MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 1230 December, 1988

Olympic Robot Building Manual

Edited by Anita Flynn

with contributions from

Colin Angle, Rodney Brooks, Jon Connell, Anita Flynn,
Ian Horswill, Maja Mataric, Henry Minsky, Peter Ning,
Paul Viola, and William Wells

The 1989 AI Lab Winter Olympics will take a slightly different twist this year from previous Olympiads. Although there will still be a dozen or so athletic competitions, the annual talent show finale will now be a display not of human talent, but of robot talent. Spurred on by the question, "Why aren't there more robots running around the AI Lab?", Olympic Robot Building is an attempt to teach everyone how to build a robot and get them started. Robot kits will be given out the last week of classes before the Christmas break and teams have until the Robot Talent Show, January 27th, to build a machine that intelligently connects perception to action.

There is no constraint on what can be built; participants are free to pick their own problems and solution implementations. As Olympic Robot Building is purposefully a talent show, there is no particular obstacle course to be traversed or specific feat to be demonstrated. The hope is that this format will promote creativity, freedom and imagination.

This manual provides a guide to overcoming all the practical problems in building things. What follows are tutorials on the components supplied in the kits: a microprocessor circuit 'brain', a variety of sensors and motors, a mechanical building block system, a complete software development environment, some example robots and a few tips on debugging and prototyping. Parts given out in the kits can be used, ignored or supplemented, as the kits are designed primarily to overcome the inertia of getting started.

If all goes well, then come February, there should be all kinds of new members running around the AI Lab!

Funding for the Robot Olympics was provided by the Systems Development Foundation.

Table of Contents

1. Implementing Your Imagination
   1.1 Goals
   1.2 Groundrules
   1.3 How to build stuff

2. The Motorola 6811 Microprocessor
   2.1 OLM2 and OLM2S microcontroller boards
   2.2 Using serial I/O
   2.3 Assembling the OLM2 board
   2.4 The OLM2S surface mount board

3. Sensors
   3.1 Pyroelectric motion sensors
   3.2 Microswitched touch sensors
   3.3 Photodiodes
   3.4 Near-infrared emitters and detectors
   3.5 Microphones
   3.6 Photocells
   3.7 Inclinometers
   3.8 Mercury switches
   3.9 Cameras
   3.10 Using the Polaroid ultrasonic ranging sensor
   3.11 The Futaba gyro
   3.12 The Zemco flux-gate digital compass

4. Much Random Information Concerning Motors
   4.1 Introduction
   4.2 DC Motors
      4.2.1 Rated voltage
      4.2.2 "No load" shaft speed
      4.2.3 Stall torque
      4.2.4 Starting current
      4.2.5 Loaded performance
   4.3 Gearboxes
      4.3.1 Spur gear boxes
      4.3.2 Planetary gear boxes
      4.3.3 Lead screws
      4.3.4 Worm gears
   4.4 Drivers
      4.4.1 H-bridge
      4.4.2 Relays
      4.4.3 Power problems
   4.5 Control
      4.5.1 Position sensors
      4.5.2 Rate sensors
   4.6 Stepping Motors
   4.7 Types of Stepping Motors
      4.7.1 The stator
      4.7.2 The rotor

4.7.2.1 The variable reluctance rotor
4.7.2.2 The permanent-magnet rotor
4.7.3 The controller
4.7.4 Implementation of stepper drivers
4.8 Futaba Servo Motors
4.9 Conclusion

5. Software Development System - 6811 Assembler and Subsumption Compiler
5.1 Programming Tools for the M68HC11
5.2 The Assembler
5.2.1 Assembling code
5.2.2 Code format
5.2.3 Assembler directives
5.2.4 Data statements
5.2.5 Instruction specification
5.2.6 Macros
5.2.7 Iterations
5.3 The Down Loader
5.4 The Subsumption Compiler
5.4.1 Organization
5.4.2 Invoking the compiler
5.4.3 Selectively compiling only parts of the system
5.4.4 Specifying the target machine
5.4.5 Compilable forms
5.4.6 Available primops
5.4.7 The squirt-op environment

6. Cross-Development Using a Host Computer
6.1 Running the Development Environment
6.2 Cables and Connectors
6.2.1 Flavors of connectors
6.2.2 Cable connections
6.3 Bootstrap Mode

7. Batteries

8. Connectors
8.1 Prototyping electronics
8.2 Mobot connectors

9. Debugging - "If it doesn't work, it's probably the connectors..."

10. An Example Robot Control System - Squirt's Brain

11. Designing the Photovore - A Case Study
11.1 Task
11.2 Sensors
11.3 Geometry
11.4 Chassis
11.5 Motors
11.6 Logic
11.7 Construction

1989 AI Lab Winter Olympics

Olympic Robot Building

"Implementing Your Imagination"

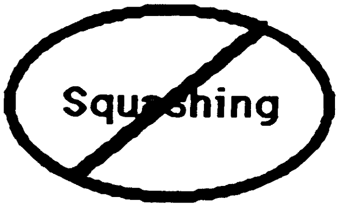
Anita Flynn

Goals

There are two primary goals for the Robot Olympics, the first and foremost of which is to-have fun. In the spirit of the AI Lab Olympics, the idea is to get to know other members of the laboratory in a setting where we not only work hard, but also play hard.

The second goal for the robot talent show is to give everyone a chance to get some hands-on experience and insight into the problems involved in actually building a robot. Hopefully, we'll create a labful of jacks-of-all-trades and there will be less temptation to build abstraction barriers. The spirit of the robot contest then, is to learn a bit about the things at which you're not already expert. So if you're a software guru, give a shot at the mechanical aspects; if you're an electronics wizard, take a crack at software; if you're an ace mechanical designer, try building some sensors.

Groundrules



Golden Rule #1 - <u>No Squashing</u>!

We want the robot olympics to be creative, imaginative and most of all, fun. As a creative environment is a very fragile thing, we have to always be sure we maintain the open atmosphere of our laboratory. Brainstorming, daydreaming and wild ideas are welcome and encouraged. Be careful you don't squash others when brainstorming in groups. Hammering someone's idea at the conception stage can ruin them for life. One Helpful Hint is to try playing "Can you top this?".

Golden Rule #2 - <u>Treat the Tools with Love and Respect</u>

After daydreaming, it's time to start implementing your imagination. The leaders of our lab have been more than generous in funding this tournament and everyone therefore is welcome to keep the robots they build. However, there aren't enough tools to go around and they must be returned after the contest. Don't mangle, destroy or lose the tools provided. For the most part, tools, especially wiring tools, are ridiculously expensive, so please make sure they aren't lost.

How to Build Stuff

Part of the impetus for this robot building undertaking is the hope that we can transfer the skills of quick prototyping to the general lab audience, unveiling the mysteries of craftsmanship. If all goes well, then we foresee a new research tool for every office - an "AI I/O" device; a real-time, real-world, sensory-input/actuator-output mechanism that brings Artificial Intelligence off a 2-D bit mapped display and into everyday activity.

The kits you've received are designed to help overcome the inertia of getting started. They contain a wide variety of sensors, a mechanical building block system, a single-chip microcontroller with an associated software development system, and a complete assortment of connectors, fasteners and prototyping equipment. You're free to build anything you like and you can choose to add to the kit or not use anything from the kit. As Olympic Robot Building is purposefully a talent show instead of a contest *per se*, the problems to be solved are intentionally unconstrained. You get to choose your own problem and solve it (as in general, much of the essence of good research is choosing the right problems on which to work).

Bon Voyage and have fun!

Lesson 1: How to Get Stuff

Where does all this material come from and how can you build things long after the Olympics are over? A big part of knowing how to build stuff is knowing where to get stuff. The first thing you should do before you get started, if you want to be a real robot hacker, is call manufacturers and ask for free catalogs. They'll be more than happy to send you one, but it often takes three weeks as they come third class mail. Get this in the queue as soon as possible. Below is a list of the manufacturers from which most of the parts in your kit came. Local distributors are also listed, as many parts aren't sold directly from the manufacturer. A few other vendors are also included even though we didn't actually get anything from them for these kits. However, you'll probably want to collect their catalogs for future reference. Many of the parts in your kit can be found in the lab's electronic parts stockroom, Ron Witken's office, room 908. These parts are marked by an asterisk.

Vendors	Tel. Nos.	Part Nos.	Price	Page No.
3M Electronic Products	800-328-SPEC			
225-1N 3M Center				
St. Paul, NM 55144				
Scotchflex prototype wiring technology. Distributed by Aztech Electronics.				
*Plug strips, 24 pos.		_3397-1240_	_.82_	
*Sockets, 16 pin		_3370-1000_	_2.02_	
*Sockets, 24 pin		_3374-1000_	_2.62_	
Aztech Electronics	301-995-6800			
8940-E Route 108				
Columbia, MD 21045				
Electronics distributor. Carries 3M Scotchflex wiring technology.				
Berg	516-599-5010			
499 Ocean Ave.				

E. Rockaway, NY 11518
Mechanical parts - gears, linkages, pulleys, etc.

Bicc Vero 203-288-8001
Electronics Handbook Catalog
1000 Sherman Ave.
Hamden, CT 06514
Speedwire equipment

Speedwire pen	*244-26213E*	*$31.10*	*p. 1.72*
** Reel of 1000 terminals*	*244-262195*	*138.68*	*p. 1.72*
Insertion tool	*244-299845*	*65.47*	*p. 1.72*

Cronin Electronics 449-5000
77 Fourth Ave.
Needham, MA 02194

Crystals	*8.0000 Mhz crystals FOX080*	*.95*	*p. 198*

Digi-Key 800-344-4539
701 Brooks Ave. S.
PO Box 677
Thief River Falls, MN 56701-0677.

Microphones	*P9930*	*.97*	*p.94*
Piezoelectric buzzers	*P9924*	*1.05*	*p.94*
Microswitches	*SW146-ND*	*2.58*	*p.97*

Douglas Electronics 415-483-8770
718 Marina Blvd. 415-357-7305 (uploading)
San Leandro, CA 94577
6811 PC board fabrication (circuit diagrams sent in over modem)

Edmund Scientific 609-547-3488
101 E. Gloucester Pike
Barrington, NJ 08007
All sorts of good stuff: lenses, optical equipment, electromechanical widgets, tools, etc.

ELI's Solid State Sales 547-7053
139 Hampshire St.
Cambridge, MA 02139
Close by and has all kinds of chips, power supplies, electronic instruments, etc.

Eltec Instruments 800-874-7780
PO Box 9610
Central Business Park
Daytona Beach, FL 32020-9610

Pyroelectric sensor with amplifier	*442-3*	*25.00*
Fresnel lens with 1" focal length	*6800408*	*1.00*

Emtel 1-769-9500
375 Vanderbilt Ave.
Norwood, MA 02062

Low voltage inhibitor for 6811 reset	*Seiko part 8054HN*	*1.59*

Ferranti Dege 547-8600
Harvard Square

Cambridge, MA
Camera store - sells good lithium batteries.

Duracell 3V, 160mAh lithium battery	*DL1/3N*	*3.45*
Duracell 3V, 1300mAh lithium battery	*DL123A*	

Also sold through Gerber Electronics.

Fordham 800-645-9518
260 Motor Parkway
Hauppauge, NY 11788

Wahl Cordless Soldering Iron	*Model 7800*	*47.95*	*p.49*
Soldering iron tips	*Models 7566, 7545, 7535*	*3.95*	

Gerber Electronics 1-769-6000
128 Carnegie Row
Norwood, MA 02062

** Perfboard - Epoxy glass Vector boards (for 4.8" x 17" x 1/16")*	*170H48WE*	*7.03*	*p. 681*

Hallmark Electronics 1-508-667-0902
6 Hook St.
Billerica, MA 01821
Distributor for the Motorola XC68HC811A2FN microprocessor *26.00*

Hamilton-Avnet 1-508-532-9682
50 Tower Office Park
Woburn, MA 01801
Huge distributor for many semiconductor manufacturers.

Jameco 415-592-8097
1355 Shoreway Road
Belmont, CA 94002
Great catalog for electronic parts, wire, tools, etc.

Lego Educational Dept. 800-527-8339
PO Box 39
Enfield, CT 06082

Technic Control II Set	*1092*	*195.00*
Touch sensor packs	*1346*	*15.00*
Chain link - small	*1317*	*14.00*
Gears - large	*1319*	*3.60*
Worm gears, racks	*1321*	*4.50*
Connecting leads	*1337*	*7.25*
Pinions	*1345*	*4.50*
Motors	*1334*	*21.00*
Angle plates, turntables	*1338*	*7.25*
Differentials	*1320*	*3.60*
Universal joints	*1339*	*4.50*
Gears - small	*1318*	*3.60*

Lego Mail Order 800-243-4870

Shock absorbers	*5251*	*4.00*
Battery boxes	*5005*	*6.00*
Gear reduction kit	*872*	*22.00*
15x15 grey base	*815*	*8.00*

Pneumatic tubes	*5102*	*2.50*
Pneumatic cylinders	*5104*	*2.50*
Pneumatic valves	*5106*	*4.00*
Hinges and couplings	*5179*	*3.00*
Chain link - large	*5244*	*3.50*

Maxim Integrated Prods. 408-737-7600
510 N. Pastoria Ave.
Sunnyvale, CA 94086

MAX233 serial driver chip - call Maxim for the special data sheet on the MAX233 level translator chip for driving RS232 serial ports. The chip is distributed by Pioneer Electronics.

Maxon Precision Motors 415-697-9614
838 Mitten Rd.
Burlingame, CA 94010

3V DC motor	*2313-910-21-141-001*	*28.00*
9V DC motor	*2312-916-21-141-010*	*28.50*

Methode Electronics 312-867-9600
7444 W. Wilson Ave.
Chicago, IL 60656

Socket for the 68HC811A2 - 52 pin through board chip carrier socket

Part No. 213-052-101	*2.38*	*p. 3G*

Micro Mo Electronics 1-813-822-2529
742 2nd Ave. S.
St. Petersburg, FL 33701

6V DC motor	*1212N006G*	*28.50*

Motorola Sales Office 932-9700
300 Unicorn Pk, 4th floor
Woburn, MA

Photodiodes (optoelectronics book)	*MRD721*	*.95*	*p.4.3*
Low volt. inhibit for 6811	*MC34046P-5*	*.95*	

Data books for all Motorola parts including 68HC811A2 data sheets. You can call and ask for data books (which will take three weeks) or you can drive there and pick them up yourself.

NASCO 201-625-5870
270 Rt 46 E
Rockaway, NJ 07866

Extraction tool for 52 pin (PLCC) sockets	*AT1767*	*18.35*

Newark Electronics 935-8350
10 G. Roessler Rd.
Woburn, MA 01801-6284

**Heat shrink tubing*	*FIT-221-3/64*	*41.46/100feet*	*p.784*
**Cinch serial port connectors DB-9 plug*	*DE-9P*	*2.47*	*p. 606*
**Cinch serial port connectors DB-9 socket*	*DE-9S*	*3.48*	*p. 606*
**Cinch serial port connectors DB-25 plug*	*DB-25P*	*3.74*	*p. 606*
**Cinch serial port connectors DB-25 socket*	*DB-25S*	*5.25*	*p.606*
**Cinch DB-9 hoods*	*DE-24657*	*3.94*	*p.610*
**Cinch DB-25 hoods*	*DB-24659*	*2.38*	*p.610*

North Star Electronics 1-508-657-5155
100 Research Dr.
Wilmington, MA 01887

5V-5V DC-DC converter	*S5R5*	*30.75*

Pacer Electronics 935-8330
70 Holton St.
Woburn, MA 01801

* *Kynar 30 gauge wire, solder*		
* *Belden 3-wire serial port cable*	*8641*	

Pioneer Electronics 861-9200
44 Hartwell Ave.
Lexington, MA 02173

Level translator chip for serial ports	*MAX233*	*3.00*

Polaroid Corporation 577-4681
Commercial/Battery Division
575 Technology Square - 3
Cambridge, MA 02139

Instrument-grade transducer	*604142*	*16.00*
Environmental transducer	*607281*	*18.00*
Single frequency driver board	*607089*	*26.50*
Cable assembly	*604789*	*2.00*

Radio Shacks:

Central Square	547-7332
Harvard Square	354-7836
197 Mass Ave., Boston	536-4773

Microphones	*270-092B*	*2.79*
Infrared emitter and detector	*276-142*	*1.99*
Photocells	*276-116A*	*1.79*
Flux-gate analog compass	*63-641*	*49.95*

Reliability Incorporated 713-492-0550
PO Box 218370
Houston, TX 77218

Small DC-DC converters.	*S5R5 5V-5V*	*30.00*

Distributed by North Star Electronics

Samtec 1-812-944-6733
PO Box 1147
New Albany, IN 47150

**Terminal strips for 6811 connectors*	*TS-132T-AA*	*1.00 per 20 pins*
**Socket strips for 6811 connectors*	*SS-132-T-2*	*1.00 per 20 pins*

These get expensive. They add up fast! Distributed by Pacer Electronics and Schaal.

Schaal 272-2506
Burlington, MA

Samtec socket strips	*SS-132-T-2*	*1.45 per 32 pins*

Schweber Electronics 275-5100
25 Wiggins Ave.

Bedford, MA 01730
Motorola's replacement for the 6811A2 *XC68HC811E2FN* *24.00*

Small Parts Inc. 1-305-751-0856
Small gears, pinions, mechanical linkages

Spectron 1-516-582-5600
595 Old Willets Path
Hauppauge, NY 11788

Inclinometers	*L-211U*	*196.00*
Mercury Switches	*M1202*	*4.00*

Tower Hobbies 1-800-637-4989
PO Box 778
Champaign, IL 61820

Futaba Standard model airplane servo motors S-28	*BH1116*	*26.00*
Futaba single axis gyros	*FP-G132*	*74.99*
Glue		

Yardney Battery 1-203-599-1100
82 Mechanic Street
Pawcatuck, CT 02891
The best batteries.

Western Micro 273-2800
20 Blanchard Rd.
Corporate Place 3
Burlington, MA 01803

Siemens IR sensors	*SFH484 IR LED emitters*	*1.00*
	SFH217 IR photodiode detectors	*1.50*
	BPW34 IR photodiode detectors	*1.00*

Zemco 1-415-866-7266
3401 Crow Canyon Rd., Suite 201
San Ramon, CA 94583

Flux-gate digital compass	*DE710*	*74.94*

Note that to build a robot, it takes parts from many vendors. For these Robot Olympics alone, there are over 40 different vendors. Finding suppliers, ordering parts and hustling to make sure everything comes in on time becomes a minor nightmare. Once we have our own micromachining fabrication line, we'll be able to design integrated robots in software, picking and choosing motors and sensors from among a variety of software libraries. Then we can mass produce them by the thousands like integrated circuits . But that's a bit of a project (let's save that one for next year's Olympics). Enough of daydreaming though, let's get on to the here and now of building robots!

The Motorola 6811 Microprocessor

Henry Minsky

The microprocessor we've chosen to distribute with the kits is the Motorola XC68HC811A2FN 8-bit microcontroller (or 6811 for short). Although we're only giving out 12 large kits of motors, sensors, tools, etc., (one to each subcaptain) there are over 40 kits of microprocessor parts which you'll find in small baggies. Each group building a robot should take a 6811 processor and then hook up with a subcaptain to choose among the variety of sensors, motors and mechanical building blocks available in their kit.

The 6811 was chosen because it comes in a small package and has some features that make it useful for robotics. The chip has 8 A/D converters onboard which means external circuitry for thresholding sensors and such can be eliminated. There are also 38 general purpose I/O pins and a variety of timers available. The chip includes 256 bytes of RAM and 2K bytes of electrically erasable programmable read only memory (EEPROM) for program space.

Since the 6811 comes with 2K bytes of EEPROM, there is no need for burning EPROMs as electrically eraseable PROMs can be burned in software. The 6811 has a serial port onboard and when the 6811 is powered up in a special bootstrap mode, it immediately begins to listen to whatever comes in over the serial port. It writes the first 256 bytes into the top of memory and then jumps there and begins executing that code. Downloading code to the 6811 then, is as simple as putting the processor in bootstrap mode, powering it up and sending a 256 byte program that knows how to listern for more stuff coming in over the serial port and is able to move that user program to another location in memory (EEPROM space) and burn it in. After you've downloaded your code, you power the 6811 down, place it in single-chip mode and power it up again. It will immediately start running your target code. This code stays in EEPROM and will continue to run no matter how often you power down and on again.

We give you a software development system that takes care of all this. The development system allows you to write code in 6811 assembly language and download it to your robot. In addition to the assembler and downloader, there is also a subsumption compiler which you can use if you want. It allows you to specify robot behaviors in terms of subsumption networks, and it compiles directly to 6811 code. The software development system is written in Common Lisp and runs on several types of machines around the lab: Macs, Suns, Symbolics and HPs. Further documenation on the software development system can be found at the end of these notes.

If you look in your baggie full of 6811 parts, you'll find a PC board, a 6811 microprocessor, a MAX233 serial port level translator chip, a hex display/buffer chip, a low voltage inhibit circuit for reset, a 74HC244 buffer and some other parts such as a switch, a socket for the 6811, a crystal, a few resistors and a handful of connectors called socket strips.

Two printed circuit boards have actually been fabricated for these Olympics. One is for classic DIP style packages of integrated circuits and the other is for surface mount components (these boards are called OLM2 and OLM2S respectively, as they're Rev 2 editions). You'll find the OLM2 version in your kit. The PC boards have been made as general as possible. They contain the minimum amount of real estate necessary to get you going, yet there is room onboard to allow you to add your own interfacing circuity by leaving room for

Speedwire terminals to be inserted, which you can then wire up any way you please. On the other hand, if you're trying to build a very small robot, you can cut the two sides off this board, including the parts we've devoted to serial drivers and buffers.

One note of caution: a CMOS microprocessor should always have its outputs buffered. Failure to do this can result in blowing out the microprocessor.

Here is a rundown of the PC boards supplied in your kit. Block diagrams, schematics and board layouts are given in the next few pages.

OLM2 and OLM2S MC68HC11 Microcontroller Boards

FEATURES

Board OLM2
- 8 bit 6811 HCMOS CPU
- Buffered output on Port B using 74HC244
- Numerous VSS and VDD connections on IO port connectors
- Optional board sections:
    - RS232 driver
    - LED display/buffer on bits 3-6 of Port A
    - Two user-prototype areas with VSS, VDD connections

Board OLM2S (Surface Mount)
- Smaller size
- Buffered output on Port B using 74HC244

DESCRIPTION

The OLM2 microcontroller board contains a MC68HC11 microcontroller which contains 2K of nonvolatile EEPROM program memory. See the Motorola Technical Data booklets in your subcaptain's kit for all details on the internal architecture of the 6811A2.

The OLM2 board has .100" spaced connector areas for all I/O ports of the 6811, as well as the interrupt lines, clock and address and data strobes.

There is a serial-port connection area directly below the 6811 socket. This area is connected directly to the 6811, and thus has TTL level 5V signals. *Connecting this port directly to an RS232 level signal will likely damage the CPU and other components on the board.* Please see the **USING SERIAL I/O** section of this data sheet.

A 74HC244 octal driver chip is mounted next to the 6811 to buffer the outputs of PORT B. You should use these for driving any power circuits, like transistors or relays. Check the specs on how much current you can pull with the 244 driver before trying to drive any large loads such as small motors or relays.

I/O Connectors for ports B and C have alternating VSS,VDD supply pins running next to them, for powering sensors or other peripherals.

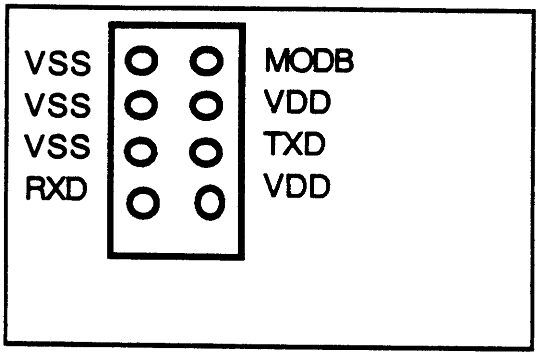
The board is designed to be cut into sections if you want to save space. The optional sections contain an RS232 driver chip, an LED display/buffer, and two prototype areas drilled with .100" spaced .062" holes. The boards can be cut with the shears in the machine shop or Ron's office, room 908.

The TI309 LED BCD display/buffer displays the value of PORT A bits 3:6, and also buffers these lines as outputs.

USING SERIAL I/O

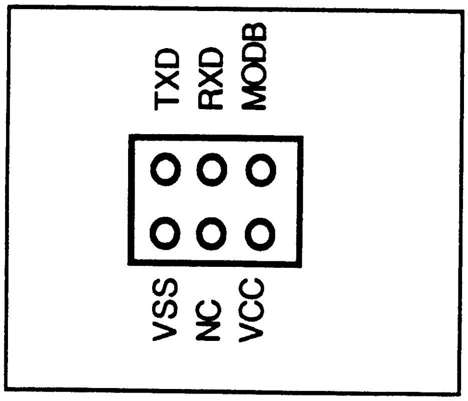
To talk to the OLM2 board, you will need to make a serial port cable. In order to program your board directly from the serial-port of your host machine, you will want to use the RS232 serial-port on the board. This is the connector area located directly above the MAX233 RS232 driver chip on the left of the board.

Pinout of RS232 port:



When you make your cable, have it connect MODB to Ground. That will automatically put the 6811 in bootstrap mode upon reset. MODB is pulled up on the board, so when your board is programmed, you can remove the cable and upon reset the processor will come up in run-mode.

If you want to save space or conserve power, you can omit the RS232 section, and talk directly to the TTL level serial port of the 6811. These TTL signals are available from a connector area right below the 6811 socket. The trick to making this space saving hack work is to get the MAX233 driver chip off your board by building it into the end of the serial port cable coming from your host computer. Then you can actually saw your 6811 board down to about half its original size. Pinout of TTL level serial port:



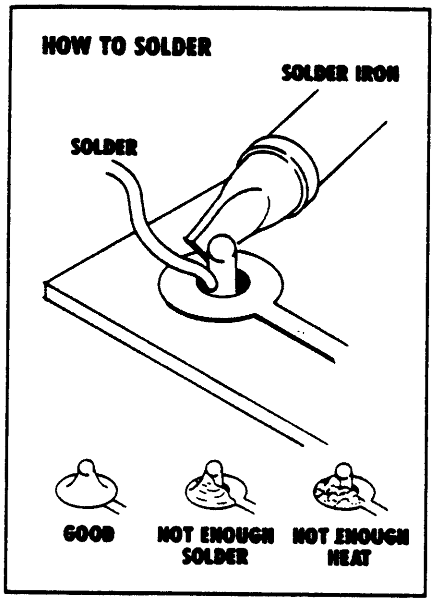
Again, your cable should jumper MODB to Ground, putting the 6811 into bootstrap mode.

ASSEMBLING THE OLM2 BOARD:

PARTS PLACEMENT

Since there is no silkscreen on the board, parts should be placed by referring to the included diagram, and comparing with an actual finished board.

The 6811 socket should be soldered into place first, and then the crystal and reset switch. Pullup resistors are next, and then connector strips. Make sure to make good solder joints by heating the pads slightly with the soldering iron tip before touching the solder to the pad. Note you shouldn't touch the piece of solder to the soldering iron directly. Rather, it should melt by heat conduction from the soldering iron to the pad and then to the piece of solder. As soon as it melts, hold the soldering iron tip there just for a second and then take it away. You just need a small amount of solder. Don't glob it on. Also, make sure the solder pool hardens smoothly so as not to make cold soder joints.



The 5V voltage regulator (not to be confused with the low-voltage detect/reset device) is optional. Only the LED display chip needs a 5 volt TTL supply level. The other components have a wider range of allowable power-supply voltages. Note that with the low-voltage sense/reset device installed, power supplies below about 4.5 volts will shut down the CPU.

There is one slight oddity about the 6811A2 version, the version which contains EEPROM. There is a problem on power down in which if the reset signal follows the supply voltage, certain instructions may stop working correctly and overwrite the CONFIG register (which is implemented in EEPROM and sets the memory map of the microprocessor) before power has completely been shut off. Then next time power is applied, the memory map is pointing off to somewhere strange, and the processor doesn't know where to find your program. Consequently, a low voltage inhibit circuit is required on the reset line, which forces the reset line low whenever the power supply falls below a certain threshold.

Note that power to the board should be supplied from the points near the voltage regulator.

THE OLM2S SURFACE MOUNT BOARD

This board contains a minimal configuration of a surface mounted 6811 and an octal buffer on port B. If we can figure out how to solder the chips to this board, it will be a good alternative for designs which need to save space and weight.

Note that there is no RS232 driver chip on this board, so an external level converter must be used. The serial port area has the same pinout as the OLM2 board. Other connector areas have different pinouts. See the attached placement sheet for details.

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

BR289

MC68HC811A2

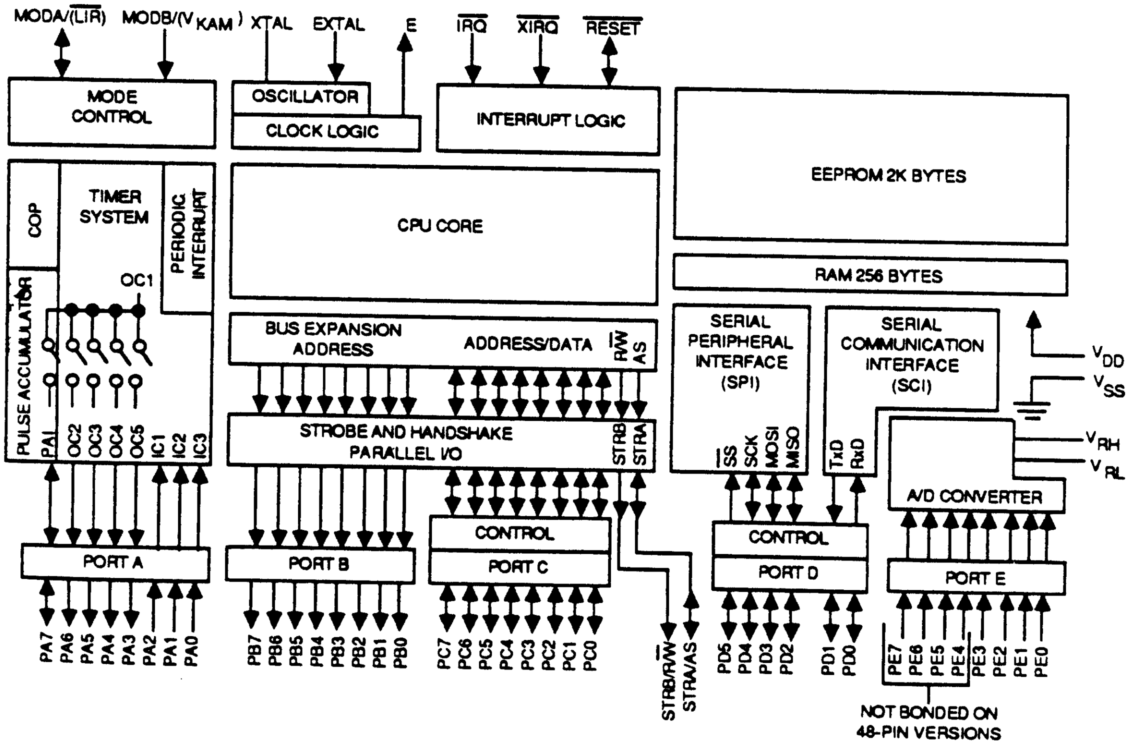
Technical Summary

8-Bit HCMOS Microcomputer

The HCMOS MC68HC811A2 is an advanced microcomputer (MCU) containing highly sophisticated on-chip peripheral functions. An improved instruction set provides additional capability while maintaining compatibility with the other members of the M6801 Family. The fully static design allows operation at frequencies down to dc, further reducing its already low power consumption. Features include:

- Power Saving STOP and WAIT Modes
- Separate RAM Voltage Supply Pin (V_{KAM})
- 2K Bytes of EEPROM (Byte Eraseable)
- 256 Bytes of Static RAM (All Saved During Standby)
- Enhanced 16-Bit Timer System
  - Four Stage Programmable Prescaler
  - Three Input Capture Functions
  - Five Output Compare Functions
- A Real Time Interrupt Circuit
- An 8-Bit Pulse Accumulator Circuit
- An Enhanced Non-Return-to-Zero Serial Communications Interface (SCI)
- A New Serial Peripheral Interface (SPI)
- Eight Channel 8-Bit A/D Converter
- A Computer Operating Properly (COP) Watchdog System
- Multilevel Interrupt Priorities (21)

BLOCK DIAGRAM



This document contains information on a new product. Specifications and information herein are subject to change without notice.

MOTOROLA

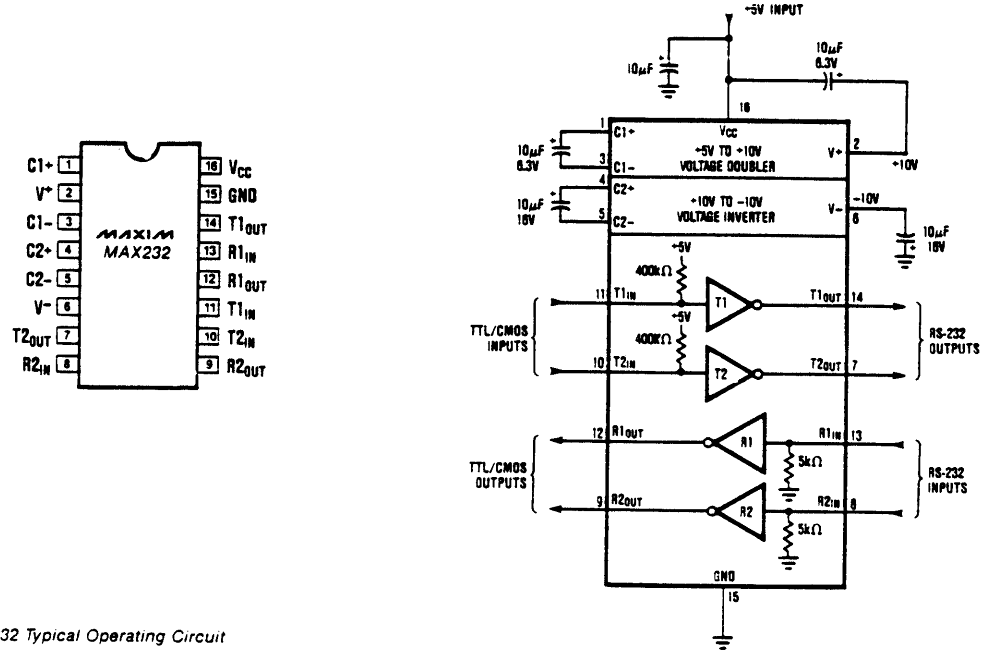
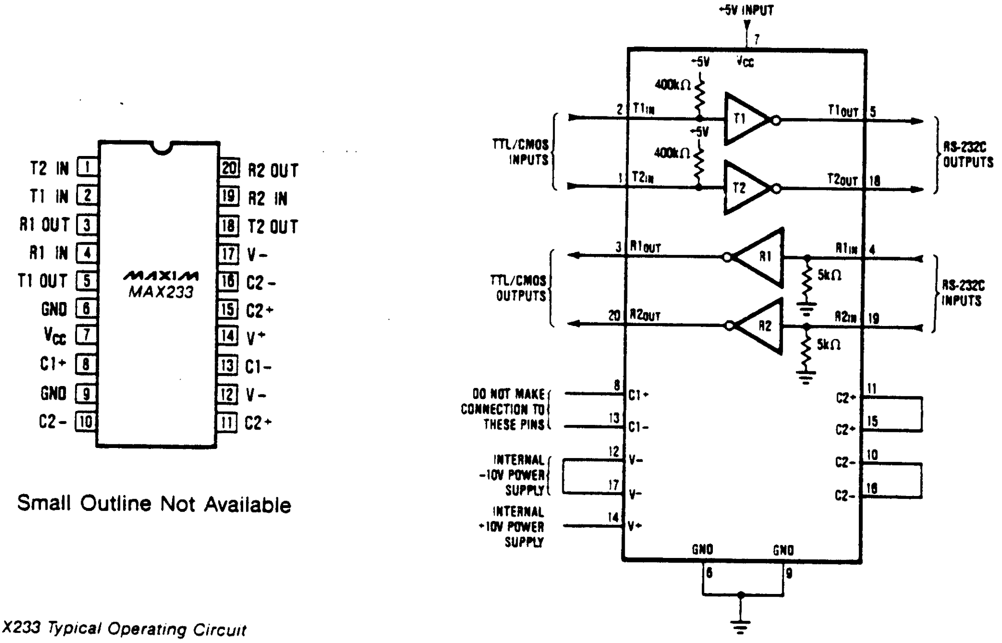
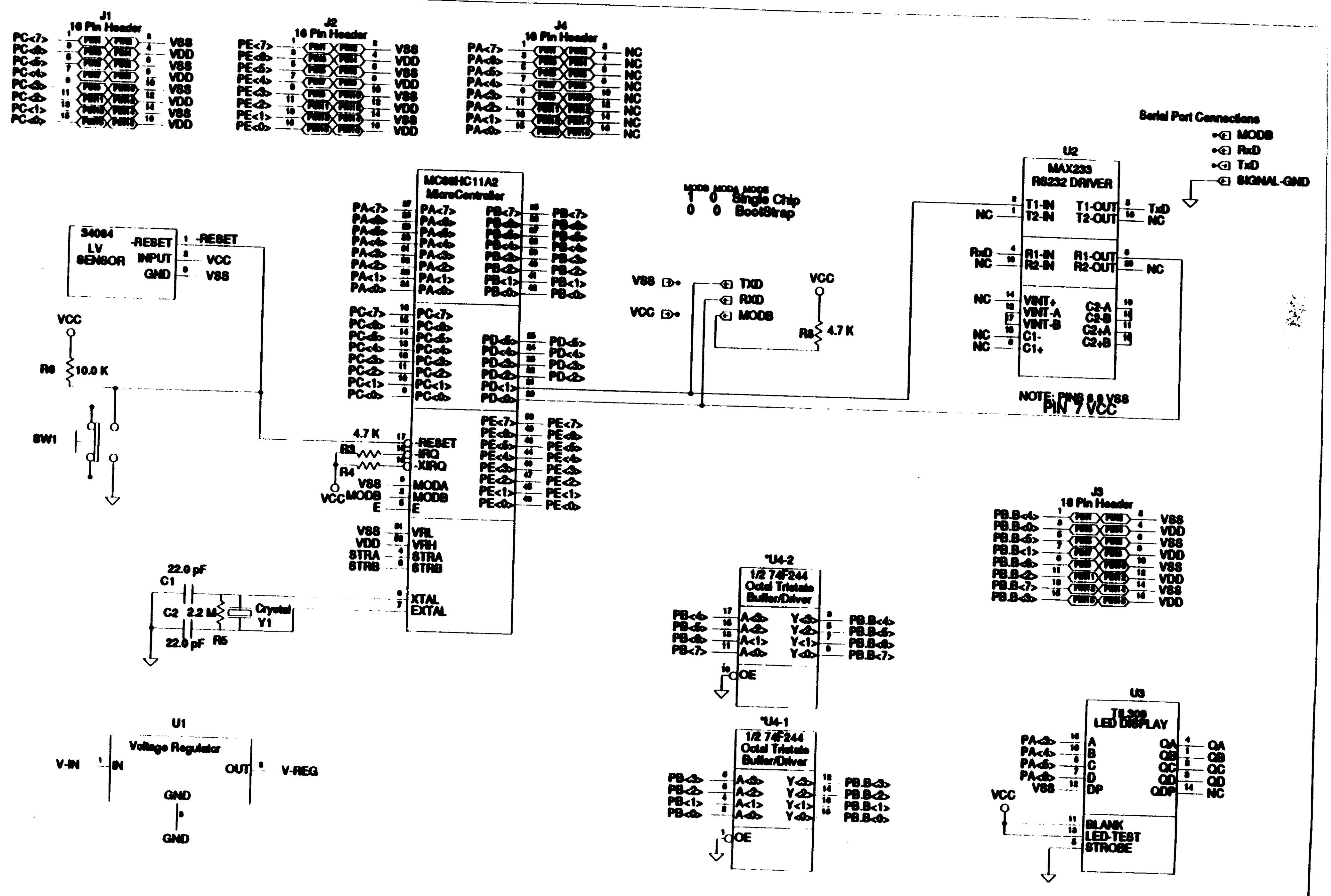


Figure 5. MAX232 Typical Operating Circuit



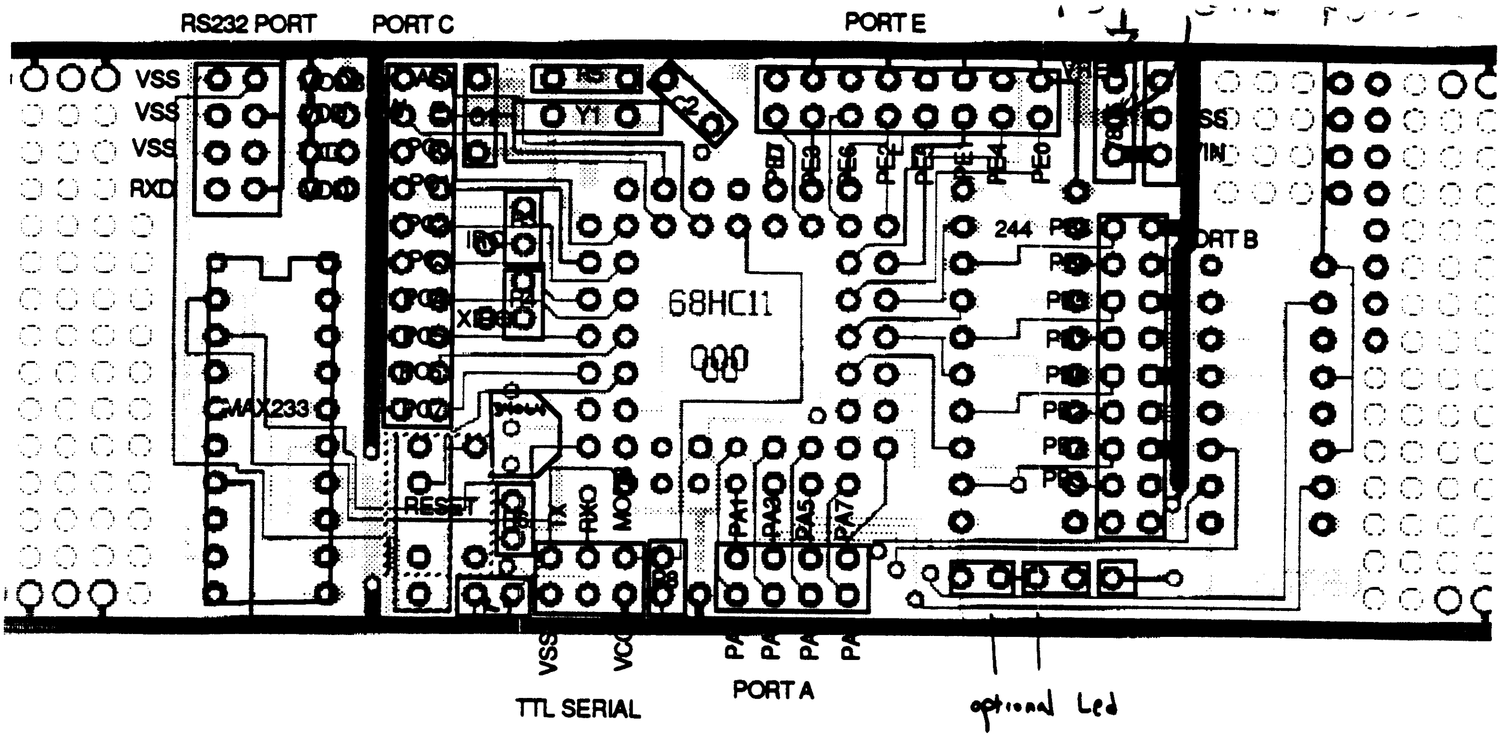
Small Outline Not Available

Figure 6. MAX233 Typical Operating Circuit



ROBOT:OLYMPIC-BOT

HQM | 11/16/88 12:32



COMPONENT RAIL IS VDD
SOLDER SIDE RAIL IS VSS

Jasmine HD:Lisps:Allegro CL:squirt:blink.lisp *11/30/88 11:44:43 AM* *Page 1*

```
;;;Code for blinking portb leds

(defprog count-test
  :machine 6811
  :start #xf800
  :code ((=v portb #x1004)
         (=v porta #x1000)
         (=c stack #xff)

     start
         (lds ! stack)
         (ldaa ! #x00)
      loop
         (staa porta)
         (jsr delaysome)
         (inca)
         (jmp loop)

     delaysome
         (ldx ! #x1fff)
     dlots
         (iterate ((i 4)) (nop))
         (dex)
         (bne dlots)
         (rts)

         (= #xfffe)
         (!16 start)
         ))
```

MOTOROLA
SEMICONDUCTORS
P.O. BOX 20912 • PHOENIX, ARIZONA 85036

MC34064
MC33064

Product Preview

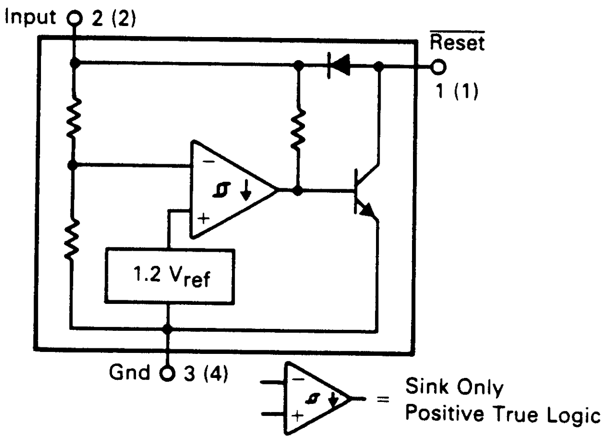
UNDERVOLTAGE SENSING CIRCUIT

The MC34064 is an undervoltage sensing circuit specifically designed for use as a reset controller in microprocessor-based systems. It offers the designer an economical solution for low voltage detection with a single external resistor. The MC34064 features a trimmed-in-package bandgap reference, and a comparator with precise thresholds and built-in hysteresis to prevent erratic reset operation. The open collector reset output is capable of sinking in excess of 10 mA, and operation is guaranteed down to 1.0 volt input with low standby current. These devices are packaged in 3-pin TO-226AA and 8-pin surface mount packages.

Applications include direct monitoring of the 5.0 volt MPU/logic power supply used in appliance, automotive, consumer and industrial equipment.

- Trimmed-In-Package Temperature Compensated Reference
- Precise Comparator Thresholds Guaranteed Over Temperature
- Comparator Hysteresis Prevents Erratic Reset
- Reset Output Capable of Sinking in Excess of 10 mA
- Internal Clamp Diode for Discharging Delay Capacitor
- Guaranteed Reset Operation with 1.0 Volt Input
- Low Standby Current
- Economical TO-226AA and Surface Mount Packages

REPRESENTATIVE BLOCK DIAGRAM



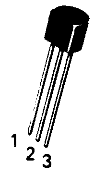
Pin numbers adjacent to terminals are for the 3-pin TO-226AA package.
Pin numbers in parenthesis are for the D suffix SO-8 package.

This document contains information on a product under development. Motorola reserves the right to change or discontinue this product without notice.

UNDERVOLTAGE SENSING CIRCUIT

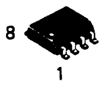
SILICON MONOLITHIC INTEGRATED CIRCUIT

P SUFFIX
PLASTIC PACKAGE
CASE 29-04



PIN 1. $\overline{\text{RESET}}$
2. INPUT
3. GROUND

D SUFFIX
PLASTIC PACKAGE
CASE 751-02
SO-8

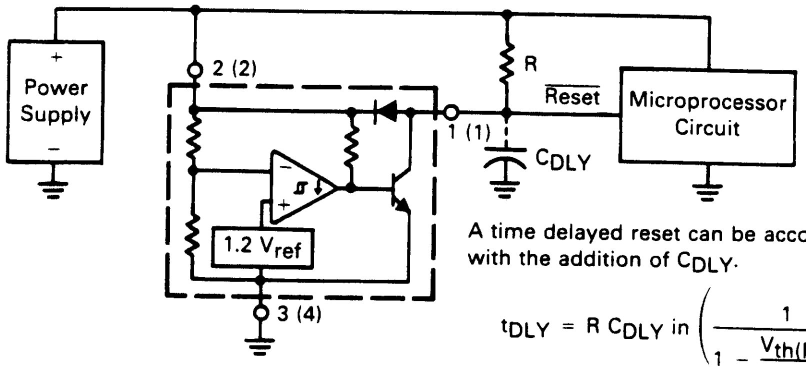


PIN 1. $\overline{\text{RESET}}$	5. N.C.
2. INPUT	6. N.C.
3. N.C.	7. N.C.
4. GROUND	8. N.C.

ORDERING INFORMATION

Device	Temperature Range	Package
MC34064D-5	0°C to +70°C	Plastic SO-8
MC34064P-5		Plastic TO-226AA
MC33064D-5	−40°C to +85°C	Plastic SO-8
MC33064P-5		Plastic TO-226AA

FIGURE 7 — LOW VOLTAGE MICROPROCESSOR RESET



A time delayed reset can be accomplished with the addition of C_{DLY}.

$$t_{DLY} = R\,C_{DLY}\ \ln\left(\frac{1}{1-\frac{V_{th(MPU)}}{V_{in}}}\right)$$

FIGURE 8 — VOLTAGE MONITOR

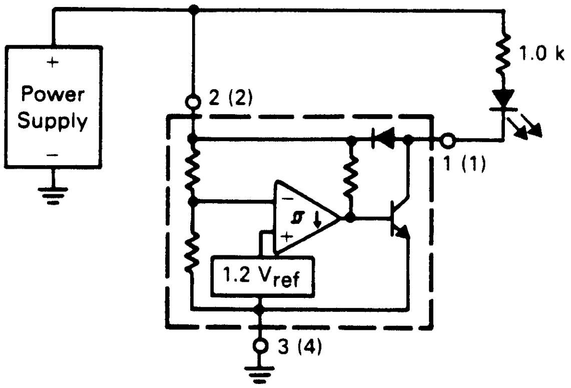


FIGURE 9 — SOLAR POWERED BATTERY CHARGER

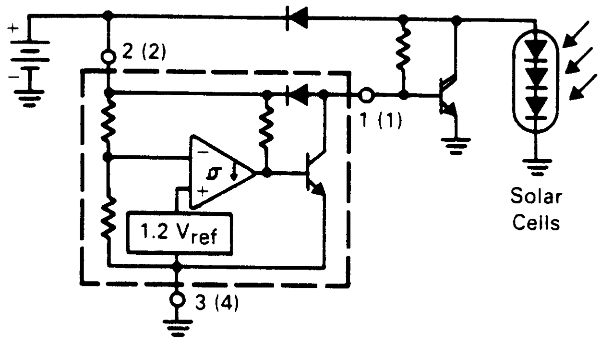
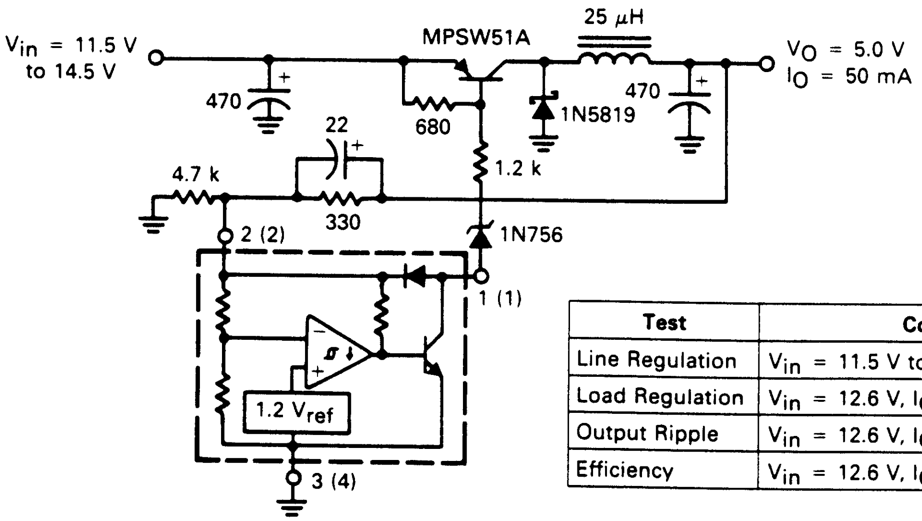


FIGURE 10 — LOW POWER SWITCHING REGULATOR



Test	Conditions	Results
Line Regulation	V_{in} = 11.5 V to 14.5 V, I_O = 50 mA	35 mV
Load Regulation	V_{in} = 12.6 V, I_O = 0 mA to 50 mA	12 mV
Output Ripple	V_{in} = 12.6 V, I_O = 50 mA	60 mV_{p-p}
Efficiency	V_{in} = 12.6 V, I_O = 50 mA	77%

RELATIVE LUMINOUS INTENSITY
vs
CASE TEMPERATURE

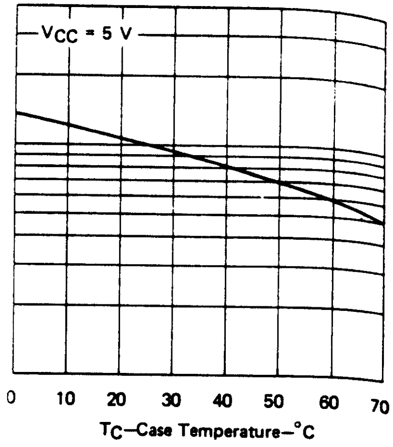


FIGURE 3

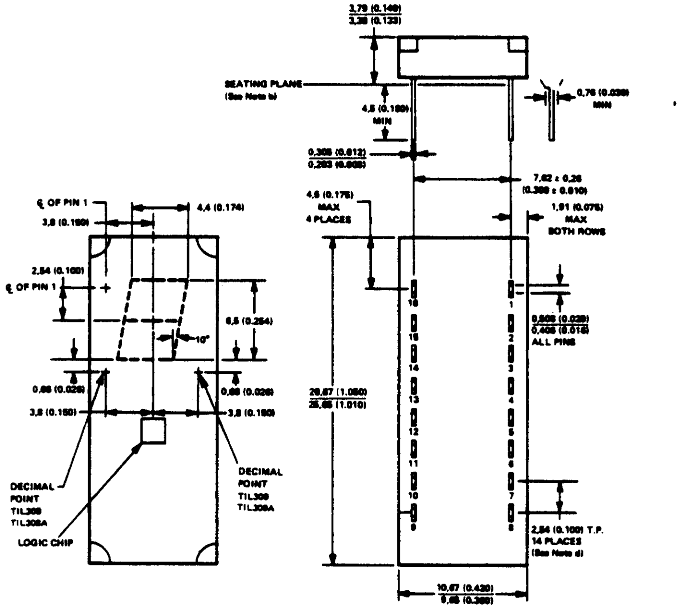
D1096, MARCH 1072 – REVISED JUNE 1982

SOLID-STATE DISPLAYS WITH INTEGRAL TTL MSI CIRCUIT CHIP FOR USE IN ALL SYSTEMS REQUIRING A DISPLAY OF BCD DATA

- 6,9-mm (0.270-Inch) Character Height
- TIL308 and TIL308A Have Left Decimal
- TIL309 and TIL309A Have Right Decimal
- Easy System Interface
- Wide Viewing Angle
- Internal TTL MSI Chip with Latch, Decoder, and Driver
- Constant-Current Drive for Light-Emitting Diodes

mechanical data

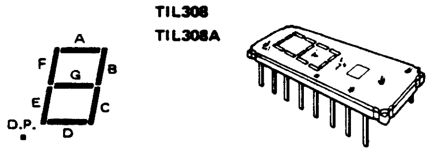
These assemblies consist of display chips and a TTL MSI chip mounted on a header with either a red molded plastic body for the TIL308 and TIL309 or a red plastic cap for the TIL308A and TIL309A. Multiple displays may be mounted on 11,43-mm (0.450-inch) centers.

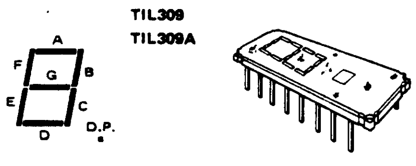


PIN ASSIGNMENTS
FOR BOTH TYPES

PIN 1	LATCH OUTPUT Q_B (BINARY WEIGHT 2)
PIN 2	LATCH OUTPUT Q_C (BINARY WEIGHT 4)
PIN 3	LATCH OUTPUT Q_D (BINARY WEIGHT 8)
PIN 4	LATCH OUTPUT Q_A (BINARY WEIGHT 1)
PIN 5	LATCH STROBE INPUT
PIN 6	LATCH DATA INPUT C (BINARY WEIGHT 4)
PIN 7	LATCH DATA INPUT D (BINARY WEIGHT 8)
PIN 8	GROUND
PIN 9	NO INTERNAL CONNECTION
PIN 10	LATCH DATA INPUT B (BINARY WEIGHT 2)
PIN 11	BLANKING INPUT
PIN 12	LATCH DATA INPUT DP
PIN 13	LED TEST
PIN 14	LATCH OUTPUT DP
PIN 15	LATCH DATA INPUT A (BINARY WEIGHT 1)
PIN 16	SUPPLY VOLTAGE, V_{CC}

NOTES: a. All linear dimensions are in millimeters and parenthetically in inches.
b. Lead dimensions are not controlled above the seating plane.
c. Centerlines of character seqments and decimal points are shown as dashed lines. Associated dimensions are nominal.
d. The true-position pin spacing is 2,54 mm (0.100 inch) between centerlines. Each centerline is located within 0,26 mm (0.010 inch) of its true longitudinal position relative to pins 1 and 16.
e. On TIL308A and TIL309A devices, the 4 mold indentations are not present.

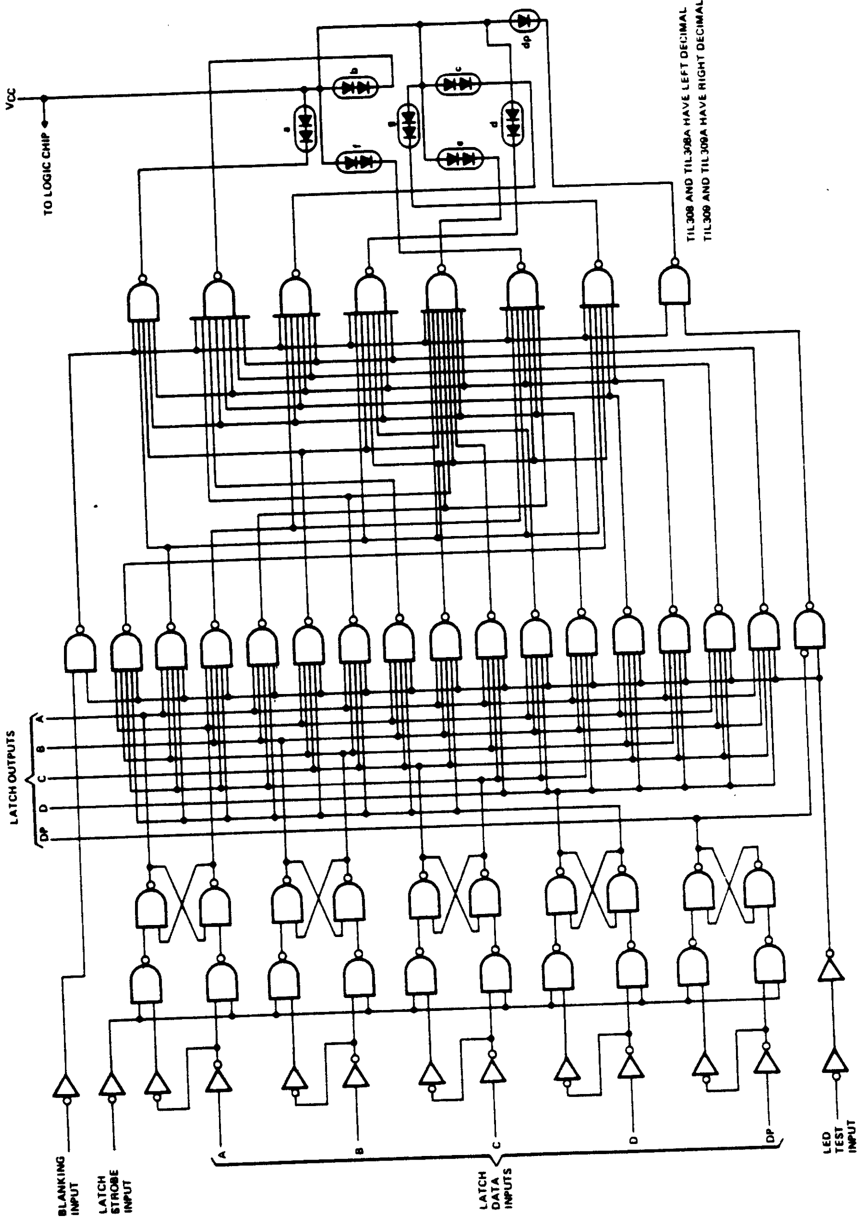




PRODUCTION DATA documents contain information current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

TEXAS INSTRUMENTS
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

functional block diagram



description

These i
driver i

F

LATCH

LATCH
A, B, C

LATCH
QA, QE

BLANK

LED TE

FUNCTION
0
1
2
3
4
5
6
7
8
9
A
MINUS SIGN
C
BLANK
E
F
BLANK
LED TEST

H = high level,
DP input has a

Sensors

Anita Flynn

We claim that robotics is a superset of AI because robotics is the study of the computations that connect perception to action, and robotics therefore forces AI to deal with the real world. However, it's hard to build an interesting thinking component if the decision system has no variables to juggle. Getting these 'perception variables' into an intelligent control system seems to be one of the black holes of current research.

In these Olympics, we're going to attempt to push through that problem by supplying as wide a variety of sensors as we possibly can. In your kit, you'll find such things as pyroelectric motion sensors, sonar rangefinders, microswitched touch sensors, infrared emitters and detectors, photodiodes, microphones, photocells, flux gate compasses, inclinometers, gyros and mercury switches. Such a large variety of sensors should be inspirational for all sorts of intelligent robot action. Notice there aren't any cameras in the kit. As of this writing, cameras are too expensive to give away ($800 typically), but there are a number of cameras around the lab which are available and you should feel free to use them.

That brings up a point about choosing whether or not to go with onboard or offboard computation in these projects. The microprocessor you're supplied with contains only 2K of eeprom program space and 256 bytes of RAM. If your code is lean, this should suffice. If you need more computational power, you can attach more memory through one of the 6811's ports and run it in extended mode. On the other hand, another very viable solution is to connect all sensors and actuators to the A/D ports or other port pins and use the serial port on the chip to communicate to a larger computer offboard. This could be a very convenient way to interface such things as your thesis or other research code you've already finished, to a variety of sensors and actuators. Thus a camera could be attached to a small robot, computations done offboard, and then commands shipped down to the 6811 over a cable. The point is then, that there is a wide assortment of sensors available and you shouldn't feel constrained in terms of logistics.

What follows is a rundown of some of the sensors provided, including the physics of their operation, a synopsis of their capabilities and limitations and some electronic interfacing examples.

Pyroelectric Motion Sensors

A pyroelectric sensor detects a change in temperature per unit time. These types of sensors are most commonly used as burglar alarms in security systems. Filters are usually placed over the sensors to pass the range of infrared energy emitted by humans. Pyroelectric sensors can be thought of as warm body motion sensors. Relative motion is always required between the pyroelectric and the target for there to be any signal. Note that this is very different than a thermocouple or an infrared night vision camera. Those types of sensors will output different voltages for different steady state temperatures. Pyroelectrics have no response in the steady state. If a person walks into the field of view (which can vary depending upon the optics used), the sensor will trigger. However, if the person stands very still, the sensor will no longer be able to detect that person.

Fresnel lenses are most often used with pyroelectrics because they absorb less infrared energy than a typical convex glass lens. A fresnel lens is a thin plastic sheet with a surface texture equivalent to discretizing the curvature of a convex lens and laying it out on a flat sheet. This type of optical system has a side effect of segmenting the image hitting the sensor, an effect which is used advantageously. The data sheet for the sensor given in your kit, the Eltec 442-3 is shown on the next page. The 442 is actually two pyroelectric crystals connected in a parallel-opposed configuration. This means that the voltage actually output by the sensor is the difference in voltages between the two crystals (an amplifier is also integrated into this sensor to boost the gain). Due to the segmentation effect of the optics, only one of the crystals sees the person at a given time. Nominally, the output voltage would float at 2.5V, but if a person walks into the area of detection, the output signal will first rise above 2.5V and then below 2.5V as the person walks through each sensor's field of view. If a person walks throught the field of view in the other direction, the signal will do the opposite - first drop below 2.5V and then rise above it.

A simple way to interface this sensor to your robot then, is to apply +5V and Ground to the appropriate sensor pins, and connect the signal wire straight into one of the analog to digital converter channels (Port E on the 6811). Then just write software that polls this pin and when the value goes over a threshold, declare a person has been found. In general, pushing your technology to the software stage as fast as possible is a good heuristic for getting things up and running quickly.

The 442 sensor has roughly a six degree field of view, so it is relatively straighforward to determine the orientation to a target, but no range calculation is possible with this simple sensor. You may want to use a pyroelectric sensor in conjunction with a rangefinding sensor if this information is desired.

The fresnel lenses given with the kit have 1" focal lengths. You'll need to build some type of structure to hold the lens. Construction paper cut to the appropriate cone shape with a few drops of super glue works well.

Microswitched Touch Sensors

Microswitches are some of the simplest sensors available. The microswitches given in your kit are the smallest ones I've been able to find, but they come in all shapes and sizes and have a wide variety of specifications for the force required to close the switch. Microswitches could be used as whiskers or touch sensors: attach a feeler to the lever switch and connect the normally open output to a port pin on your 6811 and you have a very simple but effective one-bit sensor. You may want to add a simple debouncing circuit so that your 6811 polls a nice clean signal.

You'll also find some microswitches in your Lego kit. They come in the form of a Lego block and fit right in with the rest of the system. There aren't very many of these, but you can build your own out of the ones given in the kit.

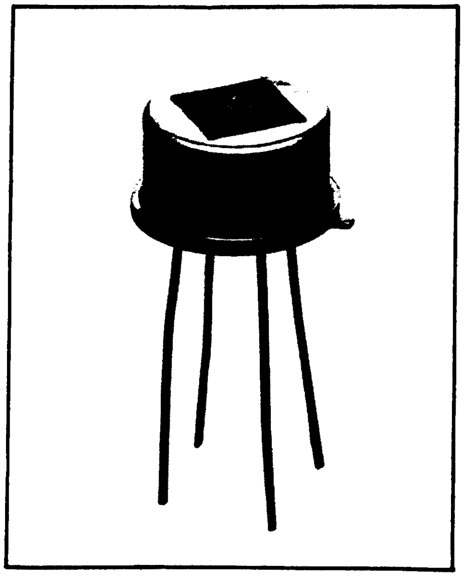
Photodiodes

Photodiodes are diodes that are sensitive to light and create a current when photons hit the junction. They can come in several varieties. Most commonly, they are sensitive to either visible light or infrared radiation. A specification sheet for the photodiode given in your kit, the Motorola MRD721, is shown on the next page. A Motorola Optoelectronics Data Book is also included in your kit, and you should scan through it for a better idea of the types of optical sensors available. An simple interface circuit, that of a photodiode in a logarithmic amplifier configuration connected to an A/D on the 6811, is given in the chapter on the example robot, Squirt, at the end of these notes.

ELTEC MODEL 447

PRELIMINARY DATA

IR-EYE™ INTEGRATED SENSOR
Parallel Opposed Dual IR Detector With Integrated Signal Processing*



Eliminate Burn-In Tests	Miniaturize Circujtry
Improve RF Immunity	Reduce Components
Eliminate False Alarms	Reduce Repairs

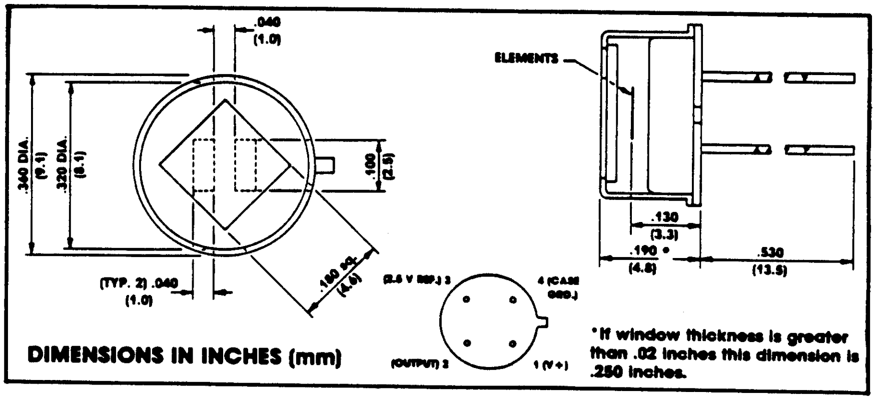
The **Model 447 IR-EYE™** Integrated Sensor is a Lithium Tantalate pyroelectric parallel opposed dual element high gain detector with complete integral analog signal processing. This unit offers greatly improved detection capability over an extended temperature range of -40 to +70°C with no significant change in noise or sensitivity.

Features

100 x Signal Amplification
100 x Voltage Regulation
2 x Detection Capability
Wide Operating Temperature

Applications

People/Object Detection
Intrusion Detection
Lighting Control
Robotics
Motion Sensing
Automatic Door Control
Safety Warning
High Stability Industrial & Military Applications



DIMENSIONS IN INCHES (mm)

*If window thickness is greater than .02 inches this dimension is .250 inches.

MODEL 447 Specifications

Operating Characteristics

D* (cm Hz$^{1/2}$/W, BW-1Hz)	2.0 x 10^8
NEP (W/Hz$^{1/2}$, BW-1Hz)	7.4 x 10^{-10}
Responsitivity (V/W)	2.7 x 10^5
Common Mode Rejection (Min.)	5/1
(Typ.)	15/1
Noise (mV/Hz$^{1/2}$)	0.2
Breakpoint:	
Thermal	0.15Hz
Electrical	5Hz
Power Supply	
Voltage	5-15 VDC
Current (Max.)	2.0 mA

Output Characteristics

Voltage (Max.)	V+
Current (Rec.)	0.02 mA
Output Load (min.)	15 Kohm
Reference Voltage** pin 3/4	+2.5 V
Offset Voltage	±30mV

Ambient Operating Conditions

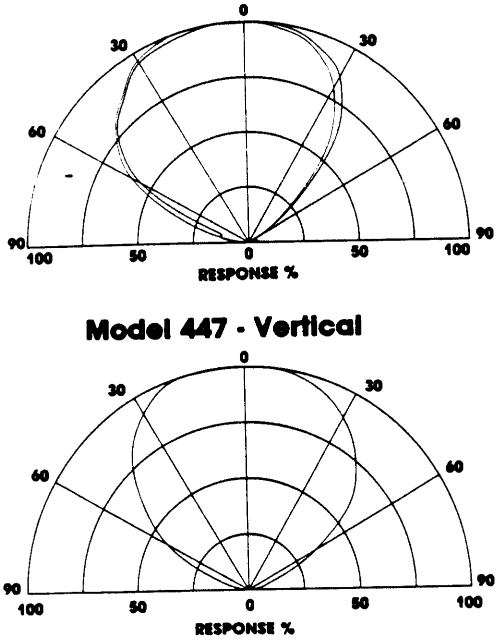
Storage Temp.	-55 to +125° C
Operating Temp.	-40 to +70° C
Sensitivity to:	
Temperature	+.3%/°C

NOTE 1- Characteristics are at 25°C, 14.7 psia, V+ = 5VDC, f = 1Hz, Bandwidth of 8-14 micrometers.
NOTE 2- The information contained in this sheet has been obtained from development samples. Data is believed to be representative.

*Patent pending. Manufactured under one or more of the following U.S. patents: 3,839,640 - 4,218,620 - 4,326,663 - 4,384,207 - 4,437,003 - 4,441,023 - 4,523,095

**See reverse for additional information.

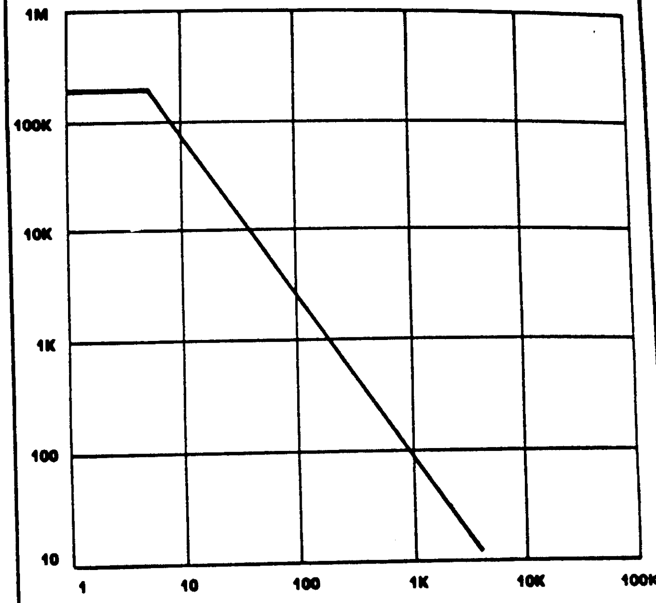
FIELD OF VIEW
Model 447 - Horizontal



Model 447 - Vertical

For -3 window only. For other windows consider refractive index and thickness.

FREQUENCY RESPONSE
MODEL 447
(EACH ELEMENT)



FREQUENCY (Hz) - HORIZONTAL AXIS
RESPONSITIVITY (V/W) - VERTICAL AXIS

Mounting, Soldering and Handling:

These Sensors have been improved over previous Models and can withstand normal handling and automatic assembly as well as wave soldering at 280°C for 10 seconds, at 1/4" (6.3mm) from the case bottom.

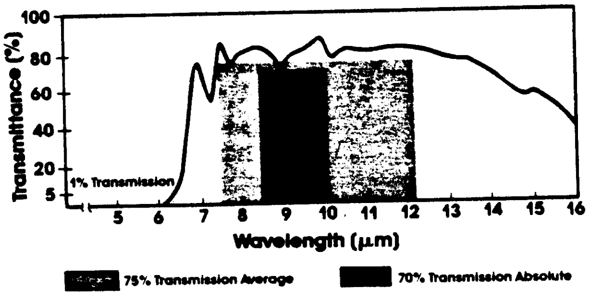
Contamination and fingerprints on the window surface should be cleaned with alcohol and a soft cloth.

Avoid mechanical stresses on case and leads.

Static Discharge

Additional safety features are used internally to make these sensors almost immune to electrostatic discharge.

Transmission Characteristics of -3 Window (HP-7)



Reference Voltage

The internal biasing voltage is accessible on pin 3. This voltage is used to drive the internal output amplifier. Offset voltage is referred to this point.

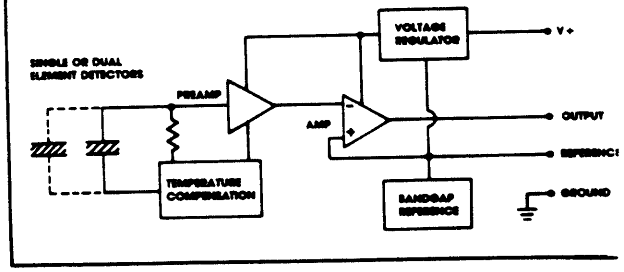
This reference provides a low drift zero to allov for direct DC coupling of a subsequent comparator or A/D converter.

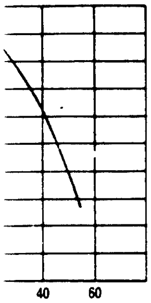
The recommended maximum load on this pin is 20 uA (source only) to maintain electrical and thermal stability. Current loads greater than 20uA may adversely affect performance; however, the output is short circuit proof.

Light Leakage

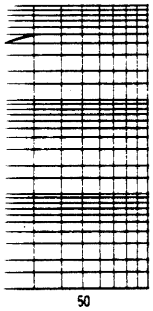
Slight sensitivity to visible light leaking througt the glass-to-metal seal on the base may be observed.

BLOCK DIAGRAM





ɔonse



ıllector Light
ɹ Current

INCHES	
MIN	MAX
).135	0.185
).110	0.130
).080	0.125
).017	0.024
).045	0.055
0.100 BSC	
0.060 BSC	
).009	0.022
).505	0.750
).120	0.130
).030	0.060
).150	0.185

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

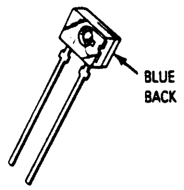
Photo Detector
PIN Diode Output

MRD721

PHOTO DETECTOR
DIODE OUTPUT
100 VOLTS

. . . designed for application in laser detection, light demodulation, detection of visible and near infrared light-emitting diodes, shaft or position encoders, switching and logic circuits, or any design requiring radiation sensitivity, ultra high-speed, and stable characteristics.

- Ultra Fast Response — (<1 ns Typ)
- Sensitive Throughout Visible and Near Infrared Spectral Range for Wide Application
- Annular Passivated Structure for Stability and Reliability
- Economical, Low Profile, Miniature Plastic Package
- Lens Molded Into Package
- Designed for Automatic Handling and Accurate Positioning



CASE 349-01
PLASTIC

MAXIMUM RATINGS (T_A = 25°C unless otherwise noted)

Rating	Symbol	Value	Unit
Reverse Voltage	V_R	100	Volts
Total Power Dissipation @ T_A = 25°C Derate above 25°C (Note 1)	P_D	150 2	mW mW/°C
Operating and Storage Junction Temperature Range	T_J, T_{stg}	−40 to +100	°C
Lead Soldering Temperature (5 sec. max, 1/16" from case) (Note 2)	—	260	°C

ELECTRICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic	Fig. No.	Symbol	Min	Typ	Max	Unit
Dark Current (V_R = 20 V, R_L = 1 MΩ; Note 3) T_A = 25°C T_A = 100°C	3 and 4	I_D	 — —	 0.06 14	 10 —	nA
Reverse Breakdown Voltage (I_R = 10 μA)	—	$V_{(BR)R}$	100	200	—	Volts
Forward Voltage (I_F = 50 mA)	—	V_F	—	—	1.1	Volts
Series Resistance (I_F = 50 mA)	—	R_s	—	8	—	Ohms
Total Capacitance (V_R = 20 V; f = 1 MHz)	5	C_T	—	3	—	pF

OPTICAL CHARACTERISTICS (T_A = 25°C)

Characteristic	Fig. No.	Symbol	Min	Typ	Max	Unit
Light Current (V_R = 20 V, Note 4)	2	I_L	1.5	4	—	μA

Sensitivity (V_R = 20 V, Note 5)	—
	—
		S(λ = 0.8 μm)
		S(λ = 0.94 μm)
			—
			—
				5
				1.2
					—
					—
						μA/mW/cm^2

Response Time (V_R = 20 V, R_L = 50 Ω)	—	$t_{(resp)}$	—	1	—	ns
Wavelength of Peak Spectral Response	6	λ_s	—	0.8	—	μm

Notes: 1. Measured with the device soldered into a typical printed circuit board.
2. Heat sink should be applied to leads during soldering to prevent case temperature from exceeding 100°C.
3. Measured under dark conditions. (H ≈ 0).
4. Radiation Flux Density (H) equal to 5 mW/cm^2 emitted from a tungsten source at a color temperature of 2870 K.
5. Radiation Flux Density (H) equal to 0.5 mW/cm^2.

TYPICAL CHARACTERISTICS

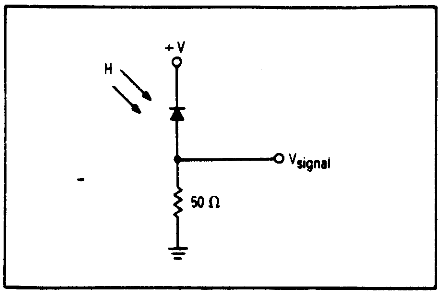


Figure 1. Operating Circuit

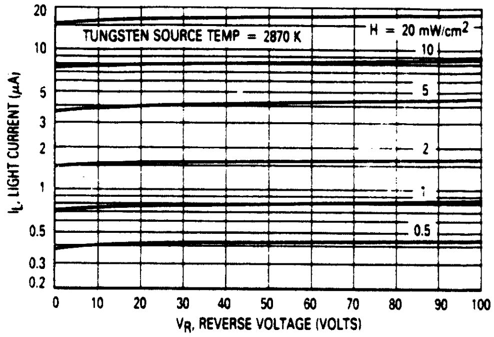


Figure 2. Irradiated Voltage — Current Characteristic

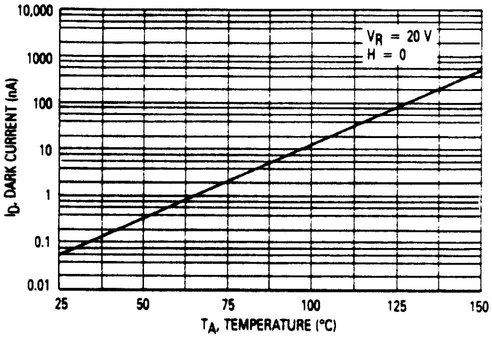


Figure 3. Dark Current versus Temperature

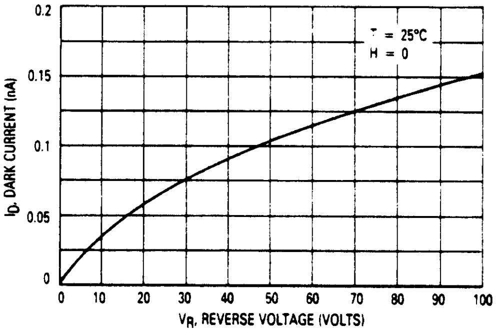


Figure 4. Dark Current versus Reverse Voltage

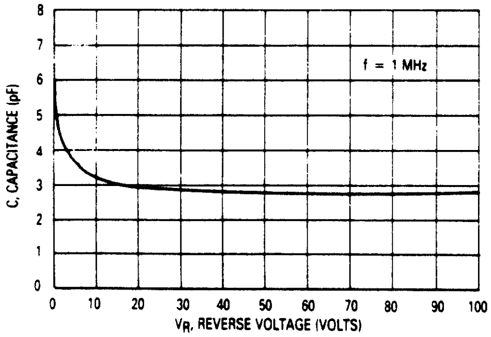


Figure 5. Capacitance versus Voltage

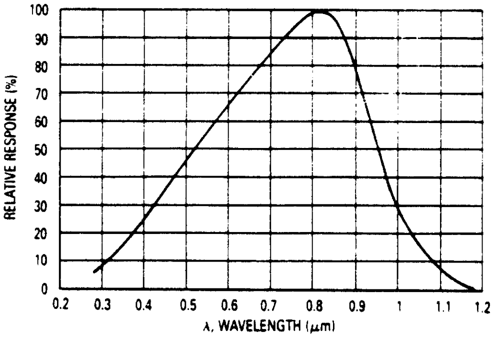


Figure 6. Relative Spectral Response

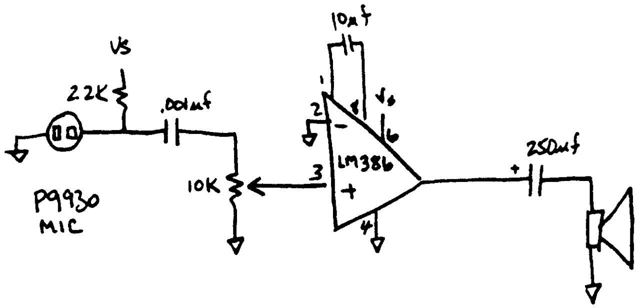
Near-Infrared Emitters and Detectors

Transduction of near-infrared energy can be used as a basis for building fast-response time proximity detectors. As proximity sensors, infrareds have an advantage over sonars in that they work at the speed of light instead of the speed of sound. They also have a much narrower cone of detection. The disadvantage however, is that infrared sensors give almost no ranging information and are very dependent on the surface albedo of what they're looking at. Nevertheless, they're cheap to make and do a fine job for close-in collision avoidance.

IR proximity sensors are made out of two components: a near-infrared LED emitter with a typical wavelength of 880nm, and a matching receiver - either an infrared photodiode or a phototransistor. In order to get around confusion with ambient infrared energy levels, the transmitting LEDs are usually pulsed at a certain frequency and the receiving circuit is designed so as to bandpass filter for that frequency. Shown on the next page is an example from the infrared heads on Herbert, ("The Collection Machine") robot. The tranmitters are Radio Shack XC880A near-infrared LEDs and the detectors are TIL414 phototransistors, also from Radio Shack. In your kits, however, you'll find parts from Siemens (feel free to drop by Radio Shack and get some of their parts) - data sheets are on the next page. Note the detectors given in your kit are photodiodes, not phototransistors, so adjust your circuit accordingly. Also note, in the receiver schematics, that the first stage op-amp circuit is a bandpass filter tuned to the frequency that Herbert's emitters were pulsed. Your design should bandpass for the frequency you choose.

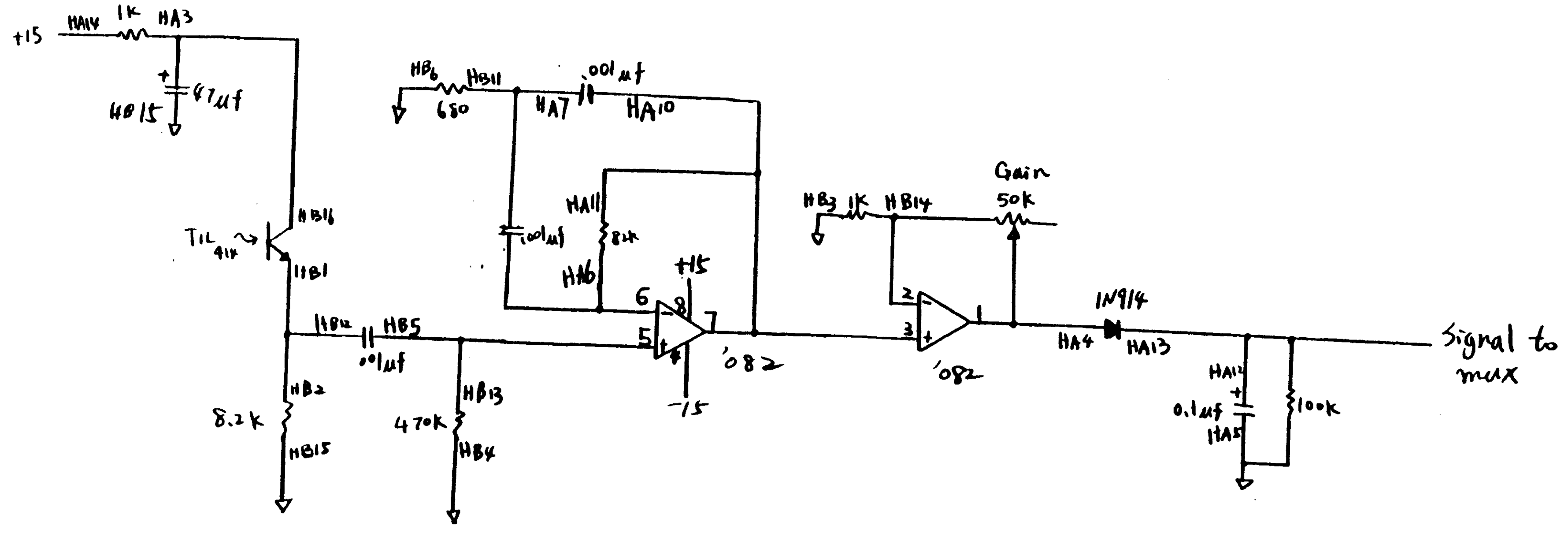
Microphones

Electret microphones are sensitive to acoustic energy and can be useful in simple noise recognition sorts of tasks. The microphones included in the kit come from Digi-Key and are fairly small. They're simple two-wire devices and can be biased and amplified according to the example circuit shown below (Digi-Key catalog, p. 94). Radio Shack also carries microphones. They sell a three terminal device which takes +5V, Ground, and which outputs the signal on the third pin. The signal should be amplified before being interfaced to your microprocessor. Depending on how you want to use this information, there are several options for connecting to the 6811.

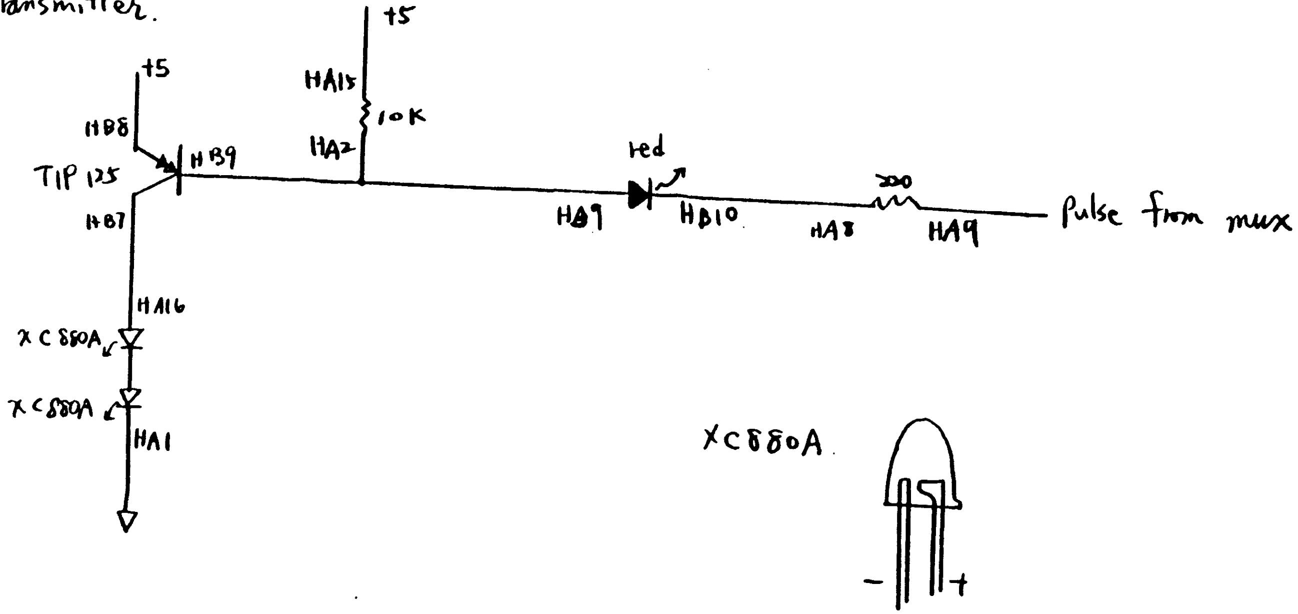


Example circuit for microphone amplifier.

Receiver



Transmitter.



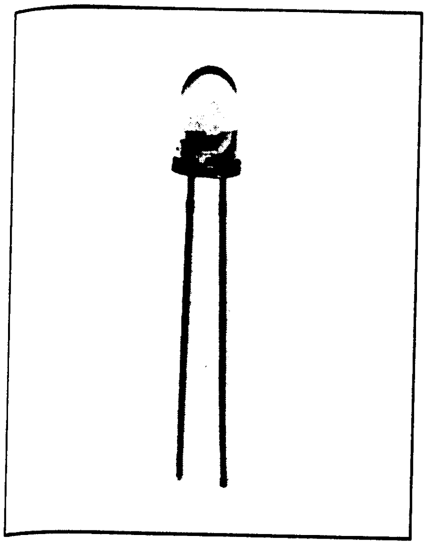
10-15-86

FILE

SIEMENS

SFH 484

GaAIAs INFRARED EMITTER



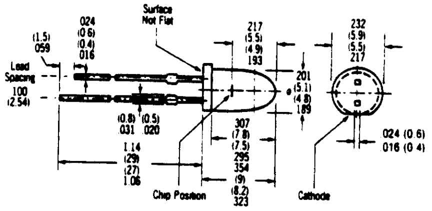
FEATURES

- **Good Spectral Match with Silicon Photo Detector**
- **Gallium Aluminum Arsenide Material**
- **Low Cost**
- **T-1¾ Package**
- **Clear Plastic Lens**
- **Long Term Stability**
- **Narrow Beam, 16°**
- **Very High Power, 20 mW Typical at 100 mA**
- **High Intensity, 100 mW/sr at 100 mA**
- **For Smoke Detection Application: Use SFH484-E7517**

DESCRIPTION

SFH 484, an infrared emitting diode, emits radiation in the near infrared range (880 nm peak). The emitted radiation, which can be modulated, is generated by forward flowing current. The device is enclosed in a 5mm plastic package. Uses for SFH 484 include: IR remote control of color TV receivers, smoke detectors, and other applications requiring very high power, such as IR touch screens.

Package Dimensions in Inches (mm)



Maximum Ratings

Storage temperature	T_{stg}	-55 to +100	°C
Soldering temperature at dip soldering: (≥2 mm distance from the case bottom; soldering time t≤5 sec)	T_{sold}	260	°C
Soldering temperature at iron soldering: (≥2 mm distance from the case bottom; soldering time t≤3 sec)	T_{sold}	300	°C
Junction temperature	T_j	100	°C
Reverse voltage	V_R	5	V
Forward current	I_F	100	mA
Surge current ($\tau = 10$ µs)	i_{SC}	2.5	A
Power dissipation (T = 25°C)	P_{tot}	200	mW
Thermal Resistance*	R_{thA}	375	K/W

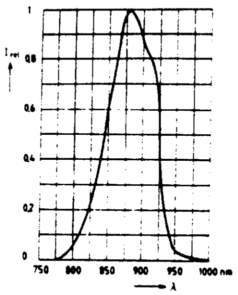
Characteristics ($T_{amb} = 25$°C)

Wavelength at peak emission at $I_F = 10$ mA	λpeak	880	nm
Wavelength at peak emission at $I_F = 100$ mA; $t_{pulse} = 20$ ms, Duty cycle = 1:12	λpeak	883	nm
Wavelength at peak emission at $I_F = 1$ A; $t_{pulse} = 100$ µs, Duty cycle = 1:100	λpeak	886	nm
Spectral bandwidth at $I_F = 10$ mA	Δλ	80	nm
Half angle	φ	±8	degrees
Active chip area	A	0.16	mm²
Dimensions of active chip area	L × W	0.4 × 0.4	mm
Distance chip surface to case surface	D	4.9...5.5	mm
Switching time: (I_e from 10% to 90%; and from 90% to 10% $I_F = 100$ mA)	t_r, t_f	0.6/0.5	µs
Capacitance ($V_R = 0$ V, f = 1 MHz)	C_o	25	pF
Forward Voltage ($I_F = 100$ mA; $t_{pulse} = 20$ ms)	V_F	1.5 (≤1.8)	V
($I_F = 1$ A; $t_{pulse} = 100$ µs)	V_F	3.0 (≤3.8)	V
Breakdown voltage ($I_R = 10$ µA)	V_{BR}	30 (≥5)	V
Reverse current ($V_R = 5$ V)	I_R	0.01 (≤10)	µA
Temperature coefficient of I_e or Φ_e	TC	−0.5	%/K
Temperature coefficient of V_F	TC	−0.2	%/K
Temperature coefficient of λpeak	TC	0.25	nm/K
Radiant intensity I_e in axial direction at a steradian $\Omega = 0.01$ sr or 6.5°			
Radiant intensity ($I_F = 100$ mA, $t_{pulse} = 20$ ms)	I_e	100 (≥50)	mW/sr
($I_F = 1$ A; $t_{pulse} = 100$ µs)	I_e	900	mW/sr
Φ_e (Total) typ. ($I_F = 100$ mA)	Φ_e	20	mW

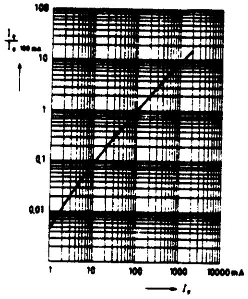
*At 10mm maximum clearance between PC board and bottom of plastic body.

Specifications are subject to change without notice.

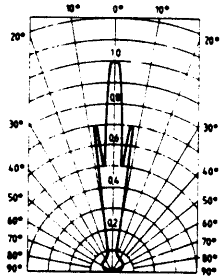
Relative spectral emission
$I_{rel} = f(\lambda)$



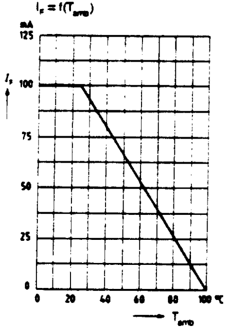
Radiant intensity
$\frac{I_e}{I_e\ 100\ mA} = f(I_F)$



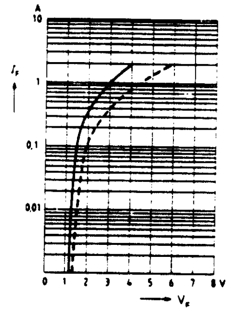
Radiant characteristics
$I_{rel} = f(\varphi)$



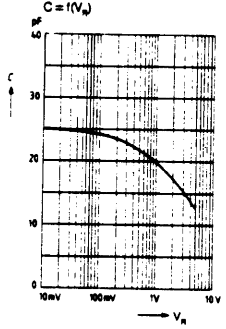
Maximum permissable forward current
$I_F = f(T_{amb})$



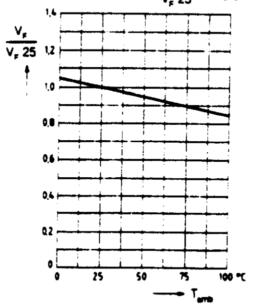
Forward current
$I_F = f(V_F)$



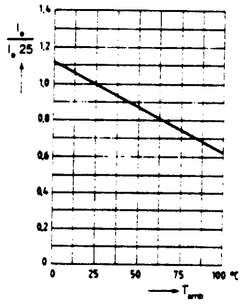
Capacitance
$C = f(V_R)$



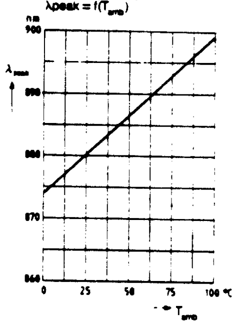
Forward voltage $\frac{V_F}{V_F\ 25} = f(T_{amb})$



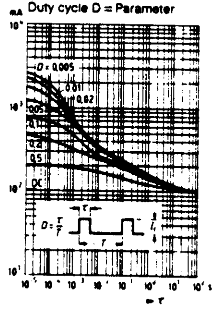
Radiant intensity $\frac{I_e}{I_e\ 25} = f(T_{amb})$



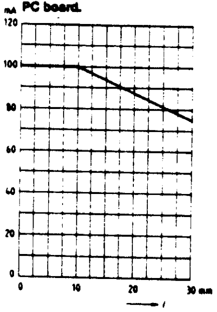
Wavelength at peak emission
$\lambda peak = f(T_{amb})$



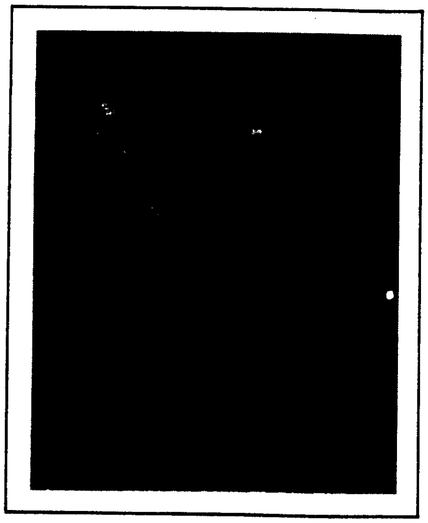
Permissable pulse load
$I_F = f(\tau)$
Duty cycle D = Parameter



Forward current (max): dependent upon the lead length from the package bottom to the PC board.



SFH217
SFH217F
PIN PHOTODIODE



FEATURES

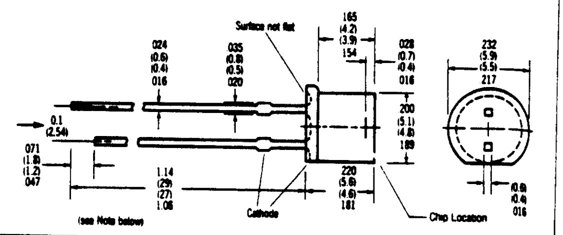
- **Silicon Planar Pin Photodiode**
- **Cost Effective Device**
- **T-1¾ Package**
- **Flat Top**
- **High Speed, 1 ns**
- **Low Dark Current, 1 nA**
- **IR Filter (SFH217F)**

DESCRIPTION

The SFH217 and SFH217F are planar PIN photodiodes in a plastic T-1¾ package with a flat lens. The flat window has no effect on the beam path of optical lens systems. It is characterized by its low junction capacitance and fast switching speeds.

Because of its high cut-off frequency, this diode is particularly suitable for use as an optical sensor of high modulation bandwidth.

Package Dimensions in Inches (mm)



Note: Temporarily these devices may be supplied with lead lengths of $\frac{65\ (16.6)}{62\ (15.8)}$

Maximum Ratings

Reverse voltage	V_R	30	V
Storage/operating temperature range	T	− 40 to + 80	°C
Power dissipation	P_{tot}	100	mW
Soldering temperature (Solder 2 mm distance from case t ≤ 3 sec)	T_L	300	°C

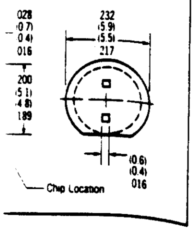
Electrical/Optical Characteristics ($T_{amb} = 25°C$)

		SFH217	SFH217F	
Radiant sensitive area	A	1	1	mm²
Dimensions of radiant sensitive area	LxW	985x.985	985x.985	mm
Distance chip surface to package surface	H	0.4 0.7	0.4 0.7	mm
Wavelength of the max. sensitivity	$\lambda_{S\,max}$	850	900	nm
Quantum yield (Electrons per photon) (λ = 850 nm)	η	0.89	0.89	$\frac{\text{Electrons}}{\text{Photon}}$
Spectral sensitivity (λ = 850 nm)	S	0.62	0.62	A/W
Rise time of the photocurrent (load resistance R_L = 50 Ω; V_R = 5 V; λ = 880 nm, I_p = 14 μA)	t_r	2 (≤ 4)	2 (≤ 4)	ns
Forward voltage (I_F = 100 mZ, E_e = 0, T_A = 25°C)	V_F	1.3	1.3	V
Capacitance (V_R = 0 V, f = 1 MHz, E = 0 lx)	C_0	11	11	pF
Dark current (V_R = 20 V; E = 0)	I_R	1 (≤ 10)	1 (≤ 10)	nA
Photosensitivity (V_R = 5 V, standard light A, T = 2856 k)	S	9.5 (≥ 5)	—	nA/lx
Photosensitivity (V_R = 5 V, λ = 950 nm, E_e = 0.5 mW/cm²	S	—	3.0 (≥ 1.8)	μA
Spectral range of photosensitivity (S = 10% of S_{max})	λ	400 ... 1100	800 1100	nm
Open circuit voltage (E_V = 1000 lx, standard light A, T = 2856 K)	V_O	350 (≥ 300)	—	mV
(E_e = 0.5 mW/cm², λ = 950 nm)	V_O	—	300 (≥ 250)	mV
Short circuit current (E_V = 1000 lx, standard light A, T = 2856 K)	I_S	9.3 (≥ 5)	—	μA
(E_e = 0.5 mW/cm², λ = 950 nm)	I_S	—	3.1 (≥ 1.8)	μA
Noise equivalent power (V_R = 20 V)	NEP	2.9×10^{-14}	2.9×10^{-14}	$\frac{W}{\sqrt{Hz}}$
Detection limit (V_R = 20 V)	D*	3.5×10^{12}	3.5×10^{12}	$\frac{cm\sqrt{Hz}}{W}$
Temperature coefficient for I_S	TC	0.2	0.2	%/K
Temperature coefficient for U_O	TC	− 2.6	− 2.6	mV/k

[1] The illuminance indicated refers to unfiltered radiation of a tungsten filament lamp at a color temperature of 2856 K (standard light A in accordance with DIN 5033 and IEC publ. 306-1)

Specifications are subject to change without notice.

SFH217
SFH217F
PHOTODIODE



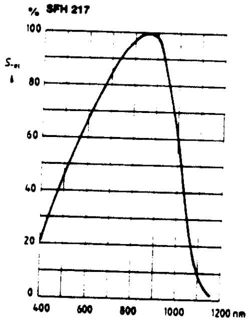
30	V
− 40 to + 80	°C
·00	mW
300	°C

5°C)

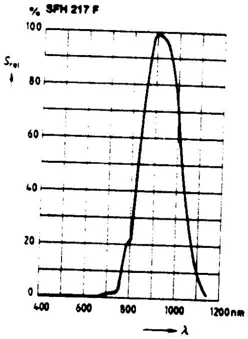
SFH217F	
1	mm²
985x 985	mm
0.4 0 7	mm
900	nm
0 89	Electrons Photon
0 62	A/W
2 (≤4)	ns
1 3	V
11	pF
1 (≤10)	nA
—	nA/lx
3 0 (≥1 8)	µA
)0 800 1100	nm
—	mV
300 (≥250)	mV
—	µA
3 1 (≥1 8)	µA
2 9 x 10⁻¹⁴	W √Hz
3 5 x 10¹²	cm √Hz W
0 2	%/K
− 2 6	mV/K

· amp at a color temperature of

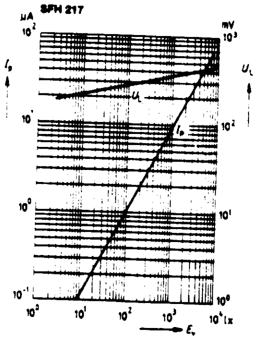
Relative Spectral Sensitivity $S_{rel} = f(\lambda)$



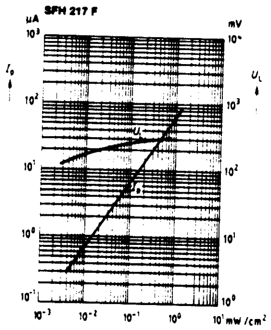
Relative Spectral Sensitivity $S_{rel} = f(\lambda)$



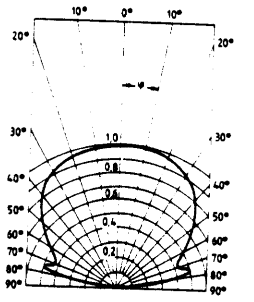
Photocurrent $I_P = f(E_V)$



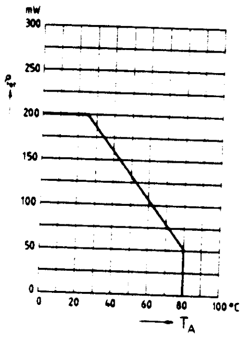
Photocurrent $I_P = f(E_e)$



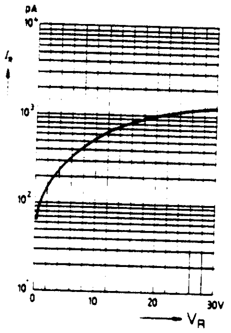
Directional Characteristics $S_{rel} = f(\varphi)$



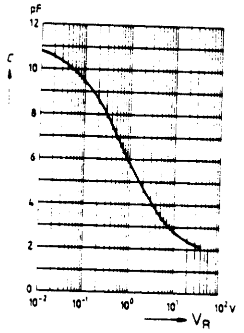
Power Dissipation $P_{tot} = f(T_A)$



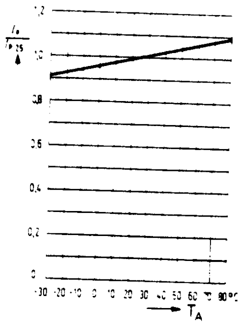
Dark Current $I_R = f(V_R)$



Capacity $C = f(V_R)$
$T_{amb} = 25°C$



Photocurrent $\frac{I_P}{I_{P\,25}} = f(T_A)$



Photodiodes

SIEMENS

BPW 34
PIN PHOTODIODE

FEATURES

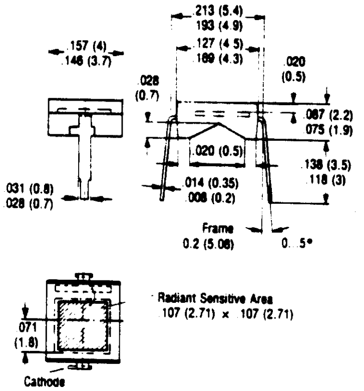
- **Silicon Planar PIN Photodiode**
- **Transparent Plastic Package**
- **2/10″ Lead Spacing**
- **Low Junction Capacitance**
- **Short Switching Time**
- **High Sensitivity**
- **Lead Bend Option (for SMD)**

DESCRIPTION

The BPW 34 is a silicon planar PIN photodiode, which is incorporated in a transparent plastic package. Its terminals are soldering tabs arranged in 5.08 mm (2/10″) lead spacing. Due to its design the diode can also very easily be assembled on PC boards. The flat back of the epoxy resin case makes rigid fixing of the component feasible.

Arrays can be realized by multiple arrangements. This versatile photodetector can be used as a diode as well as a voltaic cell. The signal/noise ratio is particularly favorable, even at low illuminances. The open circuit voltage at low illuminances is higher than with comparable mesa photovoltaic cells. The PIN photodiode is outstanding for low junction capacitance, high cut-off frequency and short switching times. The photodiode is particularly suitable for IR sound transmission.

Package Dimensions in Inches (mm)



Maximum Ratings

Reverse Voltage (V_R)	32 V
Operating and Storage Temperature Range	-40 to +80°C
Soldering Temperature in a 2 mm Distance from the Case Bottom ($t \leq 3$ s) (T_S)	230°C
Power Dissipation (T_{amb} = 25°C) (P_{tot})	150 mW

Characteristics (T_{amb} = 25°C)

Photosensitivity (V_R = 5 V, Note 1)	S	80 (≥50)	nA/lx
Wavelength of Max. Photosensitivity	λ_{Smax}	880	nm
Spectral Range of Photosensitivity (S = 10% of Smax)	λ	400...1100	nm
Radiant Sensitive Area	A	734	mm²
Dimensions of the Radiant Sensitive Area	L × W	2.71 × 2.71	mm
Distance Between Chip Surface and Package Surface	H	0.5	mm
Half Angle	φ	±60	Deg.
Dark Current (V_R = 10 V)	I_R	2 (≤30)	nA
Spectral Photosensitivity (λ = 850 nm)	S	0.62	A/W
Quantum Yield (λ = 850 nm)	η	0.90	$\frac{\text{Electrons}}{\text{Photon}}$
Open Circuit Voltage (E_V = 1000 lx, Note 1)	V_O	365 (≥300)	mV
Short Circuit Current (E_V = 1000 lx, Note 1)	I_{SC}	80 (≥50)	μA
Rise and Fall Time of the Photocurrent from 10% to 90% and from 90% to 10% of the Final Value (R_L = 1 KΩ, V_R = 5 V, λ = 830 nm I_P = 70 μA)	t_r, t_f	350	ns
Forward Voltage (I_F = 100 mA, E_e = 0 T_{amb} = 25°C)	V_F	1.3	V
Capacitance (V_R = 0 V, E = 0, f = 1 MHz)	C_0	72	pF
Temperature Coefficient of V_O	TC_V	-2.6	mV/K
Temperature Coefficient of $I_K$or I_P	TC_I	0.18	%/K
Noise Equivalent Power (V_R = 10 V)	NEP	4.1×10^{-14}	$\frac{W}{\sqrt{Hz}}$
Detection Limit (V_R = 10 V)	D	6.6×10^{12}	$\frac{cm \sqrt{Hz}}{W}$

[1] The illuminance indicated refers to unfiltered radiation of a tungsten filament lamp at a color temperature of 2856 K (standard light A in accordance with DIN 5030 and IEC publ. 306-1).

Specifications are subject to change without notice.

BPW 34
PHOTODIODE

)20
) 5)
087 (2 2)
075 (1 9)
138 (3 5)
118 (3)

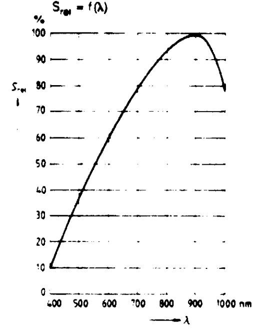
5°

32 V
-40 to +80°C

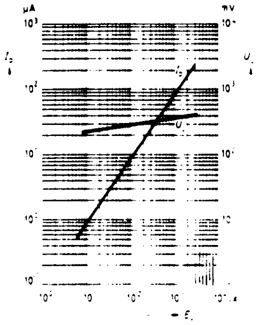
230°C
150 mW

90 (≥50)	nA/lx
880	nm
400...1100	nm
734	mm²
71 × 2 71	mm
0 5	mm
±60	Deg.
2 (≤30)	nA
0 62	A/W
0 90	Electrons / Photon
55 (≥300)	mV
30 (≥50)	μA
350	ns
1 3	V
72	pF
-2 6	mV/K
0 18	%/K
× 10⁻¹⁴	W / √Hz
6 × 10¹²	cm √Hz / W

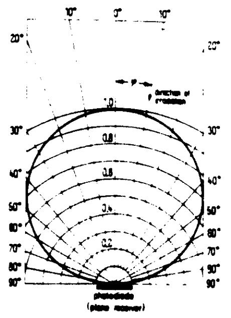
ment amp at a color
and IEC publ 306-1)



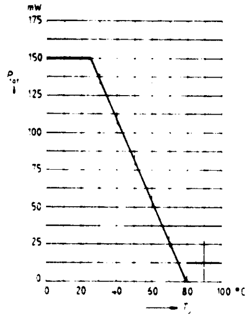
Relative spectral sensitivity
$S_{rel} = f(\lambda)$



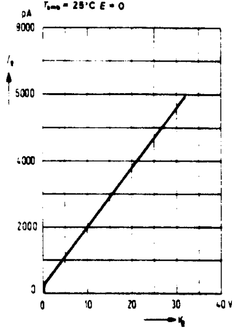
Photocurrent $I_p = f(E_V)$
Open circuit voltage $V_L = f(E_V)$



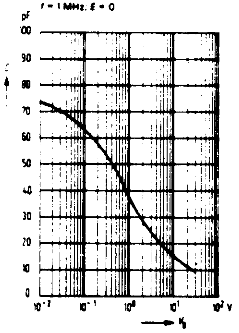
Directional characteristic $S_{rel} = f(\varphi)$



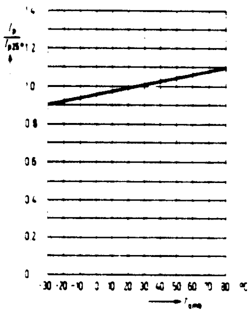
Power dissipation $P_{tot} = f(T_{amb})$



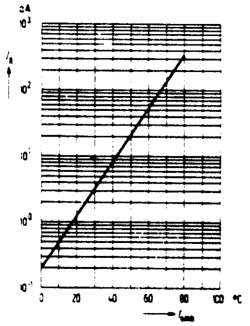
Dark current $I_R = f(V_R)$
$T_{amb} = 25°C$ $E = 0$



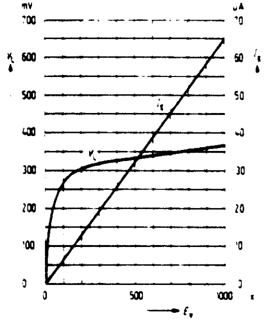
Capacitance $C = f(V_R)$
$f = 1$ MHz; $E = 0$



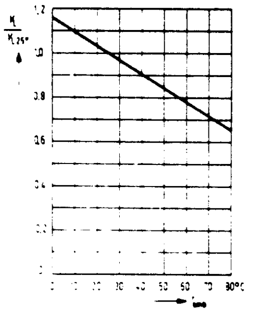
Photocurrent $\frac{I_p}{I_{p\,25°}} = f(T_{amb})$



Dark current $I_R = f(T_{amb})$
$V_R = 10$ V $E = 0$



Open circuit voltage $V_L = f(E_v)$
Short circuit current $I_K = f(E_v)$



Open circuit voltage $\frac{V_L}{V_{L\,25°}} = f(T_{amb})$

You may want to use the A/D port, or alternatively, you may want to do some timing and interface the microphones to the timer pins. An example of the latter is shown in the Squirt chapter. Your 6811, running with an 8Mhz signal should be able to measure time steps of 0.5 microseconds, which should be good enough for two microphones to discriminate orientation to a noise source.

Photocells

Photocells are variable resistors that change their resistance in proportion to the incident light. Photocells are available from Radio Shack, as are a wide variety of sensors. Browse through a Radio Shack catalog for ideas. An example of a robot which uses photocells to avoid dark corners and run towards light spots is given in a later chapter on Photovore, The Light-eater.

Inclinometers

Inclinometers measure the angle of tilt of a body. Typically, they measure tilt along one-axis, but sometimes two. Inclinometers can range from something as simple as a ball rolling in a curved tray to electrolytic fluid-based sensors. Inclinometers are useful for walking machines, flying objects or any machine not restricted to a flat plane. There are only a few electrolytic inclinometers, so see me if you need one for your project.

Mercury Switches

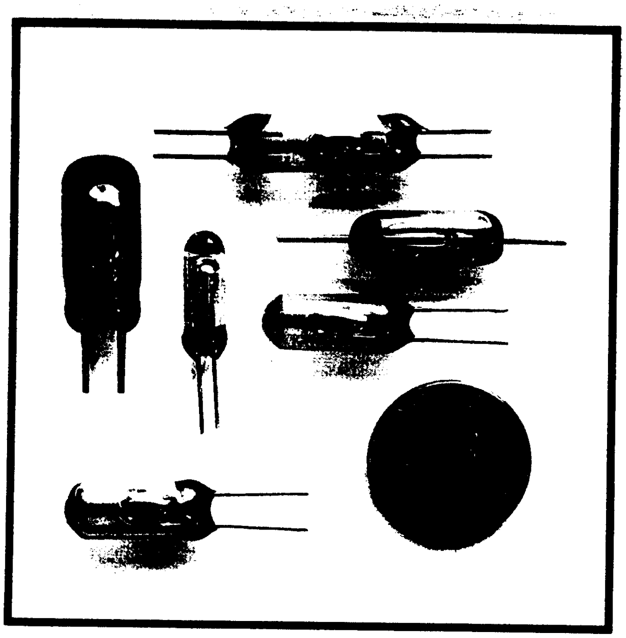
Mercury switches are inclinometers that produce only one bit of tilt information. Consequently, they're much cheaper than inclinometers. Put a bunch of them together however, and you can get a spectrum of information. A mercury switch is basically a switch closed by a flow of mercury. Consequently, if your vehicle tips, you can discern that fact if an upright mercury switch is connected to one of the 6811's port pins.

Cameras

Cameras are more expensive than most of the sensors listed here, but they're getting cheaper all the time. The best buy in town for a video camera has got to be the Pixar 2000, sold by Toys 'R Us for $99. It's a low-resolution, slow-scan system that records video and audio information onto a normal (but fast running) audio cassette. You get approximately 5 minutes of recording information per side, but it really works and you can play it back on a regular television set.

If you're looking for something smaller, Sony Watchcams are available in pocket size, with good resolution and normal NTSC video output for about $800. Single chip video digitizers have just come out too (Analog Devices AD9520); so extremely small video systems are possible, but it's a bit of a project.

I recommend running a video cable offboard to either the Sun, Macintosh or Symbolics-based frame grabbers we have around the lab. Run your code there and send commands out the serial port to the 6811 to control your robot. This way you'll have plenty of computer power to do image processing and very little new hardware will need to be built.



spectron
glass and electronics incorporated
595 Old Willets Path, Hauppauge, N.Y. 11788
Telex 968.296 (516) 582 5600

Automatically Produced MINIATURE MERCURY SWITCHES

GENERAL INFORMATION

SPECTRON mercury switches are Small, Silent, Sealed, Safe and Low Cost.

★ Small – less than one inch long and under half an inch in diameter, they are capable of switching currents of 0.5 to 2 amps.

★ Silent – quietly switching with tilt, there is no sound and no electrical "noise" as the mercury moves to close or open a circuit.

★ Sealed – hermetically sealed, they are immune from dirt, water, oil, and other contamination.

★ Safe – sealed and filled with an inert gas, they cannot generate exposed electrical arcs during switching.

★ Low cost – made in volume on automatic sealing equipment, they are low in initial cost. Because of their long useful life and high reliability, they also have a low useful life-cost.

Ideal for use in hostile environments, these miniature tilt switches will not degrade in switching action or develop "hot spot" high resistance contacts. Neither will they generate reactive shock forces normally encountered in spring loaded switches. Absence of this shock adds to the life of surrounding electronic assemblies and components as well as to the life of the mercury switch.

SWITCH SELECTION:

In selecting a miniature mercury switch from the accompanying table, the primary consideration is the level of current to be switched. The M 1205 tilt switch has been designed to switch 2 amps at 220 volts with a resistance load. As such, it is most used for large power applications.

For low power applications such as PC board power supplies, the M 1204 switch is appropriate because of its low contact resistance.

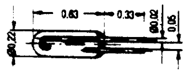
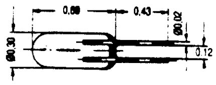
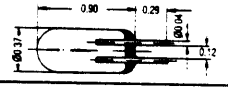
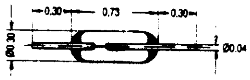
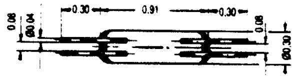
The M 1207 is a switch to be considered for applications requiring switch closure in the inverted as well as the upright position. By commoning one electrode on either side, the switch becomes a SPDT device.

Our engineers are always available to answer any question you may have regarding the application and installation of our miniature mercury switches.

Whenever there is a need for a versatile switching device actuated by a change in attitude, choose a SPECTRON mercury switch. These rugged, miniature mercury switches will provide hundreds of thousands of switching cycles in a power efficient and reliable manner and greatly add to the life of your equipment.

spectron
glass and electronics incorporated
595 Old Willets Path, Hauppauge, N.Y. 11788
Telex 968.296 (516) 582 5600

TECHNICAL DATA

Ordering Reference	Contact Form	Max. Switching Current (A)	Max. Carrying Current (A)	Max. Switching Voltage (Vrms)	Min. Switching Voltage (Vrms)	Max. Switching Capacity (W)	Max. Contact Resistance (mΩ)	Min. Differential Angle (°)	Dimensions
M 1201	SPST	0.5	1	115	10	25	300	15	
M 1202	SPST	1	2	220	10	100	300	10	
M 1204	SPST	0.5	2	220	0.1	20	50	15	
M 1205	SPST	2	4	220	10	200	150	15	
M 1206	SPST	2	4	220	10	100	100	15	
M 1207	SPDT	1	2	220	10	100	300	15	
M 1216	SPDT	0.5	2	220	0.1	20	50	15	see dimensions below

Notes:

Min. Differential Angle (MDA):

This is the included angle between the "just closed" and "just open" positions. For long term, high reliability of closure, a change in angle of 1.5 to 2 times the MDA is recommended.

Terminals:

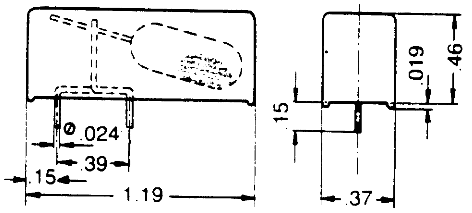
All terminals are solid, easily soldered to leads, 0.022'' diameter min. and extend out from the body of the level a min. of 0.30''. As an option, switches can be supplied with leads as specified by the user.

Mounting & Encapsulating:

Standard clip type mountings are available as well as potting or encapsulation services.

Technical Services:

In addition to providing technical application information, SPECTRON provides design and fabrication services for unique switches and interface hardware.

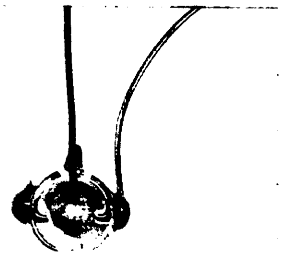


Ref. 1216
Available from stock in
plastic housing and specifically
designed for PCB mounting

ELECTROLYTIC TILT SENSOR L-211 U

This SPECTRON series of vertical sensing electrolytic potentiometers is composed of miniature single-axis units that provide a linear voltage output as the unit is tilted about the horizontal axis. The unit is a onepiece glass enclosure, with platinum terminals and contacts sealed flush into the glass walls, thus eliminating any drain off of electrolyte from protruding contacts. Each sensor comes wired with all terminals and wire connections covered by gyro grade epoxy cement. As such, it qualifies as a truly hermetically sealed tilt sensor.

The series has characteristics similar to the L-210 series, but also has a dampening orifice which controls the rate at which the fluid responds to sudden angular inputs. This improves operation in moderate dynamic environments.



TECHNICAL CHARACTERISTICS L-211 U (at 20 Degrees C) **(11 C 4100)**

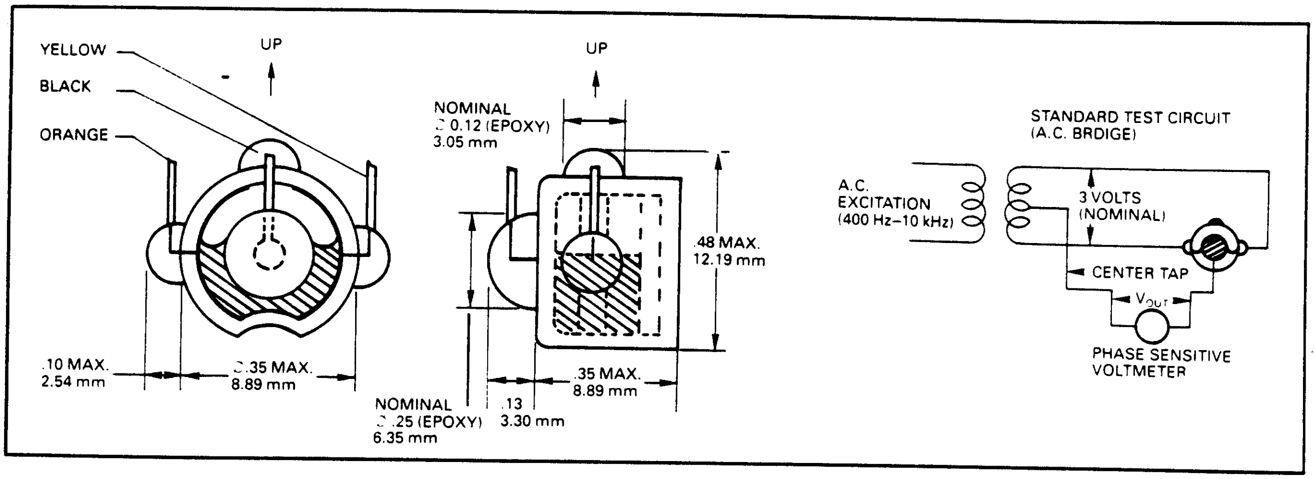
	STANDARD	AVAILABLE
1. Tilt Angle Range (Degrees)	± 60	—
2. Output (mV / Degree / Volt Excitation)	7.2 ± 20%	± 5%
3. Total Null (mV at 3 V 400 Hertz Excitation)	2	0.5
4. Null Repeatability (Degrees)	0.03	0.008
5. Repeatability at any angle (Degrees)	0.03	0.008
6. Linearity, Full Scale (% of Full Scale)	5	2
1/2 Scale (% of Full Scale)	2	1
7. Symmetry at 1/2 Scale (%)	5	2
8. Time Constant (Seconds)	0.3 ± 25%	—
9. Impedance, end to end (K Ohms)	6 ± 20%	± 5%
10. Excitation Voltage (Volts AC)	0.5 to 5	—
11. Excitation Frequency (Hertz)	20 to 20,000	—
Operating Temperature Range (Degrees C)	− 40 to + 80	
Storage Temperature Range (Degrees C)	− 55 to + 100	
Dimensions: See Drawing		
Weight: 2 Grams		

NOTE: See Graphs on Other Side for Additional Characteristics

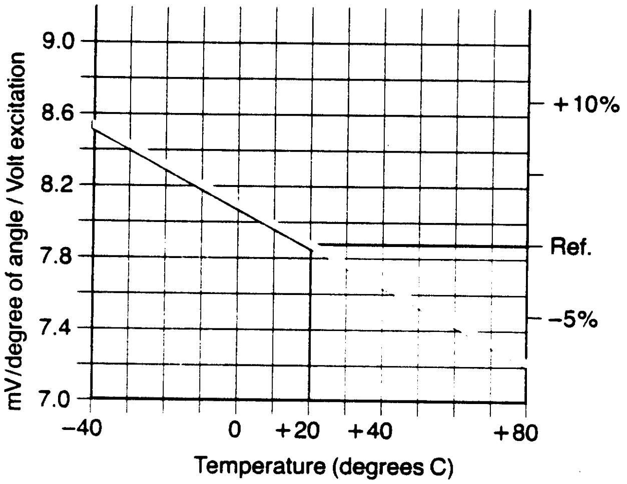
Time Constant Vs. Temperature:

Temperature (Degrees C)	Time Constant (Seconds)
+ 80	<0.1 ± 25%
+ 20	<0.1 ± 25%
− 40	0.1 ± 25%

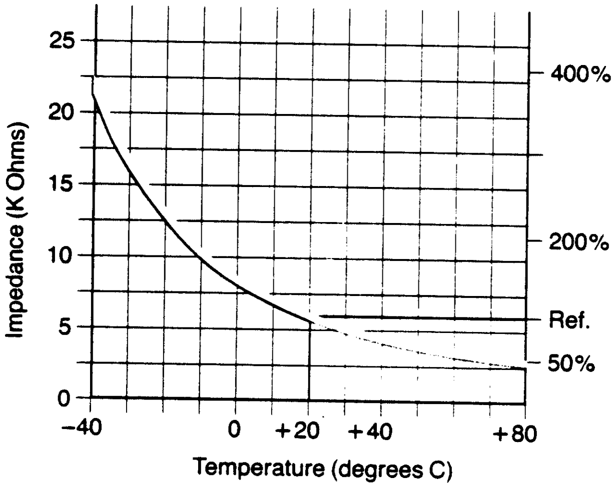
TECHNICAL DATA L-211 U



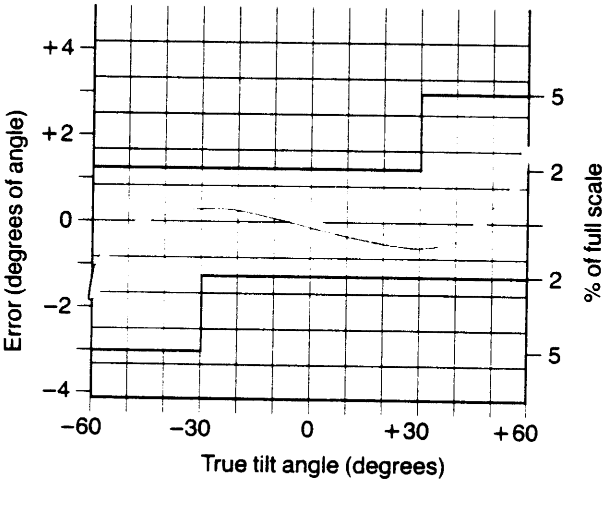
Scale factor vs temperature



Null impedance vs temperature



Accuracy



Using the Polaroid Ultrasonic Ranging Sensor

Maja Mataric

The Polaroid Ranging Sensor is one of the simplest and best sensors available for directly measuring distances to objects. The sonar can be used to measure the distance to the nearest point within a 30-degree cone and can deliver range information from 0.9 to 35 feet. The sensor system supplied by Polaroid consists of a transducer and a controller board. The boards we are supplying are single-frequency boards, not to be confused with the four-frequency boards also described in the available Polaroid documentation.

A complete manual for the Polaroid sonar transducer is included in your kit and you should refer to it for complete details. What follows is a quick overview.

The Polaroid sonar controller board receives a signal to ping the sonar from your 6811 microprocessor (this signal is called VSW in the Polaroid documentation). The sonar board returns a flag (called XLG) which identifies the exact instant at which the transducer actually fires a burst of acoustic energy. Another flag (called FLG in the documentation), is returned by the sonar board when an echo is detected. Connect these three wires to three port pins on your microprocessor and then write appropriate software to assert VSW, which will direct the controller board to ping the transducer. Then simply measure the time interval between the arrival of the XLG and FLG flags. By multiplying this time by the speed of sound, software can determine distances to obstacles. Measurement of this time interval can be facilitated by using the 6811's timer inputs on port A to interface to XLG and FLG.

Assembling the Sonar

Connect the transducer to the controller board with a coax cable (Figure 1). Connect the three flags (green, blue, and red) to your 6811 microprocessor. Use a pull-up resistor on the flags indicating the signas have returned (e.g. 2.2K Ohms).

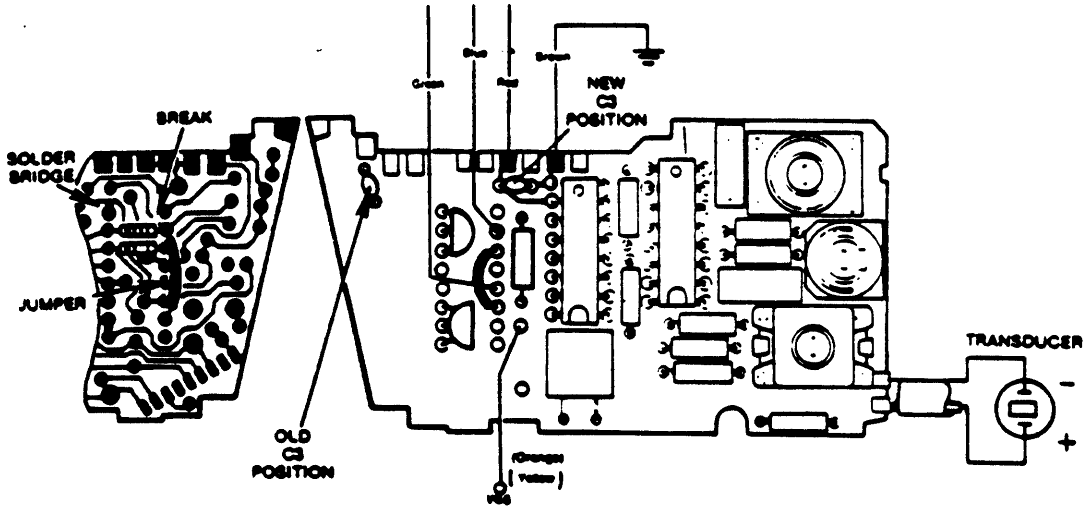


Figure 1. The Polaroid sonar controller board.

Power Supply and Related Issues

The Polaroid transducer requires 2.5A of current for about a millisecond in order to ping (about 250ma when not pinging). The sonar board requires a 6V power supply whereas 5V is required for the microprocessor. Either use separate power supplies, or a voltage regulator, to get around the different power requirements. Additionally, a transistor is needed between the microprocessor and the sonar board in order to amplify the signal telling the board to ping.

Do not apply power to the sonar board if the transducer is not connected.

The VSW signal telling the sonar to ping must be timed properly. It should be high for 100ms and low for a minimum of another 100ms. (See the waveforms in Figure 2). If VSW is applied for too long, this will probably send a load to the transducer which pulls the current for longer than a millisecond. Consequently, the MPS-A14 Motorola transistor on the Polaroid board (#22 in Figure 3) will promptly fry. This may happen without you realizing it. The transistor may fry without emitting the telltale smoke and noxious fumes, and make debugging more difficult. If you do fry it, rather than just soldering a new one directly, consider attaching a socket onto the Polaroid board for easier removal in case of future failures of the same component.

LC is the reference.
LE min. is the earliest echo received, .9 feet (1.6ms).
LE max. is the furthest echo received 35.0 feet (62.2ms).
OF is 177.8ms from LC.

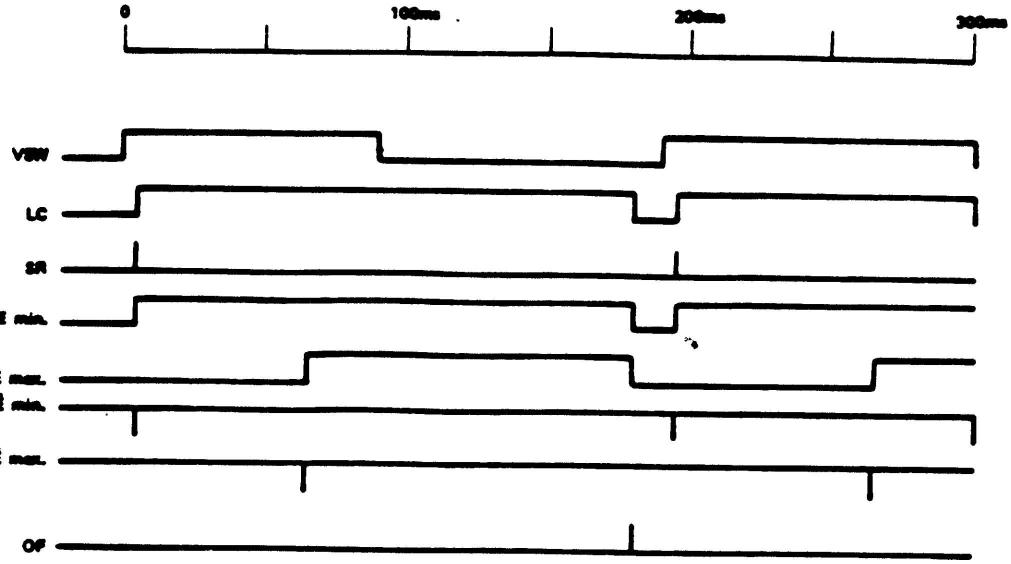
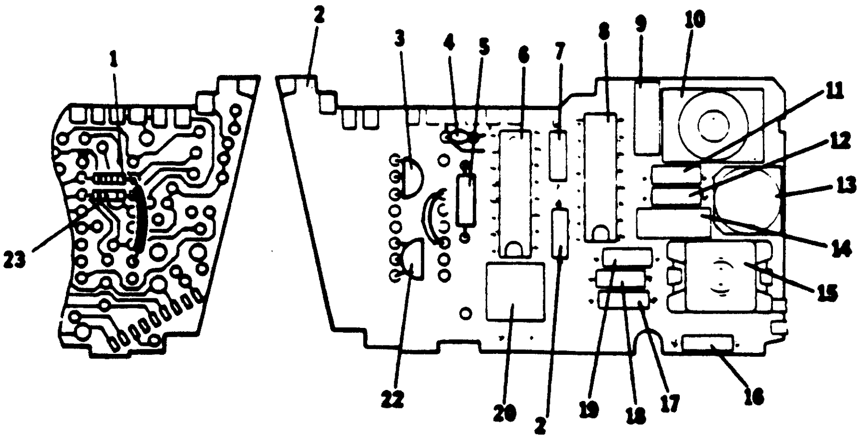


Figure 2. Timing diagram for Polaroid controller board signals.

General Debugging Tips

1) Power-related problems are not unusual with the Polaroid sonar. Always be sure that the sonar board is receiving the proper voltage, and that the VSW signal is strong enough.

2) Build good connectors.



ITEM	REF.	DESCRIPTION
1	R10	RESISTOR (22K 5%, ¼W)
2		P.C. BOARD
3	Q1A	TRANSISTOR 2N-4401
4	C3	CAPACITOR 1μf, 35V
5	R9	RES. 150 Ω
6	U2	DIGITAL IC
7	C4	CAPACITOR .001μf, 10V
8	U1	ANALOG IC
9	C1	CAPACITOR .01μf, 10V
10	L1	TUNED CIRCUIT INDUCTOR
11	C2	CAPACITOR .01μf, 10V

ITEM	REF.	DESCRIPTION
12	R4	RESISTOR (62K, 5%, ¼W)
13	R6	VARIABLE RESISTOR (25K, ¼W) (see note 2)
14	C5	CAPACITOR .0022μf, 400V
15	T1	TRANSFORMER
16	R1	RESISTOR (1-10K, 5%, ¼W) (see note 1)
17	CR1	DIODE ZENER (IN4006)
18	CR2	DIODE ZENER (IN4006)
19	R2	RESISTOR (130K, 5% ¼W)
20	C10	CAPACITOR .01μf, 10V (see note 2)
21	XTAL	CRYSTAL 420 kHz
22	Q2A	TRANSISTOR MPS-A14 MOTOROLA
23	R11	RESISTOR (2.2K 5%, ¼W)

Figure 3. Parts placement on the Polaroid controller board.

The Futaba Gyro

Paul Viola

The Futaba gyro is a one axis rate gyro. It contains a single gyro whose rotation axis is perpendicular to the intended rntation of the chassis. As the chassis is rotated, the gyro experiences a torque that is dependent on the rate of rotation. Since the gyro is spring mounted, its deflection angle is linearly related to that torque. The deflection angle is measured and is interpreted as the rate of rotation of the chassis.

Pulse Width Encoding

The gyro assembly is controlled by a small box packed full of complex analog and digital hardware. This gyro assembly needs to be driven with a pulse width modulated signal in order to run. In return, it emits a pulse width modulated signal signifying the rate of turn. That is, the Futaba gyro controller takes a pulse encoded value and modifies it (increasing or decreasing the pulse width) based on the current rate of rotation.

The Futaba standard pulse width encoded output signal is a 20 msec cycle where 1500 micro seconds ON is the center of the range that can be measured (i.e. a 50 hertz square wave with a 7.5 percent duty cycle). A smaller ON period (lower duty cycle) implies smaller pulse width encoded values while a larger ON period (or higher duty cycle) implies a larger pulse encoded value (the range is approximately 1000-2000 micro secs).

Interface to the 6811

To interface a 6811 to the gyro controller amplifier, the 6811 needs to provide a 1.5ms wide pulse train tuned to 50 hertz. This provides the base from which the rate of rotation is measured. The output is then another square wave at 50 hertz with a modified pulse width. This width corresponds to the rate of rotation.

The gyro has two inputs and one output. The input of interest is labeled RX 1-4 on the data sheet. It is normally connected to a radio controlled receiver for a model airplane. This should be connected to your 50 hertz 1.5ms pulse generator (one of the pins on your 6811 which you've programmed). The output, labeled SX, should be connected to one of the 6811's timer input capture lines.

On each of the connectors, the output signal is white, ground is black and red is power.

Snip off the external battery power lines (unless you have a 6 volt battery power supply around). You will power the the gyro through the red and black power lines on the RX 1-4 input port. The 6811 has a sophisticated set of timers and a timer output compare register can be used to generate the base 1.5ms signal. This should be a straightforward use of some simple interrupts. Similarly, the timer input capture register can be used to measure the width of the incoming pulses; another simple interrupt application. Both of these facilities are described in the 6811 data sheet's section on timers.

Adjustments

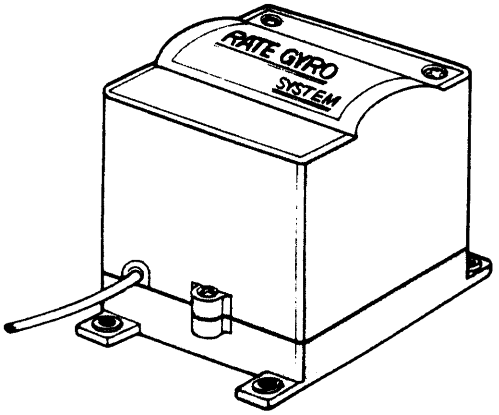
The documentation included with the gyro describes some adjustments such as gain, center and reverse. The gain adjustment controls the amount of amplification. Center controls the neutral or center of the pulse width encoding (what I have been assuming is 1.5ms). Reverse inverts the effect on the pulse width of a particular rotation (this is more important when used in a model aircraft - you could do the same by changing a sign somewhere in your program).

Warnings

Readings are not especially reliable. The device is a *rate* gyro; it is useless for determining angular position over even a small period of time. Even when stationary, the gyro will "drift" and additionally, there is a hysteresis effect.

Futaba®

**DIGITAL PROPORTIONAL
RADIO CONTROL**



INSTRUCTION MANUAL

SINGLE AXIS RATE GYRO

FP-G152 (For J,M, and SG Series)
FP-G132 (For E, F, G, H, and L Series)

FUTABA CORPORATION OF AMERICA
FUTABA CORPORATION

D60226

Thank you for buying a Futaba digital proportional radio control set.
Please read this manual carefully before using your new set.

FUTABA CORPORATION OF AMERICA
555 West Victoria Street Compton Calif. 90220 U S A
Phone 213 537 9610 Telex 23-0691227 Facsimile 213 637 8529
FUTABA CORPORATION
Tokyo Office Daido Bldg., 3-1-16, Sotokanda, Chiyoda ku, Tokyo, Japan
Phone (03) 255 5881 Telex J26532

Printed in Japan 851120

The FP-G132/G152 is a single axis rate gyro designed to stabilize aircrafts. Like full size aircrafts, stabilisation is accomplished by detecting angular acceleration with the rate gyro. Detected motion information is fed to the control amplifier, which then sends a counteraction signal to the appropriate control surface.

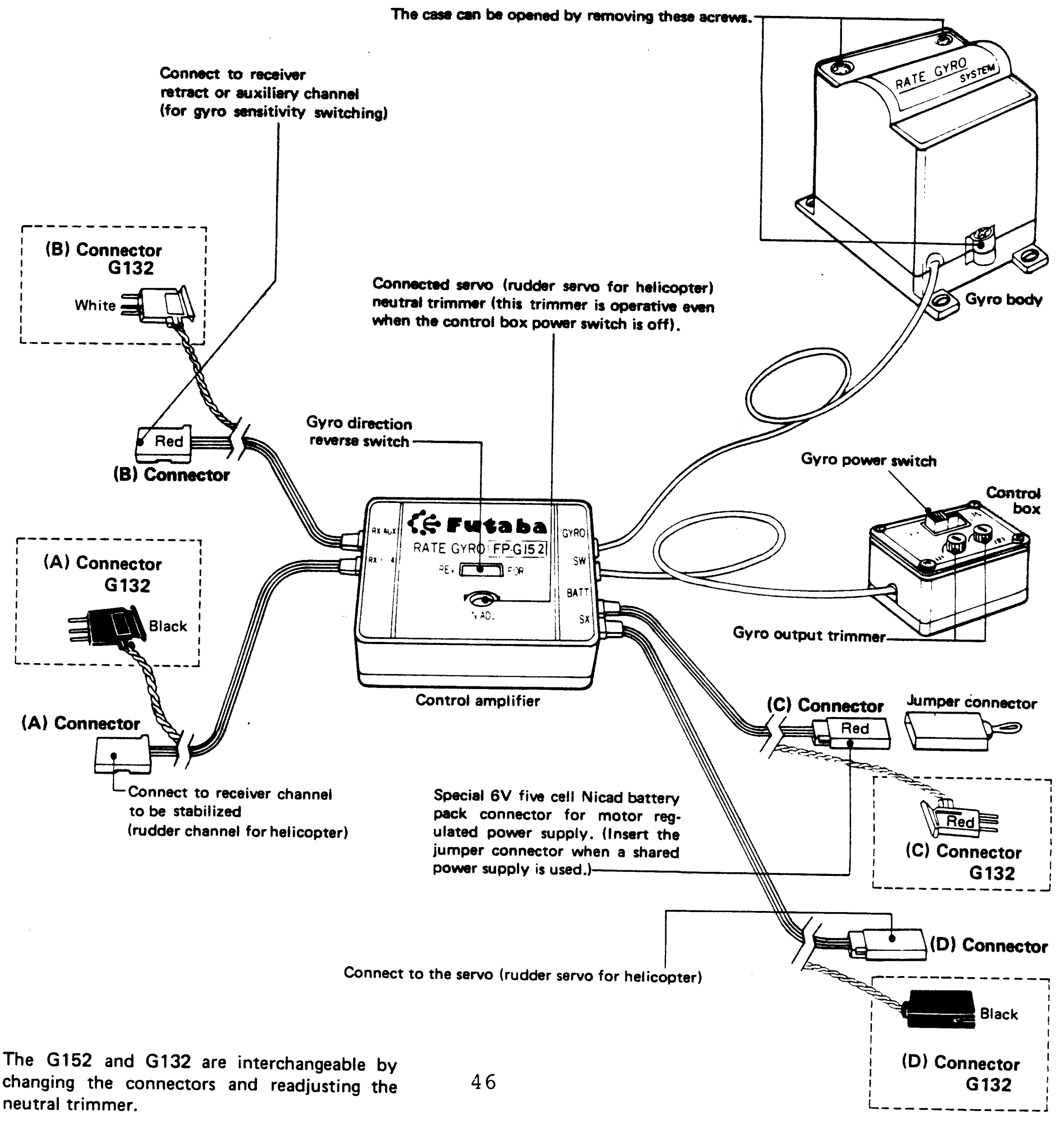
FEATURES OF FP-G152
FEATURES OF FP-G132

- The FP-G152 is for Futaba J, M, and SG Series (1520 μs, neutral) digital proportional radio control sets.
- The FP-G132 is for Futaba E, F, G, H, and L Series digital proportional radio control sets.
- Voltage regulated gyro motor supply maintains constant motor speed and allows consistent gyro performance. The voltage regulator is effective only when used with an external 6V. (five cell Nicad battery).
- Direction of correcting mix can be switched at the control amplifier (internal reverse amp switch).
- Centering of the channel being stabilized can be adjusted by a neutral trimmer built into the control amplifier.
- A very sensitive magnetic motion sensor with excellent voltage characteristics, linear sensitivity, high speed response is used. This results in superior neutral characteristics. Such characteristics make it ideal for use in the rudder channel of a model hilicopter or in the aileron/elevator channel of a model aircraft.
- Large 2mm diameter gyro motor shaft for long life and strength.
- The gyro can be bypassed without affecting normal operation by turning the gyro power switch "off".
- Mount the gyro with grommets and screws or use two-sided foam tape.
- Gyro output ("sensitivity") can be switched to one of two preset outputs at the transmitter (use the retract switch if possible, or any avail. channels).

RATEINGS

Power supply voltage	4.8V shared with receiver (6V for external supply)
Current drain	Motor: 100mA, Amplifier: 20mA (at 4.8V)
Dimensions and weight	Gyro body: 1.57 x 1.65 x 1.69 in.(40 x 42 x 43mm) - 2.86 oz.(80g)
	Control amplifier: 1.73 x 2.28 x .63 in.(44 x 58 x 16mm) - 1.61 oz.(45g)
	Control box: .94 x 1.34 x .59 in.(24 x 34 x 15mm) - .54 oz.(15g)

CONNECTIONS


The case can be opened by removing these acrews.
RATE GYRO SYSTEM
Gyro body
Connect to receiver retract or auxiliary channel (for gyro sensitivity switching)
(B) Connector G132
White
Red
(B) Connector
Connected servo (rudder servo for helicopter) neutral trimmer (this trimmer is operative even when the control box power switch is off).
Gyro direction reverse switch
Futaba
RATE GYRO FP-G152
RX AUX
RX
REV
FOR
N.AD.
GYRO
SW
BATT
SX
Control amplifier
Gyro power switch
Control box
Gyro output trimmer
(A) Connector G132
Black
(A) Connector
Connect to receiver channel to be stabilized (rudder channel for helicopter)
(C) Connector
Red
Jumper connector
Special 6V five cell Nicad battery pack connector for motor regulated power supply. (Insert the jumper connector when a shared power supply is used.)
Red
(C) Connector G132
(D) Connector
Connect to the servo (rudder servo for helicopter)
Black
(D) Connector G132

The G152 and G132 are interchangeable by changing the connectors and readjusting the neutral trimmer.

The Zemco Flux-Gate Digital Compass

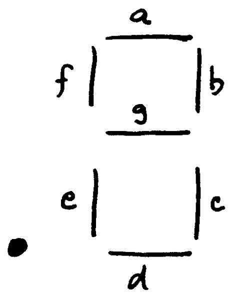
Peter Ning

The Zemco Digital Compass will provide a reasonably accurate measure of orientation with respect to the earth's magnetic field. You can extract 4-bits of resolution from the compass. There are basically two ways of interfacing the compass with your main CPU, either digitally or analog; the former requires no hardware modification to the compass electronics but new data may take up to 3 second to stabilize. The latter method provides data on the order of 1/10 second but requires another level of data translation from analog to digital. Here are all the hints you need to use this compass:

1. Digital Interface Method

There is essentially one chip in the compass electronics that you need to be concerned with, the microprocessor (COP421-MLA). This processor sends out three signals D0 (Pin 24), SK (Pin 16), and SD (Pin 15). D0 is low when SK and SD signals are valid. The SK signal is used to clock in serial data on the SD line on the RISING-EDGE.

What is this data? - The first four bits are starting bits and should be ignored. Then come three 8-bit numbers, each representing a digit of degrees (0-360) for the direction. The last 8-bit number is always zero and should be ignored. Unfortunately, the 8-bit numbers are encoded in 7-segment led display format.



Number	Bits: . c b a f g e d
0	0 1 1 1 1 0 1 1
1	0 1 1 0 0 0 0 0
2	0 0 1 1 0 1 1 1
3	0 1 1 1 0 1 0 1
4	0 1 1 0 1 1 0 0
5	0 1 0 1 1 1 0 1
6	0 1 0 1 1 1 1 1
7	0 1 1 1 0 0 0 0
8	0 1 1 1 1 1 1 1
9	0 1 1 1 1 1 0 1

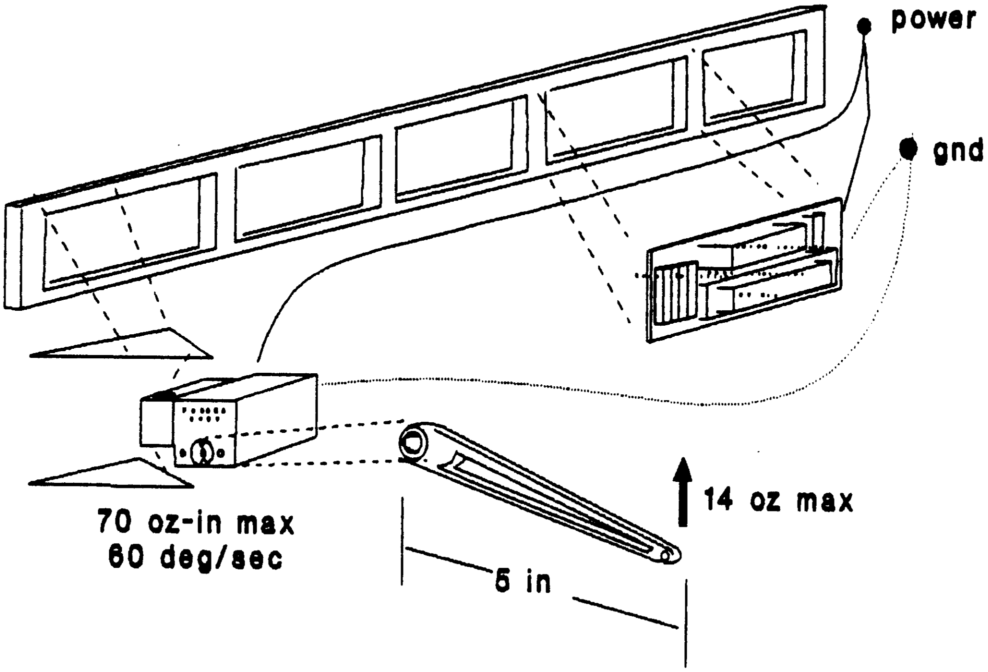
Scope these signals on an oscilloscope or a logic analyzer to verify the data bits as you move the compass around. For both methods be sure to have proper GROUND signals between the interface of the compass electronics and your CPU circuitry. Remember that the compass is NOT foolproof; there will be occassional spurious readings when the compass nears metal objects, as with most needle-based hand held compasses.

2. The Analog Method

You will not have to deal with the COP421-MLA processor in this case. All you need from the compass are the analog outputs X (Pin U1-8, U1 is the CA3403 quad op-amp) and Y (Pin U1-14). As you rotate the compass 360 degrees you should see these two outputs produce sine waves with a phase shift of 90 degrees.

How does one get these signals to a CPU? - Use an A/D converter. Fortunately this is easy if you use the 68HC11 CPU which has an on-chip A/D converter. As with the digital method, you should look at the data on an oscilloscope to verify the X and Y analog outputs.

MUCH RANDOM INFORMATION CONCERNING MOTORS



Colin Angle

MUCH RANDOM INFORMATION CONCERNING MOTORS
Colin Angle

Introduction

This paper is meant to provide practical knowledge about how to choose the right motor for a particular actuator, and how to drive it.

DC MOTORS

Nothing is more pitiful than a wimpy, under-powered robot. It is important to choose a motor which meets your requirements. Some of the motors provided for the contest have data sheets which characterize them, while for the others a few simple tests will give an idea of what to expect. Listed below are some of the basic characteristics of DC motors.

Rated voltage

This voltage is the voltage at which all the tests were run, and is the voltage at which the motor is guaranteed to operate for long periods of time without burning up. It is a very conservative number. DC motors can be overdriven by over one hundred percent if the use of the motor is intermittent. This will appropriately increase the motor's performance. If your motor gets very hot, however, you are driving it too hard and therefore you should not be surprised if it dies. If this voltage is unknown, assume it is a 5v motor, because most small DC motors are either 4.5v or 6v.

"No load" shaft speed

This speed is, as the name implies, the speed of the output shaft with no load on it. This speed is a good indication of whether or not you will have to gear down the output shaft in order to suit your application, and by how much to do it.

If you must measure this quantity, and do not have access to a dynamometer, hook the shaft to a gear train which slows the output speed of the gear train to something you can count and multiply by the gear ratio. If an appropriate gear train is unavailable, it is possible to hook the output shaft to a spindle of known diameter, and winch in string for a specified amount of time, and calculate the shaft speed from that as shown on figure 1.

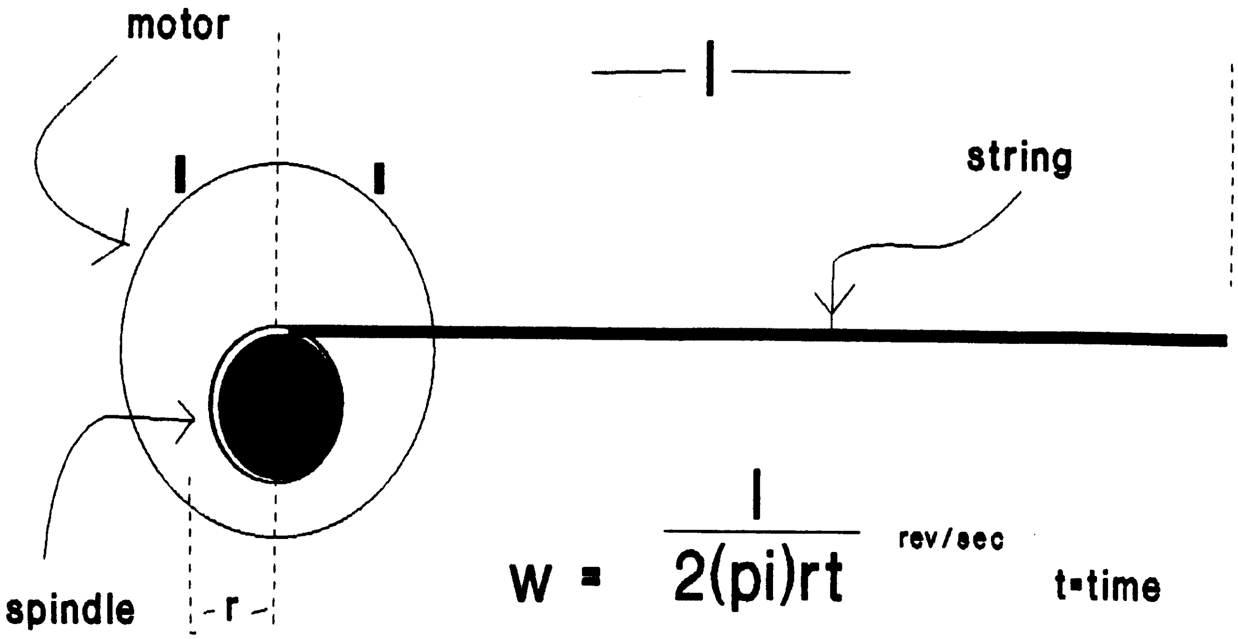


figure 1
finding shaft speed

Stall torque

Knowing the amount of torque a motor will provide perhaps the most important single piece of knowledge required to prevent your robot from falling on its face. It is important to remember that this number represents torque and not force. A 1 oz-in stall torque motor turning a 2 inch diameter spindle will winch in a string attached to it with a maximum force of 1 oz before it stalls. If the spindle was .25 inches in diameter, the force would be 8 oz. To find the stall torque of a motor, attach an arm to the .output shaft, and set up a device such as the one shown in figure 2. Decrease the weight in the basket until the arm pulls the weight, then measure the weight.

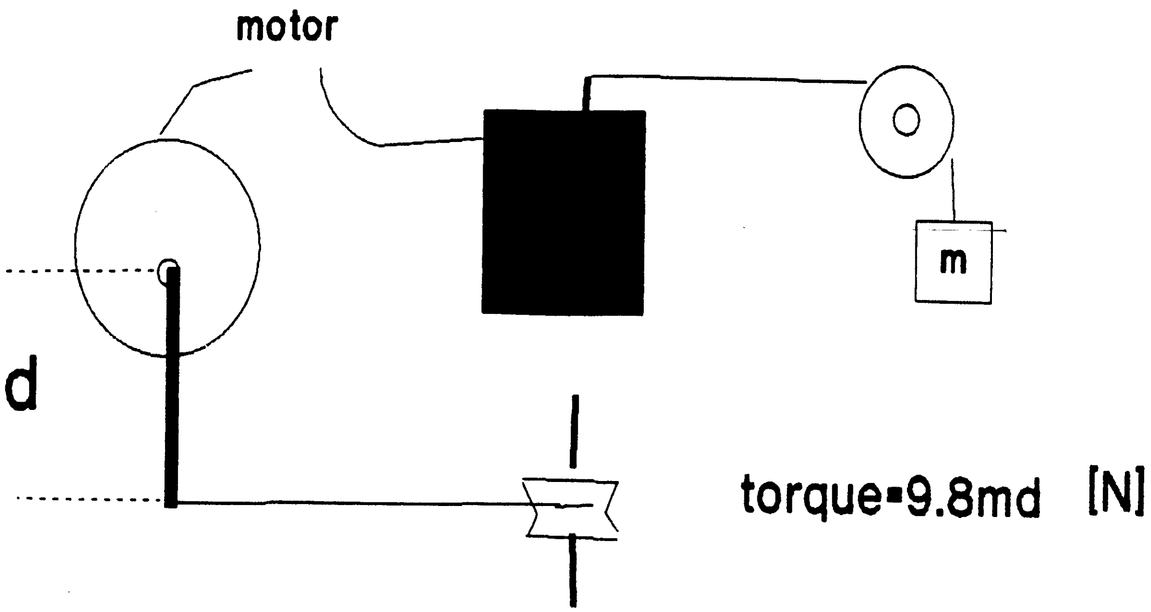


figure 2

finding the stall torque of a motor

Starting current

A rotating DC motor creates a voltage which tends to cancel the applied voltage, and is proportional to the output speed. If the motor is not rotating, this back voltage is zero, and the motor draws its maximum current. This current can be very high. For example, a Maxon 10 watt motor draws .067amps with no load, but has a starting current of 17amps. This fact should be considered when choosing a power source for your motor.

It is possible to determine the starting current by measuring the resistance between the power leads. If you know the voltage which will be used,

Starting current= V/R.

Loaded Performance

The motor will not run at *no load speed* for all loads until it reaches *stall torque*. A linear approximation of speed vs load performance, as shown in Figure 3, is useful if loaded data is needed. Note: A motor loaded to .5 *stall torque* has an output shaft speed of .5 *no load speed.*

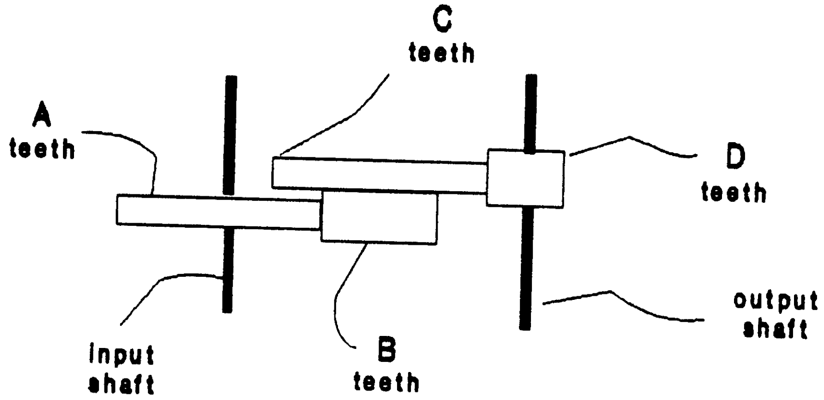
GEARBOXES

DC motors typically will offer much higher shaft speeds and much lower output torques than desired. There are many different ways to *gear down* a motor.

Spur gear boxes

Using spur gears boxes is the most common way of *gearing down* a motor. The output shaft speed is merely the input shaft speed multiplied by the ratio of the number of teeth on the gears (called the gear ratio) as shown in Figure 4. Theoretically, the output torque is the input torque multiplied by the inverse of the gear ratio. This however, is not the actual case. Efficiency, defined as what percentage of the theoretical output torque actually is realized, declines with the amount of reduction used. Typical numbers for efficiency of a spur gearbox are 80% for reduction of 10:1, 65% for a reduction of 100:1, and 50% for a reduction of 500:1.

A second consideration when choosing a gearbox is the maximum output torque that the gearbox can handle. This number is again dependent on the amount of reduction the gearbox gives. Typical numbers are .1 oz-in continuous/0.3 oz-in intermittent for 10:1, and .6 oz-in continuous/1.8 oz-in intermittent for 100:1.



Gear ratio = $\frac{A}{B}\ \frac{C}{D}$

Figure 4
spur gear train

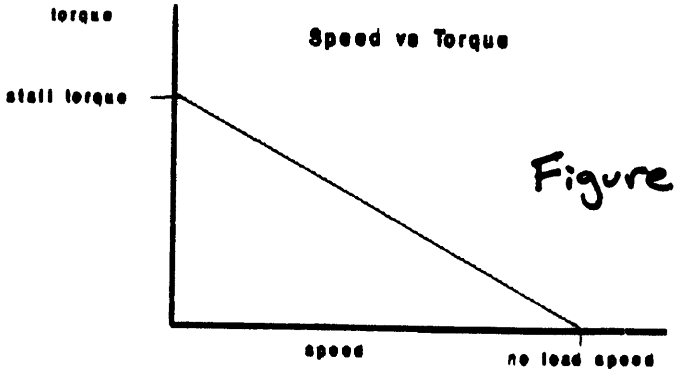


Figure 3

0,5 Watt

S-Motor 2312.…
Precious Metal Brushes

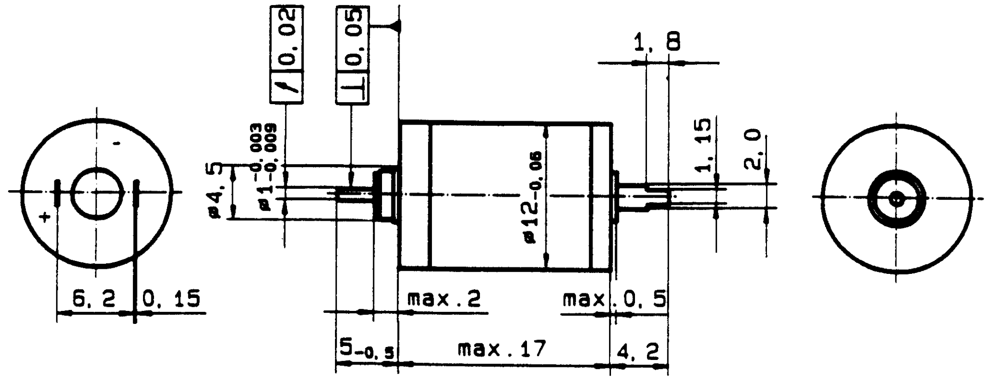
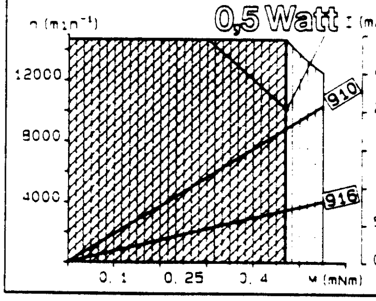


Figure 3a

2312.___-21.1 4 1-010

Motor Data	Winding Number (Order Number)	903	904	905	906	907	908	909	■ 910	911	912	■ 913	□	□	□	□
1 Assigned power rating	W	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5				
2 Nominal voltage	Volt	0.80	1.20	1.20	1.50	1.50	2.40	3.00	3.00	4.50	6.00	9.00				
3 No load speed	rpm	10000	13800	11300	11600	9270	12100	12300	9480	10800	13200	12800				
4 Stall torque	mNm	0.505	0.611	0.496	0.498	0.405	0.514	0.507	0.400	0.450	0.505	0.485				
5 Speed/torque gradient	rpm/mNm	21000	23700	24200	24600	24500	25000	25700	25400	25500	27800	28100				
6 No load current	mA	38.4	39.4	30.3	24.9	18.8	16.6	13.5	9.67	7.58	7.47	4.77				
7 Starting current	mA	701	774	520	427	281	288	231	142	120	124	76.8				
8 Terminal resistance	Ohm	1.14	1.55	2.31	3.51	5.33	8.33	13.0	21.1	37.4	48.6	117				
9 Max. permissible speed	rpm	14700	14700	14700	14700	14700	14700	14700	14700	14700	14700	14700				
10 Max. continuous current	mA	500	500	500	424	344	275	221	173	130	114	73.4				
11 Max. power output at nominal voltage	mW	127	214	141	145	93.8	157	157	94.6	121	168	156				
12 Max. efficiency	%	59.7	61.3	58.8	58.8	56.2	59.1	58.8	55.9	57.4	58.3	57.8				
13 Torque constant	mNm/A	0.721	0.790	0.954	1.17	1.44	1.79	2.20	2.81	3.74	4.09	6.32				
	oz-in/A	0.10	0.11	0.14	0.17	0.20	0.25	0.31	0.40	0.53	0.58	0.90				
14 Mechanical time constant	ms	31.0	31.0	30.8	30.7	30.5	30.5	30.5	30.4	30.4	30.7	30.7				
15 Rotor inertia	gcm^2	0.141	0.125	0.122	0.119	0.119	0.117	0.113	0.114	0.114	0.106	0.105				
16 Terminal inductance	mH	0.02	0.03	0.04	0.06	0.09	0.14	0.21	0.35	0.61	0.73	1.75				
17 Thermal resistance housing-ambient	K/W	59.00	59.00	59.00	59.00	59.00	59.00	59.00	59.00	59.00	59.00	59.00				
18 Thermal resistance rotor-housing	K/W	18.00	18.00	18.00	18.00	18.00	18.00	18.00	18.00	18.00	18.00	18.00				

Operating Range



Comments
(details on page 19)

□ Recommended operating range

▨ Continuous operation
In observation of above listed thermal resistances (lines 17 and 18) the maximum permissible rotor temperature will be reached during continuous operation at 25°C ambient.
≙ Thermal limit

▥ Short term operation
The motor may be briefly overloaded (recurring).

Important Points

■ **Stock program**

● Axial play	0.05–0.15 mm
● Max. sleeve bearing loads	
axial (dynamic)	0.25 N
radial (5 mm from flange)	0.60 N
press-fit force (static)	20 N
● Radial play/sleeve bearings	0.012 mm
● Ambient temperature range	−20/+65°C
● Max. rotor temperature	+85°C
● Number of commutator segments	5
● Weight of motor	9 g

Micro Mo® GEARHEADS

Gearhead Series 10/1

Figure 4a

- Fits Motors Series 1016M ... 1212M ... & 1219M ...
- Case Material: Nickel-plated brass
- Sintered bearings

Maximum Ratings:

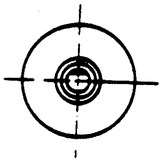
Input Speed: 5,000 RPM.
Temperate Range: -20°C to +125°C
Shaft Loading:
AXIAL: 7.2 oz. (2N).
Press-Fit Force: 36 oz. (10N).
Bearing Play:
RADIAL : 0.03 mm
(.0012")
AXIAL: 0.10 mm
(.0039")
Backlash: ≤3°
Continuous Torque Output: 14.16 oz.-in
Intermittent Torque Output: 28.32 oz.-in

Dimensions are given in mm (in.)

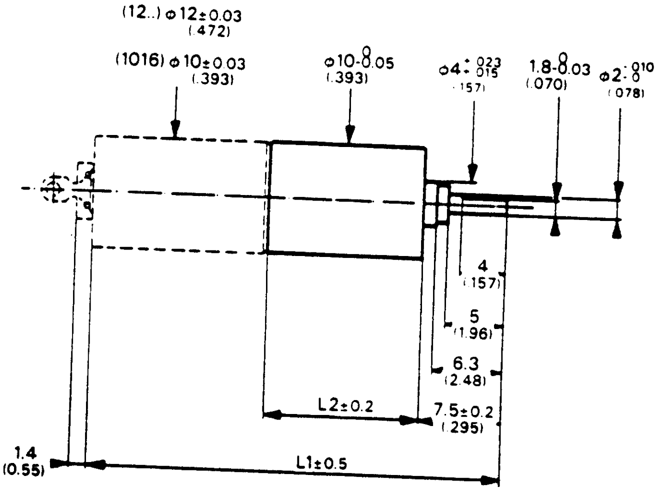
Dimensions with no tolerance indicated are as follows:

For Dimensions:	Tolerance
Less than or equal to 6 mm.	±.1 mm. (.0039")
Less than or equal to 30 mm.	±.2 mm. (.0089")
Less than or equal to 120 mm.	±.3 mm. (.0118")

Dimensional Outlines:



Front View



1	2		3		4				5		6*	7
Gear Ratio	Weight Without Motor		Length Without Motor		Length With Motor 1016		Length With Motor 1212		Max. Continuous Duty Torque Output		Rotation Direction	Efficiency
			L2	L2	L1	L1	L1	L1				
	g	oz	mm	in ·	mm	in	mm	in	mNm	oz-in	R=CW L=CCW	%
4:1	7.1	.25	7.8	0.307	25.4	1.000	22.6	0.890	5	0.71	R	90
16:1	8.0	.28	10.9	0.429	28.5	1.122	25.7	1.012	15	2.12	R	80
64:1	8.3	.30	14.0	0.551	31.6	1.244	28.9	1.138	54	7.65	R	70
256:1	8.7	.31	17.1	0.673	34.7	1.366	32.0	1.260	100		R	60
1024:1	9.2	.33	20.2	0.795	37.0	1.457	35.2	1.386	100	14.16	R	55

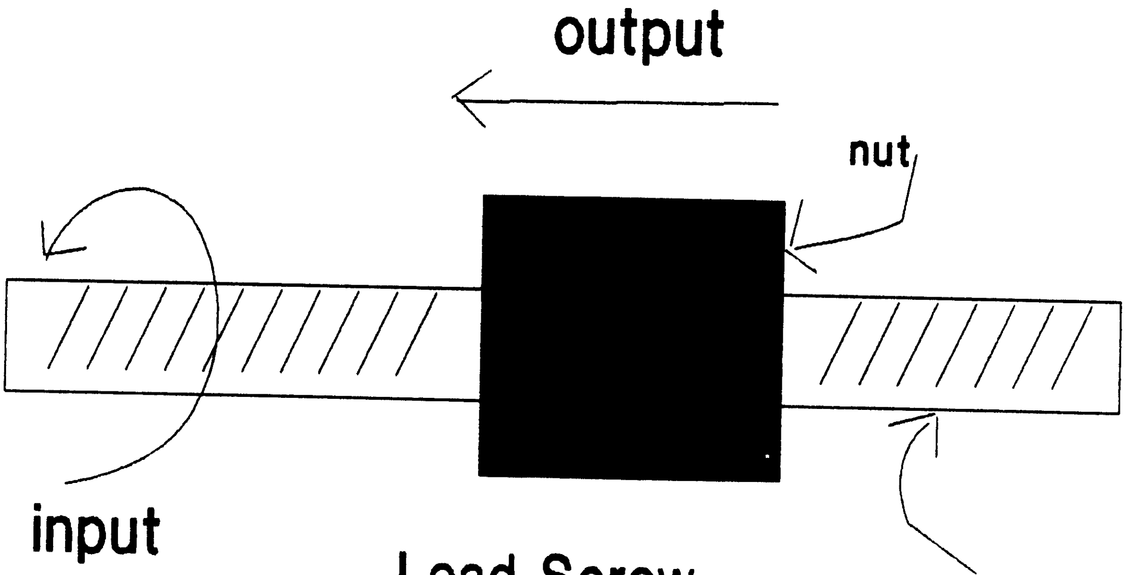
*R=Clockwise. L=Counterclockwise as viewed from Shaft End with Driving Motor Turning Clockwise.
All Ratios Are Bidirectional.

Planetary gear boxes

Greater efficiency and higher output torques can be achieved through the use of planetary gear boxes. They use fewer gears to achieve the required reduction, and therefore can make the gears bigger allowing higher output torques. Typical efficiencies and torques are 72% efficient, with a max output torque of 2 oz-in at 20:1, and 61% efficient, with a max output torque of 3 oz-in at 150:1.

Lead screws

Lead screws are used to change rotary motion into linear motion. The relation between angular velocity of the screw and linear speed of attached nut depends on the number of threads per inch on the screw as shown in figure 5. Typically, lead screws cannot be back driven. This means pushing on the nut will not turn the screw. When the nut is heavily loaded, the efficiency of a lead screw drops dramatically (<30%).



Lead Screw

Figure 5

Worm gears

Worm gears are very similar to lead screws, except the nut is replaced by a disk with threads on the outside(see figure 6). The worm gear offers high reduction and high output torque. However, as with lead screws, the efficiency of the gearing drops drastically under high loads.

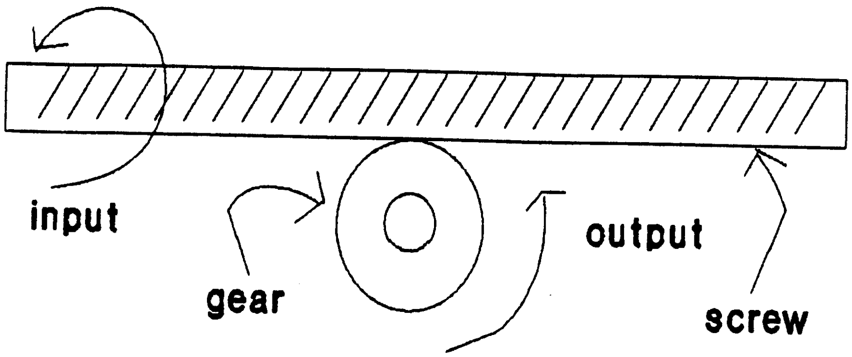


figure 6
Worm Gear

DRIVERS

Providing power to DC motors can be as simple as soldering wires from the motor to a battery, but if the motor is to be controlled electronically, some sort of motor driver is necessary. Below are three different types: an H-bridge driver made of discrete components, an H-bridge driver on an IC, and a driver made of relays.

H-bridge

H-bridge drivers allow the motor to be controlled by applying a TTL level signal to either of its inputs. Direction is specified by choosing which input, forward or backward (as shown on Figure 7), to pull down. A problem with these drivers is that there is a voltage drop across the device, so some of the applied voltage is wasted. A driver made of discrete parts has a much higher power rating than the discrete IC.

H—BRIDGE DRIVER

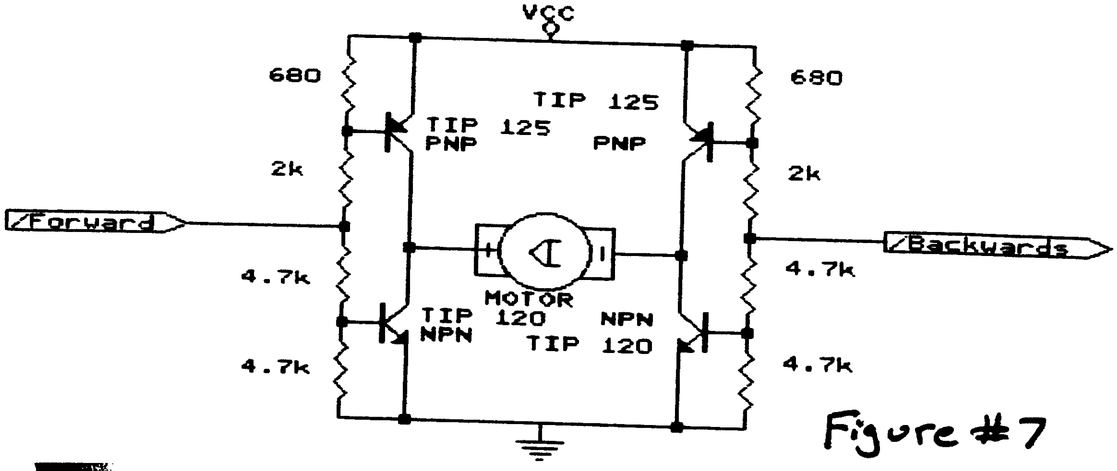


Figure #7

SH-PULL FOUR CHANNEL DRIVER WITH DIODES

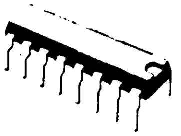
- A OUTPUT CURRENT CAPABILITY CHANNEL
- PEAK OUTPUT CURRENT (NON RE- TIVE) PER CHANNEL
- BLE FACILITY
- ERTEMPERATURE PROTECTION
- GICAL "0" INPUT VILTAGE UP TO V (HIGH NOISE IMMUNITY)
- ERNAL CLAMP DIODES

293D is a monolithic integrated high volt- h current four channel driver designed to standard DTL or TTL logic levels and ductive loads (such as relays solenoides, d stepping motors) and switching power tors.

To simplify use as two bridges each pair of channels is equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included.

This device is suitable for use in switching applications at frequencies up to 5 kHz.

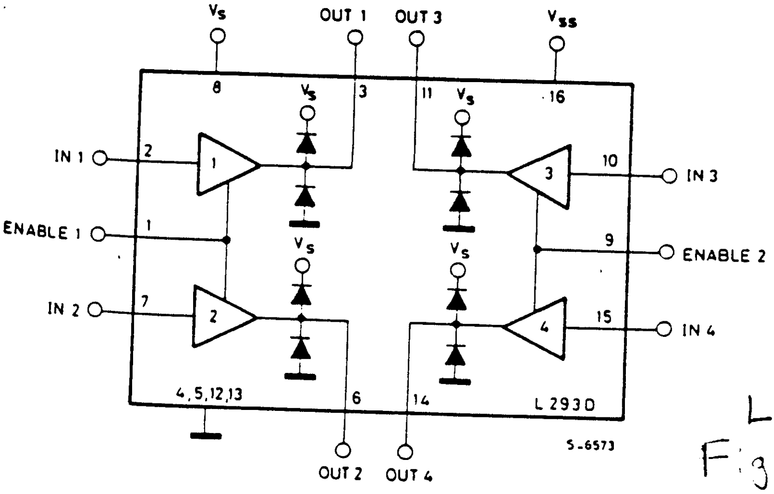
The L293D is assembled in a 16 lead plastic package which has 4 center pins connected together and used for heatsinking.



Powerdip
12 + 2 + 2

ORDERING NUMBER: L293D

CK DIAGRAM



L293D
Figure 8

Relays

The relay driver circuit is harder to control because it is necessary to drive the coils of the relay. This may require open collector buffers, or discrete transistors. There is no voltage drop across the relay however, and use of the appropriate relay will allow the circuit to handle very high motor currents. An example driver circuit is shown in figure 9.

RELAY DRIVER CIRCUIT

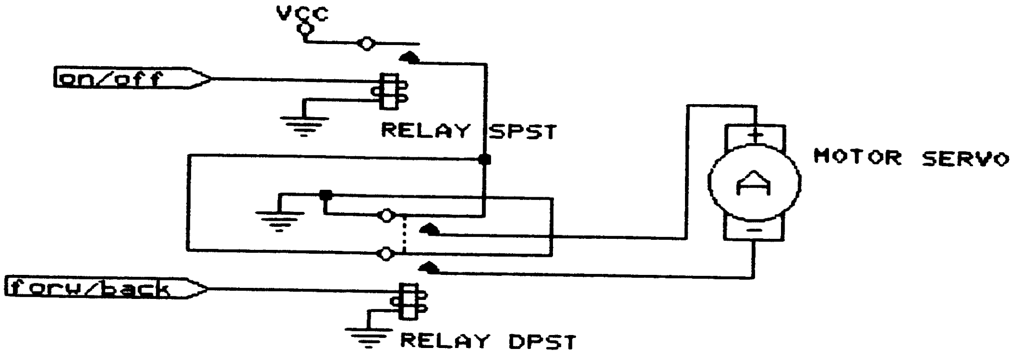


figure 9

Power problems

Driving motors requires great care if the motors are going to be even remotely connected to the rest of the electrical circuitry. This is because voltage spikes caused by the inductance of the motor and current spikes caused by starting a motor can wreak havoc with the sensitive digital components. These problems can be avoided by careful circuit design, or you can decouple your motors from your circuit with relays.

Capacitors go a long way in reducing the effect of voltage spikes. If you are driving motors in your circuit, every chip should have a .01 microfarad capacitor across power and ground. In addition, a VBFC (very big capacitor, ~100mf) should be across the master power and ground going to a particular board.

Improved spike protection and voltage stabilization can be had if each board has a regulated DC-DC converter on it. These converters output a steady voltage as long as the input is in a range of acceptable values. The bigger the range of these values the better.

Connecting the motors to the power supply should be done using what is called a *single point ground*. A single point ground connects the power and ground going to the motor with the power and ground going to the rest of the circuit at the power source. This minimizes inductive voltage drops in the wires due to current spikes. A *single point ground* power structure is shown in figure 10.

CONTROL

DC motors can be controlled very precisely given the proper feedback sensors. Position and velocity sensors are available which can be used as feedback elements to construct a full blown PID controller. Usually position sensors are sufficient.

Position sensors

There are two main types of position sensors. The first, a potentiometer, is the simplest to use. A potentiometer attached to the output shaft yields a voltage proportional to position. This voltage can be hooked to the analog to digital port on the 68HC11 microprocessor to give a position reading. A limitation to this method is that there is a limited range of motion for the potentiometer, and accuracy is limited to about 1%. A second, and more complex method is with an optical encoder. It consists of a rotating disk attached to the output shaft, and an LED emitter/detector pair. The disk has many slots and the emitter/detector pair gives a pulse when each slot comes between them. Thus a pulse train results from spinning the output shaft. By counting the number of pulses, position can be determined. The supporting hardware for this is fairly involved, however.

Rate sensors

Again there are two types of rate sensors. Both operate on the same idea. A spinning motor generates a voltage proportional to its velocity. If your motor is being driven using pulse width modulation, that is, if the motor is being given power for only a fraction of each period and then allowed to spin freely, it is possible to measure the voltage generated directly from the motor during the power off period. If the motor is not being driven in this manner, it is possible to attach a small generator to the output shaft in parallel with the rest of the motor's load. A voltage generated by this device will give a voltage proportional to velocity.

Single Point Ground Wiring

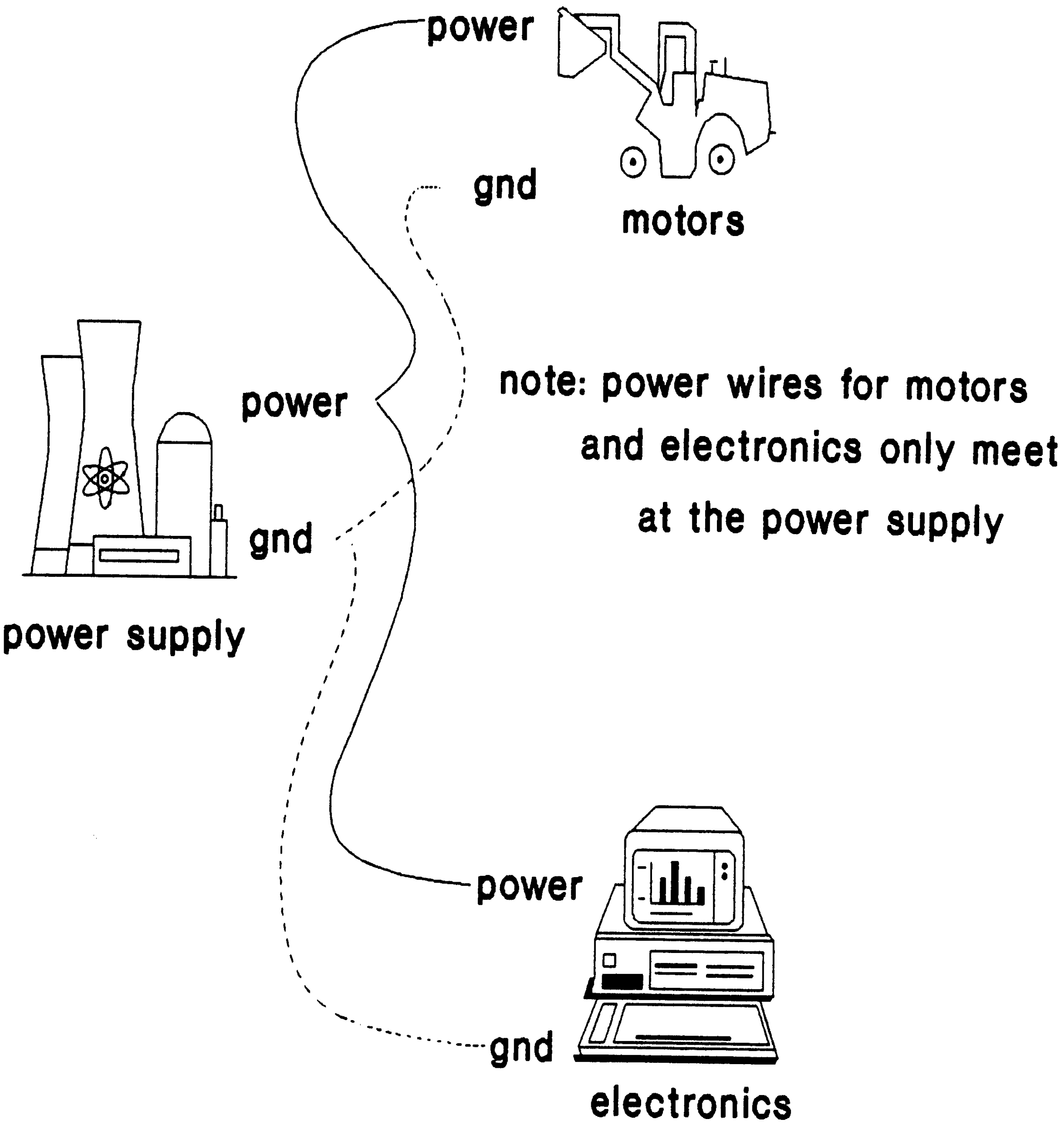


Figure 10

Stepping Motors

A stepping motor is a motor which has a finite number of discrete states at which the rotor can align. This property is very useful in the area of control. To illustrate the advantage of this in a common application, consider a computer printer. The print head of the printer must traverse the page which is being printed, stop in the appropriate position to type a letter, and then move to the next position. The accuracy of the print head positioning must be extremely high, or else the letters will appear too close together, or too far apart. In order to implement a print head positioner using an ordinary DC motor, a position transducer would have to be mounted on the system and a complex feedback system would have to implemented. While this is possible, it would probably require using a computer in the feedback path.

A stepping motor can simplify this control problem greatly. Because it has only a finite set of discrete states it can be in, a single number can completely describe where the rotor, and thus the print head is. More important is that a stepping motor does not run continuously. The rotor turns from discrete state to discrete state one at a time. This allows the motor to be controlled "open loop". No feedback is required. In the case of the printer, if the print head needed to be moved one inch, the computer would only have to tell the stepping motor to make say, 136 steps.

A stepping motor does have one major drawback. If you connect it to a battery, it will not run. It requires some sort of drive sequencing circuitry to operate. This circuit, however, in most cases does not need to be very complex, and the ease of controlling the motor far outweighs the driver's complexity.

Types of Stepping Motors

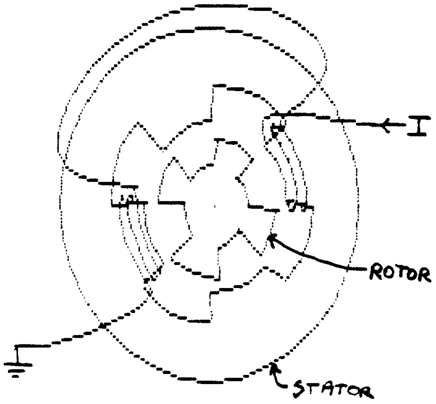
The stepping motor has a rotor and a stator like traditional motors. However, the arrangement and operation are quite different.

The Stator

The stator on a typical stepping motor is shown in the diagram below. It consists of many separate windings which can better be thought of as a series of electromagnets sequentially attracting the rotor poles, rather than as producing a traveling flux wave which carries the rotor along with it. The electrical connection of these windings depends on the type of rotor the stepper motor has. In the case of the variable reluctance rotor which is discussed later, these windings typically occur in pairs. Each pair of windings is called a phase. An example of a winding for a variable reluctance rotor is shown below. Windings for a permanent magnet rotor, also discussed in following section, are independent. The number of windings is equal to the number of poles in the stator which is equal to the number of phases.

STATOR WOUND FOR VARIABLE RELUCTANCE ROTOR WITH ONE PHASE AND TWO POLES

FIGURE 11



The Rotor

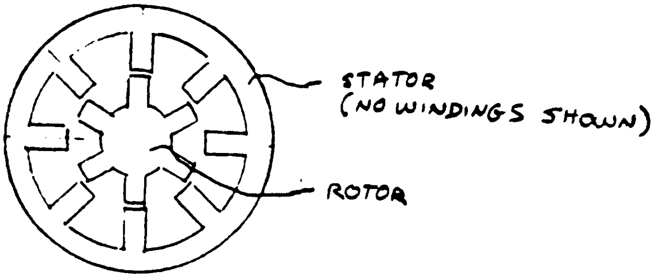
There are two main types of rotors which are used in stepping motors. The type of rotor used is the most important factor in characterizing the operation of the motor. The rotor may either be of the variable-reluctance type, or the permanent magnet type.

The Variable Reluctance Rotor

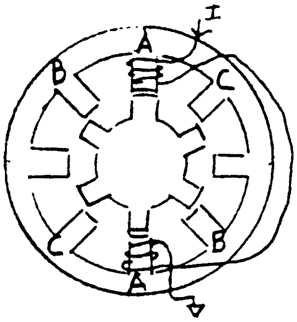
A variable-reluctance type rotor is made out of a highly permeable material and looks like a cylinder with many square beams running lengthwise down the cylinder. A typical rotor cross-section is shown below, along with a four phase stator.

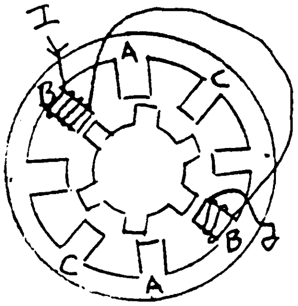
MOTOR ASSEMBLY

FIGURE 12



The operation of the motor involves sequentially exciting the individual stator windings. A typical drive sequence is illustrated below. In Figure 13, winding A is excited. The rotor experiences no torque. This can be shown by looking at the energy method. The energy method tells us that the rotor will move, in the absence of other torques, to a position which maximizes the inductance of the system. Since only winding A is on, this arrangement occurs when a pair of the poles on the rotor is directly aligned with each of the poles in stator winding A. In Figure 14, winding B is excited. There will therefore be a clockwise torque on the rotor which will cause it to turn until it reaches an equilibrium similar to that in Figure 14. Note that if winding C is excited instead of winding B, the rotor turns in the opposite direction. Thus the rotor can be stepped around in either direction.





STEPPING MOTOR STATOR WINDINGS

Figure 13

Figure 14

The variable reluctance rotor is used because very small step sizes can be achieved. For example, in the configuration above, which has 8 stator poles and 6 rotor poles, the step angle is 15 degrees. By stacking rotors on top of one another with a slight offset, it is easy to get step angles of less than one degree. This technique of stacking rotors and stator windings is shown below (Figure 15).

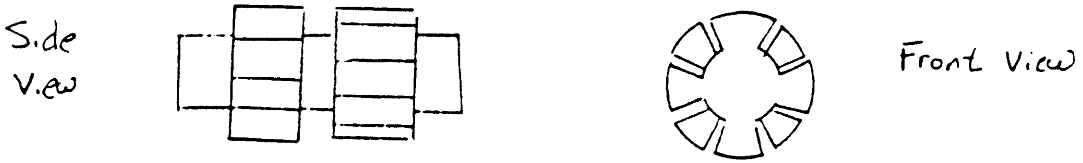
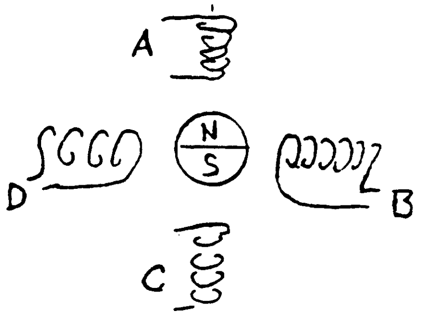


Figure 15

HYBRID ROTOR

The Permanent-Magnet Rotor

A permanent-magnet rotor is intuitively simpler in operation the than variable-inductance type. A simple rotor could just be a cylindrical bar magnet with a shaft through the center such as the rotor in a simple permanent magnet motor shown below. To understand the motor operation, we model the stator windings as electromagnets which can be independently controlled. If winding A is driven with a current such that an induced south pole faces the rotor, the rotor's north pole will turn, in the absence of other torques, until it is directly lined up with winding A. If winding B is also turned on, the rotor's north pole will be attracted by both windings and will align itself at a 45 degree angle to both of them. By turning off winding A, the rotor will turn another 45 degrees, and so on. Thus with four individual windings, the motor is capable of 45 degree steps.



PERMANENT MAGNET ROTOR WITH STATOR WINDINGS

FIGURE 16

By increasing the number of poles on the rotor, it is possible to significantly decrease the step size of the motor. However it is more difficult to achieve small step sizes than with the variable reluctance rotor type. Permanent magnet rotors are commonly used in smaller stepping motors because they can provide greater torques for their size than the variable reluctance rotors.

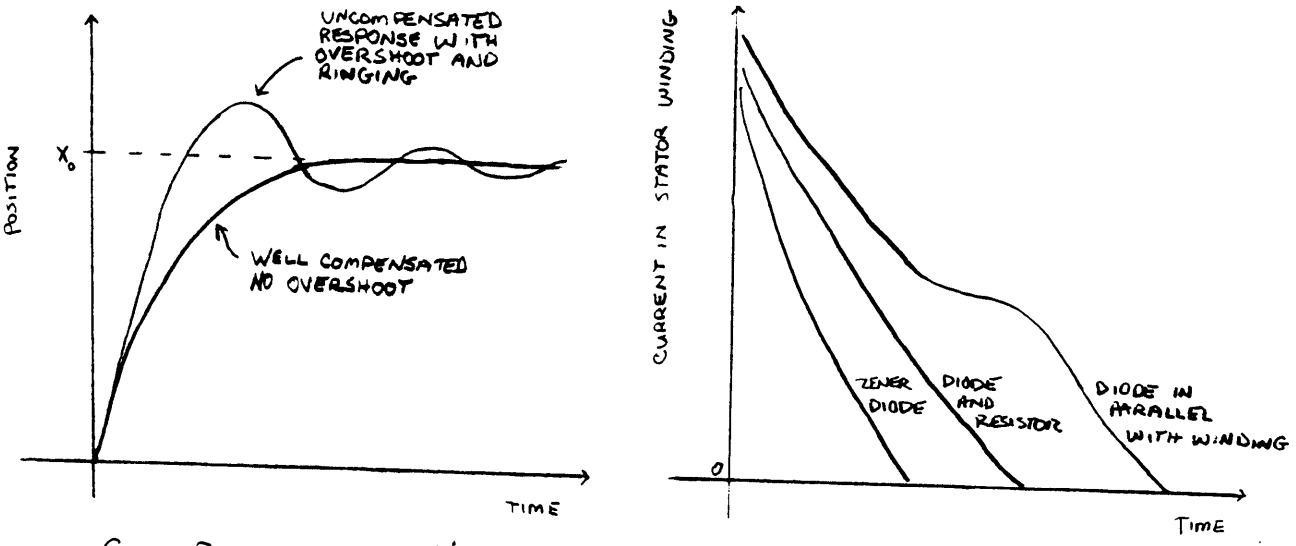
The Controller

As noted above in the descriptions of the basic operation of the different kinds of stepper motors, there must be some kind of drive sequencing device in order for the stepping motor to run. There are many different kinds of controllers, but they all do basically the same thing. That is, a sequencer is given an input square wave which has a peak for every step the motor is expected to make, and the sequencer must drive the various stator windings in the correct sequence to accomplish this.

The reason for the various kinds of controllers is that performance of the stepping motor can be dramatically improved with better power control. For example, the stator windings can have very high inductances. For optimal performance of a stepping motor the currents in these windings should build up as quickly as possible when the winding is turned on, and die out as quickly as possible when the winding is turned off.

The failure of a controller to deal with inductive current spikes may result in the destruction of the power electronics driving the motor. The inductance of the stator will attempt to drive current through a power transistor in the off state and blow it out. Even if the transistors survive the spikes, performance will suffer. Problems such as the output shaft overshooting its desired position, and ringing for a long time can happen. Worse still, the motor may miss steps completely at higher stepping rates.

These problems are beyond the scope of this paper, but they are dealt with by various means such as adding viscous damping to the motor, or by putting a diode in parallel with the winding, or more complex means such as using more power electronics to actively kill the winding current, or using feedback to control the winding current. To give a better idea of the problem of overshoot, the following graph is included (Figure 17) which shows the position of the rotor relative to the desired location as a function of time in response to a single step. Also shown is a graph of current through a stator winding as a function of time following the shut off of the driving power transistor.



STEP RESPONSES WITH VARIOUS CONTROLLERS

WINDING CURRENT WITH DIFFERENT FLYBACK DIODE CONFIGURATIONS

FIGURE 17

Implementation of stepper drivers

Each different type of stepper motor requires its own type of driver circuit. The most common stepper motor types are two and four phase with permanent magnet rotors. Figures 18 and 19 are schematics for a four phase and a two phase driver circuit respectively.

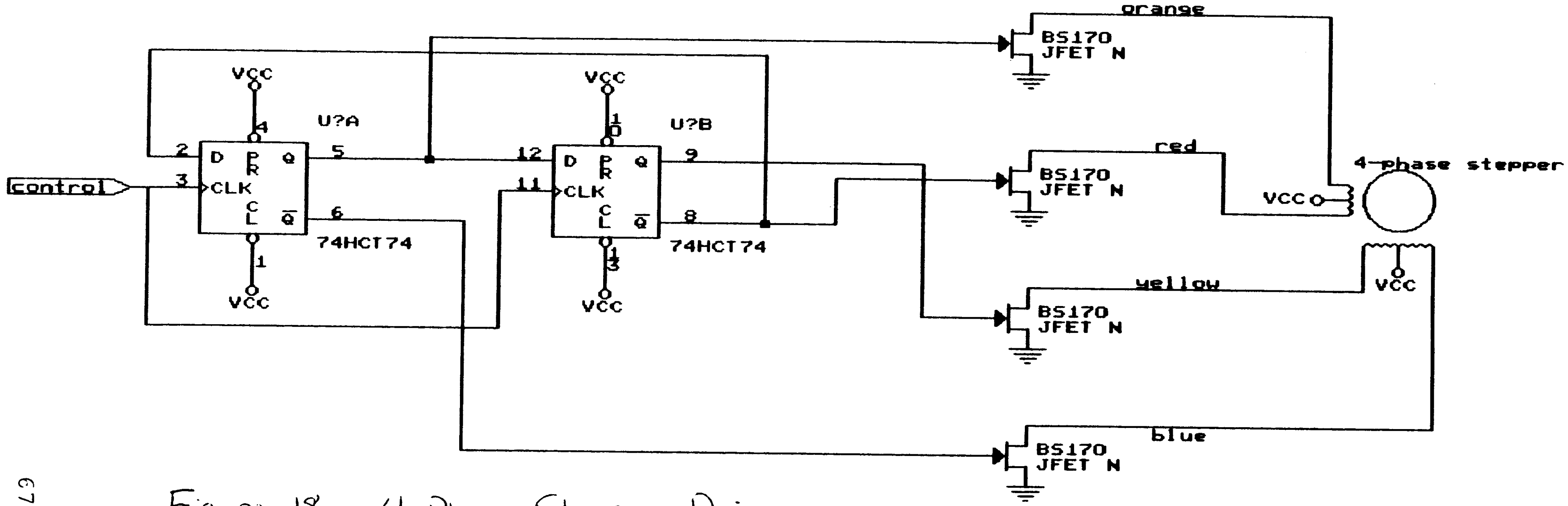
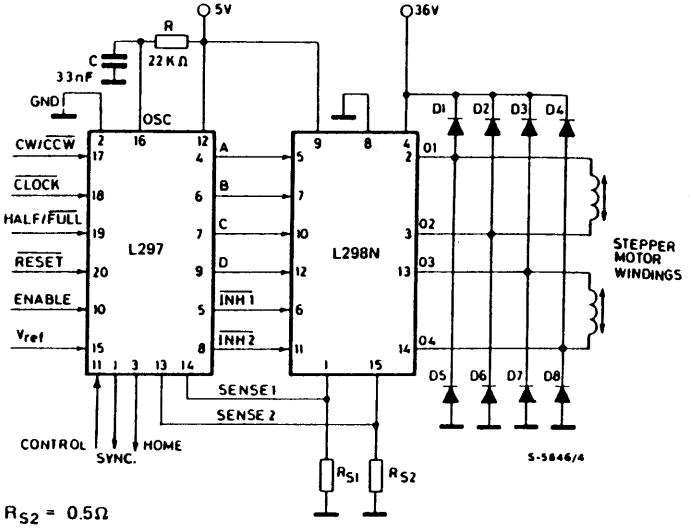


Figure 18 4 Phase Stepper Driver

Two phase bipolar stepper motor control circuit

This circuit drives bipolar stepper motors with winding currents up to 2A.

Fig. 19



R_{S1} R_{S2} = 0.5Ω

D1 to D8 = 2A Fast diodes $\left\{ \begin{array}{l} V_F < 1.2V @ I = 2A \\ trr < 200 ns \end{array} \right.$

FUTABA SERVO MOTORS

A Futaba servo motor is essentially a DC motor with gear train, feedback potentiometer, and servo processor all integrated in one box. Controlling these servos requires merely sending the correct control signal to the motor. This control signal is a *pulse code modulated signal*. This means the signal is a repeated waveform which is high for a given time corresponding to an output shaft position. An example of this is given in Figure 20. The center positioning frequency for a Futaba servo is approximately 1.3 milliseconds, and the signal should be repeated approximately every 20 milliseconds. If the output shaft needed to be turned 45 degrees to the right, all that would be required would be to shorten the pulse.

It is possible to get an approximate force measurement on these position based servo motors. The servo processor chip takes the actual position the servo is in, and compares it with the desired output shaft position. Subtracting these two signals, it creates an error signal which tells it how hard to drive the actual motor in the servo. The greater the error, the harder it will drive the motor. Since, in steady state, the error in shaft position is proportional to the torque on the shaft, measuring the error signal gives an approximation of the force.

Output shaft position can be detected in the same way the internal servo chip detects it. This is by using the potentiometer, and tapping off the center lead.

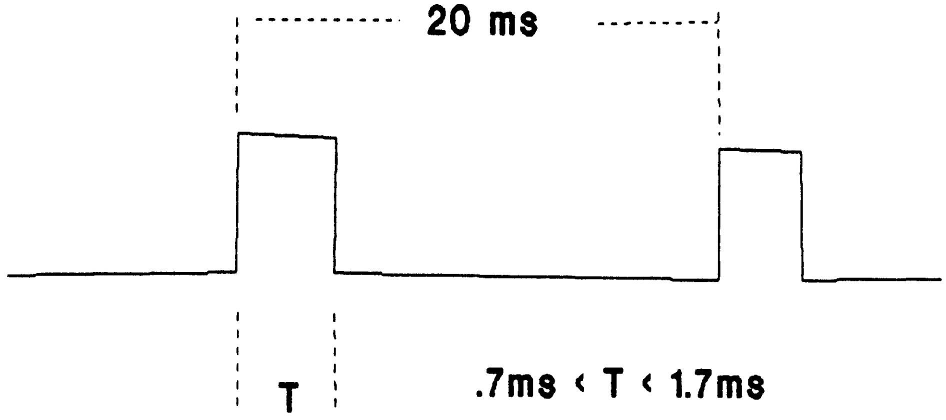


figure 20
Futaba Control Timing diagram

CONCLUSION

This is a very general and shallow overview of DC motors, but it should provide enough material to make an educated decision as to the correct motor to use. Make sure you use a powerful enough motor, and if at all possible add a factor of two in all your loading estimates. GOOD LUCK.

Software Development System

6811 Assembler and Subsumption Compiler

Rodney Brooks

Programming Tools for the M68HC11.

There is a collection of tools available for programming the M68HC11, written in Common Lisp, that run on a variety of machines: Macs, Suns, HPs and Symbolics . They include a macro assembler, a downloader/eeprom burner, a subsumption compiler, and a serial line monitor.

All these tools can be loaded by loading the appropriate load.lisp file, or for smaller machines, the assembler and downloader can be selectively loaded by loading only a subset of the files.

The Assembler

The assembler is a dynamically retargetable macro assembler. The file ass.lisp defines it and describes target machines for the Hitachi 6301 and the Intel 8085A. A second file, m68hc11.lisp, provides a target definition for the m68hc11.

The assembler is used by the down loader and by the subsumption compiler as its back end. If you only want to use the subsumption compiler as your development system you can completely skip this section.

The assembler runs in two passes. In the first pass, it resolves all symbol values and instruction lengths. In the second pass, it computes the actual code bytes. This could all be done in the first pass, save for the the fact that branches, jumps and subroutine calls are allowed to do forward referencing. All other references are resolved in the first pass and so no other forward references are allowed. E.g. in using the == directive described below, any symbols referenced in definition must precede the directive.

Assembling Code

There are two components to assembling code. First, a program, an instance of the %program defstruct, must be created, and then it must be passed to the assemble procedure.

The simplest way of defining a program is with the defprog macro. Putting it at top level in a file and loading that file is sufficient. (Note that there are some problems with compiling files with defprogs in them on the Macintosh---it is not recommended.) The general format of defprog is:

```
(defprog <name>
  :machine 6811
  :start <some address>
  :code <code>)
```

Note that none of the arguments are evaluated.

Instead of 6811 you can also use m68hc11; they are completely synonomous.

The <name> must be a symbol. It gets globally set to the created %program. The <some address> entry should be a number. It is the address at which assembly will start. On the version of the chips used for the AI Olympics (the A2 version, which contains 2K eeprom) this should be #xf800.

The <code> entry is a list of assembler statements. See the following sections for complete descriptions of their format. In the meantime, here is an example program:

```
(defprog example
  :machine 6811
  :start #xf800
  :code ((=v portb #x1004)
         (=c stack #xff)
         (def-ass-subst set-led ()
            (ldaa ! #b11111111)
            (staa portb))
         (def-ass-subst clear-led ()
            (clr portb))
   start
         (lds ! stack)
   flash
         (set-led)
         (jsr delaylots)
         (clear-led)
         (jsr delaylots)
         (bra flash)
   delaylots
         (ldx ! #xffff)
   dlots
         (iterate ((i 4)) (nop))
         (dex)
         (bne dlots)
         (rts)
         (= #xfffe)
         (!16 start)
         ))
```

The result of evaluating this is that the symbol EXAMPLE has as its value a %program defstruct. It can be assembled using the procedure assemble. The result is a data structure which is stashed in a slot of the same %program defstruct. As a side effect a listing can be produced on the terminal, or in a listing file. The format of the call is:

```
(assemble <name>)              ;assembles the %program which is <name>'s value

(assemble <name> t)            ;also generates an on screen listing

(assemble <name> <filename>)   ;writes a listing to the designated file.
```

For instance:

```
? (assemble example "exampl.txt")
32
```

assembles the above defined program and generates a listing in the file "exampl.txt". The 32 which is returned is the total number of bytes of code assembled. Notice that these are not contiguous bytes. The listing file would be something like:

```
Program EXAMPLE, assembled for the M68HC11, 6 Nov 1988, 12:57:37

        F800            (= 63488)
        1004            (=V PORTB 4100)
          FF            (=C STACK 255)
        F800  START
F800  8E 00 FF          (LDS ! STACK)
        F803  FLASH
F803  86 FF             (LDAA ! 255)
F805  B7 10 04          (STAA PORTB)
F808  BD F8 13          (JSR DELAYLOTS)
F80B  7F 10 04          (CLR PORTB)
F80E  BD F8 13          (JSR DELAYLOTS)
F811  20 F0             (BRA FLASH)
        F813  DELAYLOTS
F813  CE FF FF          (LDX ! 65535)
        F816  DLOTS
F816  01                (NOP)
F817  01                (NOP)
F818  01                (NOP)
F819  01                (NOP)
F81A  09                (DEX)
F81B  26 F9             (BNE DLOTS)
F81D  39                (RTS)
        FFFE            (= 65534)
FFFE  F8 00             (!16 START)

Program EXAMPLE, 32 (decimal) bytes of code (20 hex)
```

The leftmost column is the hex address of the assembled instruction. The following two digit hex numbers are the assembled bytes. On the m68hc11, a single instruction may occupy up to five bytes. Then the post macro expansion source code is shown. For lines which are labels, the address of the label and its name are shown. For other directives (e.g., =, =v, =c, etc.) the value of the directive is shown. In the source code all numbers are printed in decimal. Elsewhere they are printed in hexadecimal.

Code Format

Each line of code can either be a symbol, which is taken as a label, or a list, which we will call an expression.

As with all assemblers an address is associated with every expression which assembles into a non-zero number of bytes. Likewise, the current address becomes the value of any symbol used as a label. Thus in the example above the symbol DLOTS has value #xf816.

Expressions can be assembler directives, data statements, instruction specifications, macro definitions or an iteration construct. In each case it is valid to include at the end of the list defining the expression an arbitrary number of comment subexpressions. A comment subexpression has the form:

(comment <format-string> . <args>)

Such comments are ignored except in producing listings, when they are printed following a tabbed semi-colon. They are printed by feeding the <format-string> and any <args> to FORMAT. The <args> must all be constants, special variables, or functions of constants and special variables as there is no lexical environment available. Human writers will probably just use a simple string without additional arguments (except perhaps in a macro or iteration construct). This extra functionality is really for the benefit of compilers which wish to produce comment annotated code.

Assembler Directives

There are two forms of assembler directives. The first, =, specifies a new address, or origin, at which the following instructions should be assembled, while the others (==, =c, =v, and =v2) provide ways of defining values for symbols with some error checking.

The origin specification simply takes the form:

(= <some-address>)

where <some-address> is a number. In the example program above, the address for the reset vector (#xfffe) is specified with an = construct, and we see in the listing that the assembler introduced an additional = construct to specify the :start address from the %program defstruct.

To specify the value of a symbol the simplest thing is to use == as in:

```
(== foo #xbfcd)
(== bar #xf3)
(== baz (+ foo bar 3))
```

The first argument must be a symbol, whose value is to be defined, and the second argument must be a lisp expression whose terminals are either numbers or symbols which have previously been defined.

The == directive provides all the functionality that this assembler has, but there are three similar versions which provide an extra level of error checking for the user. They obey precisely the same syntax as ==. They are:

1. =c. This defines a symbol as being a numeric constant. If the symbol is anywhere used as an address, then an error is signalled.

2. =v. This defines a symbol as the name of a location. If another symbol is defined to name the same location, an error is signalled.

3. =v2. The same as =v, except that this reserves both the named location and the following location. This can be thought of as defining a two byte variable.

Data Statements

Data can be placed at the current location within a program with the following statements:

(!8 <expression>)
(!16 <expression>)
(!string <string>)
(!table <exp1> <exp2> ... <expn>)

For !8, !16, and !table, each expression can be an arbitrary lisp expression whose terminals are all numbers or previously defined symbols. !8 produces one 8 bit byte, !16 produces two, and !table produces as many as it has argument expressions. For !16 the high order byte comes first. The !table construct macro expands to a series of !8 constructs in a listing.

For !string statements the <string> must be enclosed in double quotes as in:

(!string "Some random string")

In a listing of such a statement only the first three data bytes are printed.

Instruction specification

Every instruction has the form:

(<operator> . <operand-specs>)

where for some instructions (e.g. nop, or dex) there are no <operand-specs>. Simple instructions have precisely one operand. Weird instructions have more.

First consider the simple instructions which have an operand. There are six addressing modes: immediate, direct, extended, pc-relative, index-x, and index-y. There is no syntactic difference between direct, extended and pc-relative---it simply depends on whether the address comes out to be smaller than 256 or if the operator (a branch) requires a pc-relative mode. The six modes are:

(<operator> ! <expression>)	immediate
(<operator> <expression>)	direct or extended
(<branch> <label>)	pc-relative
(<operator> & <expression>)	index x
(<operator> &x <expression>)	index x (alternate form)
(<operator> &y <expression>)	index y

The weird instructions are BSET, BCLR, BRSET, and BRCLR which have three possible addressing modes each: direct, index-x and index-y. The first two of these operators, BSET and BCLR, take two arguments, while the latter two take three. They are specified in the same order as they appear in memory in the assembled version of the instructions. Here are the formats for these instructions. Note that <address> in the direct mode must evaluate to the range 0 to 255, as must the <index> in the two indexed modes. Note also that the <mask> is not prefixed with ! as one might expect as it is always unambigously an immediate operand.

direct:
    (bset <address> <mask>)
    (bclr <address> <mask>)

```
    (brset <address> <mask> <branch-label>)
    (brclr <address> <mask> <branch-label>)

index x:
    (bset (& <address>) <mask>)
    (bclr (& <address>) <mask>)
    (brset (& <address>) <mask> <branch-label>)
    (brclr (& <address>) <mask> <branch-label>)

or
    (bset (&x <address>) <mask>)
    (bclr (&x <address>) <mask>)
    (brset (&x <address>) <mask> <branch-label>)
    (brclr (&x <address>) <mask> <branch-label>)

index y:
    (bset (&y <address>) <mask>)
    (bclr (&y <address>) <mask>)
    (brset (&y <address>) <mask> <branch-label>)
    (brclr (&y <address>) <mask> <branch-label>)
```

Macros

There are two flavors of macro definitions. Once a macro is defined it can be used in the place of a normal operator and macroexpanded into zero or more instruction expressions which are then spliced into the instruction stream. These instruction expressions can be labels, directives, data statements, instructions, calls to macros, or anything else that is normally valid as a top level form within program code. Macros are defined in line as though they were instruction expressions.

One of the macro forms is a substitution form. The other can invoke general lisp code to produce the resulting set of instructions.

They are:

```
(def-ass-subst <name> (<param1> .... <paramn>)
  <form1>
  ...
  <formk>)
```

and

```
(def-ass-macro <name> (<parm1> ... <paramn>)
  <body>)
```

In a def-ass-subst, the supplied parameters are simply substituted textually in the k body forms and those forms are spliced into the output stream.

For instance:

```
 ...
 (def-ass-subst incf (var amount)
    (ldab var)
    (adab ! amount)
    (stab var))
 (incf foo 3)
 (jsr bazola)
 (incf bar 6)
 ... is the same as:
...
(ldab foo)
(adab ! 3)
(stab foo)
(jsr bazola)
(ldab bar)
(adab ! 6)
(stab bar)
...
```

In a def-ass-macro, the supplied arguments are bound to the formal parameters and then the body is evaluated.

For instance:

```
...
(def-ass-macro incf (var amount)
  (if (numberp amount)
     `((ldab ,var)
       (adab ! ,amount)
       (stab ,var))
     `((ldab ,var)
       (adab ,amount)
       (stab ,var))))
(incf foo 3)
(jsr bazola)
(incf bar foo)
... is the same as:
...
(ldab foo)
(adab ! 3)
(stab foo)
(jsr bazola)
(ldab bar)
(adab foo)
(stab bar)
...
```

Iteration

The ITERATE special form allows iteration at assembly time. One way to think of it is as loop unrolling. It has the general form

```
(iterate (<iteration1>
          <iteration2>
      ...
          <iterationn>)
<bodyform1>
<bodyform2>
...
<bodyformm>)
```

The idea is that n variables are stepped in parallel over k values each, and that the m forms with each of these values substituted for the variables, are repeated k times in the instruction stream.

There are two forms for an iteration:

Form1: (<variable> <number>)
Form2: (<variable> (<val1> <val2> ... <valk>))

In form1, the variable is stepped through values 0, 1, 2, through one less than the <number>. In form2, the variable is stepped though each of the specified k values. In form2, a value can be anything. Examples are:

```
(iterate ((i 4))
 (nop))

==>
 (nop)
 (nop)
 (nop)
 (nop)
```

```
(iterate ((j 3)
          (v (val2 val1 val0)))
 (ldaa & j)
 (staa v))
==>
 (ldaa & 0)
 (staa val2)
 (ldaa & 1)
 (staa val1)
 (ldaa & 2)
 (staa val0)
```

Here is an example taken from the six legged walker. We are repeating the same code for six different legs. Each code instance stores the contents of accumulator B in a different location, depending on whether a particular (different in each case) bit in accumulator A is set. Note that this example also iterates over a label set and generates the labels. Notice also, that this example is more powerful than loop unrolling as there is no data-driven indexable bita instruction.

```
(iterate ((bit (1 2 4 8 16 32))
          (bt (bt0 bt1 bt2 bt3 bt4 bt5))
          (forb (forb0 forb1 forb2 forb3 forb4 forb5)))
         (bita ! bit)
         (beq bt)
         (stab forb)
       bt)
```

The Down Loader

The down loader provides precisely one user level procedure, dl. It takes one argument which should be a %program defstruct instance which is to be loaded into the eeprom of the m68hc11.

The serial port used by default on the host machine must be connected to the serial port of the m68hc11, with appropriate level translator chips. The m68hc11 must be in bootstrap mode. Some downloader cables may arrange this automatically. Consult your schematics.

The downloader defaultly operates over the 'A' serial port of a Macintosh.

The downloader operates at 1200 baud. This is a requirement of the m68hc11 and cannot be altered.

The downloader assembles the %program it is given as an argument and then proceeds in two phases. First it downloads an eeprom programmer into the RAM of the mc68hc11. This is only 256 bytes, and as each byte is loaded it is echoed as a period on the host, in four rows of 64 periods. Then it proceeds to download the requested program and burn it into the eeprom. This is rather slow and is limited not by the baud rate, but by the 10ms per byte programming time necessary in the eeprom. The downloader is smart and only writes actually specified bytes, so small programs load faster than long ones. Progress is monitored by printing a period roughly every 32 bytes of downloaded code.

Besides the assembler it is necessary to load the files dload and load6811. Note that on the Macintosh it is buggy to compile load6811. On the Macintosh it is also necessary to have the library file serial-streams loaded. On other machines there will also be a machine specific file to load.

The Subsumption Compiler

The subsumption compiler provides a way of compiling Brooks' subsumption architecture to run on a single processor simulating many parallel processors. The subsumption architecture provides an incremental method for building robot control systems linking perception to action. A properly designed network of finite state machines, augmented with internal timers, provides the robot with a certain level of performance, and a repertoire of behaviors. The architecture provides mechanisms to augment such networks in a purely incremental way to improve the robot's performance on tasks and to increase the range of tasks it can perform. At an architectural level, the robot's control system is expressed as a series of layers, each specifying a behavior pattern for the robot, and each implemented as a network of message passing augmented finite state machines.

The network can be thought of as an explicit wiring diagram connecting outputs of some machines to inputs of others with wires that can transmit messages. On the m68hc11, the messages are limited to 8 bits. Each augmented finite state machine (AFSM, see Figure 1), has a set of registers and a set of timers, or alarm clocks, connected to a conventional finite state machine which can control a combinatorial network fed by the registers. Registers can be written by attaching input wires to them and sending messages from other machines. The messages get written into them replacing any existing contents. The arrival of a message, or the expiration of a timer, can trigger a change of state in the interior finite state machine.

Finite state machine states can either wait on some event, conditionally dispatch to one of two other states based on some combinatorial predicate on the registers, or compute a combinatorial function of the registers, directing the result either back to one of the registers or to an output of the augmented finite state machine. Some AFSMs connect directly to robot hardware. Sensors deposit their values to certain registers, and certain outputs direct commands to actuators.

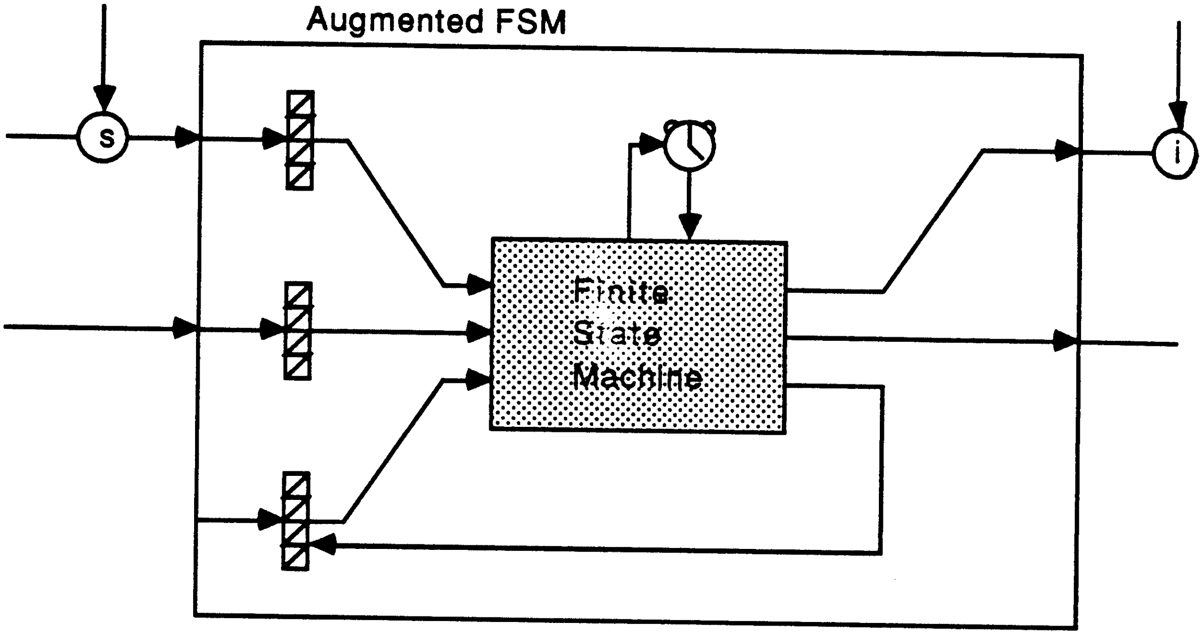


Figure 1. An augmented finite state machine consists of registers, alarm clocks, a combinatorial network and a regular finite state machine. Input messages are delivered to registers, and messages can be generated on output wires. AFSMs are wired together in networks of message passing wires. As new wires are added to a network, they can be connected to existing registers, they can inhibit outputs and they can suppress inputs.

A series of layers of such machines can be extended by adding new machines and connecting them into the existing network in the ways shown in the figure. New inputs can be connected to existing registers, which might previously have contained a constant. New machines can inhibit existing outputs or suppress existing inputs, by being attached as side-taps to existing wires. When a message arrives on an inhibitory side-tap (see figure 1, circled 'i'), no messages can travel along the existing wire for some short time period. To maintain inhibition, there must be a continuous flow of messages along the new wire. (In previous versions of the subsumption architecture explicit, long, time periods had to be specified for inhibition or suppression with single shot messages. Recent work has suggested this better approach.).

When a message arrives on a suppressing side-tap (circled 's'), again no messages are allowed to flow from the original source for some small time period, but now the suppressing message is gated through and it masquerades as having come from the original source. Again, a continous supply of suppressing messages is required to maintain control of a side-tapped wire. One last mechanism for merging two wires is called defaulting (which can be indicated in wiring diagrams by a circled 'd'). This is just like the suppression case, except that the original wire, rather than the new side-tapping wire, is able to wrest control of messages sent to the destination.

All clocks in a subsumption system have approximately the same tick period (0.03277 seconds on the m68hc11), but neither they, nor messages, are synchronous. The fastest possible rate of sending messages along a wire is one per clock tick. The time periods used for both inhibition and suppression are recommended to be two clock ticks or say 0.08 seconds. Thus, a side-tapping wire with messages being sent at the maximum rate can maintain control of its host wire. In later versions of the compiler, this recommendation may be enforced unilaterally.

On the m68hc11, it is realistic to expect to simulate no more that 20 to 25 augmented finite state machines (AFSMs), running at about 30Hz (a Hz measurement indicates the peak number of event-dispatch states which an AFSM might enter in a second). The limiting factor is neither RAM nor cycle time, but rather the amount of eeprom available.

Organization

The optimizing compiler takes a file (which may contain 'include' statements for other files) and compiles it into a %program instance which can be assembled by the standard assembler, then downloaded with the downloader. In fact, only the downloader need be explicitly called, as it invokes the assembler.

Shown below is the contents of a file called Test.lisp which contains subsumption code describing two augmented finite state machines. The assembler code that this compiles down to (with the embedded operating system, called the scheduler, included) is shown at the end of this chapter.

```
;;; level 0

(defafsm test1 0
  :inputs ()
  :outputs ()
  :instance-vars (led-state)
  :states ((nil (event-dispatch (delay 1.0) s2))
           (s2 (progn (set-led))
               s3)
           (s3 (event-dispatch (delay 1.0) s4))
           (s4 (progn (unset-led))
               nil)))

;;; level 1

(defafsm test2 1
  :inputs ()
  :outputs ()
  :instance-vars (char)
  :states ((nil (event-dispatch (delay 0.001) n2))
           (n2 (setf char 0) n3)
           (n3 (event-dispatch (delay 0.5) n4))
           (n4 (cond (= char 26) n2 n5))
           (n5 (progn (write-hex (+ #.(char-code #\A) char))) n5a)
(n5a (event-dispatch (delay 0.04) n5b))
(n5b (progn (write-string '" foo "))
     n6)
(n6 (setf char (+ char 1))
   n3)))
```

The source file(s) specify a set of augmented finite state machines and the wires connecting them. Some such AFSMs and wires are specified directly. Others are the results of macro expansion. The compiler produces a customized scheduler plus the code of the bodies of the augmented finite state machines and the message delivery networks specified by the wires. It imbeds this in an 'operating system' which lets the complete compiled program run stand alone on a processor. The bodies of AFSMs can include a susbset of lisp expressions to compute functions of registers and robot i/o ports. The results of these computations either get sent out to other AFSMs as messages, get stored in local registers, or are used to set motor parameters via the I/O ports. These expressions rely on all the primitive lisp functions which they reference having been described to the compiler as primops.

The compiler compiles both an open coded scheduler and the bodies of the AFSMs. The scheduler works by checking every event-dispatch state in every AFSM in turn. If the scheduler finds such a state in which an AFSM is suspended, it checks the event conditions which the AFSM is waiting on. If one of them is satisfied, the scheduler transfers control to the appropriate state of the AFSM. That AFSM runs until it reaches an event-dispatch state, and immediately suspends itself without bothering to check whether any of the event conditions are satisfied. The AFSM returns control to the scheduler which goes on to the next AFSM in some predetermined sequence. The scheduler is thus quite simple. The penalty for this simplicity is that the user must guarantee that no AFSM will compute for more than a small time period before reaching an event-dispatch state. Thus, busy waiting, for instance, will starve out all the other AFSMs.

The 'operating system' is specified in the file squirt-op.lisp. It provides a number of macros needed by the compiler. Certain features the operating system provides are vital to successful running of subsumption code. It also provides tools for a debugging interface back to a host computer. In that sense it is self contained. However, if you need to add new device drivers you may well need to copy squirt-op.lisp and roll your own version. Notice that addressing modes are not flattened out as in the final version of assembly code; rather, they are grouped as in:

```
(ldaa (! 4))
(staa (&x offest))
```

It takes some care to write the device drivers this way, but the added inconvenience is necessary in order for the compiler to work so easily.

Invoking the Compiler

The simplest ways to call the compiler are:

```
(subcompile <filename>)
(subcompile <filename> :listing <listfile>)
```

Both <filename> and <listfile> should be strings. If a listing is requested, it is written into the <listfile> with detailed comments which let the user deduce what target code comes from what part of the source.

The result is a %program instance bound to a symbol produced by processing the <filename> string in some way. In fact, it will be the upper case name of the "first part" of the directory-free file name.

E.g.:

```
(subcompile "test.lisp")
```

will produce a %program, and setq TEST to it.

Selectively Compiling Only Parts of the System

The compiler scans all the forms within a file (and any indirect file it refers to). Typically, these will be defafsm and defwire statements. There is a mechanism to selectively compile only some of these defafsm's and defwire's. In its simplest form all AFSMs and wires are split into numbered levels. E.g., some machine might be designated level 3. There is a reserved position in the syntax for specifying both an AFSM and a wire where the level is specified. This means that only when level 3 or higher is requested will that machine be compiled. An additional keyword argument to subcompile lets the user specify what level of network should be compiled. E.g.,

```
(subcompile "test.lisp" :max-level 5)
```

says that all machines in levels 0, 1, 2, 3, 4, or 5 should be compiled. If max-level is bound to T then that says that all known levels should compile.

In fact, the 'max-level' facility is much more general than this.

Suppose for instance you have written a control system which has three major components; a navigation system, a striper orienter system, and a manipulation system. Suppose further that each of these subsystems is logically arranged in layers. Then you might at some point want to compile all of the navigation systems and the manipulation system, but only the orientation system up through level 2. This could easily be achieved by giving each AFSM and wire a level specification like (NAV 3), or (ORIENTER 1), or (MANIP 2), and then supplying a value like (NAV (ORIENTER 2) MANIP) for max-level.

The general mechanism is that all AFSMs and wires can implicitly be organized into subsets which live at the nodes of a tree. The max-level argument to subcompile specifies a set of subtrees which are to be compiled. One can think of the AFSMs and wires being organized like a file/directory tree in a file system, with optional version numbers on leaf nodes. The max-level argument specifies a number of subdirectories, with the wrinkle that any version number means 'include all versions of the same file with a lower number'.

In more concrete terms each AFSM and wire must have a level specification that is either:

a number	specifying a simple number level
an atom	specifying a simple named class

list of atoms	specifying a specialized AFSM or wire
list end in #	similar, with an ordering

Thus a level of A says that the AFSM (or wire) is in the A class. A level of (A B C) says it is in the C subclass of the B subclass of the A class. And a level of (A B C 3) says that is in the third layer of that same class. Now the max-level argument must be a list of subclasses to include in the compilation. E.g., ((A B) E) would include all members of the B subclass of class A, and all members of class E. Hence it would include an AFSM with a level of (A B C) or even (A B C 50). A max-level of ((E F) (A B C 4)) would include an AFSM specification of (A B C 3) but not one with specification (A B C 5). It would not include AFSMs whose level was simply E, but it would include those with (E F) or (E F G) or (E F G H 3).

More generally any entry within max-level can be a list of subclasses at that level. So for instance, a valid value for max-level would be:

```
((a (b d)) (e f (h j)))
```

which says to include all b and d subclasses of a, and all h and j subclasses of the f subclass of e. So a (a b x) AFSM will be included, but an (a c) AFSM will not. Likewise an (e f j) AFSM will work, but not an (e f i).

Specifying The Target Machine

The compiler is dynamically retargetable. I.e., it can have multiple backends present in core. For this reason it is necessary to specify which backend is to be used. One way is to use an additional keyword argument in the call to it, as in:

```
(subcompile <filename> :target 'm68hc11)
```

which says that <filename> should be targeted to the m68hc11. Another way to do it more globally is to say:

```
(set-current-machine 'm68hc11)
```

Compilable Forms

There are four compilable forms with the subsumption compiler: 'include', 'defafsm', 'defwire', and 'defthings'.

*** The 'include' form is used to include other files. The call is of the form:

```
(include <filename>)
```

where <filename> is a string. All the usual defaults apply to the file name in order to find which file is meant.

*** The 'defafsm' form is used to define an augmented finite state machine. The general form is:

```
(defafsm <name> <level>
   :registers <register-list>
   :outputs <output-port-list>
   :monostables <monostable-list>
   :states <state-list>)
```

The <name>, <level>, and <state-list> items are compulsory and the others are optional. In more detail each entry is as follows.

<name>

A symbol that is the name of the specified augmented finite state machine.

<level>

The level is a number which gets compared to the :max-level argument of subcompile in order to decide whether this AFSM should get included in the current compilation.

<register-list>

An unquoted list of symbols, each of which is the name of an 8 bit register. These registers contain all the state of the augmented finite state machine, besides the actual state of the machine.Any register can be an input to the AFSM, by connecting it as the destination of a wire coming from the output of another AFSM.

<output-port-list>

An unquoted list of symbols, each of which is the name of an output port. This must be referenced as the source of a defwire in order to direct the output to an appropriate place.

<monostable-list>

An unquoted list of things of the form (<mname> <time>) where <mname> is the name of a monostable element, and <time>, a number, is the monostable's characteristic period measured in seconds. Monostables are two-state elements which are set by a 'trigger' state and which time out after the characteristic period. The monostable is again quiescent until retriggered.

<states>

An unquoted list of state descriptors. See below for more details. #####The first state in the list must be named NIL.#####

The <states> describe the possible states of an Augmented Finite State Machine (AFSM). The possible forms of a state are (where asterisks mean you can have as many instances of the things in { }s as you want):

(<name> (event-dispatch {<event> <dispatchstate>}*))

(<name> (setf <register> <expression>) <nextstate>)

(<name> (conditional-dispatch <cond-expression> <thenstate> <elsestate>))

(<name> (output <port> <expression>) <nextstate>)

(<name> (trigger <monostable>) <nextstate>)

(<name> (sequence . <sequence of expressions>) <nextstate>)

There are certain synonyms for many of the state identifier words. They are:

Formal name	Synonyms
event-dispatch	ed
conditional-dispatch	cond, cd
sequence	progn

In all of the above state forms, <expression> can be an arbirary lisp-like form which uses available primops. All the leaves of the expression must be constants or register names, where the registers have been specified for the particular AFSM. Example expressions are:

```
(+ A B 3)
(+ (ASH B 3) (- A 57))
(MAX (LOGAND A 15) (- (+ B B) 3)
```

A <cond-expression> above is similar, except the outermost primop must be a predicate. Examples include:

```
(< (+ A B 3) 37)
(= (MAX A B) (MIN A B))
```

Each of <dispatchstate>, <thenstate>, <elsestate> and <nextstate> must be names of states; i.e., they must appear as the heads of other state descriptions.

In more detail now, the types of allowable states can be described as:

(<name> (event-dispatch {<event> <dispatchstate>}*))

> There can be multiple (more than one) pairs of <event>s and <dispatchstate>s. Each time the scheduler checks an event-dispatch state it goes through each pair in order, and triggers at the first <event> clause that is satisfied. An <event> clause can be a register name, a monostable name, a delay clause, or any boolean combination (using and, or, or not) of these. The semantics of each of these possibilities are:
>
> register
> is true if a message has arrived as an input to this register since the previous event-dispatch state was left.
>
> monostable
> is true if the monostable is in its triggered state and false otherwise.
>
> (delay <n>)
> where <n> is a number measuring seconds. Is false until <n> seconds after the event-dispatch was entered (i.e. the time at which the AFSM was suspended) and thereafter is permanently true. Note that this is equivalent to preceeding the event-dispatch clause with a trigger of a monostable of the same time period, and then checking for (not <monostable>) in the event-dispatch clause.

(<name> (setf <register> <expression>) <nextstate>)

This simply sets the named <register> to have the value given by <expression> then transfers control to <nextstate>.

(<name> (conditional-dispatch <cond-expression> <thenstate> <elsestate>))

This checks the outcome of the <cond-expression> and transfers control to <thenstate> if it was true and to <elsestate> otherwise.

(<name> (output <port> <expression>) <nextstate>)

This passes the value of <expression> to the output named <port> then passes control to <nextstate>.

(<name> (trigger <monostable>) <nextstate>)

This triggers the named monostable for its declared time period then transfers control to <nextstate>. If the <monostable> was already triggered this retriggers it for the full time period.

(<name> (sequence . <sequence of expressions>) <nextstate>)

This is just a means of evaluating a bunch of expressions involving primops which are really meant for side effects rather than returned values. For instance most primops which actually interface to the robot (or the serial port) fall into this category.

*** The 'defwire' form is used to specify a wire connecting one output to many inputs. The general form of a 'defwire' is

(defwire <level> <output-spec> <input-spec1> <input-spec2>... <input-specn>)

The <output-spec> is the name of an output port on a particular named AFSM. It has the general form:

(<AFSM name> <output port name>)

There are five forms for specifying an <input-specj>. They are

regular

has the form (<AFSM name> <register name>) and connects the wire so that it writes into the specified register.

reset

has the form ((reset <AFSM name>)) and makes any message on the wire force the specified AFSM into the NIL state.

inhibit

has the form ((inhibit (<AFSM name> <output port name>) <time>)) and lets messages inhibit all output from the specified port for a time period specified by the number <time>. The

<time> specification may disappear from later releases.

supress

has the form ((supress (<AFSM name> <register name>) <time>)) and lets the message supress and replace all input to the specified register for a time period specified by the number <time>. The <time> specification may disappear from later releases.

default

has the form ((default (<AFSM name> <register name>) <time>)) is identical to supress except that the semantics of the two wires are swapped.

Note that 'defwire's are incremental. So that:

```
(defwire 2 (foo a) (bar b))
 ...<other stuff>
(defwire 2 (foo a) (baz c))
```

is entirely equivalent to

```
(defwire 2 (foo a) (bar b) (baz c))
```

*** The 'defthings' form is used to specify macro expansion forms.

It has the general form

```
(defthings <name> <formal-parameter-list>
  <form1>
  <form2>
  ...
  <formn>)
```

This defines a macro which expands to n top level forms by substituting the supplied arguments for the elements of the <formal-parameter-list> in each of the <formi>s. For instance after:

```
(defthings foo (a b c)
  (defafsm a 3
    :registers (b)
    :outputs (c)
    :states ((nil (ed (delay 1.0) baz))
                         (baz (output c b) nil)))
  (defwire 3 (central out) (a b))
  (defwire 3 (a c) (central in)))
```

Then we get the following expansion:

```
(foo mod1 inreg outport)
==>
(defafsm mod1 3
   :registers (inreg)
```

```
    :outputs (outport)
    :states ((nil (ed (delay 1.0) baz)
                          (baz (output outport inreg) nil)))

(defwire 3 (central out) (mod1 inreg))

(defwire 3 (mod1 outport) (central in))
```

Thus defthings is somewhat similar to def-ass-subst except that it produces multiple results.

Available Primops

Most primops concern arithmetic. All arithmetic is 8 bit signed integers. Some primops are built into the compiler. These include:

primops	args	predicate?
+	2-inf	
-	2	
+trunc	both args pos	
-trunc	first arg neg, second pos	
ashr	2 (2nd arg must be literal)	
evenp	1	t
oddp	1	t
abs	1	
max	2-inf	
min	2-inf	
logand	2	
logior	2	
logxor	2	
lognot	1	
<	2	t
>	2	t
<=	2	t
>=	2	t
=	2	t
/=	2	t
memq	2 (2nd arg is constant)	t
examine	1 (assembler symbol)	
deposit	2 (symbol, value)	

The primops examine and deposit refer to symbols anywhere in the assembled program. Sometimes they might be useful for interfacing subsumption programs to background things (e.g., a token ring that writes into and reads from certain locations).

Other primops, such as the following, are built into the squirt-op.lisp file. They all operate over the serial line of the m68hc11 at 9600 baud. For the writing primops, all operate in the background, and it is up to the caller to leave enough time for them to transmit before calling another writing primop.

primop	args
read-char	0
write-char	1

write-space	0	
write-crlf	0	
write-hex	1	
write-strin	1	string must be quoted: (write-string '"foo")

come with the squirt-op file and can be retained or deleted as you wish if you decide to make your own private version.

The Squirt-op Environment

The default squirt-op.lisp file provides an implementation of the subsumption architecture where the granularity of the clock as seen by delay clauses, monostables, and inhibit, supress and default is one tick per 32.77ms or approximately 30.5Hz.

It also provides a set of primops for communicating over the 9600 baud line. This takes about 150 bytes and can be removed from the generated program by setting the global variable *INCLUDE-IO-CODE* to nil before invoking the subsumption compiler via subcompile.

The 9600 serial convention used by squirt-op does not allow either 0 or 255 to be sent. In fact, many such characters get randomly interspersed in the character stream. On the 68HC11 end, the read-char primop returns 0 when there is no new character in the input buffer. On the MAC or host end, the procedure CV sets up the baud rate correctly then prints out any transmitted characters after disposing of 0s and 255s. Calling (CV) is useful then for checking the information coming up from the robot on its way to be processed with offboard computation.

Cross-Development Using a Host Computer

Ian Horswill

Running the Development Environment

A set of programming utilities (assembler, compiler, etc.) are available to run on a host computer (an Apple Macintosh, Unix machine, or a Symbolics lisp machine). This will allow you to edit and compile programs using the host machine, and download them to the 6811 processor using a serial cable. The utilities are written in Common Lisp, so unless you are using a lisp machine, your first task will be to get lisp running. If you are using a Macintosh, you can do this by taking the floppy disk provided with your kit and double-clicking on the file "load.lisp" in the folder "Compiler" on the floppy disk. Note that your Mac must already have a copy of Coral Common Lisp.

To invoke lisp on a Unix workstation, either a Sun or an HP, invoke Emacs, and type the command "M-x load-file <return> /com/olympics/clisp.elc". Then type "M-x run-clisp". Lisp will be run as a subprocess of emacs in an emacs buffer. Once clisp.elc has been loaded, files with the extention ".lisp" will be put in Common Lisp mode. The major feature of clisp mode is that you can pass functions from clisp mode buffers to the lisp process for evaluation by typing "C-C C-E". Note that Lisp should only be run on a single user workstation, not on a multi-user timesharing system such as rice-chex or wheat-chex.

Once you have lisp running, you need to load the development environment. To do this, first load the file "load.lisp". If you are using a Mac, then load.lisp was automatically loaded when you double-clicked on it. If you are using a Unix machine, you need to type "(load "/com/olympics/load")" at lisp. On a lisp machine, you should type "(load "wh:/com/olympics/load")" or ":load file wh:/com/olympics/load". Having loaded the file, you need only type "(loadup)" at lisp and wait for it to load its files.

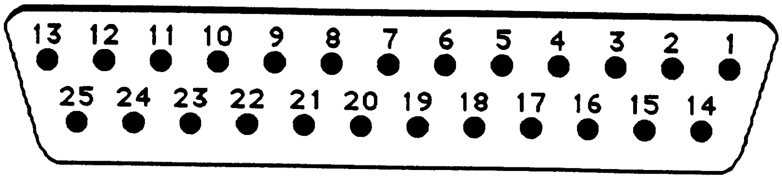
Unix and Lisp Machine users may wish to check the file /com/olympics/README occasionally for news of software changes, etc.

Cables and Connectors

To actually connect your host machine to your 6811 processor, you will need to build a cable. There are two things which make this a little complicated. One is that the connectors on the different machines have different shapes and pinouts (assignment of functions to the various pins making up the connector). The other is that the connector on the 6811 must also determine whether the 6811 is in normal mode or bootstrap mode.

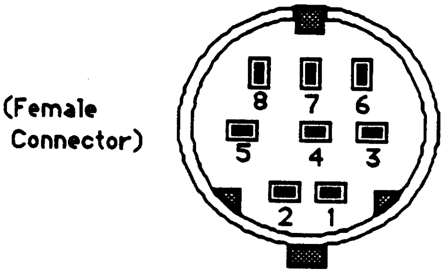
Flavors of Connectors

Most host machines use the DB-25 connector which looks like this:



The lisp machines have three DB-25 conectors on the back of the processor cage in addition to another on the back of the console monitor. Sun workstations generally have them either on the CPU box if there is one, or on the back of the base unit for the desktop models. HP workstations have them on the back of the CPU cage. By default, the development software will use the console connector on lisp machines and port A on Unix machines. Mating connectors can be obtained at most electronics stores or at the electronics stock room, room 908. Check whether the connector on the back of your host is male or female so you can be sure to get a mating connector.

The Macintosh uses a funny circular connector which looks like this:



NOTE: do *not* use the DB-25 connector on the back of the Macintosh, as it will fry the Macintosh's logic board. This is an $800 mistake, more if you have a Mac II or an accelerator card. Since the Mac connectors are hard to obtain and hard to solder, it is probably easiest to buy the Apple Peripheral cable which has one of these connectors on each end and cut it in half, soldering to the existing wires rather than trying to open up the connector. Alternatively, you could buy a ready-made cable with the Mac connector on one end and a DB-25 on the other, and make a DB-25 cable to connect to the processor. The ready-made cables are available from the the MIT Microcomputer Center.

Finally, the 6811 board uses its own type of connector fabricated from mobot connectors (terminal strips and socket strips). You will use the same mobot connectors for this that you used in building the 6811 processor board. Simply take two strips of four pins each and glue them together. Note that there are *two* serial connectors on the 6811 board - TTL and RS-232. It is very important that you use the RS-232 connector (the connector with 8 sockets for pins in the upper left-hand corner of the board), and not the TTL connector (with six at the bottom). Wiring the host straight into the TTL connector could fry the processor board.

Cable Connections

Our cable needs only needs three wires: receive, transmit and a ground. The DB-25 connector always carries ground on pin 7, and the Macintosh carries it on pin 4. This should be routed to the ground (VSS) in the connector for the processor board. The Macintosh also uses a slightly different type of serial line driver and so it also needs pin 8 (RxD+) connected to ground. The receive and transmit lines vary because some host machines use DCE format and other use DTE format. One will always be pin 2 and the other pin 3 however. Look up the connections in the table below and wire them to the receive and transmit lines on the connector for the 6811 processor.

Machine	**VSS**	**RxD**	**TxD**	
Sun 3/160	7	2	3	
Symbolics 3600	7	3	2	
HP	7			
Macintosh	4	3	5	*also tie pin 8 to VSS*

As of this writting, we don't have the pinouts for HP workstations. Check the file wheaties:/com/olympics/cables for more information on HPs or if you have any problems.

Bootstrap Mode

Finally, the MODB signal has to be jumpered to VSS on the 6811 processor in order to put the processor in bootstrap mode. Connect the MODB signal on the processor connector to the VSS signal using a little bit of wire. When the cable is plugged into the processor board and the reset button is pushed, the processor will go into bootstrap mode. To run the processor in normal mode, simply unplug the cable, reset the processor, and (optionally) plug the cable back in. If you expect to be using the cable frequently in both normal and bootstrap mode, it may be worth your while to replace the jumper with a switch.

Batteries

Anita Flynn

Power supplies are always one of the biggest problems in building autonomous robots. If your robot isn't mobile, you can save yourself alot of trouble and use an external power supply. Even if your robot is mobile, we recommend you use the power supplies given in the kit for debugging and prototyping and save the batteries for test runs.

You'll find two types of lithium batteries in your kit. One, the Duracell DL123A, (which Duracell was kind enough to donate) is a 3V, 1300mAhr battery and can deliver a nice punch of current for its size. The other is the Duracell DL1/3N, a 3V, 160mAhr battery. Although smaller, two of these 3V batteries ran the microprocessor and an (unloaded motor) on Squirt for 3 hours straight in one test with plenty of juice left over. Specification sheets for these batteries are shown on the next page.

Other batteries are available and feel free to use them, although they may be a bit heavy. NiCads are rechargeable, have high current capability and can be found in nearby Radio Shacks. Alkaline cells can be acquired from the basement supply room. Top of the line batteries are silver-zinc or silver-cadmium cells from Yardney Battery, but they typically run 10 times the cost of normal lead acids, although they also run 3 or 4 times smaller in size and weight.

Other sources of power are possible, but usually have some drawbacks. Solar cells are clean but don't supply much power in indoor environments. A small gasoline engine would be ideal to have, but they get kind of smelly indoors. An alternative is CO2 cartridges, such as the type used on model airplanes. Rubber bands work pretty well too.

Any innovations in the power supply arena are always welcome. Barring major advances, your best bet is to be conservative with your power budget. All the electronics supplied with your 6811 card are CMOS and the processor itself draws only 15mA. You can always leave out the MAX233 chip too, if you really want to trim power consumption. If your entire robot is very small, it may be possible to get by with wimpy motors. The Micro Mo and Maxon motors supplied can run at about 30mA. We haven't been able to find motors any smaller. If you want smaller motors, you'll have to micromachine your own. That's a bit of a project too.

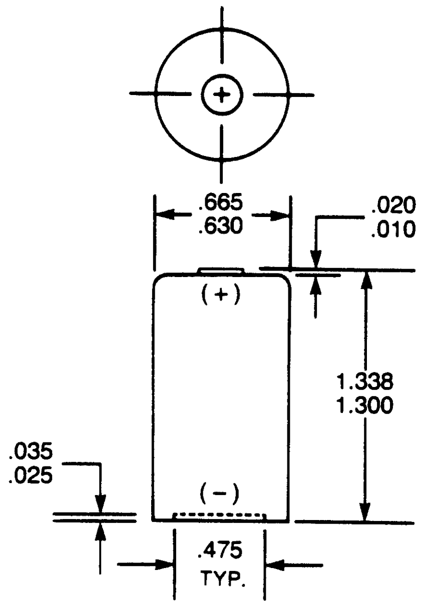
DURACELL®

DL123A

Size: 2/3A

Li/MnO_2

Lithium/Manganese Dioxide Battery



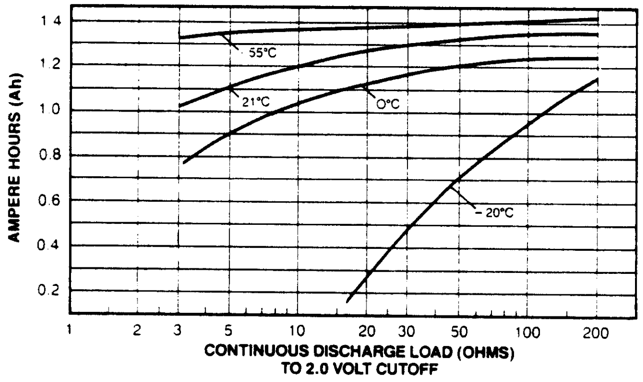
Inches	mm
1.338	33.99
1.300	33.02
.665	16.89
.630	16.00
.475	12.07
.035	0.89
.025	0.63
.020	0.51
.010	0.25

SPECIFICATIONS

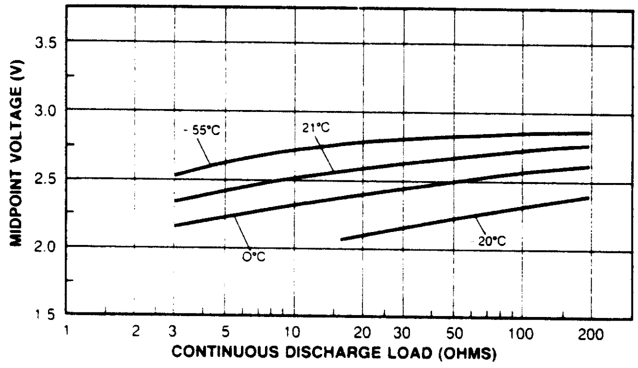
Nominal Voltage:	3.0 V
Typical Voltage:	3.2–3.3 V
Rated Capacity:	1,300 mAh on 200 Ω to 2.0 V at 21°C (70°F)
Average Weight:	0.564 oz. (16.0 g)
Volume:	0.422 in.^{3} (6.92 cm^3)
Terminals:	Flat, Neg. End Recessed
Operating Temp. Range:	−40°C to 60°C (−40°F to 140°F)
NEDA/ANSI:	5018LC
IEC:	—

Note: Designed for 3-volt user-replaceable applications. Battery is comprised of one DL2/3A cell.

DELIVERED CAPACITY vs. LOAD



MIDPOINT OPERATING VOLTAGE vs. LOAD

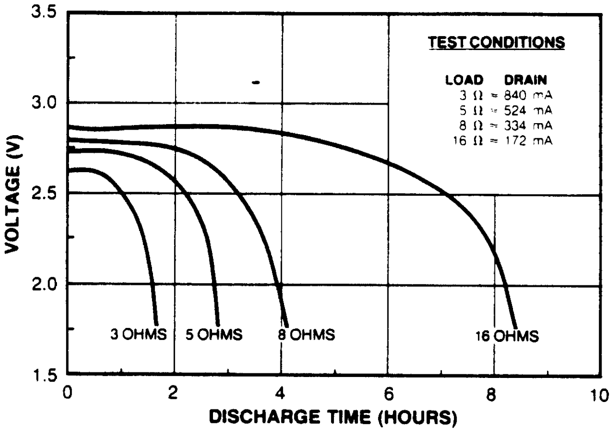


DURACELL INC.

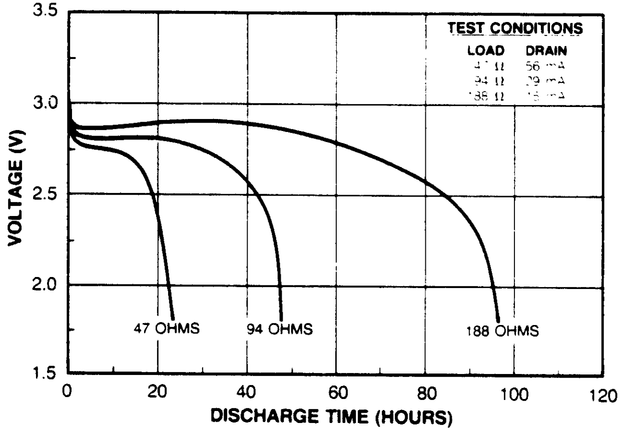
O.E.M. Technical Sales and Marketing
Berkshire Industrial Park
Bethel, CT 06801 U.S.A.
Telephone: Toll-free 1-800-431-2656 (In CT, 203-791-3274)
Telex: 697-2165 DURTSUW

PRINTED IN U.S.A.

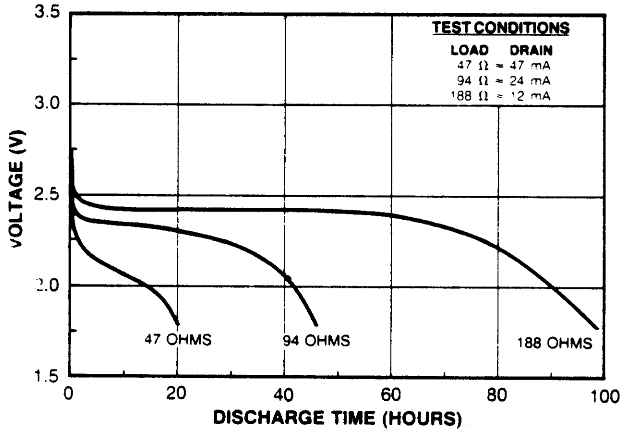
CONTINUOUS DISCHARGE AT 55°C (131°F)



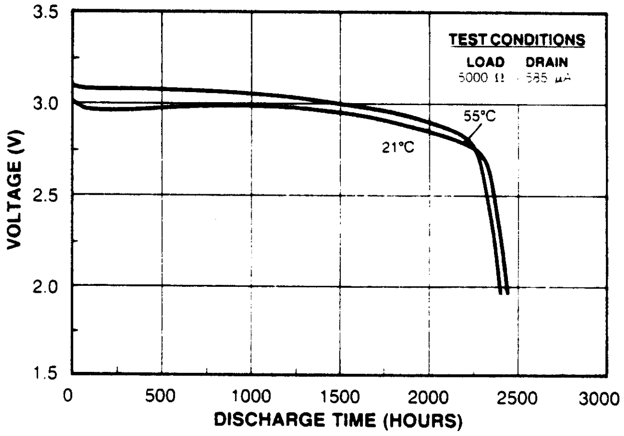
CONTINUOUS DISCHARGE AT 21°C (70°F)



CONTINUOUS DISCHARGE AT -20°C (-4°F)

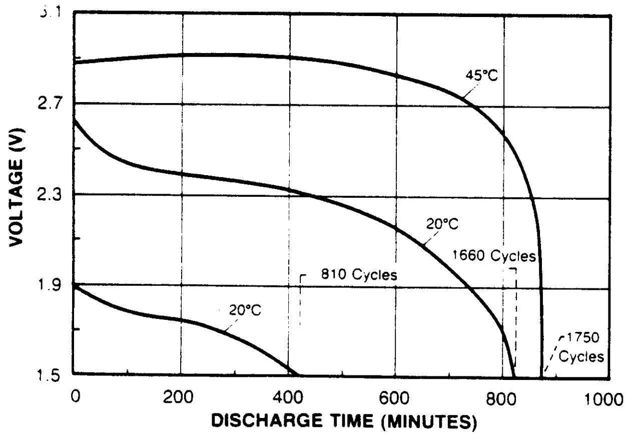


CONTINUOUS DISCHARGE AT 5000 OHMS



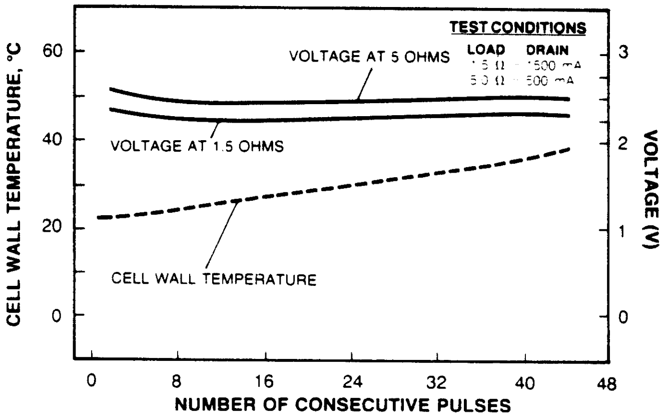
PULSE DISCHARGE AT 900 mA

3 SECONDS ON, 27 SECONDS OFF



PULSE DISCHARGE AT VARIABLE LOADS

1.5 OHM/7 SEC – 5 OHM/3 SEC, REPEATED

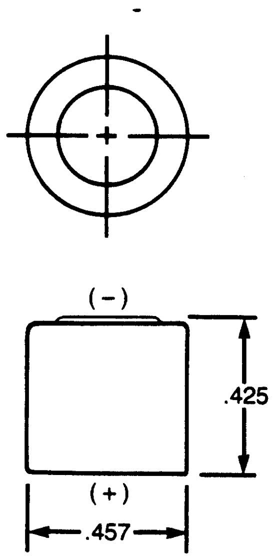


DURACELL®

DL 1/3N

Size: 1/3N

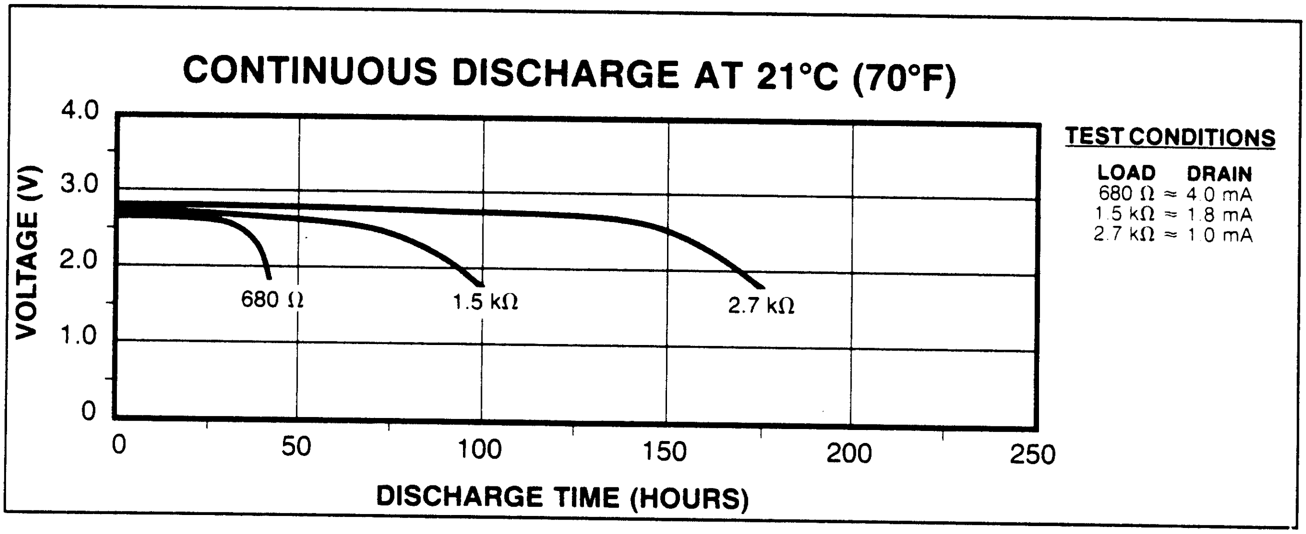
Lithium/Manganese Dioxide Battery



Inches	mm
.457	11.60
.425	10.80

Dimensions shown are maximum.

SPECIFICATIONS	
Nominal Voltage:	3.0 V
Typical Voltage:	3.2–3.3 V
Rated Capacity:	160 mAh on 2.7 kΩ to 2.0 V at 21°C (70°F)
Average Weight:	0.116 oz. (3.3 g)
Volume:	0.069 in.^{3} (1.13 cm^3)
Terminals:	Flat, PC Pins, Tabs
Operating Temp. Range:	– 40°C to 60°C (– 40°F to 140°F)
NEDA/ANSI:	5008LC
IEC:	—

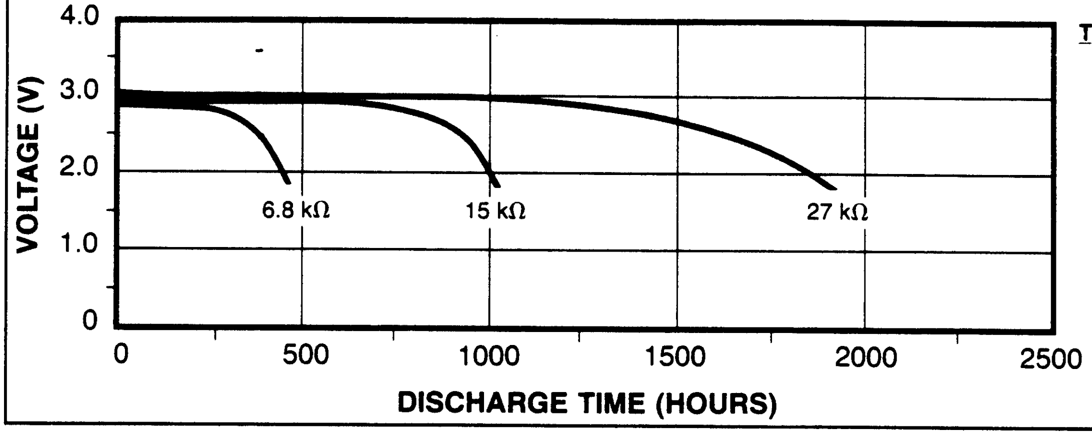


DURACELL INC.

O.E.M. Technical Sales and Marketing
Berkshire Industrial Park
Bethel, CT 06801 U.S.A.
Telephone: Toll-free 1-800-431-2656 (In CT, 203-791-3274)
Telex: 697-2165 DURTSUW

PRINTED IN U S A

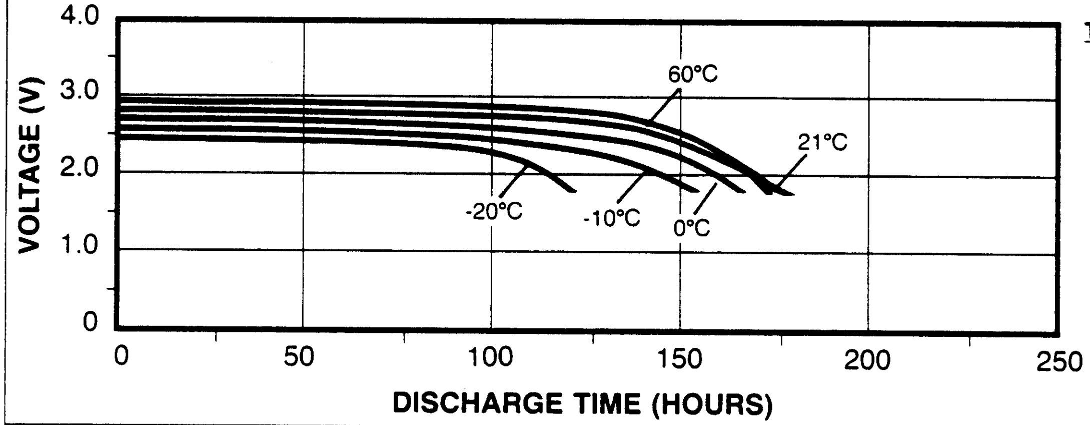
CONTINUOUS DISCHARGE AT 21°C (70°F)



TEST CONDITIONS

LOAD	DRAIN
6.8 kΩ	≈ 0.4 mA
15.0 kΩ	≈ 0.2 mA
27.0 kΩ	≈ 0.1 mA

EFFECT OF TEMPERATURE ON PERFORMANCE



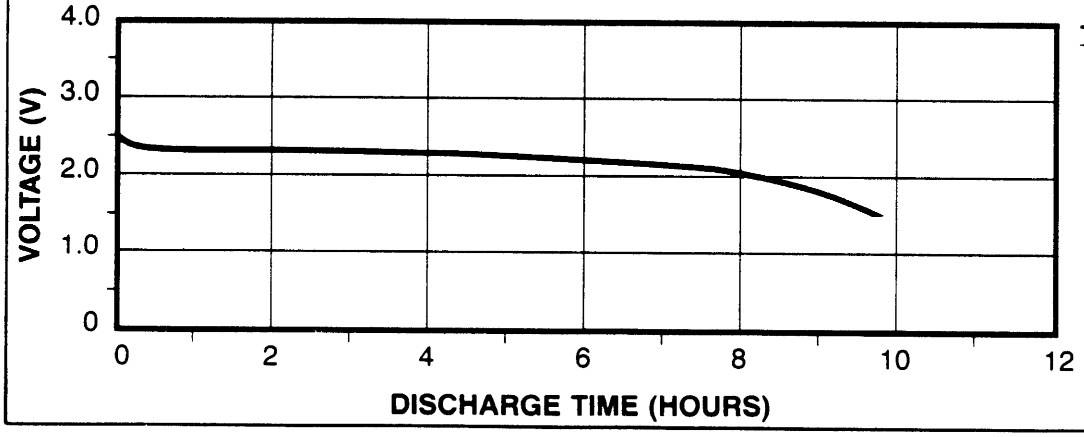
TEST CONDITIONS

LOAD	DRAIN
2.7 kΩ	≈ 1 mA

TEMPERATURE

-20°C (-4°F)
-10°C (14°F)
0°C (32°F)
21°C (70°F)
60°C (140°F)

PULSE DISCHARGE CHARACTERISTICS



TEST CONDITIONS

PULSE LOAD:
50 Ω

TIME ON:
1 second

TIME OFF:
2 seconds

TEMPERATURE:
21°C (70°F)

Connectors

Anita Flynn

The running joke in the Mobot Lab is that robotics is a deep theoretical study in connectors.

A robot has tremendous numbers of wires running between various subsystems such as power supplies, computers, sensors and actuators. Although computers, even Connection Machines, have huge interconnect problems, the topology is fairly uniform. On a robot however, geometry comes into play and signals have to be carried to sensors or actuators which are geographically scattered. Genghis, the foot-long six-legged walker, has 72 connectors. Herbert has over three hundred and Seymour is a nightmare waiting to happen.

It's important to connectorize everything, though, especially in the prototyping stages, because you'll tend to want to take things apart, fix them and add to them more often than you actually run the machine. If systems are soldered or hardwired, they'll be too much of a hassle to fix. The ideal to aim for is to be able to completely strip down your machine in a matter of minutes and to accomplish that feat without the aid of a single hand tool.

The main problem with putting connectors everywhere is that they take up too much room and a nice electronics board that you've carefully designed to be as small as possible is typically overwhelmed by the space required for connectors. After four years of deep theoretical study, we've finally chosen a connector technology that fits our bill - it easily interfaces to the prototyping technology we've selected for board layout, it's easy to fabricate and it's dense. We call these connectors Mobot Connectors, for want of a better name, as the parts we used originally had a very different purpose. First however, some background on board wiring technology for prototyping is required.

Prototyping Electronics

You'll notice some protoboards are given out in your kit. These contain power supplies and typically are used for breadboarding circuits using 22 gauge solid wire. We don't recommend you use protoboards. They're included in the kit solely to act as power supplies so that you won't drain your batteries. The problem with protoboards is that, invariably someone before you has jammed 20 gauge solid wire into the holes and then when you go to use it, you get flaky connections.

What are the alternatives available for quick and modifiable prototyping? Wire wrapping sockets into perfboard used to be the technology of choice, but it takes a long time because you have to strip both ends of each wire and daisy chaining with one piece of wire is impossible.

Scotchflex sockets and plug strips are much simpler and faster to use. The resulting profile of the bottom of the card is also much lower. Scotchflex sockets come in all the standard DIP package sizes for integrated circuits. A socket is placed on top of a piece of perfboard and corresponding plug strips are inserted from the back. Kynar 30 gauge wire is merely pushed into pins on the back. Two wires at most can be pressed into any pin, but daisy chaining is possible and no wires have to be stripped. It's possible to mix wire wrap and Scotchflex technology if necessary. For instance, if a wire wrap ribbon cable connector has to be used, wires can be wire wrapped to the ribbon cable connector and then pushed into Scotchflex pins elsewhere on the board.

There are a few drawbacks to Scotchflex technology. For long-term use, they can be unreliable. Most of the problems come from pulling chips out of the sockets (for instance, EPROMs that are often changed). Using Scotchflex technology, which is composed of two pieces on either side of the perfboard, the problem of pulling the sockets apart from the pins and off the board arises. Repeatedly doing this can cause intermittent connections. Another problem with Scotchflex is that sockets only come for DIP package sizes. Square chips, like the 6811, which come in plastic leaded chip carrier (PLCC) configurations, cannot be breadboarded using Scotchflex. However for testing out a normal quick circuit, which you may later want to turn into a printed circuit board, Scotchflex is the way to go.

What other alternative is there? Speedwire is a good choice and is the technology we recommend. Speedwiring consists of taking Speedwire terminals (which come in reels of 1000) and pushing them into the holes in a piece of perfboard. The perfboard we've supplied has holes on 0.100" spacings. The terminals are pushed through the board and on one side have a receptacle for an IC pin, and on the other side, have a pronged end for holding wire. Again, 30 gauge wire is pushed into the back side of these terminals. A special wiring tool, called a Speedwire pen, is needed to do this wiring. As with Scotchflex, daisy chaining is possible and a maximum of two wires is allowed per pin.

The advantage with Speedwire however, is that since each terminal is not made out of two pieces, there's no chance of separation when removing chips. In addition, since you place each Speedwire terminal individually on the perfboard, there is no constraint as to the package configuration of the chips you use. It would be very simple then to Speedwire a 6811 board similar to the printed circuit version we've supplied in your kit. The only disadvantage to Speedwire is that you have to spend the time pushing pins, but that's the tradeoff for reliability. Note that there's one trick in pushing the pins: they should all be lined up so that the little holes in the pins on the wiring side face 45 degrees to the rows of holes in the board. This leaves more room for bending the wires when you start wiring up your circuits.

Speedwire technology can be combined with printed circuit board technology and in fact, that's exactly what we've given you on your 6811 board. Notice the empty holes to each side of the 6811 socket. Those holes are "breadboard" space. You can insert Speedwire pins there and wire up any external circuitry you need. If you need more space, you can build a separate board out of perfboard using Speedwire pins and then use Mobot Connectors to connect the two boards.

Mobot Connectors

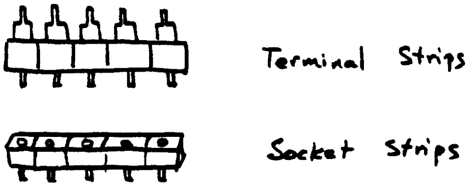
Mobot Connectors are a simple hack to solve the interconnect problem between Speedwire pins and external objects such as sensors or motors. What they really are, are things called terminal strips and socket strips, which are normally used to make a row of vertical connections between two stacked printed circuit boards. A terminal strip is a a long piece of plastic with a row of pins sticking out the top and bottom at 0.100" spacings. These normally fit into sockets strips, which are like terminal strips except that one surface has receptacles instead of pins. Socket strips would typically be soldered into a printed circuit board and then the terminal strips pushed into them to make a removable connector.

On the 6811 card you've been given, you'll solder socket strips into the holes that have solder pads meant for port output connectors. In the breadboard area, you can push Speedwire pins. In either case, terminal strips (mobot connectors) can be inserted into them. However, instead of stacking another card on top, you'll want to make connectors that go to sensors, motors or power supplies.

The procedure is to use a vise, tin the top leads of the mobot connectors, get some stranded wire, slip on a piece of heat shrink tubing (I guarantee that everyone will forget to do this at least once), solder the wire onto the pin, slip the shrink tubing over the joint and then blast it with a heat gun. Voila! - now you have a connector. This is an important skill. Learn how to do this fast and you'll be all set.

You can often do this on the other end of the cable too. For instance, for interfacing to a pyroelectric sensor, you can build a 2x2 mobot connector (use an exacto knife, super glue and a socket strip) and push the 3 pins of the pyro sensor right into the mobot connector. If you drop a glob of solder into the unused quadrant, you'll have a connector that is keyed so that you can never plug it in wrong.

Keying connectors is an important point. Always add an unused square of mobot connector so that you can key your cable. This is CRITICAL for power connections. Attention to detail in this department will save vast pain and agony later.



Debugging

"If it doesn't work, it's probably the connectors..."

Anita Flynn

What are the tricks for debugging your robot? The first heuristic is 'Attentiun to Detail'. That means build it carefully in the first place. Don't be sloppy and gloop globs of solder all over your board. Don't skimp and build shoddy cables. Don't hurry and plug power in backwards. Before you ever insert any chips or apply power for the first time, use an ohmmeter to convince yourself that +5V is not shorted to Ground and that all +5V points are connected together, as are all Ground points. Check your work and make sure there aren't ɔny solder bridges or cold solder joints. Layout your Speedwire area neatly and make sure that ends of wires don't touch other pins. Never put more than two wires in any Speedwire terminal. Use stranded wire rather than solidwire on all cables.

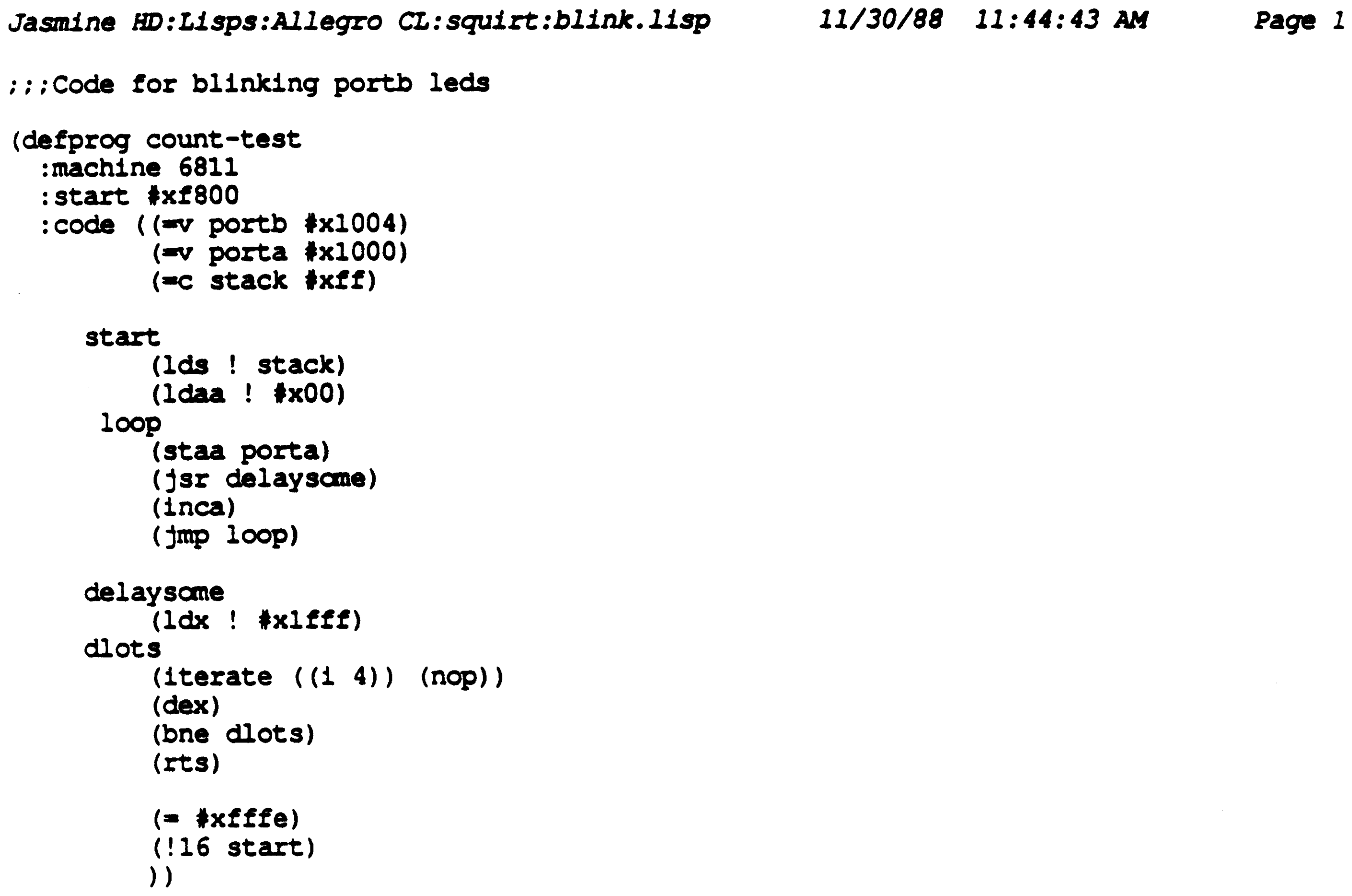
What if it still doesn't work? The best way to proceed is to go back to square one and find something that does work. Check first that power is getting to all your chips. Look at your power supply on the scope to make sure it's not too noisy. If it is, add a capacitor. It's good practice to put capacitors near the power cables running to sensors (such as pyroelectrics - if you give them junk, they'll return junk).

The next thing to do is check to make sure your 6811 is running. First check the clock. There's a signal on pin 5 called E, which is the bus clock. E is equal to the frequency supplied by your crystal divided by 4. With the 8MHz crystals you've been given, E should be a 2MHz square wave.

Now you should try and download a program. Turn the power to your 6811 off and plug in your downloader cable. Insertion of the cable causes the processor to be put in bootstrap mode. Turn the power on. Once you've evaluated a buffer containing the program you want to download, you should download it by typing (dl <prog-name>). You should see several lines of periods printed across your screen as downloading proceeds. If it just hangs instead, check to make sure your cable is correct. An easy way to do this is to run a terminal program such as Kermit and connect together the receive and transmit pins at the end of your cable. Then start typing at Kermit. You should see the characters you typed echoed back on your screen.

If that works, plug the cable back into your board and check the pin on the MAX233 which is Receive In. Trace it through to the Receive Out pin on the MAX233 and finally to the Receive pin on the 6811. Try replacing the 6811 if that all looks okay. In general, just go step by step, starting from something that has to be right, like power being correctly applied, all the way to the signal you're checking.

If downloading is successful, try to run your program. Turn the power to the 6811 off. Remove the downloader cable (now the 6811 will be in single-chip mode). Turn the power back on, and your program should run. A simple first program to run is a test routine for the hex-digit LCD display which is connected to portA, pins 3-6. Try a simple program which counts from 0 to F, as shown below:



Jasmine HD:Lisps:Allegro CL:squirt:blink.lisp *11/30/88 11:44:43 AM* *Page 1*

```
;;;Code for blinking portb leds

(defprog count-test
  :machine 6811
  :start #xf800
  :code ((=v portb #x1004)
         (=v porta #x1000)
         (=c stack #xff)

     start
         (lds ! stack)
         (ldaa ! #x00)
      loop
         (staa porta)
         (jsr delaysome)
         (inca)
         (jmp loop)

     delaysome
         (ldx ! #x1fff)
     dlots
         (iterate ((i 4)) (nop))
         (dex)
         (bne dlots)
         (rts)

         (= #xfffe)
         (!16 start)
         ))
```

When that all checks out, you're ready to hook up motors and sensors. Again, go incrementally, coercing small subsystems to work at a time. If there's a point in your circuit which has a signal that's not what you wanted or expected, try removing the load from that point on. Possibly some part of the circuit after that was shorting to Ground or loading it down.

The worst part about debugging hardware is tracking down intermittent failures. Usually the problem here is flaky connectors. Hopefully with the printed circuit boards we gave you, the number of these types of problems should be small. Check Speedwire layouts for broken wires, wires that may have been bent too acutely and are about to break, or cut ends of wires touching other pins. Try bending cut ends of wires upwards. Ohm out cables to your sensor or motor and make sure they're reliable. Flair the pins of Mobot Connectors out slightly with a pair of pliers if it wobbles in the socket strip.

If all that fails, blame it on software.

An Example Robot Control System - Squirt's Brain

William Wells

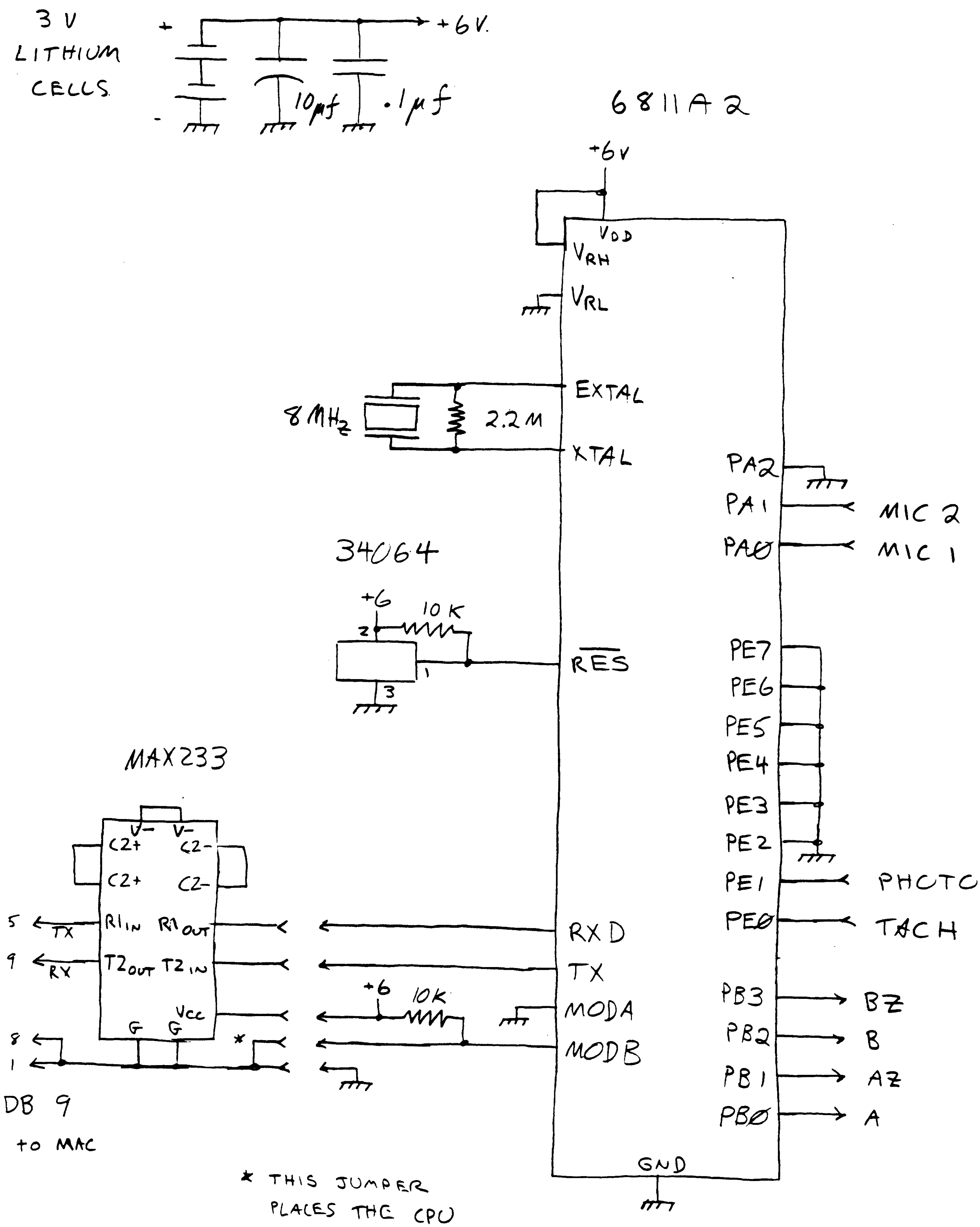
An example of how to build a very simple robot using a 6811 microprocessor and a few simple sensors and actuators is provided by the robot Squirt, presently under development in the Mobot Lab. Squirt is a three-sensor, one-actuator robot and his basic operating mode is to act like a bug. He has one photodiode light sensor to enable him to search for and hide in dark corners and he has two microphones which allow him to move in the direction of noise (but only when it's dark). The design goal in building Squirt is to fit the entire system into a one cubic inch volume and therefore create the world's smallest robot. Consequently, there isn't much room for lots of motors and sophisticated drive trains. Squirt's entire propulsion system consists of one motor driving two wheels, along with two castors - allowing the robot to only either go forward or back and turn. Minimalism is the order of the day.

Figures one and two show the control system of the robot. Squirt's circuitry is a simple example of a control system with computer, sensors, and effector. The complete circuit can be built on a one inch square printed circuit board if surface mount parts are used. Figure one shows the cpu, power, and serial downloader circuits. The entire robot is powered by two 3V lithium cells. The 34064 chip is a low voltage inhibitor connected to the reset circuit which holds the 6811A2 cpu in a reset condition if the operating voltage isn't high enough. This prevents the cpu from crashing and eating the code in the on-chip eeprom (electrically eraseable programmable ROM) when the battery voltage sags below legal operating levels.

The circuitry in the lower left is a 'downloader cable'. When the downloader cable is connected to the cpu board, it forces the cpu into a mode such that the eeprom can be loaded from the serial port. (Note that this cable can't be used as an ordinary serial link as it stands, because of the jumper which forces the cpu into bootstrap loading mode. Add a switch here if both downloading and communicating modes are desired.) The MAX233 chip is a level translator which converts the logic levels of the 6811A2 to EIA levels for the serial port of the host computer. After downloading, the cable should be disconnected and the circuit power cycled to set the 6811 running in single-chip mode.

The signals on the right side of the 6811A2 connect to the sensor and motor control circuits of figure two. The port B (PB) lines are logic signals used to drive the motor controller. The port E (PE) lines are analog to digital converter input lines. One is used to read the tachometer signal from the motor (to be used in a servo loop to control the motor's speed). Another reads the ambient light level. The rest of the A/D inputs are grounded. The port A (PA) signals are connected to timer-counter circuits in the 6811A2. They are used to measure the time difference of the onset of sounds near the robot.

The top of figure two shows the motor controller. This is an H bridge realized with CMOS bus driver parts. 74HC series parts are used for their wide operating voltage tolerance, which is convenient for battery operation. The motor driver circuit is enabled when AZ and BZ are held low. The motor may be driven in either direction by placing logically opposite signals on A and B. The motor may be proportionally controlled by duty cycle modulating the drive. Reasonable chop frequencies are in the range of 10 hz to 50 khz. The protection diodes in the CMOS parts provide adequate clamp diodes for the small motor which is used in Squirt. This driver circuit saturates at about 50 mA. A more powerful H bridge could be made from discrete transistors (and diodes), or an integrated H bridge driver chip could be used.



* THIS JUMPER PLACES THE CPU IN BOOTSTRAP LOADING MODE

FIGURE 1.

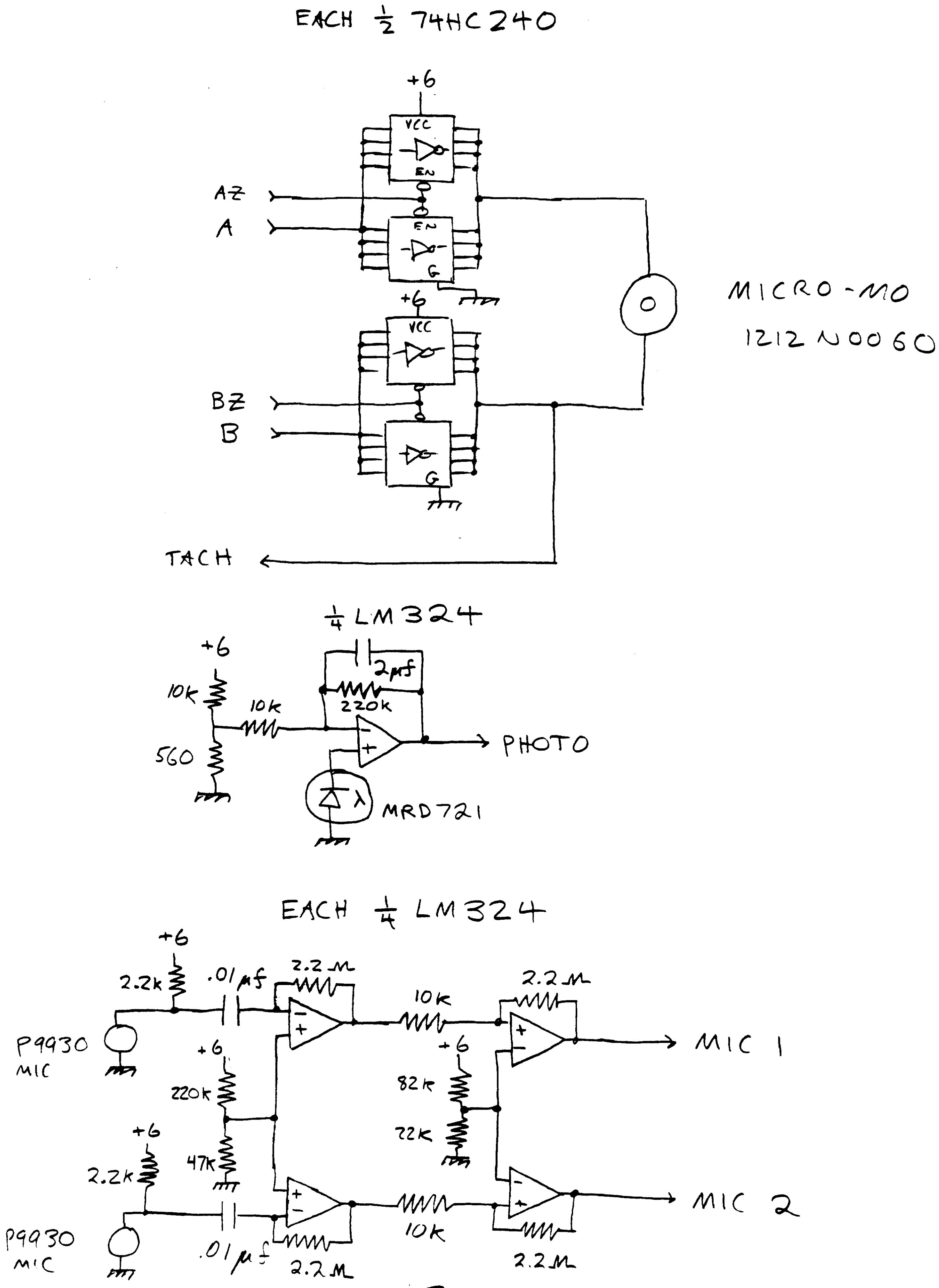
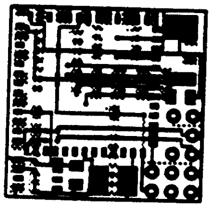

EACH ½ 74HC240
+6
VCC
EN
AZ
A
EN
G
+6
VCC
BZ
B
G
MICRO-MO
1212N0060
TACH
¼ LM324
+6
10k
10k
2μf
220k
560
PHOTO
MRD721
EACH ¼ LM324
+6
2.2k
.01μf
2.2M
10k
2.2M
P9930
MIC
+6
220k
+6
82k
MIC 1
22k
47k
+6
2.2k
P9930
MIC
.01μf
2.2M
10k
2.2M
MIC 2

FIGURE 2.

A tachometric signal for velocity control can be derived by measuring the voltage generated by the motor during the off period of the chopped drive. The switching transient dies out in about 20 microseconds. There is some commutator noise on this signal, but it isn't too bad. To insure a positive tachometer signal, the end of the motor away from the tach signal should be connected to either Ground or VCC through the driver, depending on which way the motor is spinning. The other end of the motor should be allowed to float by tri-stating that section of the driver.

The light level sensing circuit appears in the center of figure two. This circuit has a large dynamic range. The MRD721 photodiode is operated in voltage mode, which is logarithmic in the light level. The bandwidth of voltage mode isn't as good as with current mode however. The bandwidth is further reduced to about one Hz in the non-inverting amplifier circuit. The circuit is designed to span the light levels typically encountered in a room. See the Hewlett-Packard Optoelectronic Designer's Handbook for a discussion of photodiode circuits.

A primitive hearing ("noise recognition") circuit is shown at the bottom of figure two. Each section consists of a high gain microphone amplifer (an inverting amplifier) connected to a schmitt trigger. These circuits provide logic signals which go active at the onset of moderate noises (and every other half cycle, for that matter). This circuit has been tested with an oscilloscope. Given the resolution of the counters of the 6811A2, the time differential in the edges of the onset of the logic signals is adequate for a crude directional hearing sense. The code for this subsystem hasn't been written yet, and will likely be somewhat hairy. Note: the microphone amplifiers are a bit odd; one of the gain setting resistors is actually the impedance of the microphones. Furthermore, it was tested with different microphones than those provided with the kits, so some tweaking is likely to be in order.



Squirt's brain can actually fit on this one square inch printed circuit board if surface mount components are used.

Designing the Photovore: A Case Study

Jonathan H. Connell

1. Task

We report here on the design of the Photovore, a toy robot described in the October 1988 issue of OMNI magazine. The goal of this project was to design a simple autonomous robot that was cheap, fun to play with, and demonstrated the subsumption architecture. A lot of the design decisions were uriven by cost considerations. We avoided microprocessors, used the cheapest base vehicle available, and employed incredibly simple sensors - all in an effort to keep the overall price down. Given these constraints, the rest of this report goes on to describe the basic design of the robot and discusses the engineering trade-offs involved. While the details here are specific to the Photovore, we hope the general approach, and maybe even scraps of the circuitry, will be useful to other robot designers.

To give you some context for what follows, let us describe the final design. The Photovore is about 9 inches long and was built from a Radio Shack dune buggy. This car has two independent drive motors which let us drive it forward and backward as well as left and right. For sensors, we used 3 photocells hooked to comparators. Each sensor simply determines whether the environment is brighter or darker than some preset level. The first layer in the subsumption architecture uses these photocells to orient the creature toward light sources. Switching in the second layer modifies this behavior and, in certain situations, causes the robot to back away from dark areas. All the control logic for this is implemented with two quad NAND gates. By paying attention to details we arrive at a simple but interesting autonomous mobile robot which can be built for under $75. The complete circuit diagram for the Photovore is shown on the next page.

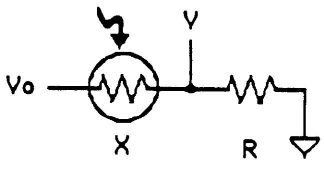
2. Sensors

The key to robotics is sensors. Finding good sensors is hard work, yet robots are not particularly interesting unless they react to their surroundings. For this project we decided to stick to established devices and avoid spending large amounts of time developing new sensor technologies. Given this bias, just about the easiest thing to build is an ambient light sensor using a photocell. Photocells are nice because they have a low impedance which makes it easy to obtain a useful signal without a lot of amplification and buffering. Furthermore, there are large changes of ambient light in the typical indoor environment which means the detector circuitry does not have to be particularly sensitive.

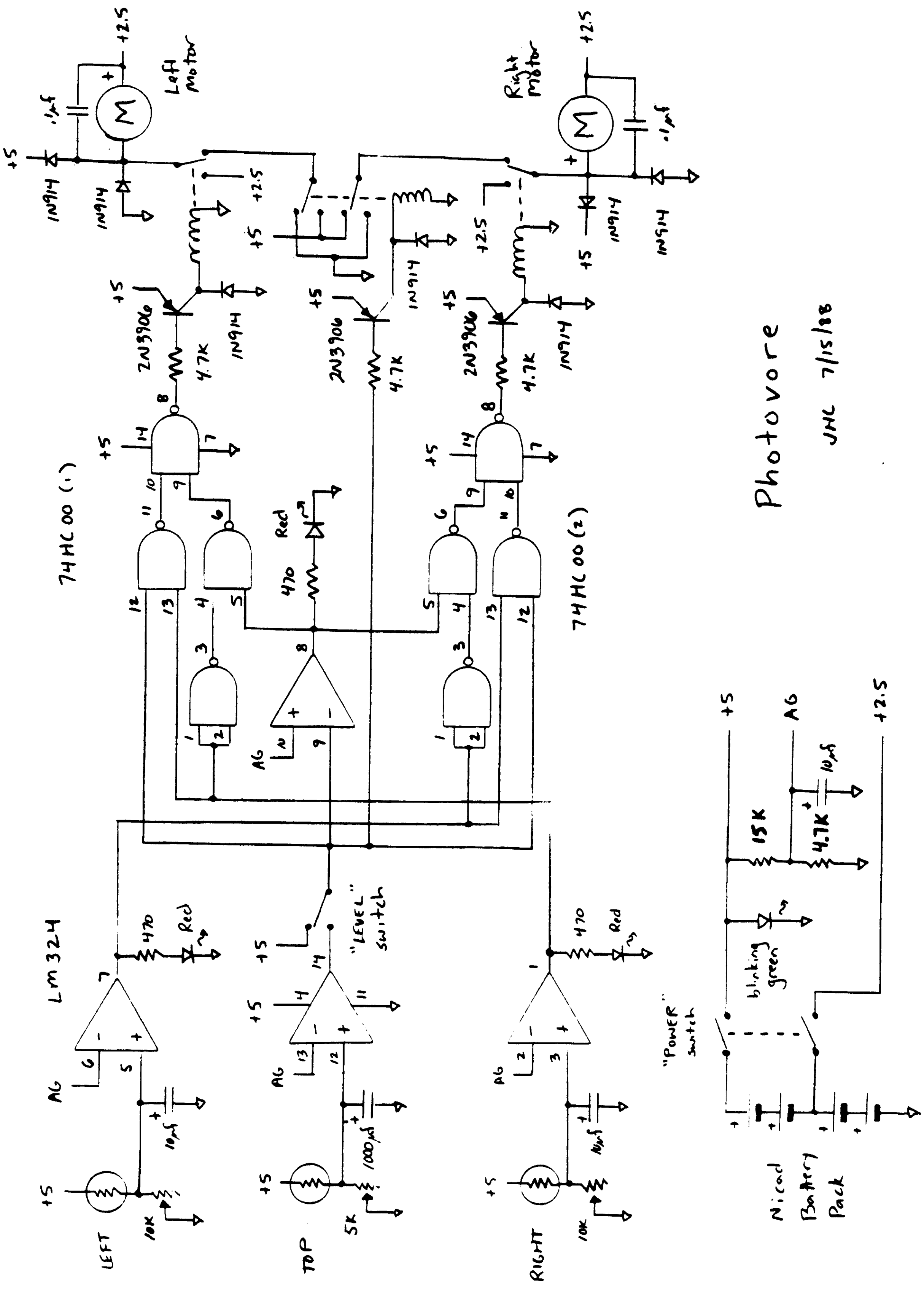
$$V = V_0R/(R+X)$$

$$G = \text{gain} = dV/dX = -V_0R/(R+X)^2$$

$$dG/dR = V_0(R-X)/(R+X)^3$$



To design with a particular sensor, it is important to first understand its capabilities and limitations. The photocells we used were made of cadmium sulfide and were roughly 1/2" in diameter. The resistance of a photocell varies with the amount of light hitting it; the more light, the lower the resistance. For the shielded configuration described later, the resistance in a bright


Photovore
JHC 7/15/88
LEFT
TOP
RIGHT
+5
10K
5K
10K
10μf
1000μf
10μf
AG
LM324
6
5
7
13
12
14
4
11
2
3
1
470
Red
"LEVEL" switch
"POWER" switch
blinking green
Nicad Battery Pack
+5
AG
+2.5
15K
4.7K
10μf
74HC00 (1)
74HC00 (2)
8
9
10
11
12
13
6
5
4
3
2
1
AG
470
Red
2N3906
4.7K
1N914
Left Motor
Right Motor
.1μf
M
+2.5
+5

room is about 5 kilo-ohms. This can go up to 30K in a dim room or down to 200 ohms in direct sunlight. To convert resistance changes into a voltage we put the photocell in one leg of a resistive divider as shown above. To get the maximum voltage swing for a given change of illumination (set dG/dR = 0), the base resistor should be the same value as the operating point of the detector (e.g. about 5K). We can then model the sensor as a variable voltage source having a source impedance of about 2.5K.

Unfortunately, although the illumination in a room may seem fairly constant, there is a large 120 Hz flicker component from the lights. For instance, when the photocell is aimed upward, its resistance oscillates by over 10%. One way to get rid of the AC flicker noise is to use a simple first order passive filter. A 10 microfarad capacitor across the output of the voltage divider gives a break frequency of about 6Hz. This still gives the car a fast response time while attenuating the flicker by a factor of 20. On the center sensor we use a 1000uf capacitor, not so much to suppress noise, but rather to give us a slow response time. We could have used a smaller capacitor by adding an extra, high value resistor between the divider and capacitor on the input of the op-amp. Yet, since a 1000uf capacitor is not exceptionally large physically, we omitted it in order to keep the parts count down. The utility of the long delay will be discussed later.

As mentioned before, we threshold the photocell signal to arrive at a binary signal. One way to do this is to take the raw analog signal and compare it to an adjustable threshold. However, remember that the divider is most sensitive when the two legs are the same. If we had a fixed resistor of 5K and the photocell was around 25K, the gain of the circuit would be reduced by a factor of 9 - the sensitivity decreases as the room gets darker. To avoid this, we use a fixed voltage reference and adjust the analog signal by changing the base resistor (i.e. we tweak the pot for the ambient light level in a particular room). Since the ratio of the base resistor to the photocell resistance is always constant, the gain does not change with the light level.

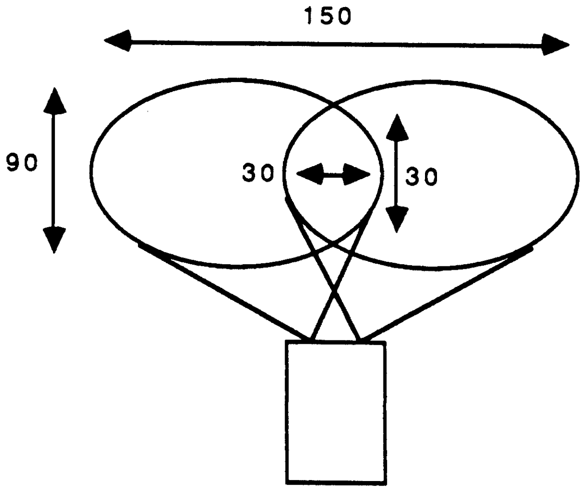
However, instead of using the optimum reference voltage of 1/2 the supply voltage, we actually use 1/4. This is because we want all the parts to be available from Radio Shack and they only carry a limited selection of trimmer potentiometers in the easy-to-grip style. At first glance, 50K seems to be a good value since we could set the pot all the way to 30K. Yet for a bright room we would need to set the pot to a mere 10% of its nominal value. Not only does this make tuning difficult, but potentiometers often have a residual resistance of around 5% of their maximum. To use the next value down, 10K, we have to fiddle with the reference voltage. To sense at 30K we must set the reference to 1/4 the supply. Now we have lost a factor of 4 in sensitivity, but for a bright room we can set the pot to 16% of its maximum.

The thresholding is actually performed by 3 sections of an LM324 quad op-amp. This device was selected because it works well at low voltages and has a reasonably high input impedance. Although the LM324 has a much slower rise time than the LM339 (a low voltage quad comparator) this does not matter for our application. Furthermore, the LM324 has true push-pull outputs (not open-collector) which makes it better for driving logic chips. We could also have given the sensors some valuable hysteresis by wiring the op-amp in a Schmitt trigger configuration. Yet this would have added a total of 6 extra resistors, so we decided against it,

3. Geometry

Once we have picked sensors, we must craft a suitable arrangement of them for our task. We start by examining the receptive field of a single photocell. Since it is a large planar sensor, its sensitivity varies as the cosine of the incident angle. To make it slightly more directional, we wrap the photocell in electrical tape to form a 1/4" high shield around the edges. This should limit the field of view to about 120 degrees full angle, although experimentally we find it to be closer to 90 degrees.

For the first level, we need a sensor geometry which lets the robot home in on a light source. We use two photocells aimed forward but toed outward by 30 degrees. This gives a "lock in" range of about 150 degrees full angle - the vehicle pays no attention to light sources behind it. Since the sensors each have a half angle of 45 degrees we get a 30 degree full angle "sweet spot" directly in front of the vehicle. This is where it will try to keep the light source. However, it is important to note that the reported size of the lock in zone and sweet spot are only valid in the plane that contains the centerlines of both sensors. This is because we are intersecting cones not square-sided pyramids. Actually, at a certain angle (about 15 degrees off the plane) there is no overlap at all and we get a dead spot instead. Since there are no interesting sources at floor level, we mount the two front sensors angled upwards about 15 degrees.



The basic first level algorithm is to turn left if only the left sensor is activated, turn right if only the right is activated, and go straight forward if both see something. Although similar to some of Braitenberg's vehicles, the non-linear thresholding nature of our sensors gives Photovore a qualitatively different type of behavior. Instead of always heading for the brightest spot, the 'Vore happily wanders around in bright areas and only turns when it gets too close to a dark region.

This algorithm works well with table lamps and reasonably well with flashlights (flashlights are not isotropic radiators so where they are aimed matters). More interesting, however, is the fact that it also works with ambient light. This is the other half of working with sensors - understanding how they respond to various environmental situations. Consider a white wall with a black molding at the bottom. Far away from the wall the photocell is looking at a mostly white surface, yet when it gets closer the dark baseboard dominates its field of view. We can tune the comparators to recognize this change, which in turn will cause the vehicle to veer away from the wall. In fact, the vehicle will try to avoid any looming dark object. This includes sofas, shadows, and people's legs (if they have dark pants on).

For the second level, our original idea was to have the robot stop when he got close enough to the light source. To do this we added a photocell mounted so that it looked straight upward. When the vehicle got close enough that the light source was 45 degrees up, our new sensor would be activated and the vehicle would stop. If only the top sensor was activated, the vehicle would back up until the front sensors saw light again. Given the sensor geometry specified above, the receptive fields of the sensors would overlap for elevations between 45 and 60 degrees. This sets the following distance - the lower down the light source, the closer the car will approach it. In ambient light, we also get interesting behavior. The robot runs around the room until it finds a sufficiently bright spot to stop in.

For several reasons, however, we rejected this algorithm. In practice it did not keep its set distance from a flashlight well. This was partly due to the fact that a flashlight is a directional radiator so sometimes the top sensor would not be in the beam. It was also due to the fact that the servo null spot was too small because, once again, we are intersecting cones not pyramids. It would probably have worked better if we had constructed a square, versus round, shield for the top sensor. Yet the real reason we flushed this behavior was it simply was not interesting in ambient light. While it seems theoretically appealing to stop in bright spots, the reality is quite boring as a toy. The car is always stopping somewhere and requires constant kicking to keep it alive and moving.

Instead, for the second level we noticed that there are certain situations in which the light seeking behavior gets the robot in trouble. If it heads directly toward a dark object or scoots under a table, both front sensors drop out at the same time. This causes the robot to freeze in place rather than veer away. To fix this, we added a third photocell to the vehicle. Since we want to sense obstacles both overhead and in front, we aimed this new sensor forward and angled it upwards 45 degrees (versus straight up as before). Now when this sensor goes dark we tell the car to back up. If both front sensors are dark, the robot backs straight away. If one is light and one is dark, it tries to face the light by turning as it backs.

Still, when the robot backs out from under a table it only goes until the top sensor is sufficiently bright, or until both front sensors see light again. This causes the car to jitter near the edges of tables and often come to a complete stop. To remedy this, we merely jacked up the value of the capacitor on the central comparator to give it a longer time constant. This is the only true piece of "state" in the robot and causes the car to continue backing up for a while after it has gotten away from a dark area. Actually, the robot usually backs out far enough to activate both front sensors which cause him to stop until the front sensor ramps up again. Thus, he seems to stop and "deliberate" before choosing a direction to go forward in. It is amazing how much smarter this makes him seem.

4. Chassis

Now that we have picked sensors and figured out how to use them for our task, it is time to choose a body for our creature. Our selection for the Photovore was motivated primarily by cost considerations. The cheapest steerable vehicle we could find was the wire-controlled Red Fox Racer from Radio Shack. Simple toys have two modes: forward and stop. More sophisticated models have three modes: forward, stop, and left turn in reverse. The Red Fox has seven modes; it can turn in either direction while going forward or backward (see table). Unlike most radio-controlled cars, the Red Fox has a tank drive configuration. There are two large wheels in the back, each of which has its own separate drive motor. To steer, we selectively drive only one of the two wheels. This causes the car to pivot around the stationary wheel and drag its front two wheels sideways. Therefore, if we want the car to be able to turn, we must be careful not to have these wheels carry too much weight.

Left Motor	Right Motor	Corresponding Motion
+	+	straight forward
+	0	forward to right
0	+	forward to left
0	0	stop
-	-	straight backward
-	0	backward to left
0	-	backward to right
+	-	swivel right (unused)
-	+	swivel left (unused)

The Red Fox has several other good features. For instance, it has a built-in rollcage to help protect any fragile electronics from violent crashes. Also, it is very fast (6 ft/sec, which is a scale speed of 70 mph!). This is important because we have found from giving many demos that the faster a robot moves, the smarter it seems (whether or not it is doing anything particularly intelligent). Finally, the Red Fox's motors normally run on 3 volts. This is an advantage because it means we can easily drive the motors from the 5 volt supply required by the logic chips and op-amps. To change the direction of the motors we must reverse the polarity of the applied voltage. A simple way to do this is to just tie one end of each motor to 2.5 volts, and then connect the other end to 5 volts or ground depending on the direction desired.

Another important part of robot design is choosing a power source. The original Red Fox ran off of two 1.5 volt C cells. Since we want at least 5 volts to run the car, we need four standard batteries. We could use two 3 volt lithium cells instead, except that they are much more expensive and are not rechargeable. Since four C cells would be awkward, we chose to use 4 AA nickel cadmium cells. Nicads are nice because they can provide high peak currents and have a fairly flat discharge profile (they are 1.25V when full and still 1.2V when mostly discharged). Four AA cells gives us 5 volts at 450 milliamp hours, or a total of 2.25 watt hours. Since the robot consumes 1.5 watts on average (motors take 300ma at 2.5V) this lets us run for an hour and a half. Many people have suggested using solar cells to power the robot but, unfortunately, you would need an unreasonably large array to do this. Solar cells deliver about 100mw per square inch in bright sunlight and about 500 times less in normal indoor lighting. Thus, to power our vehicle we would need a 7 foot square array! If we took a reasonable sized array, say 4" by 4", and let it charge the batteries for 8 hours, there would only be enough energy to run the car for 1 minute.

5. Motors

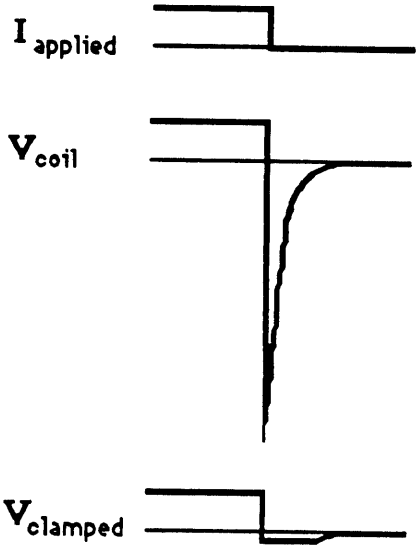
The next problem is to figure out how to drive the motors using logic level signals. A typical TTL gate can source 0.4ma and sink 4ma, while buffer chips can go up to 15ma. Still, this is a long way from the 300ma required by the motors. Obviously some amplification is needed. Another point to keep in mind is that inductive loads, like motors, typically have much higher starting currents than running currents. For instance, the Red Fox's motors have a stall current of about 1 amp, roughly a factor of 3 over its average. Whatever power device we use has to be able to withstand these peak currents. One thought would be to use power MOSFETs since they have very low "on" resistance (less than 1 ohm) and hence don't dissipate much power (about 50mw in our case). Regrettably they require about 8 volts to turn on, whereas we only have a 5 volt supply. There are some "logic level" MOSFETs but they only come in N-channel devices and we need P-channel MOSFETs as well for our half-bridge driver. While we could build charge pumps and level translators, it doesn't seem worth the work.

Another thought would be to use bipolar transistors. For instance, darlington pairs are available with current gains (betas) of over a thousand. Unfortunately, the collector-emitter saturation voltages for these units are about 1.5 volts. That is, when they are as turned on as they are going to get, they still exhibit a significant voltage drop. This ruins our plan for using a split 5 volt supply; we would have to go to a 6 cell, 7.5 volt supply to make up for the difference. More importantly, however, our motors draw 300ma at 2.5V and so consume 750mw each while the transistor driving them would consume an additional 450mw. Half of the robot's power would be going into heating up these transistors! Even using normal transistors, the ones that can handle high collector currents typically have substantial voltage drops (and the ones that don't aren't easily available).

Our final solution was to use relays. Since a relay is just a switch there is no voltage drop across its contacts and no power dissipated. To activate it, however, we must energize its coil. Typical small 5 volt relays have coil resistances from 50 to 100 ohms. Since they require at least 3.5 volts to activate, this translates into about 70ma or 250mw. Thus, they do not dissipate as much power as transistors and have an effective current gain of between 4 and 14 (stalled) for our application. Still, we must buffer the logic outputs to activate the relays. Since the current is less and we can tolerate some voltage drop, bipolar transistors seem appropriate. For currents under 100ma it is possible to find transistors with saturation voltages around 0.4 volts. These transistors typically have betas of at least 100 whereas we only really need a gain of about 20.

The second page shows the relay driver circuit we use. Since plain TTL chips can sink more current than they can source (unlike CMOS chips) we decided to use PNP transistors. To calculate the value of the base resistor we assume a beta of 100, an emitter to base voltage of 0.6V, and a logic signal of 0.4V (the maximum for a logic zero). To generate a base current of about 1ma across a voltage drop of 4V we use a 4.7K resistor. We could use a smaller resistor here without exceeding the drive capabilities of the logic chip but there is a problem with turn-off. Lightly loaded logic outputs are usually at least 4.5V when high. This back biases the transistor's base so no current flows and the transistor turns off. The problem is that one of the relays is driven by an op-amp output. The maximum high level signal for this chip is 1.5V below the power supply rail. If we lowered the base resistor to 2.2K and our transistor actually had a beta of 300, we could get a collector current of 120ma which would be enough to turn the relay on! Instead of fiddling with this resistor or hand-picking transistors, a better solution would have been to use an NPN driver for this relay since the op-amp can drive all the way to ground. We only made all the drivers the same to avoid confusing which transistor went where.

There is one more part to the relay drivers: the flyback diodes. When you drive an inductive load you generate a magnetic field. When you turn off the current to the device, this field collapses and generates a voltage across the coil. This voltage is the opposite polarity from the previously applied voltage and can soar to several hundred volts. Most transistors have a collector-emitter breakdown voltage of less than 100V and these spikes burn them out. To prevent this, we put a diode across the relay's coil. Note that when the relay is being driven, this diode is back biased and hence does not conduct. When the transistors turns off, however, this diode clamps the reverse voltage spike to ground thus saving the transistor. We use the same setup on the motors themselves and clamp the coil voltage to one of the rails using two diodes. Like transistors, relay contacts can be damaged by high voltages. When they repeatedly spark over the contacts eventually get pitted and carbonized which keeps them from switching well.

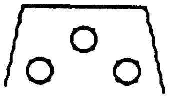


The actual relay configuration used is rather interesting. Instead of using two half-bridge drivers, we put in a reversing relay and two individual gating relays, one for each motor. This saves us one relay but means we can never simultaneously drive one wheel forward and the other in reverse. It turns out that this doesn't matter for our application. Another interesting feature is that when we are not powering the motors, they are shorted to 2.5V which causes the creature to stop faster. This works because, if the wheels keep turning, the motor acts as a generator and tries to pump current through a very low resistance load. Finally, notice that each motor is always driven off different halves of the power supply. When the right motor is across +5V and +2.5V, the left motor is across +2.5V and ground. This lowers the peak current required of any particular cell and also evens out the lifetimes of the two halves of the battery pack since they are usually equally loaded.

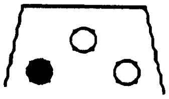
Another interesting feature of this design is that activating one of the two gating relays causes its associated motor to turn off. This lowers the peak power demand since the relay and the motor are not both dissipating power at the same time. We use a similar trick on the reversing relay. Since the vehicle is usually going forward, we make this the de-energized setting. The inversion in the gating relays also prevents oscillation when the power is running low. Imagine that the relays are wired in the opposite fashion and that the 'Vore has just enough power to activate one of the relays. This then connects the motor to the power supply which may be too much load for it. If the power supply voltage sags, the relay will drop out and then the cycle repeats. Our unusual relay wiring also has the added advantage that it actively disconnects the motor by pulling on the contacts instead of just letting them be opened by a spring. This can be important because relay contacts often begin to weld together when carrying high currents (especially true for reed relays).

6. Logic

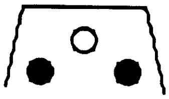
Now that we know have built the sensor and actuator interfaces it is time to connect them. For the first level following behavior we want the robot to respond as shown below. The circles represent the LEDs on the vehicle and correspond to the two forward looking sensors and the top sensor. If one of the side circles is colored in, it means that side is bright. The middle circle has the opposite semantics; if this circle is colored in, then the environment is dark overhead.



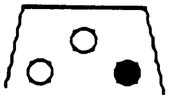
Stop



Forward to the Left



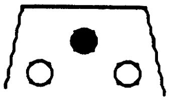
Straight Forward



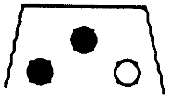
Forward to the Right

Combining this with the motor direction table shown previously, we see that we want the right motor to run forward when the left sensor is activated, and vice versa. We can use a direct connection to the relay drivers because there are two inversions in the path. The relays are activated when their control line is brought low, yet when they are activated they turn their associated motor off.

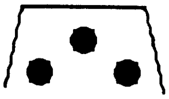
When the second level behavior is switched in, the creature should follow the additional situation-action patterns listed below. Notice that whenever the top sensor is dark (circle colored in) the car is always backing up in some manner. Once again, because of a double inversion, we can tie the output of the central comparator directly to the driver for the reversing relay. Choosing the turning direction is now a little more complicated. If the right sensor is not activated we want the left wheel to turn and vice versa.



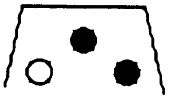
Back up



Backward to the Left



Stop



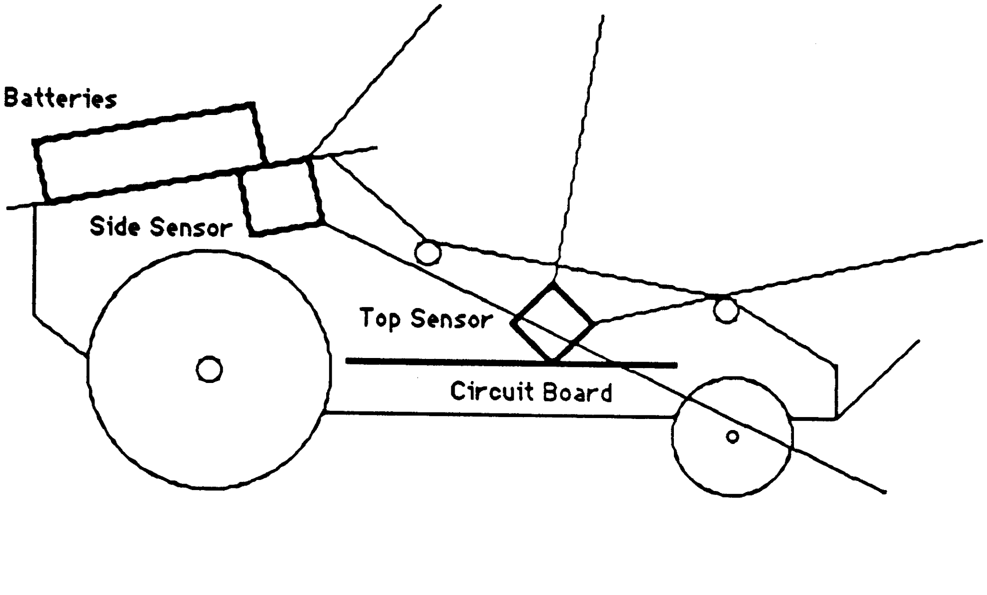
Backward to the Right

Notice that the logic for backing up is identical to that for going forward except that the side sensor signals are interpreted in the opposite fashion. A simple way to achieve this is to use an exclusive-or gate to combine each side sensor signal with the back-up signal (the inverse of the top sensor comparator output). Thus, we could use 3 sections (one wired as an inverter) of a quad XOR such as the 7486 to control the whole robot. If we built an NPN driver for the reversing relay and swapped the inputs to the top sensor's comparator we could get by with exactly 2 XORs. Unfortunately, Radio Shack doesn't stock this chip so we decided to synthesize this functionality from a collection of NAND gates, instead. To build the two XOR gates needed we require 6 NANDs and 3 inverters. Two of the inverters are made from NANDs while the third is built out from a spare section of the op-amp chip. Luckily, the reference voltage we use for the sensors is at an appropriate level to make logic comparisons as well.

The Photovore is a unusual in that it does not have a lot of power line conditioning to keep the logic from glitching. There are capacitors across the motor terminals to suppress brush noise, and the analog reference for the comparators is stiffened with a capacitor. However, there are no large power supply capacitors or individual decoupling capacitors on each chip to reduce switching spikes. In fact, the supply isn't even regulated due to the fact that nicads have fairly constant voltages and a high surge capacity. The reason we can get away with this is that the only state in our system is the voltage on the 1000uf capacitor, and this is not affected by power supply variations. It doesn't matter if the central logic goes temporarily awry. The relays and the car itself have such long response times the network will return to normal long before the creature ever notices the error.

7. Construction

The final step in creature design is to integrate all the subsystems into the body. The biggest single component is the battery holder, so let us place this first. A seemingly good location would be on the floor of the car, inside the rollcage. Yet this makes them hard to replace and, more seriously, puts so much weight on the front wheels that the car is no longer able to steer. Therefore, we instead attached a flat board to the back of the vehicle and mounted the batteries on this (see below). Unfortunately, this exacerbates the car's tendency to pop wheelies. To counteract this, we installed several lead weights at the front of the vehicle to balance the load.



The next most critical thing to mount are the sensors. We had hoped to keep them all inside the rollcage to prevent them from getting banged up, but the geometry precludes this. The upward looking sensor at least can be mounted inside the vehicle, provided that we are careful to give it an unobstructed view around two nearby structural members. Since the side sensors look forward and slightly outward, we put them on the edges of the car underneath the battery holder to give them a totally clear field of view. The fiberglass board that the batteries and controls are mounted on affords these sensors some protection from impacts.

In general, protecting the robot from smashing itself is a big problem. Fortunately, we were able to bury most of the electronics inside the car within the protective envelope of the rollcage. Still, some parts, such as the controls, had to be left exposed for accessibility. Although we attached the back board to the frame of the vehicle with sturdy nylon tie-wraps this still doesn't shelter the pots and switches. For instance, as soon as the car tries to zoom under a low bridge, the board stays put but the switches peel right off. We eventually bent a coat hanger into an extra rollbar to protect these pieces. The circuitry is not the only part that suffers, sometimes the frame itself sustains damage. For instance, after a month of banging into concrete walls the front bumper fractured and had to be replaced with a large piece of rubber tubing. To put this in perspective, however, if a human was driving at 70 mph with 30 foot visibility, his car wouldn't look so good either!

As a final note, don't be daunted by all the details presented here. We did not understand the Photovore anywhere near this clearly when we first built him. Most of the circuitry was designed using rules of thumb and then tested to see if it actually worked. Likewise, many of the component values were arrived at by trial and error substitution and the sensor geometry came from experimentation with different configurations. The final control algorithms were derived by playing with the vehicle and seeing what was fun. Only when something went wrong did we go back and analyze it to help us find a solution. In general, we believe empirical studies to be of great value in mobile robotics. You can spend years developing a theoretically perfect set of control algorithms but chances are they will not work in the real world. Our approach has been along engineering lines: we find out what works then go back and try to extract useful generalizations from it.