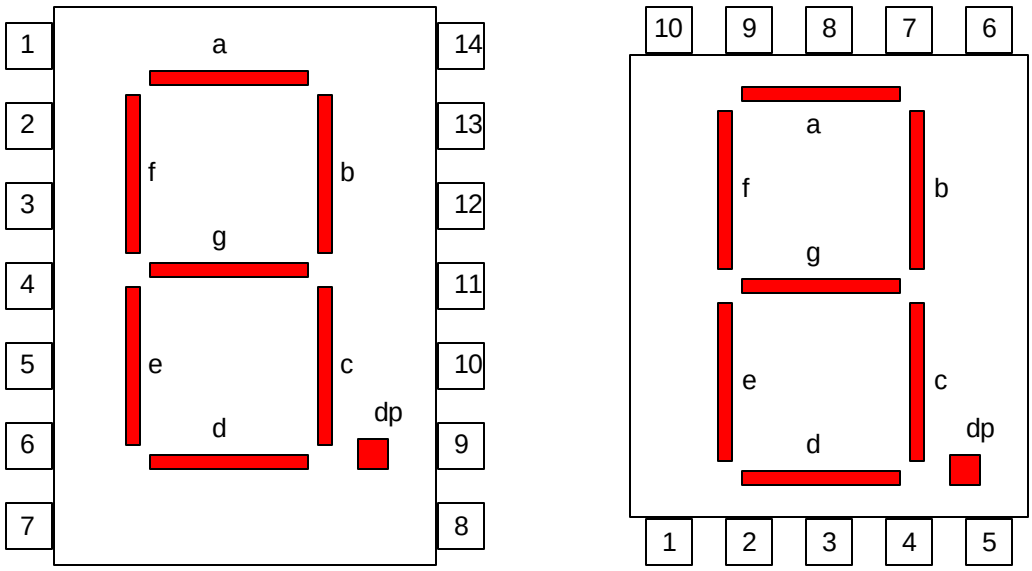


Figure 4-2 7-Segment Display Packages

Table 4-2 7-Segment Pinouts (typical)

Part#		FND507
Segment	14-pin	10-pin
ca/cc	14	3,8
A	1	7
B	13	6
C	10	4
D	8	2
E	7	1
F	2	9
G	11	10
Dp	9	5

Common Cathode / Common Anode
To save pins on 7-seg packages, all LEDs typically share one pin for a common anode, or common cathode.

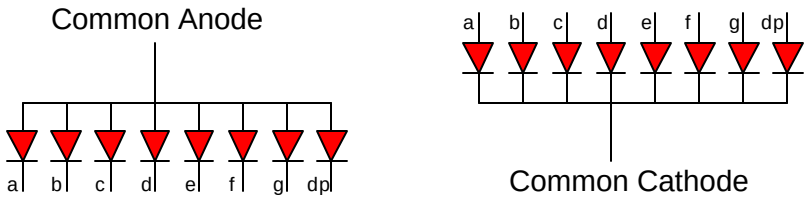


Figure 4-3 LED Configuration in 7-Segment Display Packages

Current Limiting Resistors

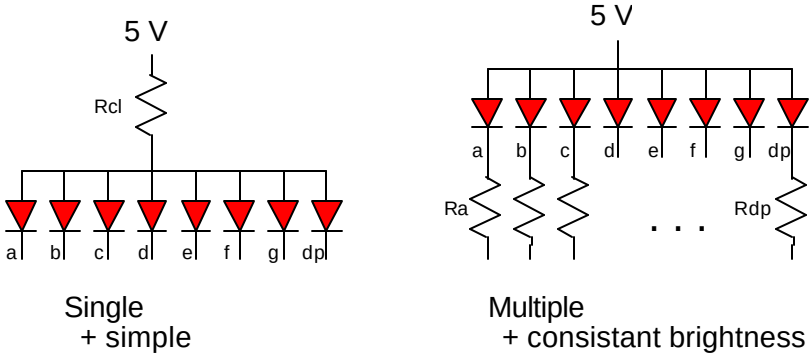


Figure 4-4 Using Current Limiting Resistors

Decimal Points
Seven-segment displays come with both right handed (RHDP), and left handed (LHDP) decimal points.

Light a **Current**
The amount of light emitted by an LED varies with current through the device. The amount of ambient light determines the required LED intensity. 7-seg. displays typically need 1 - 10 mA. Some are destroyed at 40 mA. More information is available in data books and handouts.

Overflow Digits
Overflow digits are sometimes used as the msd of a display. They can display a "+" or "-" sign, and a blank or "1". A 3 1/2 digit display would have 3 normal 7-segs. plus one overflow digit.

Multi-Segment Displays

Seven-segment displays are available in multiple segment packages. Two, or four digits are packaged together. Sometimes segment drive lines are multiplexed.

Liquid Crystal Displays (LCD)
LCDs are very similar to LED 7-segment displays. However, they do not emit light. Segments can be seen in areas where light is not reflected. LCDs are voltage controlled devices. They consume very little power.

74LS47 BCD-TO-7-SEGMENT DECODER

74LS47 BCD-to-7-Segment Decoder / Driver
Converts BCD data (digits 0 - 9, (4-bits)) into a 7-bit code used to drive 7-segment LED displays. Illegal BCD inputs are translated into indiscriminate, yet distinct display patterns. Data inputs to the code converter are: D, C, B, and A (lsb). Data outputs are: a, b, c, d, e, f, g; corresponding to the standard segment names of a 7-seg. display. Outputs are active-low, open-collector. The IC is designed to interface to common anode 7-segment displays. The 74LS47 has control inputs to test the segments, and blank leading zeros. Decimal points are not affected by the 74LS47.

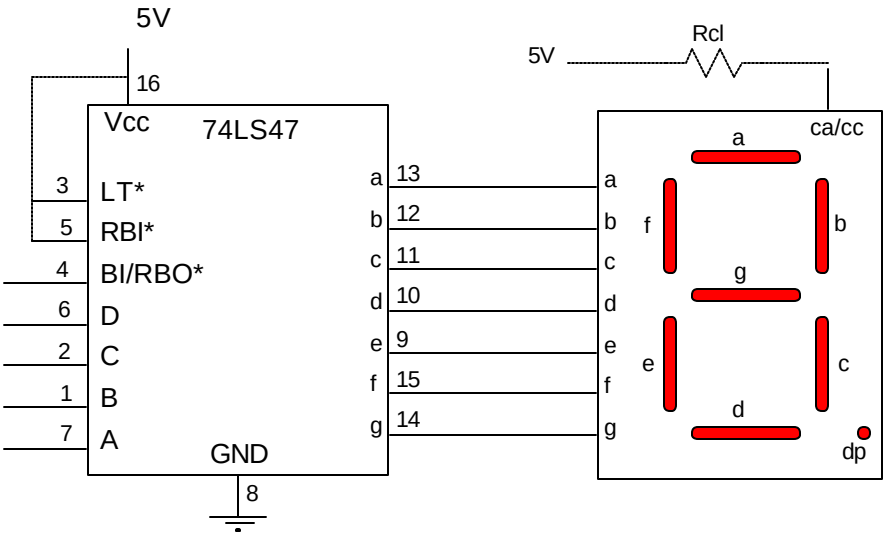


Figure 4-5 Using a 74LS47 Display Driver

Blanking Most Significant Digits

Multi-digit displays can be made where leading zeros are blanked. RBO* and RBI* control signals are daisy-chained to selectively blank digits. Any digit(s) that should always be displayed (with at least "0") should have the 74LS47 driver's RBI* set to 1. Any digit(s) that should be blanked when data is zero should have the driver's RBI* set to 0. Additional digits can be blanked when data is zero, and previous data is blanked (data is zero) by chaining the previous 74LS47 driver's RBO* to the present driver's RBI*.

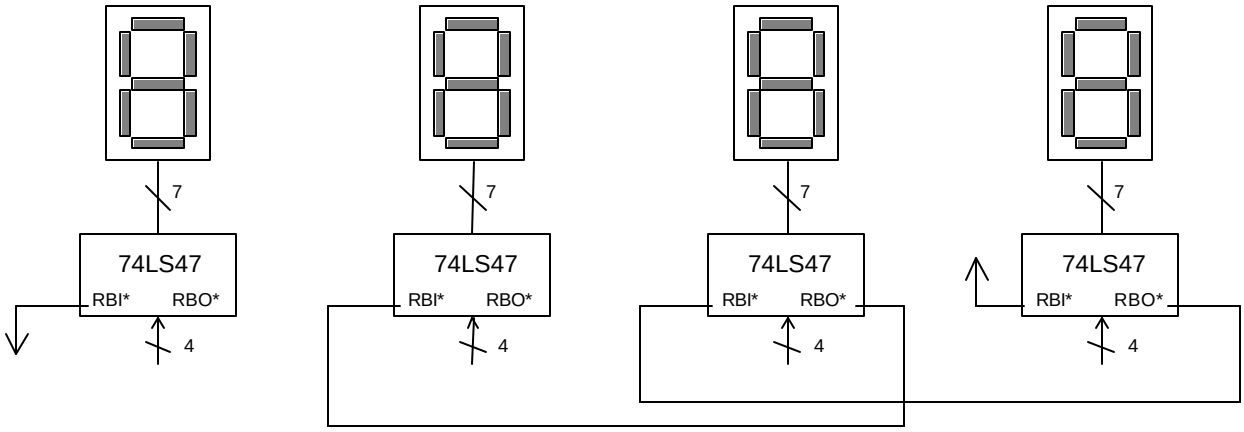
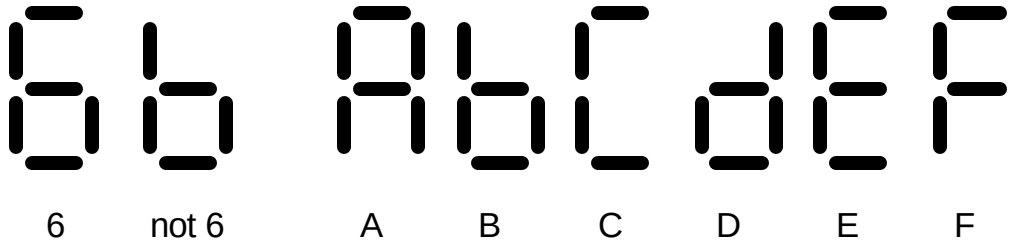


Figure 4-6 Using Multiple Displays

HEX to 7-Segment Displays

To display HEX digits on a 7-seg. display, characters A, B, C, D, E, F, must be displayed in addition to digits 0 - 9. This can be done using a mix of upper and lower case, and adjusting 6.



HEX7SEG.PLD

HEX7SEG.PLD is a source file to program a PAL16L8 programmable logic device that incorporates the functions of the 74LS47 except that it displays HEX. The source code will be available.

MULTIPLEXED DISPLAYS

Multi-digit displays require a large number of signals (≈7 signals per digit). Multiplexing LED drive lines is one way of reducing the signal count. This method can reduce the number of drive signals to 7 + N. Where N is the number of digits.

The procedure sends 7 drive signals (a - g) to each of N digits' 7 cathodes (a - g). The common anodes of each N digits are driven one at a time while the correct data is presented to the cathodes. Precise timing is needed. The total number of signals is reduced.

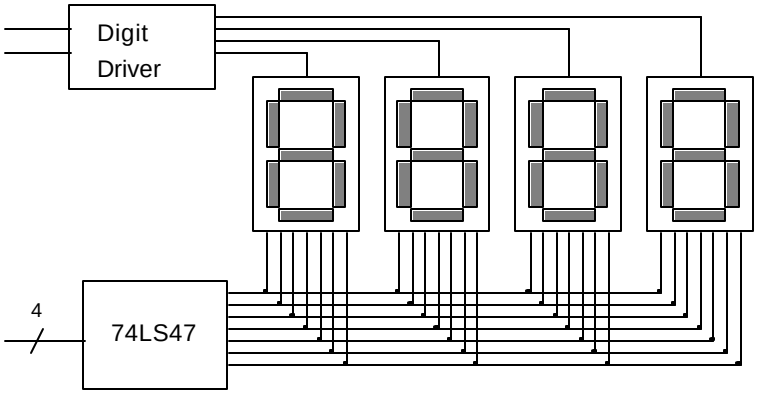


Figure 4-6 Multiplexing Displays

CHAPTER 5 SEQUENTIAL CIRCUITS,CLOCKS AND OSCILLATORS

ONE SHOTS AND CLOCK CIRCUITS FOR SEQUENTIAL DESIGN

CLOCKS & OSCILLATORS

MULTIVIBRATORS (SEQUENTIAL CIRCUITS)

Combinational Logic: A logic circuit in which the outputs are a function of the inputs. At any time, if you know the inputs, you can determine the outputs.

Sequential Logic: A logic circuit in which the outputs are a function of the present, and past inputs. The memory of past inputs involves the "state" of the system. At any time, if you the know the present inputs, and state of the circuit, you can determine the outputs.

Stable State: An internal or external signal maintains a constant magnitude (or specified range or function) for a period of time determined by external input signals.

Quasi-stable State: An internal or external signal maintains a constant magnitude (or specified range or function) for a period of time independent of external stimulus.

Unstable State: An internal or external signal varies in magnitude over time.

Table 5-1 Multivibrators

Type	Example	Stable States	Quasi-Stable
Astable	Clock	0	2
Monostable	One-Shot	1	1
Bistable	Flip-Flop	2	0

Clock

A clock is a device with no inputs and one output.

Frequency

Duty Cycle

Magnitude

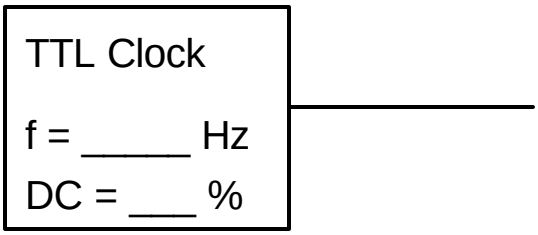


Figure 5-1 TTL Clock

LM555 TIMER IC

The 555 is a multi-function device. Function depends on external configuration and components.

Clock (Astable)
One-Shot (Monostable)
Missing Pulse Detector
Pulse-Width Modulator
Pulse-Position Modulator

LM555 Clock

f = 1.44 / [C1 * (Ra + 2Rb)]
range: ≈0.01 Hz to 1.00 MHz

dc = 100 * { 1 - [Rb / (Ra + 2Rb)] }
range: 50 to 100 %

magnitude = 0 V to +V
range: 4.5 to 16V

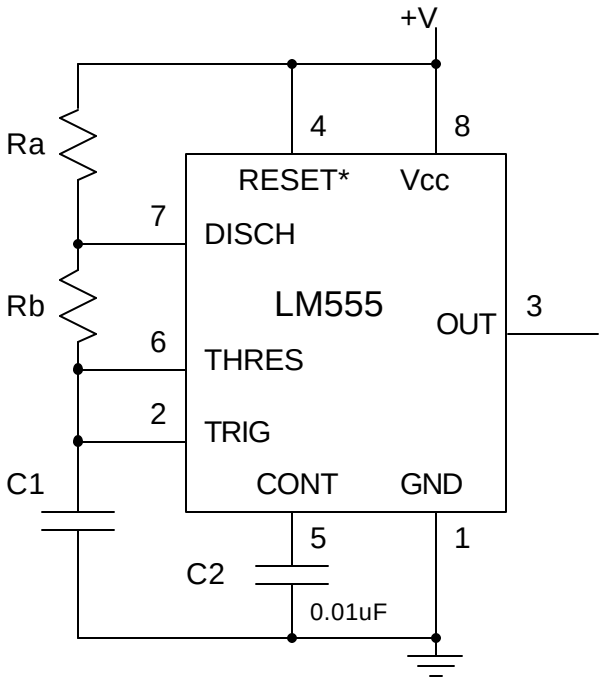


Figure 5-2 LM555 Clock Circuit

LM555 Clock Example

Design a clock circuit using a 555 timer IC to produce a TTL clock with the given specs:

f = 9600 Hz dc = 66.7 %

Step 1: Select C1. From Chart: let C1 = 0.01 μF
Step 2: Solve Ra vs. Rb ratio. 66.7 = 100 * {1 - [Rb / (Ra + 2Rb)]}
1.0 Rb = Ra
Step 3: Solve for Exact Values 9600 Hz = 1.44 / [0.01 μF * (Ra + 2Rb)]
9600 Hz / 1.44 = 1 / [0.01 μF * (3.0*Rb)]
Rb = 5000 Ω

∴ Ra = Rb = 5000 Ω

CRYSTAL OSCILLATORS

Crystals: A crystal is made from a thinly cut piece of quartz sandwiched between two metal leads. Quarts crystals force oscillation at their natural (mechanical) frequency (or harmonics). The natural frequency is primarily a function of quartz thickness. Crystals stabilize the frequency of an oscillating circuit. They provide extremely good frequency stability (0.001 %).

Table 5-2 Crystal Oscillator Types

Oscillator	Series	Pierce (Parallel)
Inverter	Two inverters	Single inverter
Reactance	negative	positive reactance
Osc. Current	0.1 - 10 mA	1 - 300 µA
Start Time	<100 ms	<1 second
Long-Term Stability		
Temp. Stability		
Crystal Removed*	Free run	Stops
Package Seal Failure	Shifts down	Stops or shifts down
Frequency Adjust	difficult	Trimmer Capacitor
Cost	less	

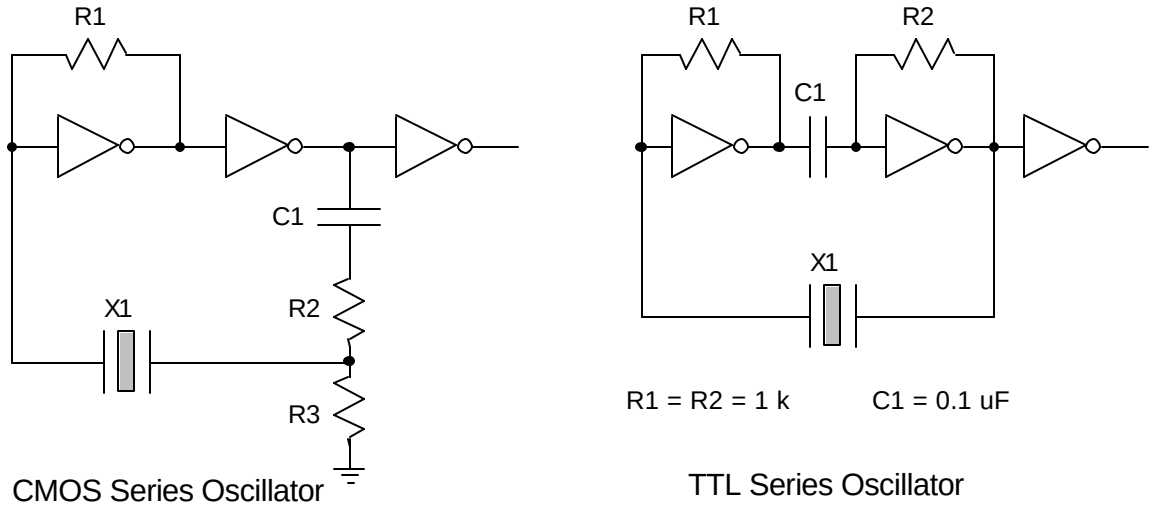


Figure 5-3 Crystal Clock Circuits

Equations for CMOS Series Oscillator: $R1 = 5\ M\Omega * e^{-(10 * 10^{-6} f)}$

$R2 = 0.12 * R1$

$R3 = R2 / (0.3\ Vcc - 0.5)$

SELECTION OF VARIABLE RESISTORS

Variable resistors or potentiometers (pots) are used in clocks and one shots to provide variable frequencies, duty cycles, and pulse widths. They are also used to fine tune circuits to exact values. Fixed resistors do not come in every value. Pots can be used to get any value. However pots should be used with a series resistor.

Example: Design a clock that can produce a variable frequency output in the range 1200 to 9600 Hz.

Step 1. Keeping C, and Ra the same, compute Rb for both frequencies.

Assume	1200 Hz	9600 Hz
Rb =	7.0K Ohms	2.7K Ohms

Step 2. Use a combination of a fixed resistor in series with a pot for Rb such that:

Rfixed < 2.7K Ohms

Rfixed + Rpot > 7.0K Ohms

Solution:

Rfixed = 2.2K, Rpot = 5K

2.2K < Rb < 7.2K

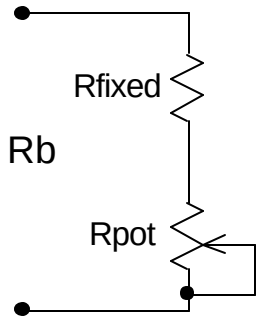


Figure 5-4 Using potentiometers to vary frequency

Keep the resistance of the pot large to have maximum variability. A small turn of the pot results in big change in frequency.

Example: A clock of frequency 9600 Hz +/- 0.1% is needed.

Rb = 5000 Ohms

- tolerance of resistors 20, 10, 5, 1%

- tolerance of capacitors +80% to -20%

Solution: Large fixed resistor in series with a small pot.

Rfixed = 4700 Ohms, Rpot = 1K Ohms

4700 Ohms < Rb < 5700 Ohms

Keep the fixed resistor value large compared to the pot to get best accuracy or maximum precision.

CAUTION: Use pots sparingly: - cost: $pots > $fixed
$ to adjust
- mechanical: noisy, unreliable

SCHMITT GATES

Schmitt Gate Characteristics

- Schmitt gates are essentially TTL inverters that treat inputs slightly different from normal CMOS or TTL.

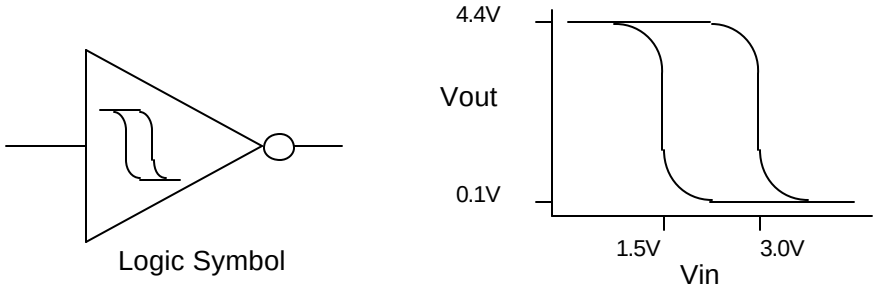


Figure 5-5. Schmitt Trigger Gate Symbol

- The input logic level is always defined.
- Schmitt-trigger inputs have different input threshold levels depending on the direction of the input signal. (Hysteresis)
  Inputs going from a low to a high voltage affect the output at V_{t+} (positive threshold).
  Inputs going from a high to a low voltage affect the output at V_{t-} (negative threshold).

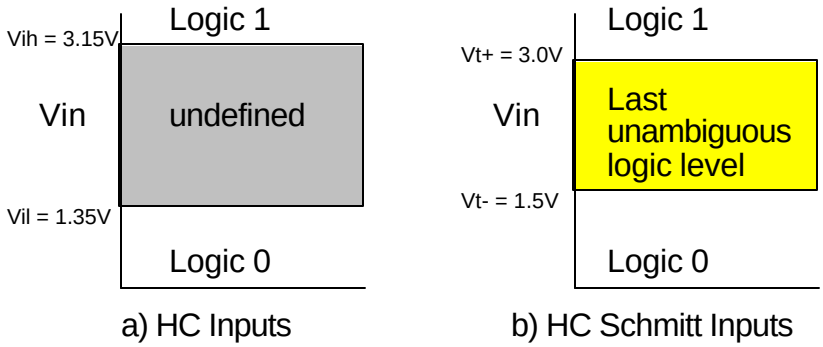


Figure 5-6. High Speed CMOS (HC) Inputs

Applications

- Signal Conditioning: cleaning up noisy, or distorted digital signals
- Line Drivers & Receivers
- Clocks & Delay Circuits

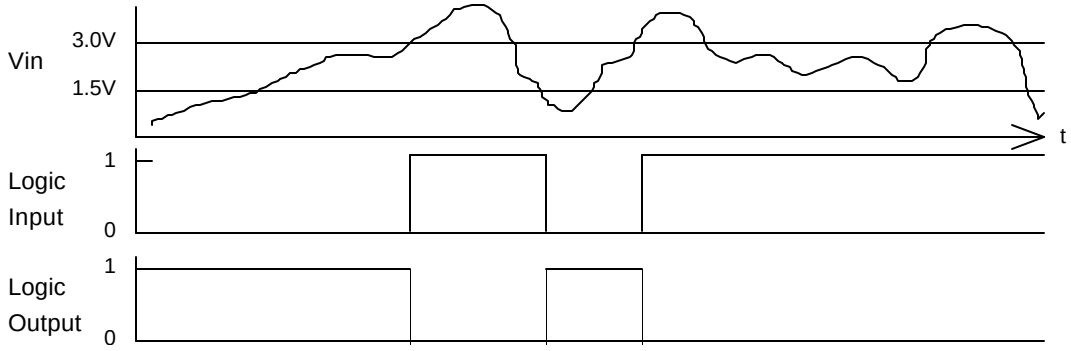
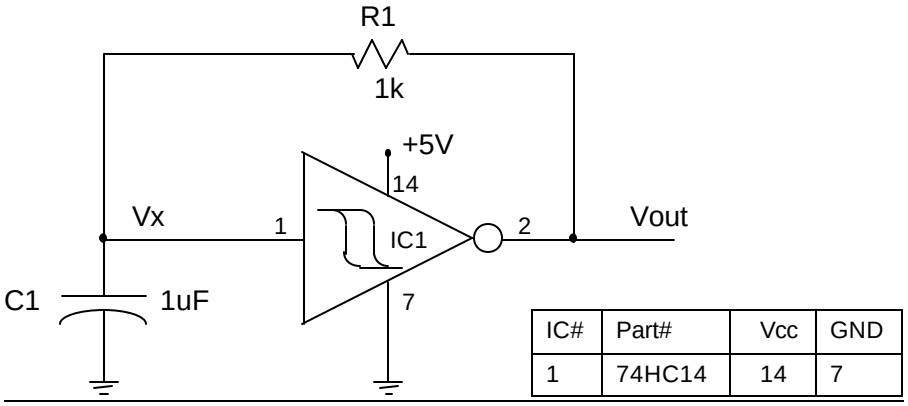


Figure 5-7. Schmitt Timing Example

SCHMITT INVERTER CLOCK

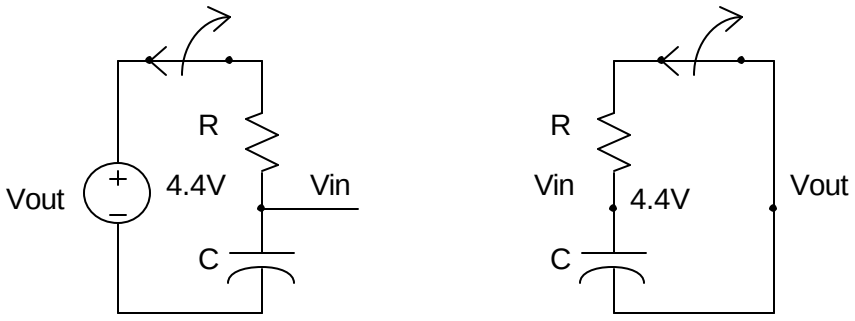
Simple Schmitt Inverter Clock



IC#	Part#	Vcc	GND
1	74HC14	14	7

Figure 5-8. Schmitt Inverter Clock

Assume C1 is discharged before power is applied. Then Vx = 0.0 V.
Since Vx = Vin = logic 0, when power is applied Vout goes high (logic 1).



a) Vin == Logic 0 b) Vin == Logic 1

Figure 5-9. Schmitt Clock Equivalent Circuits

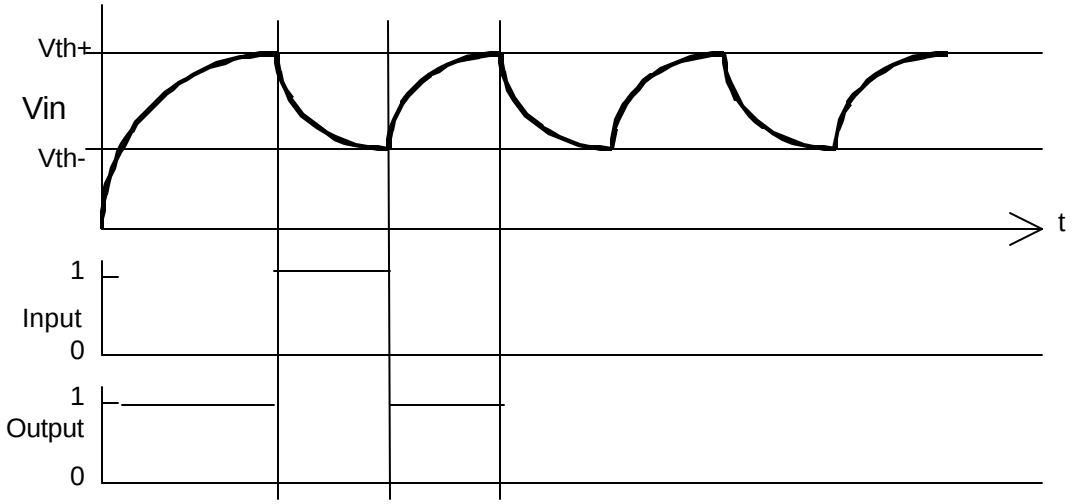


Figure 5-10. Schmitt Clock Timing Diagram

POWER-ON RESET

Flip-Flops power up to unknown states. To assure that a sequential circuit begins operation in a known state power-on/reset circuits are used.

Many registers, flip-flop, and latches have an active-low CLR* (or PRE*) input.

A power-on/reset circuit holds the CLR* input low for a short time after power is applied.

Factors to consider when designing t_{POR}:

- Response time for power supple to become stable.
- Reset time of ICs.
- User interaction.
- Clock settling time.

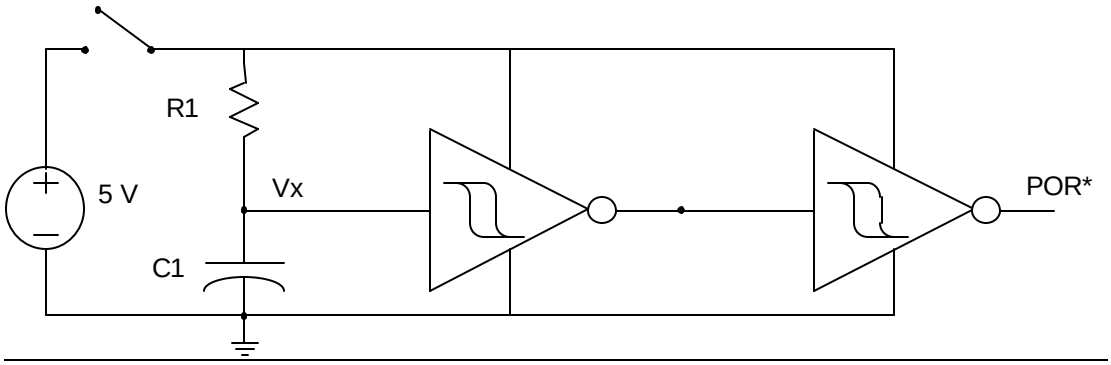


Figure 5-11 Power-On Reset Circuit

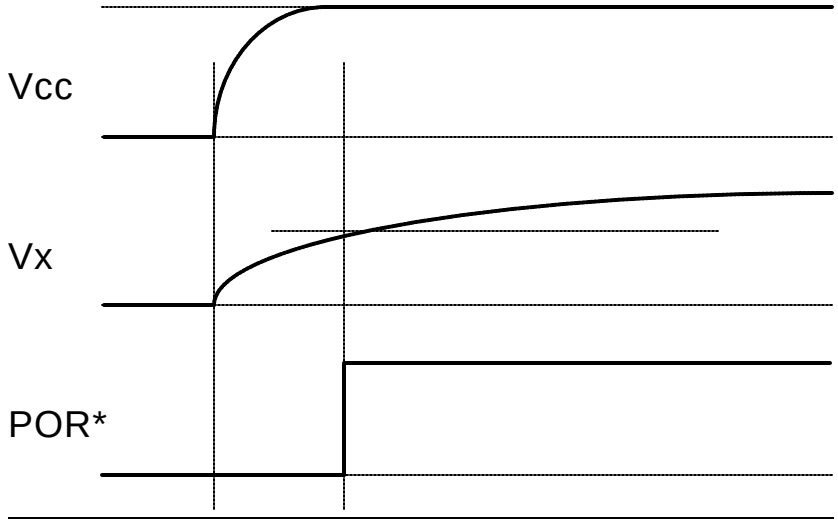


Figure 5-12 Power-On Reset Waveform

$$Vx = Vcc\,(1 - e^{-t/\tau}) \qquad \tau = RC \qquad Vth+ = Vcc(1 - e^{-(t_{POR}/RC)})$$

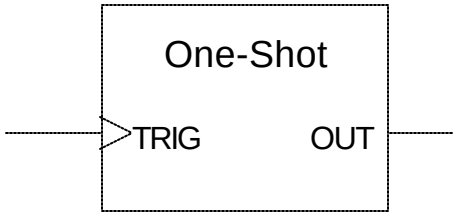
Find t_{POR} when Vx > Vth+ for a Schmitt Trigger at your operating voltage.

ONE SHOTS

One Shot -- Monostable Multi-vibrators
A one-shot is a circuit that produces a stable output (logic 1 or 0) until a trigger (+ or - edge) occurs. The trigger will cause the one-shot to produce a quasi-stable output for a time period determined by the circuit configuration. After the specified period of time, the output returns to the stable state.

The pulse width of the quasi-stable state is independent of external stimulus. Usually, the pulse with is a function of a RC time constant.

Tw = f(R,C) = 0.7RC (for a 74LS221)



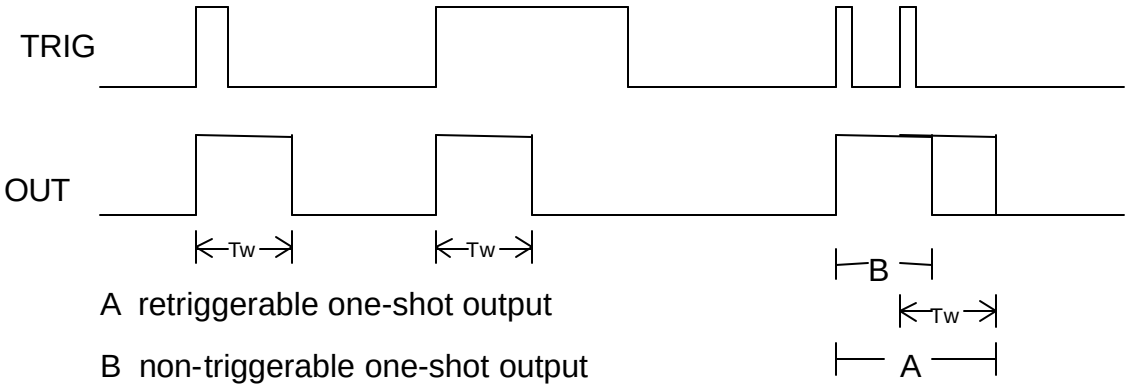


Figure 5-13 Oneshot Timing Waveform

74LS221 Dual One-Shot

The 74LS221 is a dual version of the 74LS121 TTL one-shot. The '221 has either a positive or negative edge trigger, and an active-high, or active-low output.

Refer to TTL data book for specifications.

Tw = *ln*2 * RC

range: 35 ns to 70 s

for jitter free operation:
10 pF ≤ C ≤ 10 µF

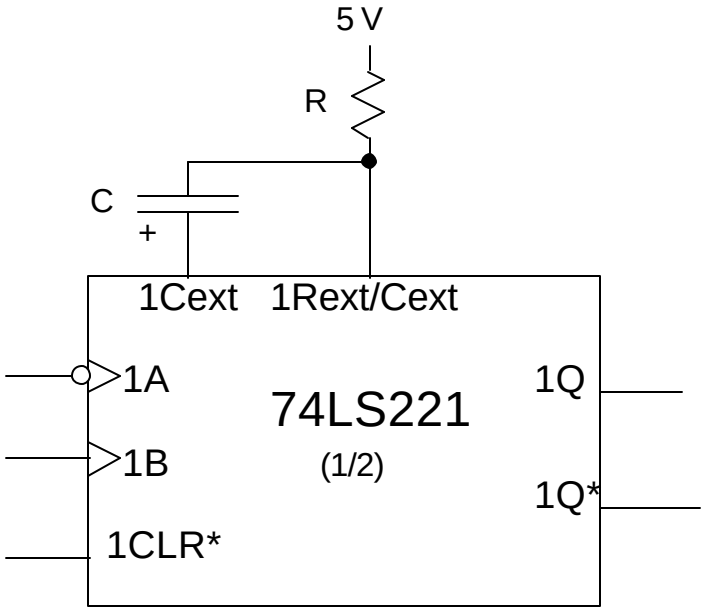


Figure 5-14 74LS221 One-Shot Circuit

CHAPTER 6 COUNTERS, FLIP-FLOPS & SHIFT REGISTERS

DESIGNING SEQUENTIAL CIRCUITS

FLIP-FLOPS

A flip-flop is a device that can maintain binary information until it is directed by an input signal to change its state. There are several different types of flip-flops, the more commonly used are the D-FF and the JK-FF. Flip-flops are used in sequential circuit design. Refer to the High Speed CMOS databook for information on using these flip-flops.

COUNTERS

Counters are sequential circuits or devices that go through a sequence of states when it is clocked. The input clock pulses could occur at discrete time intervals or they may be at random. In digital electronics we are interested in binary and BCD counters. Binary counters are counters that go through a binary sequence. An n-bit binary counter is made of n flip-flops and can count from 0 to 2^n-1 BCD counters count using BCD numbers from 0000 to 1001 and then returns to 0000 and repeats.

MOD-N COUNTERS

A Mod-N counter is one that counts through N states. Most often these mod-N counters count from 0 to n-1 then repeats. Refer to the Digital Design text for an example of the design of a mod-N counter.

FREQUENCY DIVISION USING COUNTERS

The output bits of a counter circuit can be used to divide the frequency of the input clock. Each bit divides the circuit by 2^n. Additionally, a mod-n counter can be used to divide frequency. The output frequency will be f/n where f is the input clock frequency.

Synchronous, Semi-Synchronous, & Asynchronous Counter Differences

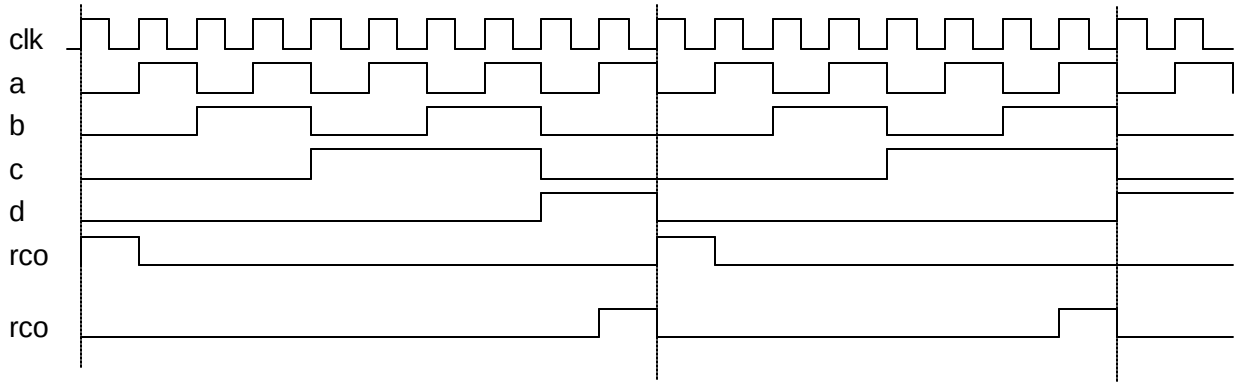


Figure 6-1 BCD Counter Timing Diagram

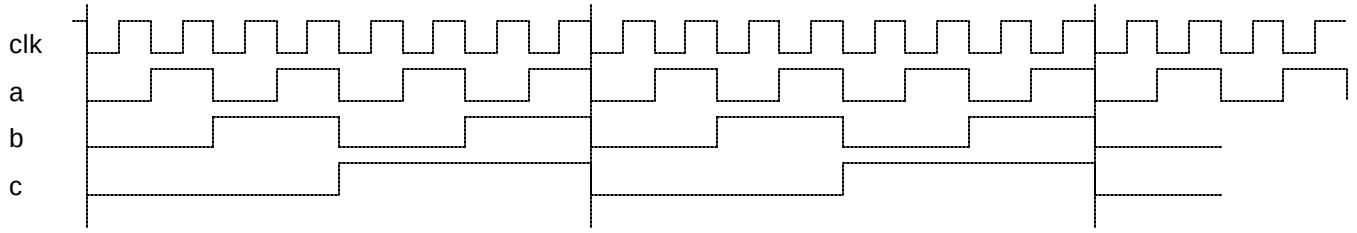


Figure 6-2 -Edge Triggered Ripple Counter

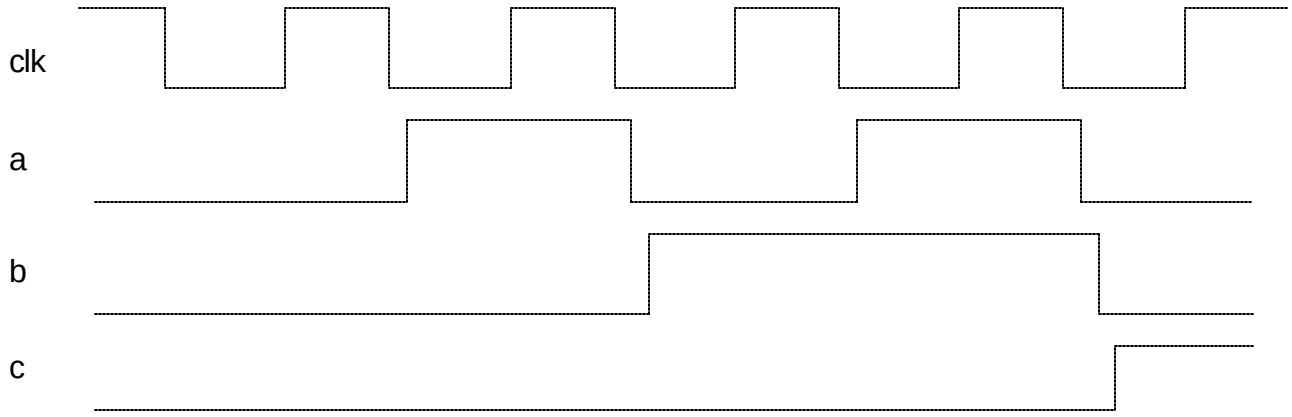


Figure 6-3 Ripple Counter -- Zoom In With Very Fast Clock

CHAPTER 7 PROGRAMMABLE LOGIC DEVICES

DESIGNING WITH PALS AND GALS

PROGRAMMABLE LOGIC DEVICES

Programmable Logic Devices (PLDs) are digital integrated circuits where the Boolean function can be determined by the user. PLDs can replace several specific purpose ICs in a digital design. A single PLD is functionally equivalent to a specific device containing from 5 to 10,000 gates. Typically PLDs implement Boolean functions using Sum Of Minterms (SOM) or Sum of Products (SOP) form. SOM and SOP use a AND-OR gate structure.

PLD Programming
PLDs are manufactured in a "blank" or "erased" form. Programming is performed in concept blowing out fuses between inputs, AND gates, and OR gates in the generic AND-OR structure. An erased PLD has all fuses intact. Actual "fuses" may be implemented as:

Table 7-1 PLD Type Comparison

Type	Function	Advantages
fuses	one-time programmable	low cost
EPROM	uv light erasable	Reprogrammable
CMOS EEPROM	Electrically erasable	fast, easy reprogramming

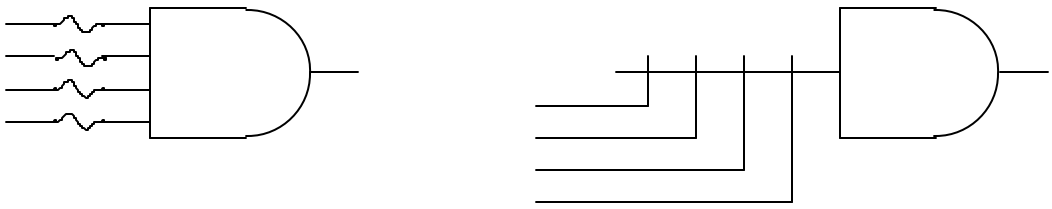


Figure 7-1 Fuses and Logic Implementation

A blown fuse acts like the input does not exist (or a logic 1 at the input).

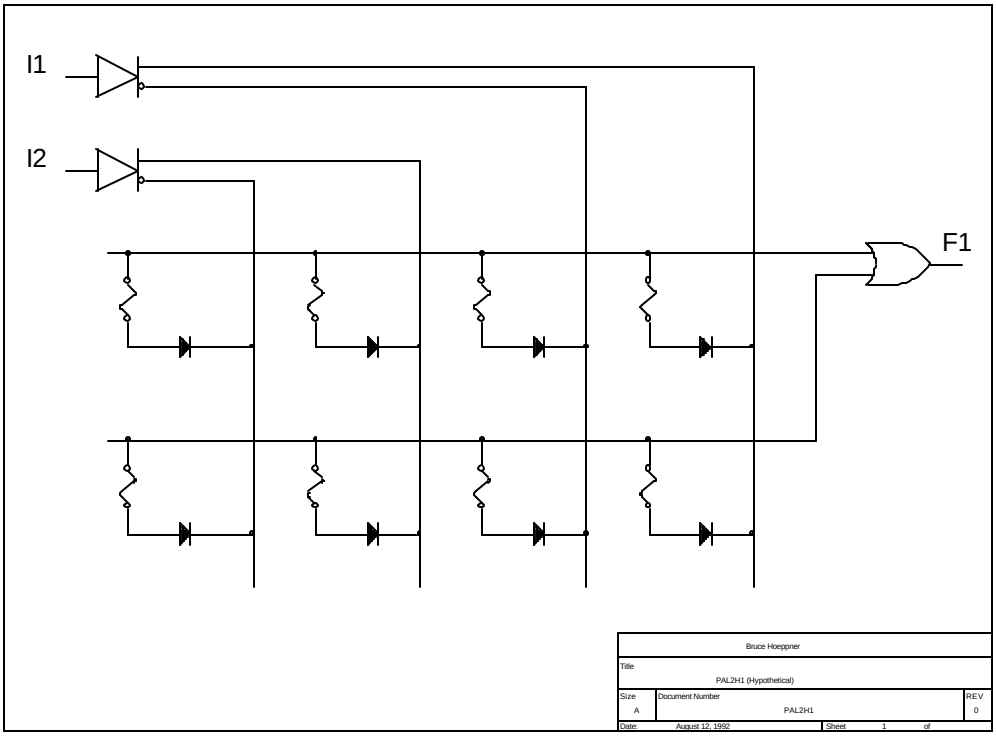
PLD ADVANTAGES:

+ reduce IC package count
  board space
  power
+ shorten design time
+ allow for future changes (maintainability)
+ improve reliability (fewer packages)

+ generally faster
+ smaller inventory

!! Design for overall lowest cost over life of product !!
IC cost
$0.25 7400 Quad 2-input NAND
$1.50 PAL16L8

SIMPLE GENERIC PALS



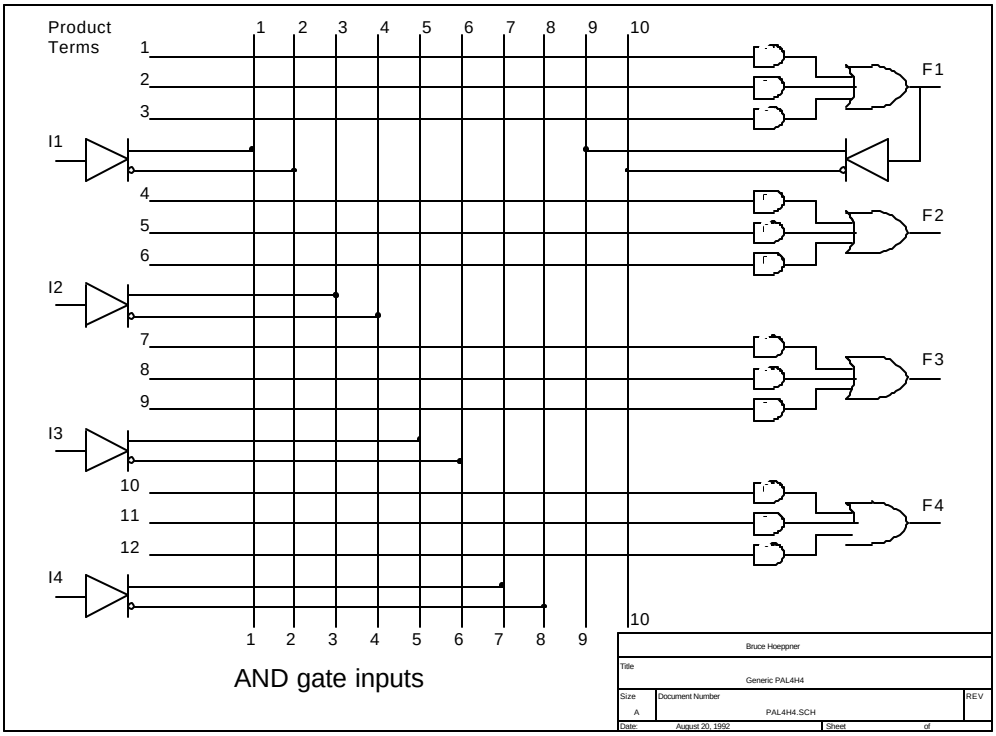


Figure 7-2 Simple PAL

PAL Example: Given functions w, x, y, and z. Implement with <u>one</u> PAL4H4.

Given: Sum of Minterms
W(A,B,C,D) = Σ(2,12,13)
x(A,B,C,D) = Σ(7,8,9,10,11,12,13,14,15)
Y(A,B,C,D) = Σ(0,2,3,4,5,6,7,8,10,11,15)
z(A,B,C,D) = Σ(1,2,8,12,13)

... After Simplification
w = ABC' + A'B'CD'
x = A + BCD
y = A'B + CD + B'D'
z = w + AC'D' + A'B'C'D

Table 7-2 PAL Programming Table

		AND Inputs					
Product	Term	A	B	C	D	w	Outputs
1	ABC'	1	1	0	-	-	w = ABC' + A'B'CD'
2	A'B'CD'	0	0	1	0	-	
3		-	-	-	-	-	
4	A	1	-	-	-	-	x = A + BCD
5	BCD	-	1	1	1	-	
6		-	-	-	-	-	
7	A'B	0	1	-	-	-	y = A'B + CD + B'D'
8	CD	-	-	1	1	-	
9	B'D'	-	0	-	0	-	
10	w	-	-	-	-	1	z = w + AC'D' + A'B'C'D
11	AC'D'	1	-	0	0	-	
12	A'B'C'D	0	0	0	1	-	

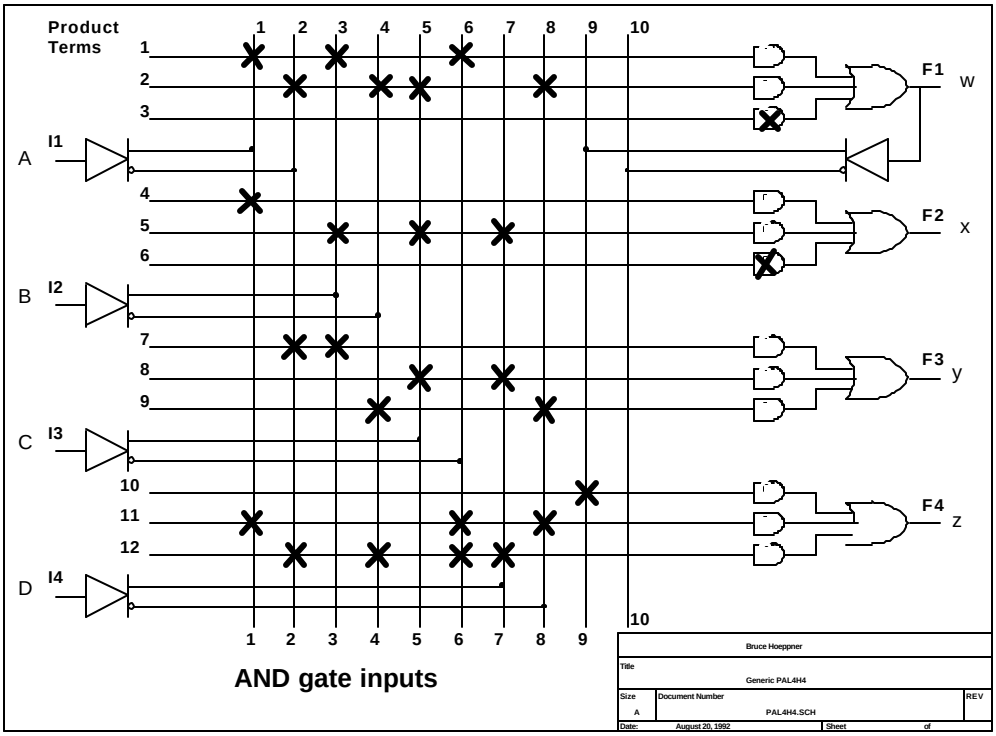


Figure 7-3 Programmed PAL Connections

CUPL PLD COMPILER

CUPL is a software package that runs on an PC. It performs most of the work in translating a PLD design into a programming file. The programming file can be used to program an IC to implement the desired logic functions. Maruette University (College of Engineering) has a site license for CUPL.

COMBINATIONAL LOGIC IMPLEMENTATION

Problem: CUPL expressions must be written for the function. Sum of Products (SOP) is the form CUPL uses. There is a limit of 7 product terms.

Given: A sum of products equation.
Find: A complement form of the equation also in sum of products.

Example: A = x'y + xy'z
In CUPL this becomes: **A = !x&y # x&!y&z;**

Example: B = x'y'z' + xy' + yz
In CUPL this becomes: **B = !x&!y&!z # x&!y # y&z;**

See sample file: **example1.pld**

SEQUENTIAL CIRCUIT IMPLEMENTATION OF PRO GRAMMABLE LOGIC

Given: The state diagram for a sequence recognizer (0110)
Problem: Implement using one PAL16R4 or GAL16V8

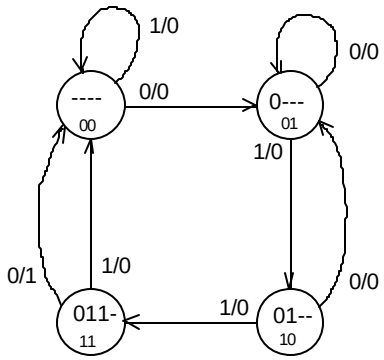


Figure 7-4 Sequential State Diagram

Table 7-3 State Table

State	Input	Next	Output
B A	S	B A	found
0 0	0	0 1	0
0 0	1	0 0	0
0 1	0	0 1	0
0 1	1	1 0	0
1 0	0	0 1	0
1 0	1	1 1	0
1 1	0	0 0	1
1 1	1	0 0	0

Insert into K-Maps:

	B			
BA s\	00	01	11	10
0				
1		1		1

DB = s(B'A) + s(BA')

	A			
BA s\	00	01	11	10
0	1	1		1
1				1

DA = s'B' + B

Let: pin 1 = clock, pin 2 = s, pin 19 = found, pin 17 = B, pin 16 = A, and pin 11 = !enable

Written in CUPL form:
B.d = s&!B&A # s&B&!A;
A.d = !s&!B # B&!A;

The output equation is:

BA s\	00	01	11	10
0			1	
1				

found = !s&B&A; **See: example3.pld & example4.pld**

CHAPTER 8 PROGRAMMING WITH CUPL

COMPILING YOUR PLD PROGRAM

1. Start CUPL application.
2. Select File..New
   This opens a new PLD file using the standard template.
3. Edit the PLD file.
   In the header portion: include the following information:
   Name XXXXX; The source logic description filename
   Partno XXXXX; Company's proprietary part number for the PLD design
   Date XX/XX/XX; Current date of source file modification
   Revision XX; Beginning with 01 and incrementing each time a file is altered
   Designer XXXXX; Designer's name
   Company XXXXX; Company name
   Assembly XXXXX; Assembly name or number on which the PLD will be used
   Location XXXXX; PC board reference or coordinate where PLD will be placed
   Device XXXXX; Default device type for compilation
   Include a description of the function of the PLD program.
   / This is a comment. */*
   In the Inputs section, specify your inputs.
   Pin 2 = x;
   In the Outputs section specify your outputs.
   Pin 19 = A;
   Depending on the function of your PLD you may need to define intermediate variables and declarations.
   You may use Logic Equations for your outputs, Truth Tables, State Machines.
4. Save your PLD file under File Menu, Save As.
   Hint: Save it on your floppy disk to bring to lab.
5. Set Compiling options. Select Option Menu, Compiler Options.
   Under Miscellaneous, select JEDEC Name = filename
   Choose Output File button. Under Download, select JEDEC/POF/PRG. Click OK.
   Click Select Device button. Select Device Type: GAL. Device Selection: g16v8 (if you are using the Gal16V8). Click OK.
   Click Select Library button if no library is shown. Enter library name: "\\Molitor\WINCUPL\CUPL.DL" Click OK. NOTE: You must be logged on to the ENGINEERING domain in order for this to work.
   Click OK on Compile Window.
6. Compile your program.
   Select Run menu item. Choose Device Specific Compile.
   Your program will start compiling.
7. Check for errors or warnings in the Message window. If there are none, you are close to being done. Click OK on the Compile Status window.

If you want to find out more about the capabilities, consult the Help menu or CUPL Starter Manual.

PROGRAMMING YOUR PLD

1. Take your disk with your PLD and JED file to a computer in the Digital Lab or Open Lab with an ALLPRO-88 Programmer.

2. Insert your GAL into the ALLPRO programmer. Be sure to place it in the proper direction. Pin 1 should face closer to the front of the programmer. **Do NOT attempt to program your chip when it is placed incorrectly. This will most likely ruin your GAL.**

3. If it isn't started already, start the ALLPRO software.

4. Select your Device Type based upon the manufacturer Use the **Choose Library and Device** option. Some common device libraries are as follows:
   AMD : PLDAM5
   Cypress: PLDCYP
   Lattice: PLDLA1
   SGS Thompson: PLDSGS

5. Perform a **Blank Check** on the device. If your device is electronically eraseable and not blank, it may still be able to be programmed, however if your chip is UV-eraseable, you need to erase the chip before programming.

6. Read in your JED file using the **Read Formatted File**. This should be off the A: drive. Take note of the checksum number returned.

7. Program your device using the **Program Device** option. Do not touch the chip while the red program light is on.

8. Programming may be verified by comparing the checksum given after the programming is complete with the checksum returned when reading your JED file.

9. Be sure to remove your disk from the A: drive.

If you are having any problems consult the TA.

The following are example programs of the CUPL programming language.

```
Name       Example1;
Partno     none;
Date       06/01/98;
Revision   01;
Designer   Bruce Hoeppner;
Company    EECE143;
Assembly   none;
Location   EN365;
Device     G16V8;

/*************************************************************/
/*  Example program to implement the following equations     */
/*  in CUPL Programming language                             */
/*                                                           */
/*     A = x'y + xy'z                                        */
/*     B = x'y'z' + xy' + yz                                 */
/*                                                           */
/*************************************************************/
/*  Allowable Target Device Types:                           */
/*************************************************************/

/**  Inputs  **/

Pin 2 = x;
Pin 3 = y;
Pin 4 = z;

/**  Outputs  **/

Pin 19 = A;
Pin 18 = B;

/** Declarations and Intermediate Variable Definitions **/



/**  Logic Equations  **/

A = !x&y # x&!y&z;

B = !x&!y&!z # x&!y # y&z;

/** End of example1.pld source code **/
```

```
Name       Example2;
Partno     example2.pld;
Date       06/01/98;
Revision   01;
Designer   Bruce Hoeppner;
Company    EECE143;
Assembly   PLD Source Code;
Location   EN365;
Device     G16V8;

/*************************************************************/
/*  Example program to implement the following equations     */
/*  in CUPL programming language using Truth Tables          */
/*                                                           */
/*    A = x'y + xy'z                                         */
/*    B = x'y'z' + xy' + yz                                  */
/*                                                           */
/*************************************************************/
/*  Allowable Target Device Types:                           */
/*************************************************************/

/**  Inputs  **/

Pin 2 = x;
Pin 3 = y;
Pin 4 = z;
/**  Outputs  **/

Pin 19 = A;
Pin 18 = B;

/** Declarations and Intermediate Variable Definitions **/

FIELD INPUT = [x,y,z];

FIELD OUTPUT = [A,B];

/**  Truth Table  **/

TABLE INPUT => OUTPUT {

0=>'b'01;
1=>'b'00;
2=>'b'10;
3=>'b'11;
4=>'b'01;
5=>'b'11;
```

```
6=>'b'00;
7=>'b'01;
}
/** End of example2.pld source code **/
```

```
Name       Example3;
Partno     example3.pld;
Date       02/19/99;
Revision   01;
Designer   JChris Perez;
Company    EECE143;
Assembly   PLD Source Code;
Location   EN365;
Device     G116V8;

/*****************************************************************/
/*  Example program to implement the sequential circuit       */
/*     in CUPL programming language using equations            */
/*                                                             */
/*****************************************************************/
/*  Allowable Target Device Types: g16v8                       */
/*****************************************************************/

/**  Inputs  **/

Pin 1 = clock;
Pin 2 = s;
Pin 11 = !enable;
/**  Outputs  **/

Pin 19 = found;
Pin 17 = B;
Pin 16 = A;

/** Declarations and Intermediate Variable Definitions **/


/**  Logic Equations  **/

B.d = s&!B&A#s&B&!A;
A.d = !s&!B#B&!A;

/** End of example3.pld source code **/
```

```
Name       Example4;
Partno     example4.pld;
Date       06/01/98;
Revision   01;
Designer   Bruce Hoeppner;
Company    EECE143;
Assembly   PLD Source Code;
Location   EN365;
Device     G16V8;

/*************************************************************************/
/*  Example program to implement the sequential circuit          */
/*  in CUPL programming language using State Machines            */
/*                                                               */
/*                                                               */
/*************************************************************************/
/*  Allowable Target Device Types:                               */
/*************************************************************************/

/**  Inputs  **/

Pin 1 = clock;
Pin 2 = s;              /* combinational data input */
Pin 11 = !enable;

/**  Outputs  **/

Pin [14..15] =[Q1..0];
Pin 19 = found;                 /* combinational output */

/** Declarations and Intermediate Variable Definitions **/

field state = [Q1..0];
/* field state= [B,A]; */

$define S0 'b'00
$define S1 'b'01
$define S2 'b'10
$define S3 'b'11

/**  State Machine  **/

Sequence state {
present S0 if s next S0;
     default next S1;
present S1 if s next S2;
     default next S1;
```

```
present S2 if s next S2;
     default next S3;
present S3 if s next S0;
     default next S3 out found;
}
/** End of example4.pld source code **/
```

CHAPTER 9 MICROPROCESSORS AND MICROCONTROLLERS

DIGITAL DESIGN WITH MICROCONTROLLERS

68HC11 INTRODUCTION AND FEATURES

Single chip microcomputer: includes CPU, Memory, and I/O.
Designed for control applications (vs data processing.)

68HC711E9 Integrated Circuit

CPU:

- 8-bit word size
- 6800 instruction set with extensions
- Accumulators: AccA, AccB (8-bit)
- Index registers: X, Y (16-bit)
- Modes of operation:
  - Single Chip
  - Expanded (64K address space)
  - Special Boot
  - Test

Memory: (On chip)

- 12K of EPROM
- 512 bytes of EEPROM
- 256 - 512 bytes of RAM

Input/Output: (Memory mapped)

- 5 parallel digital I/O ports
- RS-232 (connect to terminals or PCs)
- High Speed Synchronous Serial
- Parallel Handshake (Printer)
- Timer functions
  - Generate precise digital signals
  - Measure frequency and pulse width
  - Count external events
  - Real Time Clock
- 8-channel, 8-bit Analog to Digital Converter

68HC11EVBU Evaluation Board

- 68HC11 IC
- Single 5V Operation
- RS-232 Port
- Battery Backup Clock

68HC11 REGISTERS

Accumulators

A, B 8-bit accumulators, location of math and logic functions

D 16-bit accumulator, actually same physical device as A and B taken together

7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
AccA								AccB							
AccD															
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Index Registers

X, Y 16-bit registers, used for indexed addressing (may be used as general storage registers or counters)

Special Registers

P 16-bit program counter, keeps track of address of next instruction to be executed.

S 16-bit stack pointer, used for temporary storage of the program counter (needed for subroutines)

C 8-bit condition code register. Stores several 1-bit flags (status indicators), and interrupt masks

Condition Code Register							
S	X	H	I	N	Z	V	C
7	6	5	4	3	2	1	0

Flags are set according to the result of certain opcodes:

- **C** set when a math operation carrys or borrows
- **V** set when a math operation exceeds and 8-bit result
- **Z** set when a result of a math or logic operation is $00
- **N** set when a result of an operation leaves the MSB = 1
- **H** set when a math operation carrys or borrows between two nibbles of a byte.
- **I** user must clear I to enable I-interrupts. (=1 @ reset)
- **X** user must clear X to enable X-interrupts. (=1 @ reset)

68HC11 MEMORY MAP

Memory Address Space

The 68HC11 has a 16-bit program counter, and 16-bit address bus (external pins). This allows for 2^{16} possible memory locations (65536 ==> 64K).

68HC11 Single Chip Mode Memory Map

Top of Memory $FFFF

FFFF Reset Vectors
and
Interrupt Vectors

FFC0 (part of 8K ROM)

FFBF

Rest of 8K ROM

E000

DFFF

Unused in single chip mode
(RAM on EVB)

C000

BFFF

Unused in single chip mode

B800

B7FF

512 byte Internal EEPROM

B600

B5FF

Unused in single chip mode

8000

7FFF

Unused in single chip mode

6000

5FFF

Unused in single chip mode

1040

103F

I/O Registers

1000

0FFF

Unused in single chip mode

0100

00FF

256 byte internal RAM

0000

Bottom of Memory

68HC11 I/O REGISTERS

I/O Registers on the 68HC11 are found in the memory address space. The 5 parallel (digital) input/output ports are:

Port A $1000 8-bit I/O port.
Inputs: bits 0, 1, 2
Outputs: bits 3, 4, 5, 6
Bidirectional: bit 7 (controlled by DDRA7)

Port B $1004 Output only port. EVB pins 42 - 35 (0 -7)

Port C $1003 8-bit bidirectional I/O port.
Direction is determined by the value of the Port C Data Direction Register (DDRC)

DDRC $1007 Each bit in DDRC sets the corresponding bit in Port C as either an input (0 in DDRC) or as an output (1 in DDRC). Default: all 0's

Port D $1008 6-bit bidirectional I/O port.
Direction is determined by the value of the Port D Data Direction Register (DDRD)

DDRD $1009 Similar in function to DDRC.

Port E $100A Input only port. EVB pins 43 - 50 (0 -7)

M68HC11 INTEGRATED CIRCUIT

Table 9-1 68HC11 Pin Functions

Name	Description	PLCC	EVB	EVBU	48DIP	40DIP
Vss/GND	Ground	1	1	1		
MODB/Vstby	Mode Select, Stand-by power	2	2	2		
MODA/LIR*	Mode Select; LIR* Status Signal	3	3	3		
STRA/AS	Strobe A; Address Strobe	4	4	4		
E	E System Clock	5	5	5		
STRB/RW*	Strobe B; R/W* Control Signal	6	6	6		
EXTAL	Crystal Pin, or Clock Input	7	7	7		
XTAL	Cryatal Pin	8	8	8		
PC0/AD0	PortC Pin0; Addr/Data 0	9	9	9		
PC1/AD1		10	10	10		
PC2/AD2		11	11	11		
PC3/AD3		12	12	12		
PC4/AD4		13	13	13		
PC5/AD5		14	14	14		
PC6/AD6		15	15	15		
PC7/AD7		16	16	16		
RESET*	Reset input and output	17	17	17		
XIRQ*	"Non-maskable" Interrupt	18	18	18		
IRQ*	Interrupt	19	19	19		
PD0/RxD	PortD, Pin0; RS232 Receive Data	20	20	20		
PD1/TxD	... RS232 Transmit Data	21	21	21		
PD2/MISO	... Master In Slave Out Data Line	22	22	22		
PD3/MOSI	... Master Out Slave In Data Line	23	23	23		
PD4/SCK	... Sync. Periph. Interface Clock	24	24	24		
PD5/SS*	... Slave Select Control Signal	25	25	25		
Vdd	5 Volt Power Supply	26	26	26		
PA7/PAI/OC1	PortA, Pin7; Pulse Accum.; Output Compare 1	27	27	27		
PA6/OC2/OC1	PortA, Pin6; Output Compare 2,	28	28	28		

PA5/OC3/OC1	PortA, Pin5; Output Compare 3,	29	29	29		
PA4/OC4/OC1	PortA, Pin4; Output Compare 4,	30	30	30		
PA3/OC5/OC1	PortA, Pin3; Output Compare 5,	31	31	31		
PA2/IC1	PortA, Pin2; Input Capture 1	32	32	32		
PA1/IC2	PortA, Pin1; Input Capture 2	33	33	33		
PA0/IC3	PortA, Pin0; Input Capture 3	34	34	34		
PB7/A15	PortB, Pin7; Address Bus 15	35	35	35		
PB6/A14		36	36	36		
PB5/A13		37	37	37		
PB4/A12		38	38	38		
PB3/A11		39	39	39		
PB2/A10		40	40	40		
PB1/A09		41	41	41		
PB0/A08		42	42	42		
PE0/AN0	PortE, Pin0; ADC Input 0	43	43	43		
PE4/AN4		44	44	44		
PE1/AN1		45	45	45		
PE5/AN5		46	46	46		
PE2/AN2		47	47	47		
PE6/AN6		48	48	48		
PE3/AN3		49	49	49		
PE7/AN7		50	50	50		
Vrl	ADC Reference Voltage -	51	51	51		
Vrh	ADC Reference Voltage +	52	52	52		
Spare	Spare Connector Pins	na	nc	53-56		
Vcc	EVBU Power Rail	na	nc	57,58		
GND	EVBU Ground Rail	na	nc	59,60		

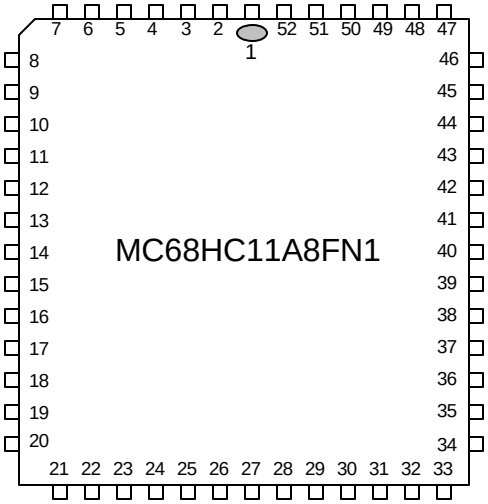


Figure 9-1: 68HC11 IC in a 52-pin PLCC Package

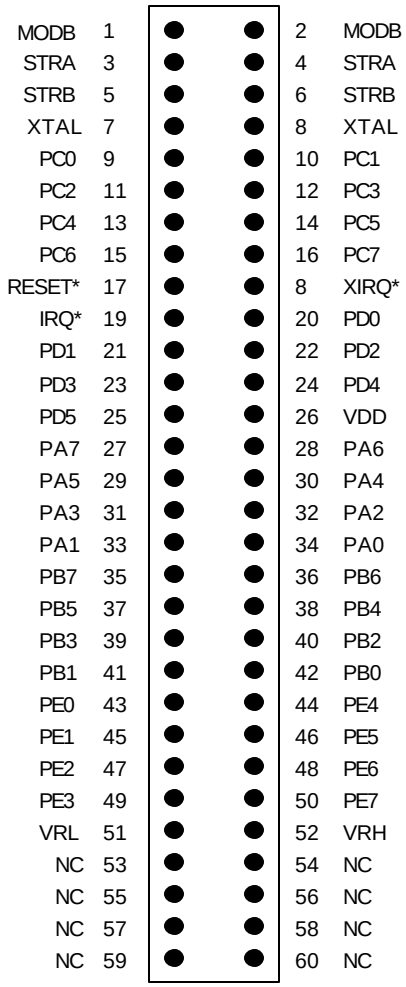


Figure 9-2: 68HC11 EVB P1 60-Pin Header

68HC11 INSTRUCTION SET INTRODUCTION

Move Instructions and Addressing Modes

LDAA LoaD Accumulator A LDAA operand
Copies data from the location specified by the operand to AccA

Immediate Addressing: data immediately follows instruction. Use # before number constant.

```
        LDAA #$64       $64 → AccA

        LDAA #100       exact same as above

CYCLES  EQU 100
        LDAA #CYCLES
```

Direct Addressing (8-bit address): 8-bit address of data follows instruction. Also called Page 0 Addressing.

```
        LDAA $64        $0064 is 16-bit address
                        value at RAM address $64 → AccA
                        [$0064] → AccA
```

Extended Addressing (16-bit address): 16-bit address of data follows instruction.

```
        LDAA $100A      [$100A] → AccA

PORTE   EQU $100A

        LDAA PORTE
```

Indexed Address: the operand's address is the content of the index register plus the ***constant*** given in the instruction.

```
        LDX #$1000
        LDAA $A,X       [[$1000] + 10] → AccA

REGBASE EQU $1000

        LDAA 10,X
```

Other Load Instructions

LDAB LoaD Accumulator B LDAB operand
Copies data from the location specified by the operand to AccB

LDD LoaD Accumulator D LDD operand
Copies 2 bytes of data from the location specified by the operand to AccD. The byte at the location specified is moved to bits 8 - 15 of AccD, then the byte at the specified address + 1 is moved to bits 0 - 7.

LDX LoaD index register X LDX operand
Similar to LDD

LDY LoaD index register Y LDY operand
Similar to LDD

LDS LoaD Stack pointer LDS operand
Similar to LDD

Store Instructions

STAA STore AccA STAA operand
Copies data from AccA to the location specified by the operand.

Other Store opcodes: STAB, STD, STX, STY, STS

Transfer Instructions

TAB Transfer AccA to AccB TAB
Copies data from the AccA to AccB. Old data in AccB is lost.
This is an example of Inherent Address Mode.

Other Transfer opcodes: TBA, TAP (to C), TPA, TSX, TXS, ...

Stack Instructions

PSHA PuSH AccA PSHA
Copies data in AccA to the memory location specified by the stack pointer (S), then decrements S by 1.

PULA PULl AccA PULA
Copies data from the memory location specified by the stack pointer (S) to AccA, then increments S by 1.

Other Stack opcodes: PSHB, PULB, PSHX, ...

Clear Instructions

CLRa CLeaR AccA CLRA, CLRB
Clears AccA (resets all bits to 0).

CLR CLeaR memory location CLR operand
Clears the data in that memory location (resets all bits to 0). May be used with Indexed or Direct addressing modes only.

Inherent & Relative Addressing Modes
(Two addressing modes do not apply to load instructions; **Inherent** (operand information is contained in the opcode), and **Relative** (used for branching, see BEQ opcode).)

Arithmetic Instructions

ADDA ADD AccA ADDA operand
Add the value specified by the operand to the value in AccA, then put the result back in AccA. Addressing Modes: Immediate, Direct, Extended, or Indexed.

ADCA ADd with Carry AccA ADCA operand
Add the value specified by the operand, plus the value of the carry bit (C flag in the Condition Register) to the value in AccA, then put the result back in AccA. Addressing Modes: Immediate, Direct, Extended, or Indexed.

Other Arithmetic Instructions: ADDB, ...

Logic Instructions

ANDA AND accumulator A ANDA operand
Perform a bit-wise ANDing of AccA with the value specified by the operand.

Other Logic Instructions: ANDB, ORAA, ...

Shift Instructions

ASLA Arithmetic Shift Left AccA ASLA
Shift a 0 into bit 0 of AccA, bit 0 into bit 1, ..., bit 6 into bit 7, and bit 7 into the carry flag (C).

$$C \leftarrow b7\ b6\ b5\ b4\ b3\ b2\ b1\ b0 \leftarrow 0$$

ASRA Arithmetic Shift Right AccA ASRA
Shift bit 7 into bit 7 and bit 6, bit 6 into bit 5, ..., bit 1 into bit 0, and bit 0 into the carry flag (C). This instruction divides a twos complement value by two without changing the sign.

LSLA Logical Shift Left AccA LSLA
Exact same as ASLA

LSRA Logical Shift Right AccA LSRA
Shift a 0 into bit 7, bit 7 into bit 6, bit 6 into bit 5, ..., bit 1 ito bit 0, and bit 0 into the carry flag (C).

$0 \rightarrow$ b7 b6 b5 b4 b3 b2 b1 b0 $\rightarrow$ C

ROLA ROtate Left AccA ROLA

C $\leftarrow$ b7 b6 b5 b4 b3 b2 b1 b0 $\leftarrow$ C

ROLA ROtate Right AccA ROLA

C $\rightarrow$ b7 b6 b5 b4 b3 b2 b1 b0 $\rightarrow$ C

Control Instructions

JMP JuMP to new address JMP operand
Load program counter with address specified by operand, then begins executing instructions at that address. Operand specifies a 16-bit address anywhere in the 64K address space. Extended or Indexed addressing modes.

BRA BRAnch to new address BRA operand
Load program counter with a new address specified by operand, then begins executing instructions at that address. Operand specifies a branch between -128 to +127 bytes from the instruction immediately following the branch instruction. The operand can be a label if it designates an address within the allowed range. Relative addressing only!

BEQ Branch if EQual BEQ operand
Branch to address specified if the Z flag is set.

BHI Branch if Higher BHI operand
Usually follows a compare or math instruction. Assumes unsigned numbers.

JSR Jump to SubRoutine JSR operand
Push the program counter onto the stack. Jump to the address specified by the operand. Execute instructions until a RTS instruction is encountered. Then return to instruction following the JSR instruction.

RTS ReTurn from Subroutine RTS
Causes a return from a subroutine call.

Other Control Instructions: ...

Imm.	Immediate	Index	Indexed
Direct	Direct (8-bit address, or page zero)	Inherent	Inherent
Exten.	Extended (16-bit address)		

Table 9-2 68HC11 Move Instructions

Instr.	Description	Imm.	Direct	Exten.	Index	Inher.
lda_	Load Acc_	√	√	√	√	
ldd	Load AccD	√	√	√	√	
ldx	Load IX	√	√	√	√	
ldy	Load IY	√	√	√	√	
Lds	Load Stack Pointer	√	√	√	√	
Sta_	Store Acc_		√	√	√	
Std	Store AccD		√	√	√	
Stx	Store IX		√	√	√	
Sty	Store IY		√	√	√	
Sts	Store Stack Pointer		√	√	√	
tab	Transfer AccA to AccB				√	
tba	Transfer AccB to AccA				√	
tap	Transfer AccA to Flags				√	
tpa	Transfer Flags to AccA				√	
Tsx	Transfer SP to IX				√	
Txs	Transfer IX to SP				√	
Tsy	Transfer SP to IY				√	
Tys	Transfer IY to SP				√	
psh_	Push Acc_ to stack					√
pul_	Pull Acc_ from stack					√
pshx	Push IX to stack					√
pulx	Pull IX from stack					√
pshy	Push IY to stack					√
puly	Pull IY from stack					√
Clr_	Clear Acc_					√
Clr	Clear Memory			√	√	
tsta	Test A					√
tstb	Test B					√
Tst	Test [M]			√	√	
xxx_	replace "_" with					
	"a" or "b"					

Table 9-3 68HC11 Arithmetic Instructions

Instr.	Description	Imm.	Direct	Exten.	Index	Inher.
add_	add Acc_	√	√	√	√	
adc_	add w/carry	√	√	√	√	
aba	add AccB to AccA					√
sub_	subtract AccB	√	√	√	√	
sbc_	subtract w/borrow	√	√	√	√	
sba	subtract AccB from AccA					√
cmp_	Compare Acc_	√	√	√	√	
cba	Compare AccB to AccA					√
inc_	Increment Acc_					√
inc	Increment Memory			√	√	
dec_	Decrement Acc_					√
dec	Decrement Memory			√	√	
neg_	Negate Acc_					√
neg	Negate Memory			√	√	
asl_	Arith. Shift Left Acc_					√
asl	Arith. Shift Left Memory			√	√	
asr_	Arith. Shift Right Acc_					√
asr	Arith. Shift Right Memory			√	√	
addd	Add AccD	√	√	√	√	
subd	Subtract AccD	√	√	√	√	
cpd	Compare AccD	√	√	√	√	
cpx	Compare IX	√	√	√	√	
cpy	Compare IY	√	√	√	√	
inx	Increment IX					√
iny	Increment IY					√
ins	Increment Stack Pointer					√
dex	Decrement IX					√
dey	Decrement IY					√
des	Decrement Stack Pointer					√
fdiv	Floating Point Divide					√
idiv	Integer Divide					√
asld	Arith. Shift Left AccD					√
daa	Decimal Adjust AccA					√
mul	Multiply AccA * AccB					√
abx	Add AccB to IX					√
aby	Add AccB to IY					√
xxx_	replace "_" with					
	"a" or "b"					

Table 9-4 68HC11 Logic Instructions

Instr.	Description	Imm.	Direct	Exten.	Index	Inher.
and_	AND Acc_	√	√	√	√	
ora_	OR Acc_	√	√	√	√	
eor_	Exclusive OR Acc_	√	√	√	√	
bit_	Bit Test	√	√	√	√	
com_	Complement (1's) Acc_					√
com	Complement Memory			√	√	
sec	Set Carry Flag					√
sei	Set Interrupt Mask					√
sev	Set Overflow Flag					√
clc	Clear Carry Flag					√
cli	Clear Interrupt Mask					√
clv	Clear Overflow Flag					√
bset	Set Bit		√		√	
bclr	Clear Bit		√		√	
lsl_	Logical Shift Left Acc_					√
lsl	Logical Shift Left Memory			√	√	
lsr_	Logical Shift Right Acc_					√
lsr	Logical Shift Right Memory			√	√	
lsld	Logical Shift Left AccD					√
lsrd	Logical Shift Right AccD					√
rol_	Rotate Acc_ Left					√
rol	Rotate Memory Left			√	√	
ror_	Rotate Acc_ Right					√
ror	Rotate Memory Right			√	√	
xxx_	replace "_" with					
	"a" or "b"					

Table 9-5 68HC11 Control Instructions

Instr.	Description	Mode	Simple	Sign	Unsign	Bit
jmp	Jump to	E,I				
bra	Branch to	R				
brn	Branch Never	R				
nop	No Operation	R				
beq	Branch if Equal or Zero	R	√			
bne	Branch if Not Equal	R	√			
bmi	Branch if Minus (MSB set)	R	√			
bpl	Branch if Plus (MSB clear)	R	√			
bcs	Branch if Carry Set	R	√			
bcc	Branch if Carry Clear	R	√			
bvs	Branch if Overflow Set	R	√			
bvc	Branch if Overflow Clear	R	√			
bgt	Branch if Greater Than	R		√		
bge	Branch if Greater or Equal	R		√		
blt	Branch if Less Than	R		√		
ble	Branch if Less or Equal	R		√		
bhi	Branch if Higher Than	R			√	
bhs	Branch if Higher or Same	R			√	
blo	Branch if Lower Than	R			√	
bls	Branch if Lower or Same	R			√	
brset	Branch if Bit Set	D,I				√
brclr	Branch if Bit Clear	D,I				√
jsr	Jump to Subroutine	E, I				
bsr	Branch to Subroutine	R				
rts	Return from Subroutine					
rti	Return from Interrupt					
swi	Software Interrupt Call					
stop	Put uP in Stop mode					
wait	Put uP in Wait mode					

BRANCH IF EQUAL

BEQ Branch if EQual **BEQ**

Operation: PC ⇐ (PC) + $0002 + Rel if (Z) = 1

Description: Tests the state of the Z bit in the CCR and causes a branch if Z is set. See BRA instruction for further details of the execution of the branch.

Condition Codes and Boolean Formulae:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected

Source Form: BEQ (rel)

Addressing Modes, Machine Code, and Cycle-by-Cycle Execution:

	BEQ (REL)		
Cycle	Addr	Data	R/W*
1	OP	27	1
2	OP+1	rr	1
3	FFFF	--	1

The following table is a summary of all branch instructions.

Test	Boolean	Mnemonic	Opcode	Comment
r > m	Z+(N⊕V)=0	BGT	2E	Signed
r ≥ m	N⊕V=0	BGE	2C	Signed
r = m	Z=1	BEQ	27	Signed
r ≤ m	Z+(N⊕V)=1	BLE	2F	Signed
r < m	N⊕V=1	BLT	2D	Signed
r > m	C + Z = 0	BHI	22	Unsigned
r ≥ m	C = 0	BHS/BCC	24	Unsigned
r = m	Z = 1	BEQ	27	Unsigned
r ≤ m	C + Z = 1	BLS	23	Unsigned
r > m	C = 1	BLO/BCS	25	Unsigned
Carry	C = 1	BCS	25	Simple
Negative	N = 1	BMI	2B	Simple
Overflow	V = 1	BVS	29	Simple
r = 0	Z = 1	BEQ	27	Simple
Always	--	BRA	20	Unconditional

LOAD ACCUMULATOR

LDA LoaD Accumulator **LDA**

Operation: AccX ⇐ (M)

Description: Loads the contents of memory into the 8-bit accumulator. The condition codes are set according to the data.

Condition Codes and Boolean Formulae:

S	X	H	I	N	Z	V	C
-	-	-	-	⇳	⇳	0	-

N R7
Set if MSB of result is set; cleared otherwise

Z R7' • R6' • R5' • R4' • R3' • R2' • R1' • R0'
Set if result is $00; cleared otherwise

V 0
cleared

Source Form: LDAA (opr); LDAB (opr)

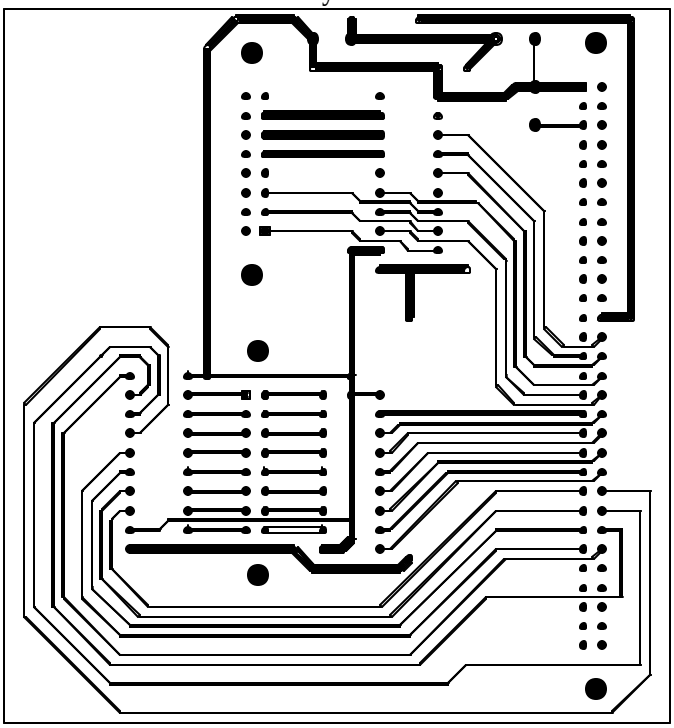
Addressing Modes, Machine Code, and Cycle-by-Cycle Execution:

Cycle	LDAA (IMM)			LDAA (DIR)			LDAA (EXT)			LDAA (IND,X)			LDAA (IND,Y)		
	Addr	Data	R/W*	Addr	Data	R/W*	Addr	Data	R/W*	Addr	Data	R/W*	Addr	Data	R/W*
1	OP	86	1	OP	96	1	OP	B6	1	OP	A6	1	OP	18	1
2	OP+1	ii	1	OP+1	ii	1	OP+1	hh	1	OP+1	ff	1	OP+1	A6	1
3				00dd	(00dd)	1	OP+2	ll	1	FFFF	--	1	OP+2	ff	1
4							hhll	(hhll)	1	X+ff	(X+ff)	1	FFFF	--	1
5													Y+ff	(Y+ff)	1

Cycle	LDAB (IMM)			LDAB (DIR)			LDAB (EXT)			LDAB (IND,X)			LDAB (IND,Y)		
	Addr	Data	R/W*	Addr	Data	R/W*	Addr	Data	R/W*	Addr	Data	R/W*	Addr	Data	R/W*
1	OP	C6	1	OP	D6	1	OP	F6	1	OP	E6	1	OP	18	1
2	OP+1	ii	1	OP+1	ii	1	OP+1	hh	1	OP+1	ff	1	OP+1	E6	1
3				00dd	(00dd)	1	OP+2	ll	1	FFFF	--	1	OP+2	ff	1
4							hhll	(hhll)	1	X+ff	(X+ff)	1	FFFF	--	1
5													Y+ff	(Y+ff)	1

HC11_143.PCB

Layer 1



Component Side

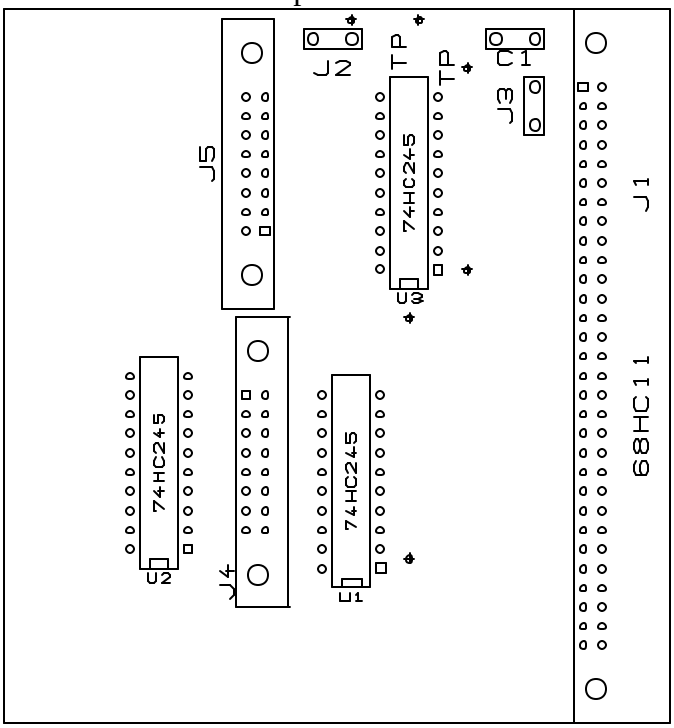


Figure ***Error! No text of specified style in document.****-3 HC11-143 Port Access Board*

BUFFALO MONITOR QUICK USERS REFERENCE

M68HC11 EVB

Command Line Format
>command [parameters] <CR>
where:

>	EVB/BUFFALO monitor prompt
command	EVB/BUFFALO command
[parameters]	required and/or optional parameters
<CR>	"RETURN" or "ENTER" key

Note: **Bolded** text in examples must be entered by the user.

Number Base Convention
All numerical data is assumed to be in HEX. "$" or "%" symbols in front of numbers are not allowed.

Editing
CTRL-H == backspace on command line
<BACKSPACE> == abort command
<SPACE> == continue modifying data on this line

EVB/BUFFALO Commands (incomplete listing)
ASM On-board Assembler/Disassembler

ASM address

Displays assembly language for specified address. User may change the instruction and/or data. Labels may not be used.

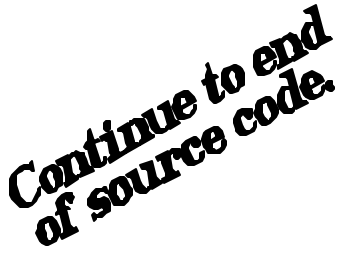
Use <CR> to advance to next line of code.

Use <BACKSPACE> to abort on-board assembly.

Example:

>**ASM C000**
C000 ??? ???
>**CLRA <CR>**
C001 ??? ???
>**LDAB D001 <CR>**
C004 ???
>**CLR D002 <CR>**
C007 ???

>**CLR D003 <CR>**
C00A ??? ???
>**CMPB #00 <CR>**
C00C ???
>**BGT C011 <CR>**
C00E ???
>**<BACKSPACE>**
>



G Go or execute command
G address

Begins execution of a program at the specified address. The program must provide a jump instruction to get back to the BUFFALO prompt.

Example:

>**G C000**

HELP Help display for monitor commands

HELP

LOAD T Download an assembled file from a PC

LOAD T
Used for downloading files assembled using AS11 from a PC.

MD Memory Display: Display memory to terminal screen.
MD address_start [address_stop]

Displays memory 16 bytes per line. Display will begin on an even 16 byte memory boundary. If no address_stop is given, 9 lines will be displayed. If address_stop is less than address_start, 1 line will be displayed.

Example:

>**MD D000 D010**

D000 04 03 00 0C 55 55 55 55 55 55 55 55 55 55 55 55...

>**MD D003 1**

D000 04 03 00 0C 55 55 55 55 55 55 55 55 55 55 55 55...

MM Memory Modify: Display and modify memory contents.
MM address

Displays memory and gives the user a chance to modify it.

Use the <SPACE> key to advance one byte. Use CTRL-H backup one byte. Use <CR> to return to the BUFFALO prompt.

Examples:

>MM D000
D000 55 **04<CR>**
>
>MM D000
D000 55 **04** 55 **03<CR>**
>

RM Register Modify: Display and modify 68HC11 registers.
RM [p,y,x,a,b,c,s]

Displays the contents of the 68HC11's registers. Also gives the user a chance to modify them.

Registers include: P - program counter
Y - index register Y
X - index register X
A - accumulator A
B - accumulator B
C - condition codes (flags)
S - stack pointer.

Examples:
>**RM**
P-C007 Y-7982 X-FF00 A-04 B-00 C-C0 S-0068
P-C007 **C000 <SPACEBAR>**
Y-7982 **6000 <SPACEBAR>**
X-FF00 **7000 <SPACEBAR>**
A-04 **05 <CR>**
>

>**RM A**
P-C007 Y-7982 X-FF00 A-04 B-00 C-C0 S-0068
A-04 **05 <CR>**

T Trace Instructions
T [n]

The trace instruction allows the user to execute a program n instructions a time. The user must set the program counter to the correct starting address before using the trace command. The machine code for the instruction will be displayed along with the registers after each instruction.

Examples:

>**T 1**

Op-C1
P-C007 Y-7982 X-FF00 A-04 B-00 C-C0 S-0068

AS11 TOP-10 BOO-BOOS

(68HC11) Assembler

This is a list of the top 10+ reasons a 68HC11 assembly language program will not run correctly. Many programs will compile and do something, but not what you want.

1. Missing a "#". Motorola defaults to direct or extended addressing mode. "#" specifies *immediate* addressing mode.

2. Using signed branches on unsigned data. AS11 has both signed & unsigned conditional branches. Ex: bgt (branch greater than) verses bhi (branch if higher).

3. Flow chart was drawn after the code was written. The program doesn't do what you wanted it to do because you didn't really know what you wanted it to do and when to do it when you started. Flow charts can be skipped for simple programs, but **do** help for larger problems. Think before you type. Programming ≠ Coding.

4. Missing a "$". AS11 defaults to decimal. The "$" is needed when using hex data with AS11. BUFFALO only allows HEX. No "$" is needed. No "$", or any other prefix can be entered.

5. Loading a 16-bit value into an 8-bit location. (or vice-versa)

6. Improper ending for embedded code. Use (in)finite loop or jump to monitor.

7. Motorola: Little Endian -- On reads: high byte first, low byte second.

8. Saving and Restoring registers from stack on subroutine calls. Pushes & Pulls.

9. Memory usage: Address space, ROM, RAM, r/w bits

10. high byte AccD == AccA, low byte AccD == AccB

11. Extra "$", or Extra "#"

12. System initialization: variables, i/o, stack, ...

13. EVB PortE, bit 0, must be logic 0 on power-up to boot BUFFALO. Otherwise boots at E^2.

14. Memory conflict with BUFFALO monitor. BUFFALO uses Page 0 memory locations: $36 to $FF

AS11.EXE REFERENCE

The IBM PC - to - 68HC11 Cross-Assembler

GENERAL

The AS11.EXE cross-assembler runs on an IBM PC compatible (80x86 microprocessor) computer, but produces source code for the 68HC11 microprocessor. Command line arguments specify the filenames to assemble.

The assemblers accept options from the command line to be included in the assembly. These options are the following:

l	enable output listing.
nol	disable output listing (default).
cre	generate cross reference table.
s	generate a symbol table.
c	enable cycle count.
noc	disable cycle count.

The command line looks like this :

>as11 file1 file2 ... [- option1 option2 ...]

Example:

>as11 file.asm -l cre >file.lst

This command assembles file "file.asm" with an output listing and a cross reference table. The output listing will be sent to a file named "file.lst" using DOS indirection.

The `S1' formatted object file is placed in file "file.S19", the listing and error messages are written to the standard output. If multiple files are assembled, the "S1" file will be placed under the first file's name.S19.

The listing file contains the address and bytes assembled for each line of input followed by the original input line (unchanged, but moved over to the right some). If an input line causes more than 6 bytes to be output (e.g. a long FCC directive), additional bytes (up to 64) are listed on succeding lines with no address preceding them.

The "file.s19" file can be uploaded from a PC to the 68HC11EVB, or an EPROM can be programmed using the Allpro Device Programmer.

Equates cause the value of the expression to replace the address field in the listing. Equates that have forward references cause Phasing Errors in Pass 2.

EXPRESSIONS

Expressions may consist of symbols, constants or the character '*' (denoting the current value of the program counter) joined together by one of the operators: + - * / % & | ^. The operators are the same as in the c programming language.

+ add
- subtract
* multiply
/ divide
% remainder after division
& bitwise and
| bitwise or
^ bitwise exclusive-or

Expressions are evaluated left to right and there is no provision for parenthesized expressions. Arithmetic is carried out in signed twos-complement integer precision (16 bits on the IBM PC). Constants are constructed with the same syntax as the Motorola MSDOS assembler:

' followed by ASCII character **TEXT FCC 'Hello!**
$ followed by hexadecimal constant **ORG $0000**
@ followed by octal constant
% followed by binary constant **FCB %01001111**
digit decimal constant **LDAA 100**

ERRORS

Error diagnostics are placed in the listing file just before the line containing the error. Format of the error line is:

Line_number: Description of error
or
Line_number: Warning --- Description of error

Errors of the first type in pass one cause cancellation of pass two. Warnings do not cause cancellation of pass two but should cause you to wonder where they came from.

Error messages are meant to be self-explanatory.

If more than one file is being assembled, the file name precedes the error:

File_name,Line_number: Description of error

Finally, some errors are classed as fatal and cause an immediate termination of the assembly. Generally these errors occur when a temporary file cannot be created or is lost during the assembly. Consult your local guru if this happens.

DIFFERENCES
For indexed addressing, the comma is required before the register; `inc x' and `inc ,x' are not the same.

Macros are not supported. (try M4 or M6)

The force size operators ('>' and '<') are implemented for all assemblers.

PSUEDO-OP-CODES
The only pseudo-ops supported are:

ORG define starting address for following code
FCC Form Constant Character; define text strings
FCB Form Constant Byte; define byte constants
FDB Form Double Byte; define a two byte constant
EQU Equate; substitute one text string for another
RMB Reserve memory space for variables
BSZ Block Store Zeros; fill memory with so many zeros
ZMB Zero Memory Bytes; same as BSZ
FILL fill memory with data; FILL value, #bytes
PAGE start new page in listing
OPT The OPT pseudo-op allows the following operands:
nol Turn off output listing
l Turn on output listing (default)
noc Disable cycle counts in listing (default)
c Enable cycle counts in listing (clear total cycles)
contc Re-enable cycle counts (don't clear total cycles)
cre Enable printing of a cross reference table
s generate a symbol table

Some of the more common pseudo-ops are not present:

SPC Use blank lines instead
END The assembly ends when there is no more input
TTL use `pr' to get headings and page numbers
NAM[E] Did you ever use this one anyway?

The above 4 pseudo-ops are recognized, but ignored.

DETAILS

Symbol: A string of characters. The first character may not be a digit. The string of characters may be from the set:

[a-z][A-Z]_.[0-9]$

(. and _ count as non-digits). The `$' counts as a digit to avoid confusionwith hexadecimal constants. All characters of a symbol are significant, with upper and lower case characters being distinct. The maximum number of characters in a symbol is currently set at 15.

The symbol table has room for at least 2000 symbols of length 8 characters or less.

Label: A symbol starting in the first column is a label and may optionally be ended with a ':'. A label may appear on a line by itself and is then interpreted as:

Label EQU ***** (defines value of label as current address)

Mnemonic: A symbol from the microprocessor's instruction set preceded by at least one whitespace character. Upper case characters in this field are converted to lower case before being checked as a legal mnemonic. Thus `nop', `NOP' and even `NoP' are recognized as the same mnemonic.

Note that register names that sometimes appear at the end of a mnemonic (e.g. nega or stu) must not be separated by any whitespace characters. Thus `clra' means clear accumulator A, but that `clr a' means clear memory location `a'.

Operand: Follows mnemonic, separated by at least one whitespace character. The contents of the operand field is interpreted by each instruction.

Whitespace: A blank or a tab

Comment: Any text after all operands for a given mnemonic have been processed or, a line beginning with '*' up to the end of line or, an empty line.

Continuations: If a line ends with a backslash (\) then the next line is fetched and added to the end of the first line. This continues until a line is seen which doesn't end in \ or until MAXBUF characters have been collected (MAXBUF >= 256).

68HC11 PARALLEL INPUT & OUTPUT

HANDSHAKE I/O SUBSYSTEM

Register	Offset
PIOC	$1002
PORTC	$1003
PORTB	$1004
PORTCL	$1005
DDRC	$1007

SIMPLE I/O

Port B is output only.

Port C bits are input or output as determined by DDRC.
1 = output, 0 = input.

SIMPLE STROBE (HANDSHAKE)

Port B == strobed output with STRB

Port C == latching input port with STRA

Independent of each other.

PORT B CODE

```
PORTB   equ$1004      define PortB address
PIOC    equ$1002      define PIOC address

        ldaa    #%00000001   bit4: HNDS=0 (Simple Strobe Mode)
        staaPIOC             bit0: INVB=1 (Active High Pulse)
        staa PORTB           write to PortB
```

PORT B TIMING

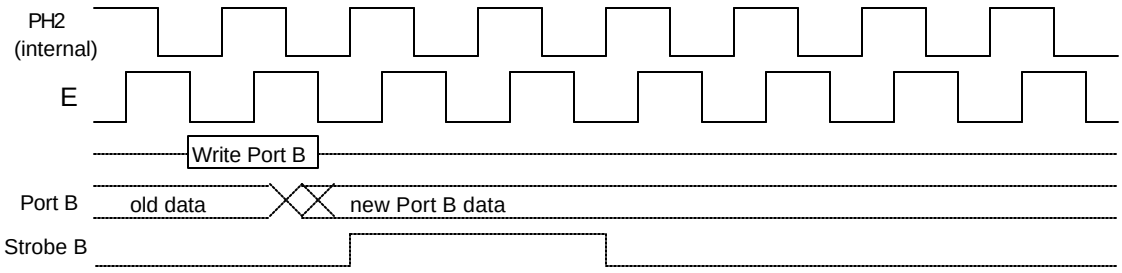


Figure 9-4 Port B Timing Waveforms

Note: INVB control bit in PIOC allows for STRB to be inverted.

PORT C CODE

```
* sample code for simple strobed input (code not tested)
        org $c000   org at USER 8K ram
_PORTCL equ$05      def PORTCL offset
_PIOC equ$02        def PIOC offset
REGBASEequ$1000   def IO Register base addr
BIT7        equ%10000000      bit 7 mask
BIT4        equ%00010000      bit 4 mask
BIT1        equ%00000010      bit 1 mask
STAF_MSK    equBIT7           def STAF mask
HNDS_MSK    equBIT4
EGA_MSK     equBIT1
PIO_MODE    equEGA_MSK | (HNDS_MSK ^ $ff)
*             EGA = 1 ---> +edge latching
*             HNDS = 0 --> simple handshake mode

* Set Parallel I/O Mode
        ldx #REGBASE
        ldaa     #PIO_MODE
        staa_PIOC,x

* Wait until data has been strobed into Port C
* (Loop while STAF (bit 7 of PIOC) is clear.)
LOOP      brclr _PIOC,x STAF_MSK LOOP

        ldaa     _PORTCL,x    read latched Port C data

* Note: Read of latched Port C resets STAF.
```

PORT C TIMING

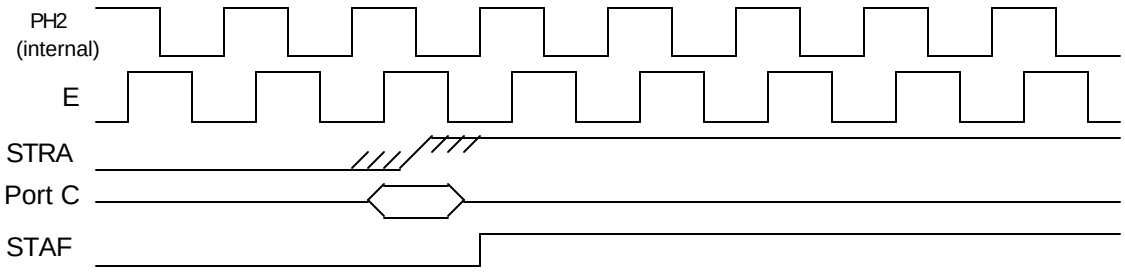


Figure 9-5 Port C Timing Waveform

FULL-INPUT HANDSHAKE MODE

Port C: Input port accessed through PORTCL register.
Strobe B: Output acting as a "ready" signal to external device.
Strobe A: Edge sensitive latching input.

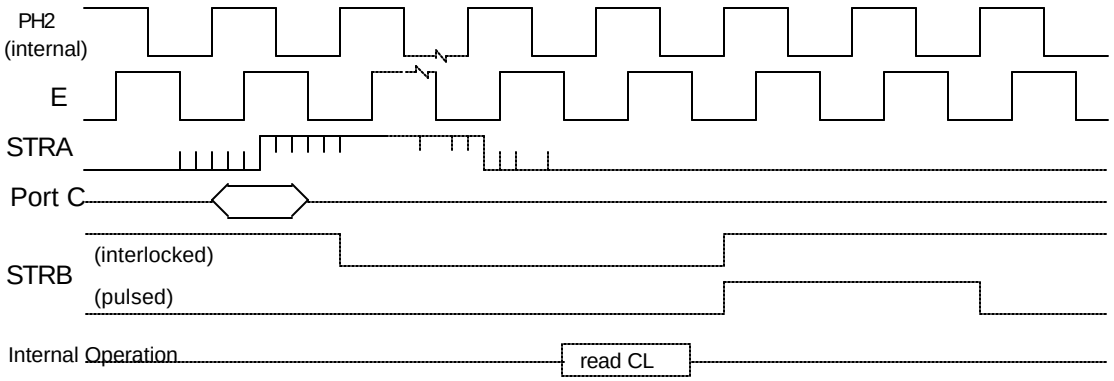


Figure 9-6 Parallel I/O Full-Input Handshake Mode

FULL-OUTPUT HANDSHAKE MODE

Port C: Output port (normal or 3-state (DDRC control).
Strobe B: Output "ready" signal that data is available.
Strobe A: Input acknowledge signal.

See Figure 7-26 (page 7.41) of HC11 Reference Manual (PINK).

CENTRONICS PARALLEL PRINTER PORT

- 8-bit parallel interface
- 1K char/sec maximum

- TTL logic levels
- 3-line handshaking

According to the PANASONIC KX-P1091i manual:
"When the printer is processing data, the BUSY signal is high. The printer will not accept new data from the computer. After the processing is completed, the BUSY signal goes low. (The BUSY signal is also high when the printer is OFF LINE). When this occurs, the ACK* signal goes low indicating to the computer that the data has been processed and the printer is ready to accept more data. This handshaking routine occurs each time a character is sent to the printer."

Table 9-6 Standard 36-pin Centronics connector

Signal Pin	Return side pin	Signal	Direction
1	19	STROBE*	INPUT
2	20	DATA1	INPUT
3	21	DATA2	INPUT
4	22	DATA3	INPUT
5	23	DATA4	INPUT
6	24	DATA5	INPUT
7	25	DATA6	INPUT
8	26	DATA7	INPUT
9	27	DATA8	INPUT
10	28	ACK*	OUTPUT
11	29	BUSY	OUTPUT
12		PO	OUTPUT
13		SLCT	OUTPUT
14		AFXT*	
15			
16		SG	
17		FG	
18		+5V	OUTPUT
31		PRIME*	INPUT
32		ERROR*	OUTPUT
33		SG	
34			
35			
36			

Table 9-7 IBM PC Printer connector (25-pin)

Pin	Signal	Pin	Direction
1	STROBE*	10	ACK*
2	DATA1	11	BUSY*
3	DATA2	12	PO
4	DATA3	13	SLCT
5	DATA4	14	AFXT*
6	DATA5	15	ERROR*
7	DATA6	16	PRIME*
8	DATA7	17	SLCT IN
9	DATA8	18	FG
19-25	SG, signal grounds		

SIGNAL DEFINITIONS

STROBE* -edge trigger (clock) input to printer.
DATA1 - DATA8 8-bit data
ACK* Acknowledge signal that printer has data.
BUSY Printer status signal. BUSY = 1 when:

- Receive buffer is full.
- Printer is processing data.
- Printer is OFF LINE.
- Printer is in an error condition.

PO PO = 1 when printer is out of paper.
SLCT SLCT = 1 when printer is ON LINE.
AFXT*When AFXT* = 0, a line feed <LF> is added to each <CR>.
SG Signal Ground.
FG Frame Ground.
+5V +5V reference signal only.
PRIME* Reset input to printer.
ERROR* Error or "fault" output status signal.

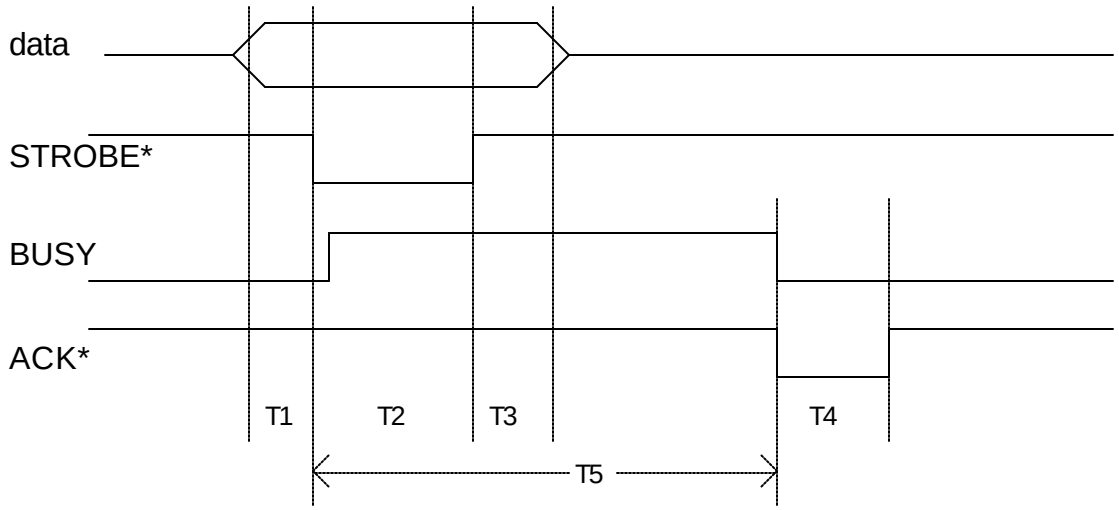


Figure **Error! No text of specified style in document.**-7 Centronics Parallel Interface Timing Diagram

*Table **Error! No text of specified style in document.**-8 Centronics Timing Specifications*

	Description	Time
T1	Data Setup	0.5 us (min)
T2	Strobe Pulse Width	1.0 us (min)
T3	Data Hold	0.5 us (min)
T4	Ack Pulse Width	5.0 us (max)
T5	Response Time	1.0 ms (typ)*

* or less when buffer not full.

68HC11 TIMING FUNCTIONS

68HC11 Timing Features

- 16-bit Main Timer
- Three Input Capture Registers (4 on E9)
- Five Output Compare Registers
- Real-Time Clock with Interrupt
- Pulse Accumulator
- COP Watchdog Timer
- Consistant Machine Cycle Times

Table ***Error! No text of specified style in document.****-9 Timer Registers*

Register	Offset	Description	BIT(S)
CFORC	$0B	Compare Force	FOC1 – FOC5
CONFIG	$3F	Configuration Control	NOCOP
COPRST	$3A	COP Timer Arm/Reset	all, $55, $AA
OC1D	$0D	Output Compare 1 Data	OC1D7 – OC1D3
OC1M	$0C	Output Compare 1 Mask	OC1M7 – OC1M3
OPTION	$39	System Option	CR1, CR0
PACNT	$27	Pulse Accumulator Count	all
PACTL	$26	Pulse Accumulator Control	DDRA7, PAEN, PAMOD, PEDGE, RTR1, RTR0
TCNT	$0E	Timer Counter Register	(16-bit)
TCTL1	$20	Timer Control 1	OM2 – OM5, OL2 – OL5
TCTL2	$22	Timer Control 2	EDG1A,B – EDG3A,B
TFLG1	$23	Timer Interrupt Flag 1	OC1F – OC5F, IC1F – IC3F
TFLG2	$25	Timer Interrupt Flag 2	TOF, RTIF, PAOVF, PAIF
TIC1	$10	Input Capture Register 1	(16-bit)
TIC2	$12	Input Capture Register 2	(16-bit)
TIC3	$14	Input Capture Register 3	(16-bit)
TMSK1	$22	Timer Interrupt Mask 1	OC1I – OC5I, IC1I – IC3I
TMSK2	$24	Timer Interrupt Mask 2	TOI, RTII, PAOVI, PAII, PR1, PR0
TOC1	$16	Output Compare 1	(16-bit)
TOC2	$18	Output Compare 2	(16-bit)
TOC3	$1A	Output Compare 3	(16-bit)
TOC4	$1C	Output Compare 4	(16-bit)
TOC5	$1E	Output Compare 5	(16-bit)

Table ***Error! No text of specified style in document.****-10*

Register	BIT	Pos	Function	Def
CFORC	FOC1	7	Write "1" to Force a Compare	0
	FOC2	6		0
	FOC3	5		0
	FOC4	4		0
	FOC5	3		0
CONFIG	NOCOP	2	0 = COP Watchdog enabled (forces Reset on timeout)	U
COPRST	all	-	Write $55, then $AA to Reset COP	U
OC1D	OC1D7	7	Data written to Port A, bit 7, on an OC1 compare (if OC1M7 is set)	U
	OC1D6	6		U
	OC1D5	5		U
	OC1D4	4		U
	OC1D3	3		U
OC1M	OC1M7	7	If OC1M7 is set, data in OC1D7 is written to Port A on OC1 compares	0
	OC1M6	6		0
	OC1M5	5		0
	OC1M4	4		0
	OC1M3	3		0
OPTION	CR1, CR0	1, 0	COP Rate: 00 == ÷1, 01 == ÷4 10 == ÷16, 11 == ÷64,	0,0
PACNT	all	all	Pulse Accumulator Count	U
PACTL	DDRA7	7	Data Direction Bit, Port A, Bit 7	0
	PAEN	6	Pulse Accumulator Enable = 1	0
	PAMOD	5	PA Mode, 0 = Event Counter 1 = Gate Time Accumulation	0
	PEDGE	4	PA Edge Control 0 = -Edge, High-Level Enable 1 = +Edge, Low-Level Enable	0
	RTR1, RTR0	1,0	RTI Interrupt Rate, 00 = $\div 2^{13}$ 01 = $\div 2^{14}$, 10 = $\div 2^{15}$, 11 = $\div 2^{16}$	0
TCTL1	OM2, OL2	7,6	Output Compare Pin Action 00 == no action 01 == Toggle OC2 output pin 10 == Clear OC2 output pin 11 == Set OC2 output pin	0,0
	OM3, OL3	5,4		0,0

	OM4, OL4	3,2		0,0
	OM5, OL5	1,0		0,0
TCTL2	EDG1B, EDG1A	5,4	Input Capture Edge Control 00 == Capture Disabled 01 == Rising Edges Only 10 == Falling Edges Only 11 == Any Edge	0,0
	EDG2B, EDG2A			0,0
	EDG3B, EDG3A			0,0
TFLG1	OC1F	7	Output Compare 1 Flag 1 ==> Compare Occured	0
	OC2F	6		0
	OC3F	5		0
	OC4F	4		0
	OC5F	3		0
	IC1F	2	Input Capture 1 Flag 1 == > Capture Occured	0
	IC2F	1		0
	IC3F	0		0
TFLG2	TOF	7	Timer Overflow Flag	0
	RTIF	6	Real Time Interrupt Flag	0
	PAOVF	5	PA Overflow Flag	0
	PAIF	4	PA Input Edge Detect Flag	0
TMSK1	OC1I	7	Output Compare 1 Interrupt Enable 1 == Enabled	0
	OC2I	6		0
	OC3I	5		0
	OC4I	4		0
	OC5I	3		0
	IC1I	2	Input Capture 1 Interrupt Enable 1 == Enabled	0
	IC2I	1		0
	IC3I	0		0
TMSK2	TOI	7	Timer Overflow Interrupt Enable	0
	RTII	6	Real Time Interrupt Enable	0
	PAOVI	5	PA Overflow Interrupt Enable	0
	PAII	4	PA Edge Detect Interrupt Enable	0
	PR1, PR0	1,0	Timer Prescaler Select Bits 00 == ÷1, 01 == ÷4, 10 == ÷8, 11 == ÷16	0,0

68HC11 ANALOG-TO-DIGITAL CONVERTER

68HC11 ADC FEATURES

- 8-bit Analog-to-Digital Converter
- 8-channel multiplexed input (8 inputs, 1 ADC)
- Successive Approximation Method
- Error +/- 1 LSB
- Reference Inputs
- Thirty-two (32) MCU E clocks per conversion
- E clock or internal RC oscillator

Table 9-11Registers

Register	**Offset**	**Description**	**BIT(S)**
OPTION	$39	HC11 Option Register	ADPU, CSEL, DLY
ADCTL	$30	ADC Control/Status	CCF, SCAN, MULT, CD, CC, CB, CA
ADR1	$31	ADC Result Register 1	all
ADR2	$32	ADC Result Register 2	all
ADR3	$33	ADC Result Register 3	all
ADR4	$34	ADC Result Register 4	all

Register	**BIT**	**Pos**	**Function**	**Def**
OPTION	ADPU	7	ADC Power Up, 1 == ON	0
	CSEL	6	ADC Clock Select, 0 == E, 1== Internal RC	0
	DLY	4	Oscillator Delay, 1==delay after STOP to allow oscillator to stabilize	1
ADCTL	CCF	7	Conversions Complete Flag, 1== done, Cleared on write to ADCTL	0
	SCAN	5	Continuous Scan Control, 0 == one set of 4 conversions, 1 == cycle of conversions	U
	MULT	4	Multiple Channel Select, 0 == 4 conversion on 1channel, CD-CA specify channel 1 == convert each channel of a 4 ch. group specified by CD, CC	U
	CD,CC, CB,CA	3,2,1 ,0	ADC Channel number, 0 - 7 are PortE inputs	U

HARDWARE INTERFACE

ANALOG DATA INPUTS

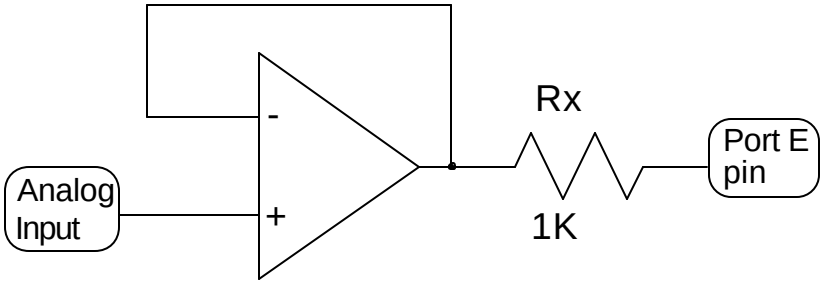


Figure 9-8 Analog Input Protection Buffer

REFERENCE VOLTAGE INPUTS

VRH High Reference Voltage typical VRH = 5.0V
VRL Low Reference Voltage VLH = 0.0V

The ADC can be operated with: VRH - VLH > 2.5 V

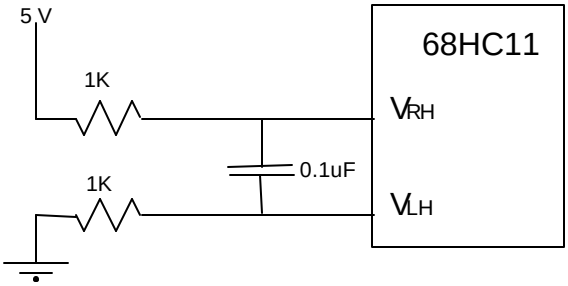


Figure 9-9: ADC Voltage Reference Inputs

SINGLE CHANNEL OPERATION, ONE TIME

MULT = 0, SCAN = 0

- Channel specified by CD, CC, CB, CA is converted 4 times.
- Four results are stored automatically in: ADR1, ADR2, ADR3, ADR4
- A write to ADCTL starts each conversion and transfer set.

Single Channel Operation, Continuous

MULT = 0, SCAN = 1

- Channel specified by CD, CC, CB, CA is continuously converted until SCAN bit is changed.
- Results are stored automatically in: ADR1, ADR2, ADR3, ADR4, then back to writing over ADR1,...
- A write to ADCTL starts the conversion and transfer process.

MULTIPLE CHANNEL OPERATION, ONE SET

MULT = 1, SCAN = 0

- Four Channels specified by CD, CC are each converted once.
- Four results are stored automatically in: ADR1, ADR2, ADR3, ADR4
- A write to ADCTL starts each conversion and transfer set.

MULTIPLE CHANNEL OPERATION, CONTINUOUS

MULT = 1, SCAN = 1

- Four Channels specified by CD, CC are continuously converted until SCAN bit is changed.
- Results are stored automatically in: ADR1, ADR2, ADR3, ADR4, then back to writing over ADR1,...
- A write to ADCTL starts the conversion and transfer process.

SAMPLING AND CONVERSION SPEED

No hardware level support is available for automating an ADC sampling process. The ADCTL register has a conversion complete flag (CCF). This flag is set after the first 4 conversions. It is only cleared by a write to ADCTL. It is not automatically cleared during continuous scan operations.

Software sampling support is available using the Real-Time Clock or Main Timer on the HC11. Each has interrupt support.

The HC11 successive approximation ADC requires 32 E-clocks per 8-bit conversion plus initialization time. The fastest conversion available is when ADR1 is read 32 E-clocks after a write to ADCTL.

FILTERING, AVERAGING, AND DATA PROCESSING

It is recommended to low-pass or band-pass analog signals before being input to the HC11. Averaging, and/or other digital signal processing may be performed with software in the HC11 to reduce noise, increase resolution, or achieve some other function.

STOP AND WAIT MODES

When STOP or WAIT modes are entered in the HC11, any conversion process is suspended at that point. Upon "Normal" operation, the channel(s) will be resampled, and conversion continued.

During STOP mode, the E clock oscillator circuit is turned off. Oscillators require some amount of time to stabilize. An unstable clock will cause errors in ADC conversions (also other timer functions, and EEPROM programming are affected). Is the DLY bit in the

OPTION register is set prior to the STOP or WAIT, the HC11 will delay all operations (approximately 4,000 E-clocks) after exit from STOP.

Table 9-12 Interrupt Vector Assignments

Vector Address	Interrupt Source	CC Mask	Local Mask Bit	Local Mask Register
FFC0, C1	Reserved			
FFD4, D5	Reserved			
FFD6,D7	SCI serial system	I bit	see below	SCCR2
FFD8,D9	SPI Serial Transfer Complete	I bit	SPIE	SPCR
FFDA,DB	Pulse Accumulator Input Edge	I bit	PAII	TMSK2
FFDC,DD	Pulse Accumulator Overflow	I bit	PAOVI	TMSK2
FFDE,DF	Timer Overflow	I bit	TOI	TMSK2
FFE0,E1	Timer Output Compare 5	I bit	OC5I	TMSK1
FFE2,E3	Timer Output Compare 4	I bit	OC4I	TMSK1
FFE4,E5	Timer Output Compare 3	I bit	OC3I	TMSK1
FFE6,E7	Timer Output Compare 2	I bit	OC2I	TMSK1
FFE8,E9	Timer Output Compare 1	I bit	OC1I	TMSK1
FFEA,EB	Timer Input Capture 3	I bit	IC3I	TMSK1
FFEC,ED	Timer Input Capture 2	I bit	IC2I	TMSK1
FFEE,EF	Timer Input Capture 1	I bit	IC1I	TMSK1
FFF0,F1	Real Time Interrupt	I bit	RTII	TMSK2
FFF2,F3	IRQ* external pin	I bit	none	
"	parallel I/O	I bit	STAI	PIOC
FFF4,F5	XIRQ* pin	X bit	none	
FFF6,F7	SWI	none	none	
FFF8,F9	Illegal Opcode Trap	none	none	
FFFA,FB	COP Failure (Reset)	none	NOCOP	CONFIG
FFFC,FD	COP Clock Monitor Fail (Reset)	none	CME	OPTION
FFFE,FF	RESET*	none	none	

Table 9-13 SCI Serial System Interrupts

Interrupt Cause	Local Mask Bit	Locak Mask Register	Flag Bit	Flag Register
Receive Data Register Full	RIE	SCCR2	RDRF	SCSR
Receiver Overrun	RIE	SCCR2	OR	SCSR
Idle Line Detect	ILIE	SCCR2	IDLE	SCSR
Transmit Data Register Empty	TIE	SCCR2	TDRE	SCSR
Transmit Complete	TCIE	SCCR2	TC	SCSR

COUNT.A11

```
* COUNT.A11  Count pulses at an input.
*          Two digit bcd output.
*    Bruce Hoeppner     11/10/92
*    Bounceless input at bit 0 of Port E
*    Output to Port B
PORTB     equ $1004
          org  $C000    ;origin in user RAM
* Initialize
MAIN      clra
            staa PORTB

*  Loop while input = 0
WAIT0     ldab $100a    ;read input
          andb #$01     ;mask off 7 msbs
          bne WAIT0
*  Loop while input = 1
WAIT1     ldab $100a    ;read input
          andb #$01     ;mask off 7 msbs
          beq WAIT1
          adda #$01     ;increment AccA
          daa           ;adjust for bcd
          staa $1004    ;write to Port B
          jmp WAIT0
```

COUNT.LST

```
 Assembling count.a11
0001                    * COUNT.A11  Count pulses at an input.
0002                    *          Two digit bcd output.
0003                    *   Bruce Hoeppner     11/10/92
0004
0005                    *   Bounceless input at bit 0 of Port E
0006                    *   Output to Port B
0007 1004                  PORTB     equ $1004
0008
0009 c000                        org  $C000    ;origin in user RAM
0010                    * Initialize
0011 c000 4f               MAIN      clra
0012 c001 b7 10 04                   staa PORTB
0013
0014                    *  Loop while input = 0
0015 c004 f6 10 0a           WAIT0     ldab $100a    ;read input
0016 c007 c4 01                       andb #$01     ;mask off 7 msbs
0017 c009 26 f9                       bne WAIT0
0018
0019                    *  Loop while input = 1
0020 c00b f6 10 0a           WAIT1     ldab $100a    ;read input
0021 c00e c4 01                       andb #$01     ;mask off 7 msbs
0022 c010 27 f9                       beq WAIT1
0023
0024 c012 8b 01                       adda #$01     ;increment AccA
0025 c014 19                         daa          ;adjust for bcd
0026
0027 c015 b7 10 04                    staa $1004    ;write to Port B
0028 c018 7e c0 04                    jmp WAIT0


MAIN            c000 *0011
PORTB            1004 *0007 0012
WAIT0           c004 *0015 0017 0028
WAIT1           c00b *0020 0022


Number of errors 0
Number of warnings 0
```

COUNT_BR.LST

```
0001                    * COUNT_BR.A11
0002                    *  Count pulses at an input.
0003                    *  Two digit BCD output.
0004                    *  Bruce Hoeppner    11/10/92
0005                    *            06/19/94 brset, brclr
0006                    * NOTE: modifications not tested
0007                    * NOTE: bset, bclr, brset, brclr instructions
0008                    *  work for direct (page 0) or indexed mode
0009
0010                    *  Bounceless input at bit0 of PORTE
0011                    *  Output to PORTB
0012                    * Constants - Hardware dependent
0013 1000                 REGBASEequ $1000
0014 0000                 _PORTA   equ $00
0015 0004                 _PORTB   equ $04
0016 000a                 _PORTE   equ $0a
0017 0001                 BIT0    equ %00000001
0018                    ***********************************************
0019
0020
0021 c000                 org $C000  ;origin in user RAM
0022                    * Initialize
0023 c000 4f               MAIN clra
0024 c001 ce 10 00            ldx #REGBASE
0025 c004 a7 04               staa _PORTB,x     ;Initialize Output
0026
0027                    * Looking for a POSITIVE EDGE
0028                    * First: Loop until input = 0 (loop while bit=1)
0029 c006 1e 0a 01 fc         WAIT0brset _PORTE,x BIT0 WAIT0
0030
0031                    * above one line replaces following three,
0032                    * and it is more clear
0033                    *WAIT0  ldab $100a ;read input
0034                    *  andb #$01 ;mask off 7 MSBs
0035                    *  bne WAIT0
0036
0037                    * Then: Loop until input = 1 (loop while bit=0)
0038 c00a 1f 0a 01 fc         WAIT1 brclr _PORTE,x BIT0 WAIT1
0039
0040                    * above one line replaces following three,
0041                    * and it is more clear
0042                    *WAIT1  ldab $100A;read input
0043                    *  andb #$01 ;mask off 7 msbs
0044                    *  beq WAIT1
0045
```

```
0046                    * Only get here if after a +edge at specified bit
0047
0048                    *Increment Count (Use adda so daa works)
0049 c00e 8b 01                 adda #$01 ;increment AccA
0050 c010 19                    daa       ;adjust for BCD
0051
0052 c011 a7 04                 staa _PORTB,x     ;write to PORTB
0053 c013 7e c0 06              jmp WAIT0 B,x     ;write to PORTB
BIT0       0001 *0017 0029 0038
MAIN       c000 *0023
REGBASE    1000 *0013 0024
WAIT0      c006 *0029 0029 0053
WAIT1      c00a *0038 0038
_PORTA     0000 *0014
_PORTB     0004 *0015 0025 0052
_PORTE     000a *0016 0029 0038
```

DELAY1M.LST

```
0001                    ********************************************
0002                    * DELAY1M.A11     N*1ms Delay Routine
0003                    * AUTHORS      DATE COMMENTS
0004                    * JACOBSON/SEVCIK 2/26/90    VERSION 1.0
0005                    *
0006                    * DESCRIPTION
0007                    *  THIS ROUTINE GENERATES INTERNAL DELAYS IN
0008                    *  MULTIPLES OF ONE (1) MILLI-SECOND. THE
0009                    *  USER ENTERS THE DURATION OF THE DELAY
0010                    *  (ms) INTO THE X REGISTER
0011                    *
0012                    * PARAMETERS
0013                    *  - X REGISTER CONTAINS DURATION (ms)
0014                    *  - INTERRUPTS ARE NOT AFFECTED
0015                    *  - SHORTEST DELAY IS 1 ms (X =1)
0016                    *  - RESOLUTION IS 1 ms
0017                    *  - MAXIMUM DELAY IS 655,36 ms (X = 0)
0018                    ************************************************
0019                    * TEST ROUTINE
0020                    *   user must enter a value into X, then run
0021 000c                  ORG $COOO
0022                    * change operand of next instruction to change
0023                    * the delay
0024 000c ce 00 64          TEST    ldx #100    ;FOR 100 ms DELAY
0025 000f bd 00 15           JSR DELAY1M    ;CALL ROUTINE TO TEST
0026 0012 7e e0 00           JMP $E000;JUMP TO BUFFALO WHEN DONE
0027                    ************************************************
0028                    * DELAY1M N*1ms subroutine
0029                    ************************************************
0030 0015 36             DELAY1M:   PSHA
0031                    * Primary Loop
0032 0016 86 c7           DELWT2    LDAA #199    ;199 * 2ND LOOP = 1ms
0033 0018 01                NOP
0034 0019 4a             DELWT3DECA      ;SECONDARY LOOP = 1ms/199
0035 001a 01                NOP
0036 001b 21 fc             BRN DELWT3    ;BRANCH NEVER = 3 CYCLE NOP
0037 001d 26 fa             BNE DELWT3    ;CONTINUE UNTIL 199 --> 0
0038
0039 001f 09                DEX        ;# OF 1ms LOOPS
0040 0020 26 f4             BNE DELWT2    ;CONTINUE UNTIL IX = 0
0041
0042 0022 32                PULA
0043 0023 39                RTS
0044                    ********************************************
0045                    * END DELAY1M
```

```
0046                    ******************************************** *

DELAY1M    0015 *0030 0025
DELWT2     0016 *0032 0040
DELWT3     0019 *0034 0036 0037
TEST       000c *0024
```

DELAY5U.LST

```
0001                    ********************************************
0002                    * DELAY5U.A11
0003                    * AUTHORS      DATE COMMENTS
0004                    * JACOBSON/SEVCIK 2/26/90    VERSION 1.0
0005                    *
0006                    * DESCRIPTION
0007                    *  THIS ROUTINE GENERATES INTERNAL DELAYS IN
0008                    *  MULTIPLES OF FIVE (5) MICRO-SECONDS. THE
0009                    *  USER ENTERS A MULTIPLIER (16-BIT) INTO THE
0010                    *  X INDEX REGISTER WHICH DETERMINES THE NUMBER
0011                    *  OF FIVE MICRO-SECOND INTERVALS
0012                    *
0013                    * PARAMETERS
0014                    *  - X REGISTER CONTAINS MULTIPLIER
0015                    *  - INTERRUPTS ARE NOT AFFECTED
0016                    *  - SHORTEST DELAY IS 10 us (X < 3)
0017                    *  - RESOLUTION IS 5 us
0018                    *  - MAXIMUM DELAY IS 327680 us (X = 64K)
0019                    ********************************************
0020
0021 000c                 ORG $COOO
0022 000c ce 03 e8           ldx #1000  ;FOR TESTING
0023 000f bd 00 15           JSR DELAY5U    ;CALL ROUTINE TO TEST
0024 0012 7e e0 00           JMP $E000;JUMP TO BUFFALO WHEN DONE
0025 0015 09              DELAY5U:   DEX   ;CORRECT FOR JSR/RTS
0026 0016 09                 DEX   ;OVERHEAD
0027 0017 01                 NOP
0028 0018 01                 NOP
0029 0019 09              DELWT1:    DEX   ;DECREMENT MULTIPLIER
0030 001a 01                 NOP
0031 001b 01                 NOP
0032 001c 26 fb              BNE DELWT1
0033 001e 39                 RTS
0034                    ********************************************
0035                    * END DELAY5U
0036                    ********************************************
DELAY5U    0015 *0025 0023
DELWT1     0019 *0029 0032
```

INNOTOUT.A11

```
* INNOTOUT.A11  Read word, NOT it, Write it
*    Bruce Hoeppner     01 JAN 94
*
*    Read an 8-bit word from PortE.
*    Complement the word.
*    Write the word to PortB.
*    Loop Continuously.
*    Essentially this turns the HC11 into an
*    Octal Inverter.

* Definitions
PORTB     equ  $1004
*****************************************
* Load program into 8k user RAM
START     org  $C000

* Read 8-bit word from PortE into AccA
        ldaa $100a
* Complement the word.
        coma
* Write the word to PortB
        staa PORTB

* Jump back to beginning of program
        jmp  START

* End of INNOTOUT.A11
```

INNOTOUT.LST

```
 Assembling innotout.a11
0001                    * INNOTOUT.A11  Read word, NOT it, Write it
0002                    *    Bruce Hoeppner     01 JAN 94
0003                    *
0004                    *    Read an 8-bit word from PortE.
0005                    *    Complement the word.
0006                    *    Write the word to PortB.
0007                    *    Loop Continuously.
0008                    *    Essentially this turns the HC11 into an
0009                    *    Octal Inverter.
0010
0011                    * Definitions
0012 1004                  PORTB     equ  $1004
0013                    ****************************************
0014                    * Load program into 8k user RAM
0015 c000                  START     org  $C000
0016
0017                    * Read 8-bit word from PortE into AccA
0018 c000 b6 10 0a                 ldaa $100a
0019                    * Complement the word.
0020 c003 43                     coma
0021                    * Write the word to PortB
0022 c004 b7 10 04                 staa PORTB
0023
0024                    * Jump back to beginning of program
0025 c007 7e 00 00                 jmp  START
0026
0027                    * End of INNOTOUT.A11


Number of errors 0
Number of warnings 0
```

MULT4BIT.LST

```
 Assembling mult4bit.a11
0001                    * MULT4BIT.A11  Multiply using repeated addition
0002                    *          P = M x N
0003                    *   Bruce Hoeppner     10/7/89
0004
0005 c000                       org  $C000   ;origin in user RAM
0006
0007 c000 f6 10 0a                ldab $100a   ;load M & N into AccB
0008 c003 17                      tba          ;Copy B to A
0009 c004 c4 0f                   andb #$0f    ;Mask off M from N
0010 c006 44                      lsra
0011 c007 44                      lsra
0012 c008 44                      lsra
0013 c009 44                      lsra         ;Move M to 4 LSBs
0014 c00a 84 0f                   anda #$0f    ;Mask off N from M
0015 c00c b7 d0 00                staa $d000   ;Store accA in temp
0016 c00f 4f                      clra         ;clear accumulator A
0017
0018 c010 c1 00           multiply  cmpb #00     ;check for N = 0
0019 c012 2e 03                bgt  addem   ;branch to addem when
0020                    *               ;accB > 0
0021 c014 7e e0 0a              jmp  $e00a   ;when N = 0 you are done
0022                    *               ;go back to BUFFALO
0023
0024 c017 bb d0 00         addem     adda $d000   ;accA = accA + M
0025 c01a 5a                    decb         ;decrement accB
0026                    *               ;accB = accB - 1
0027                    *               ;N = N - 1
0028
0029 c01b b7 d0 01                staa $d001   ;store result P
0030
0031 c01e 20 f0                 bra  multiply
0032
0033 c020                       end          ;end of MULT.A11


addem           c017 *0024 0019
multiply        c010 *0018 0031


Number of errors 0
Number of warnings 0
```