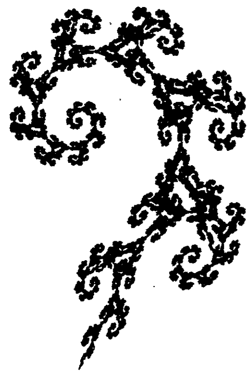
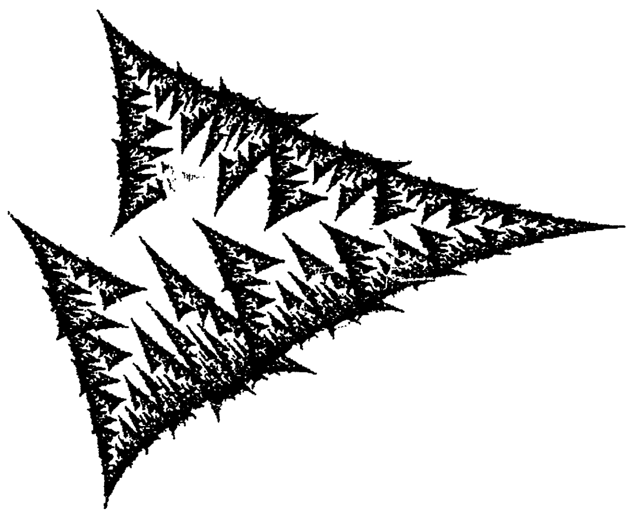


# Penn and Teller Seance Electronics



*J. Paradiso*
*MIT Media Laboratory*
*December, 1994*

# Overviews

# Notes on the Penn and Teller Hardware


*-- J. Paradiso*
*MIT Media Lab*
*21-Dec.-94*


This document is a collection of diagrams, data, wiring protocols, and notes about the hardware developed to support the Penn and Teller Seance Trick. Three components were developed; the Grouper (which contains the actual sensor electronics, microcomputer, MIDI support, etc.), the Light Driver (which contains power drivers for the 8 lighting channels and power supplies for the Grouper), and the Chair Electronics (which include an analog sensor driver, display driver, and switch transmitter). The Grouper is certainly the most complicated, as it includes the following circuit cards: a Fish (considerably modified from the stock version, but the schematic here incorporates all changes), a Chopped Fish, a Fish Peripheral, and a Penn Bit card. The Light Driver is much simpler (sporting only a pair of 4-channel light driver cards), but is much heavier, as it also includes a triple power supply for the Grouper and Chair (±12,+5 Volts), plus a big transformer and capacitor for the Light Driver (Xformer is a 12 Volt AC output, sporting a 16 Amp capacity).

The Grouper and Light Driver connect to the chair through 3 multi-connector cables that interface to the electronics rack via a breakout panel. This panel produces 8 RCA jacks (which go to the grouper hand sensor inputs H1-H4, FL, FR, S1, S2), 3 round CB connectors (which go to one of the display outputs, the keypad input [for switches], and the ±12 Volt supply for the analog drivers), one BNC jack (which goes to the Xmit output), and 2 Jones connectors (which go to the hand and foot lights on the light driver unit). All these are labeled, thus connections are straightforward.

The Grouper also can connect to a second display, currently used by Penn on stage when he plays his bass.

The Grouper has a RCA connector labeled "cal". This is an independently buffered transmitter output signal, with level set by the amplitude pot on the chopped fish. This is handy to use for checking the transmitter frequency and waveform.

The Grouper connects to the light driver through 2 cables. One multi-pin CB connector passes the DC power voltages. The other connector is a DB-9, which is used for the light outputs. Two possible DB-9 connections are available at the rear of the Grouper. The one labeled "Digital" should always be used, as it passes signals from the microprocessor. The other, labeled "Analog", is only for tests (it passes signals directly from the Fish outputs).

The Grouper also has the standard MIDI in and out jacks, which connect to the Studio 3 or other MIDI interface.

The front of the chair electronics unit has the 3 connectors that mate to the main cable that leads to the rack. It also has a connector labeled "switches". This parallels the similar connector at the rear, and can be used to connect additional switches into the setup (the circuitry can accommodate up to 7 switches; right now, we're only using 2). A spare transmit output is also provided at the front chair panel; this is to accommodate additional transmit plates that can be added to the rig. The rear of the chair electronics has many connectors, which mate to the various chair cables (this is so the chair setup can be broken down fairly easily). Everything is labeled. One of these connectors is a 4-pin CB jack labeled "Spares". This is currently unused, and provides an input to two additional channels of sensor electronics (S1, S2 at the Grouper). These don't have log amps to extend their range, hence function like the foot sensors.

When everything is cabled, and the Grouper is reset, all lights will flash on in sequence. If a hand sensor light is blown, it can be replaced with another (these are the 12 Volt, 20 Watt halogen bulbs available at Radio Shack, part # 272-1177).

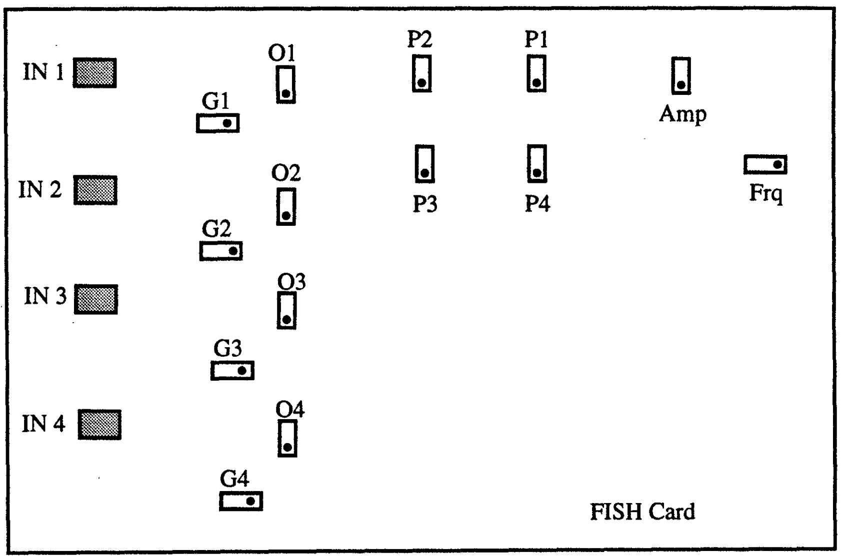
Unfortunately, due to the limited space inside the sensor canisters, these bulbs have to be soldered in. Be careful, and try not to touch the bulb itself during this process, to preserve its longevity.

The only adjustments that should be necessary are screwdriver tune pots mounted on the front of the Grouper. These are 20-turn pots, thus can take significant adjustment to produce an effect in some cases. There are 4 mounted near each LED in the picture of the chair; they are associated with the corresponding sensors. There is also 1 mounted off to the side; this is a master gain that raises and lowers all sensitivities (including the feet!). The way to use these adjustments is to sit an individual in the chair (making sure that the Penn bit is set appropriately, depending on how big the individual is; use the software calibration panel to set the user properly). Have the person put his hand in the middle of the sensor field. All lights should glow more or less evenly at medium brilliance; when he puts both hands in his lap, all lights should be off. If the lights are all running too bright or to dim, the master gain can be tweaked. If the lights are significantly unbalanced, then adjust the corresponding offenders individually. All adjustments increase sensitivity with clockwise rotation. Be careful with these potentiometers, since they are only held to the panel with epoxy.

Remember that the master gain also affects the foot sensitivity. If the feet seem too strong or weak for any reason, re-adjust the master gain so that they're OK, then adjust the individual hand sensor gains to bring the hands back into line. Normally, the feet aren't a problem, since the software can calibrate them.

After making any sensor adjustments, make sure that you run a software calibration.

The next level of sensor adjustment involves taking the top off of the Grouper, and accessing the trimpots inside. As this can be sensitive, it should be undertaken in consultation with MIT. For posterity, however, here is what everything does.




The figure above shows a layout of all trimpots on a Fish. The chopped fish is the same, except for the omission of the Frq. pot, as there's no oscillator on it. On the main Fish card, the channels 1-4 correspond to hand sensors 1-4. On the chopped fish, I believe that channels 1-2 are left and right foot, 3-4 are the spares.

The pots labeled "O" are offset pots. They adjust the linearity and span of the sensor signals. For the hand sensors, these pots are the same as the four on the front panel (if the front panel pots are pegged for any reason, these pots can be adjusted to compensate). The "G" pots are gains. They essentially control how quickly a sensor signal will move from off to full on; i.e. the physical range of the measurement. The gain and offset adjustments are somewhat coupled (especially in the case of the hand sensors, because of the log amplifiers), and sometimes have to be adjusted together to get the proper effect. I find, especially for the hands, that the offset adjustments alone (i.e. those on the front panel) are sufficient to correct essentially all normal drifts. The gains usually need adjustment only after a hardware modification.

The Phase adjustments (P1-P4) and frequency (Frq) should never be touched without an expert present. Here is how they are best calibrated, however. Connect two channels of an oscilloscope to test pins A and O on the appropriate channel (located under the AD633 multiplier). "A" provides signals from the front-end amplifier, and "O" provides signals from the reference oscillator. Trigger on "O", and have somebody sit in the chair to get a signal on "A". Adjust the relevant phase pot to bring the two sine waves that you see into perfect phase agreement. If this is not possible, adjust the frequency "Frq" slightly to make it so (this will entail re-checking the phase on all other channels, however).

The Frequency adjustment should also never be touched. I last set it to give 70.36 kHz, and things worked well. When it is actually adjusted (i.e. for new front-end cables), it is set so the hand signals give close to their maximum (since all cables are of different lengths in the current setup, this is a bit of a compromise), and all channels (inc. the feet!) can be brought into phase with the "P" pots.

The Amplitude adjustment is in series with the corresponding pot on the front panel. It should be set close to full on, so the front panel pot does all the work. If you've got a scope handy, none of the hand sensor signals should saturate when the hand moves close to a sensor; if so, back off on this adjustment (best from the front panel).

The Penn Bit has two adjustments. One (the pot on top) controls the sensitivity (actually offset) of the top two hand sensors. The other (pot on bottom) controls the sensitivity of the bottom two hand sensors. These adjustments are only relevant when the Penn Bit is asserted (done either by flipping the switch on the card to "on", or by setting the Penn status through software; if you flip the switch, remember to put it back into the middle "auto" setting when finished; otherwise it stays stuck). To adjust these pots, first tweak the sensors for a normal person (with Penn bit off) as described above, then put Penn (or a big surrogate) in the chair, have him keep hands in his lap, and adjust these pots to turn the sensor lights off. When he puts his hands in the field, the sensor response should look close to normal.

The Log Amp also has a set of potentiometers. Adjusting these can be confusing, however, so they are definitely best left untouched. In the interests of completeness, however, this is what they do. All trimpots are labelled in the card (see the PC layout diagram included in this report). The two big pots control master offsets for all inputs and outputs. The input offset adjusts global linearity, and the output offset controls where the output zero level is (the circuit clips when the output tries to drop below 0.6 Volts; the output offset raises and lowers the signal across this threshold). The 4 pots around the output amplifier control the gain of the corresponding output stage. The theory of adjusting the log amp is simple; adjust the input offset so the signal appears more or less linear across the desired range of hand motion, adjust the output offset so the signal goes down to zero or vicinity when the hands are out of the field, and adjust the individual output gains so the maximum output voltage is barely above 5 Volts. Sounds simple, but together with the possibilities in Fish adjustments, it can get confusing.

There is only one potentiometer on the Fish Peripheral. This is a master gain that controls the sensitivity of the chair lights. It should always be set full up. There is also a LED on this card that is illuminated when a successful connection is made to the switch transmitter in the chair electronics. When a switch is pressed, this LED should be seen to twinkle.

There is one potentiometer on each of the Light Driver cards. This controls a DC offset that is added into the light signal. It must be adjusted with the Grouper connected to the light driver, and no sensor signals present. It is tweaked so that there is a tiny quiescent current draw on the meter (i.e. it barely budges) for the hands (keep it full off for the feet). This is a fix that was introduced to prevent glitches that were showing up in the hand sensor signals when the lights turned on and off (i.e. when the darlington transistor switched on, it could produce a noticeable transient in the sensor response). The transistors are thus always kept on slightly with this adjustment.

There are no adjustments in the Chair electronics. There are a few LED's on the panel, however, that show system status; i.e. power, display status (these LED's should ping-pong back and forth if the display is properly connected), and Tempo monitor (thusfar used only in the Spirit Trio).

The custom Fish software written in 68HC11 assembler language by Josh Smith is also appended to this document, together with a list of Fish peripheral bus protocols and MIDI commands.

In the software written for the Penn and Teller fish, the dipswitches on the Fish card are not used, thus their settings should have no effect.

# Brief Description of Penn and Teller Sensor Chair

*-J. Paradiso,*
*MIT Media Lab*
*7-Nov.-94*

As labeled on the chair layout diagram, the copper plate (A) affixed to the top of the chair cushion is a transmitting antenna being driven at roughly 70 kHz. When a person is seated in the chair, they effectively become an extension of this antenna; their body acts as a conductor which is capacitively coupled into the transmitter plate. Four receiving antennas (B) are mounted at the verticies of a square, on poles placed in front of the chair. These pickups receive the transmitted signal with a strength that is determined by the capacitance between the performer's body and the sensor antenna. As the seated performer moves his hand forward, the intensities of these signals are thus a function of the distances between the hand and corresponding pickups. The pickup signal strengths are digitized and sent to a Macintosh computer, which estimates the hand position. A pair of pickup antennas are also mounted on the floor of the chair platform, and are used to similarly measure the proximity of left and right feet, providing a set of pedal controllers.

In order for a performer to use these sensors, he must be seated in the chair, and thus coupled to the transmitting antenna. Other performers may also inject signal into the pickup antennas if they are touching the skin of the seated individual, thus becoming part of the extended antenna system (hence the sensor instrument may be "played" by the audience member while he is tying Teller up).

Because Penn is so much larger than Teller, the Macintosh must employ a different set of sensor gains and calibrations when either is seated; otherwise the difference in body mass considerably affects the reconstructed hand position.

The sensor antennas are synchronously demodulated by the transmitted signal; this produces a receiver tuned precisely to the waveform broadcast through the performer's body and rejects background from other sources.

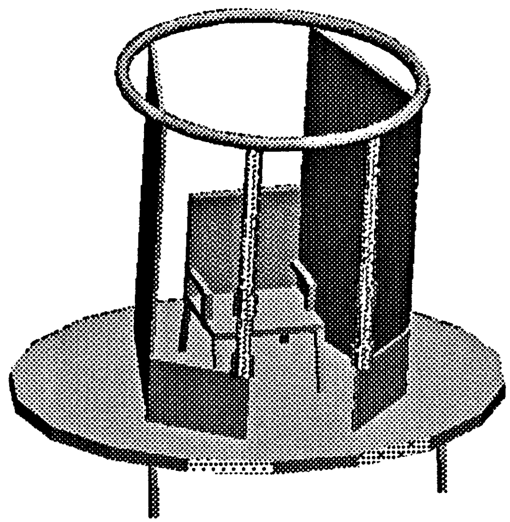
A pair of footswitches (D) are incorporated in this system to provide sensor-independent triggers. These are used for changing parameters when the foot pedals are dedicated to generating musical sounds (i.e. for getting out of the drum patch, where the two foot sensors are emulating kick drums), or for instigating triggers when the performer is not seated, hence is unable to use the sensors.

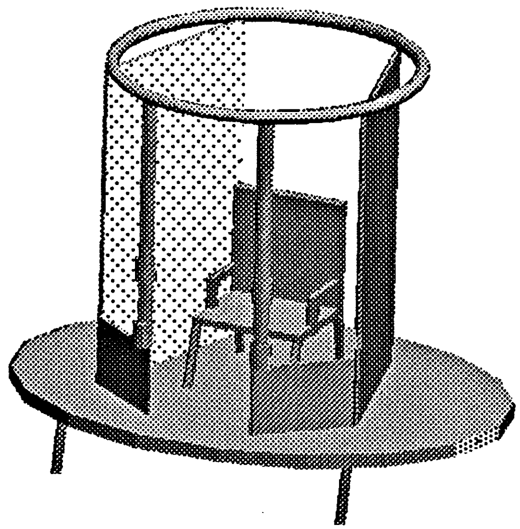
The hand sensor antennas (B) are composed of a copper mesh encased inside a translucent plastic bottle. A halogen bulb is mounted inside this mesh which is illuminated with a voltage proportional to the detected sensor signal (thus is a function of the proximity of the performer's hand to the sensor), or driven directly by the Macintosh computer as a MIDI light-instrument. Four lights are mounted below the platform (F); these are correspondingly driven by the foot-sensor signals or directly through MIDI.

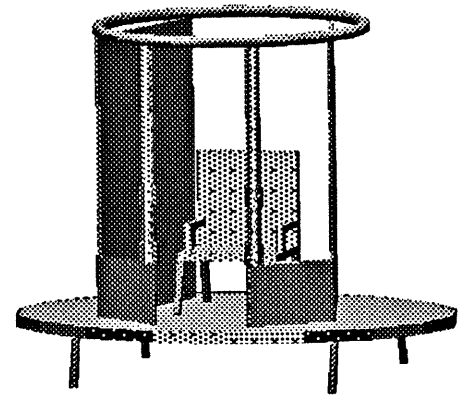
A digital display (E) is also mounted on one of the sensor posts; this is similarly defined as a MIDI device, and is driven by the Macintosh to provide performance cues (i.e. amount of time or triggers remaining in a particular musical mode, etc.).

The sensors are used to trigger and shape sonic events in several different ways, depending on the portion of the composition that is being performed. The simplest modes use the proximity of the performer's hand (or head in the case of Teller's closing bit) to the plane of the hand sensors (z) to trigger a sound and adjust its volume, while using the position of the hand in the sensor plane (x,y) to change the timbral characteristics. Other modes divide the x,y plane into many zones, which contain sounds triggered when the hand moves into their boundary (i.e. the percussion mode). Several modes produce audio events that are also sensitive to the velocity of the hands and feet.

# Instrumented P&T Booth

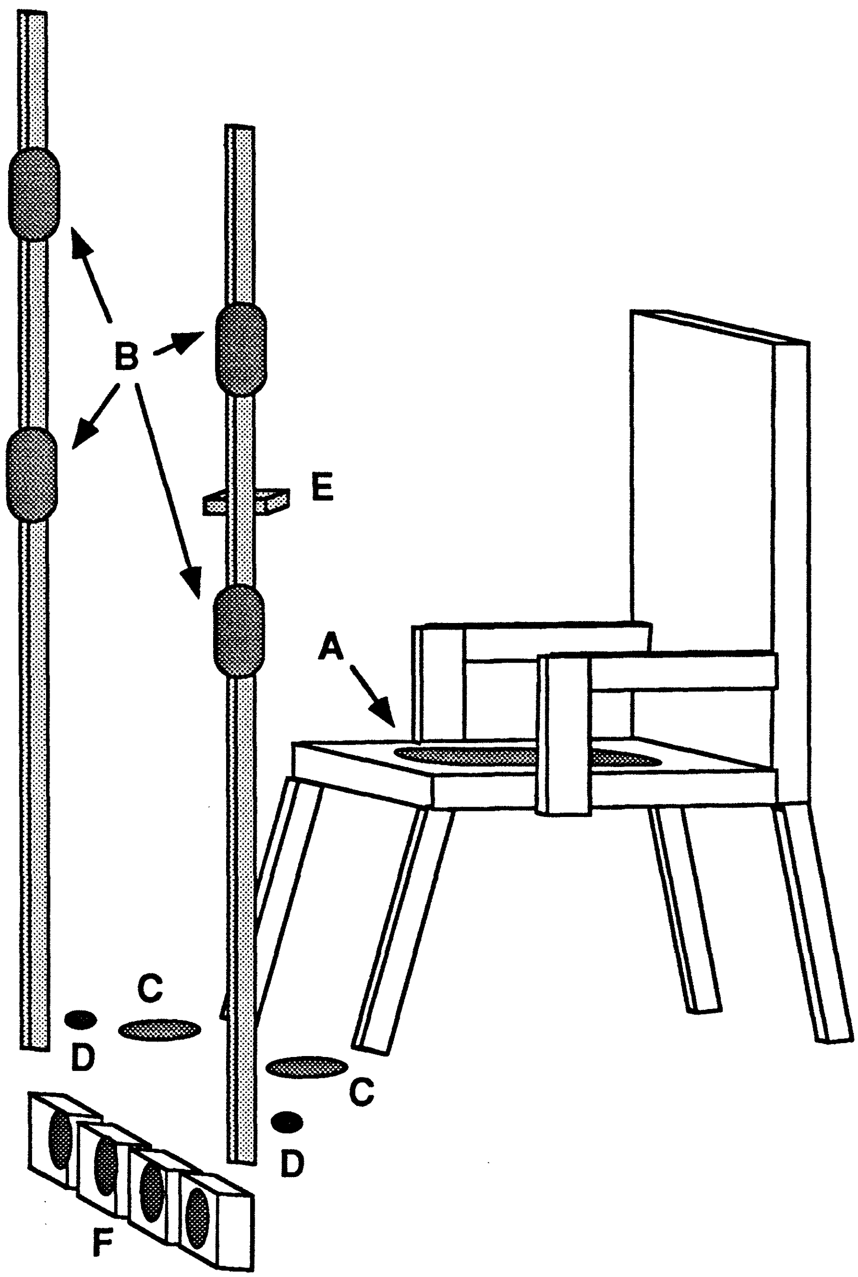






- The baffles surrounding the chair are transparent plexi, as planned.
- The thicker lengths of pipe represent hand sensor locations; 2 on each column.
- The sensor and column locations can be changed if needed.
- The sensor drive will be coupled into the performer via a plate on the seat.
- Another plate may be added on the floor near the chair to allow a standing performer to use the sensors (otherwise this performer must be touching the seated individual).

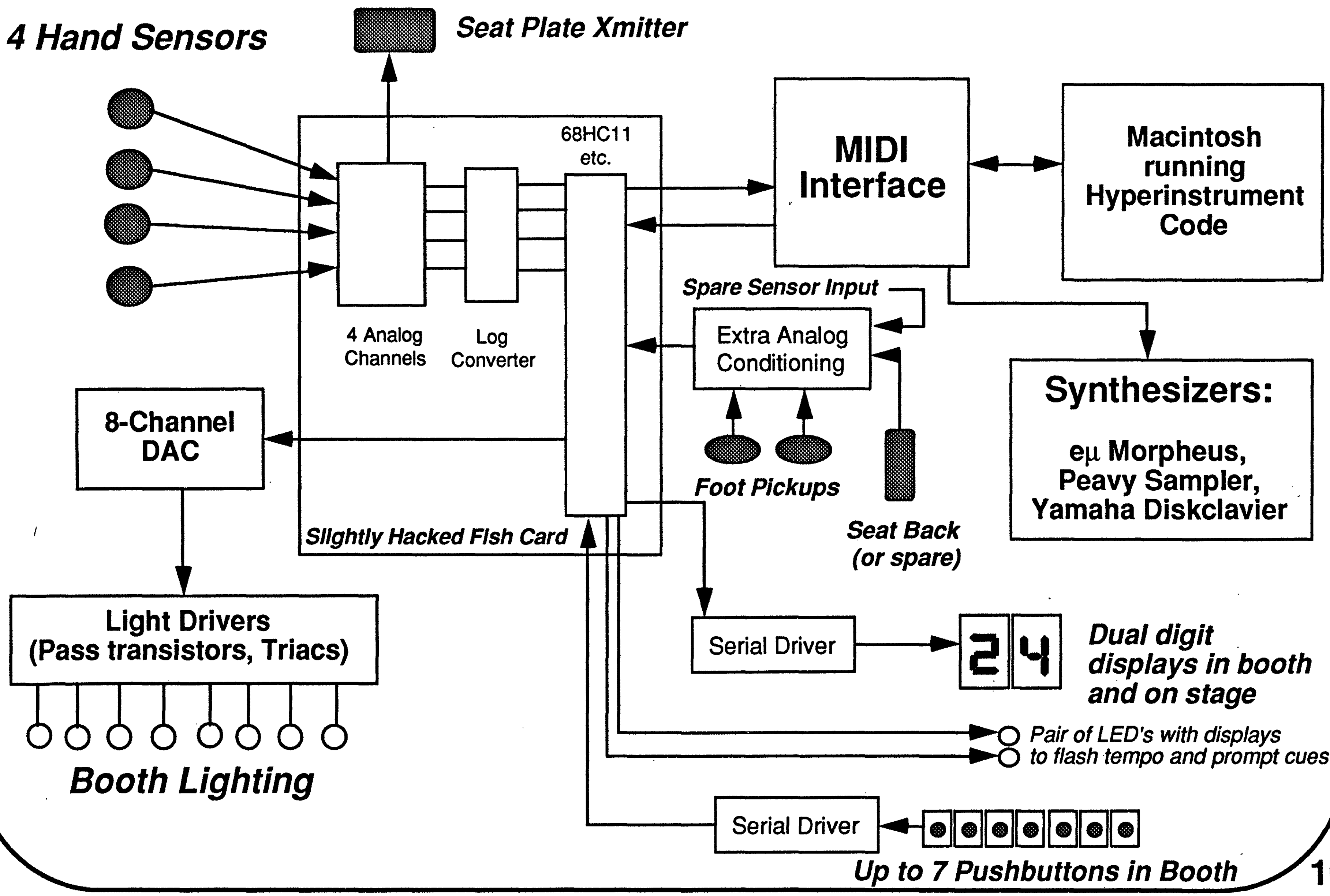
# Layout of the Penn and Teller Sensor Chair



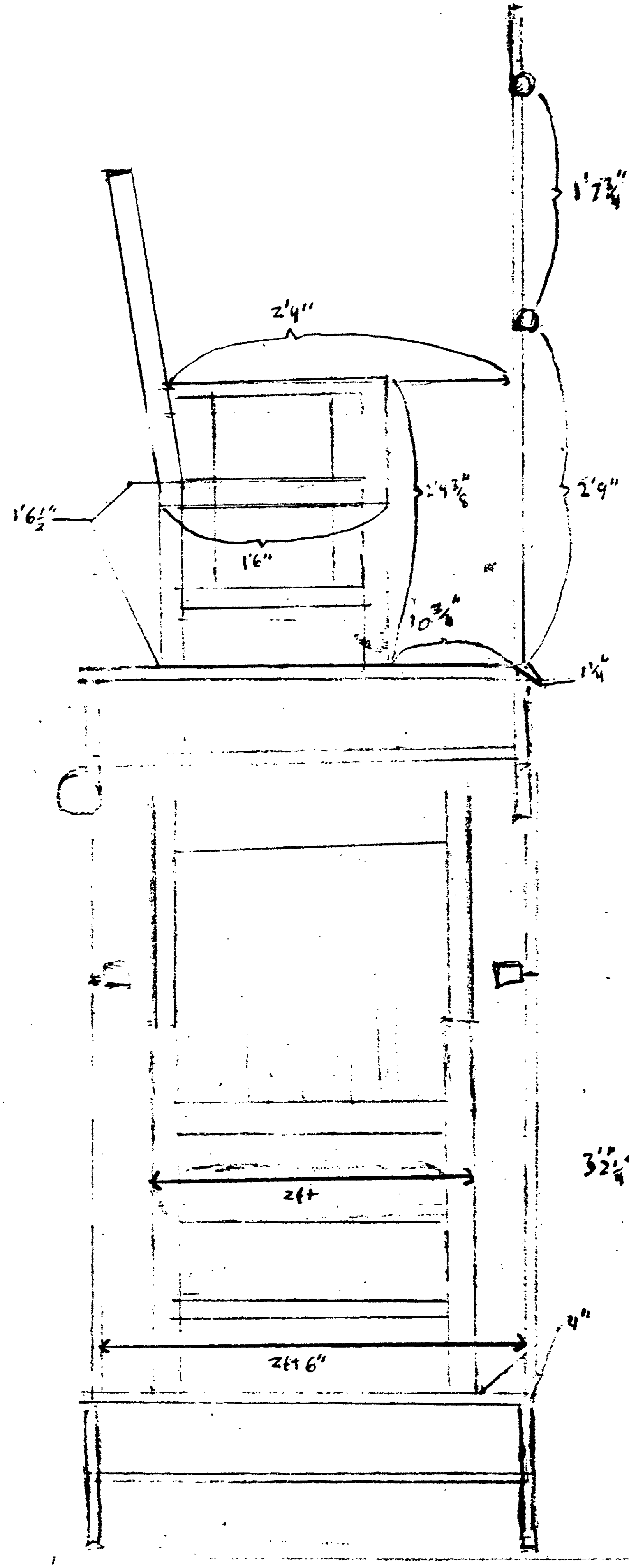

***Legend:***

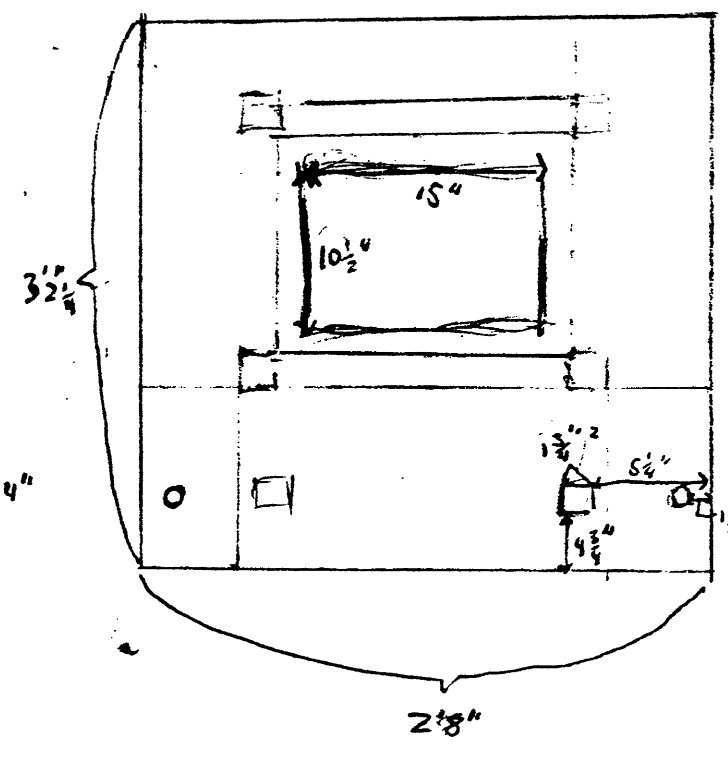
A: Copper plate on chair top to transmit 25 kHz carrier signal
B: Four illuminated antennas to sense hand positions
C: Two antennas to detect left and right feet
D: Two pushbuttons for generating sensor-independent triggers
E: Digital display for computer to cue performer
F: Four lights under chair platform, nominally controlled by foot sensors

# The P&T System



P+T
Chair
Measurements

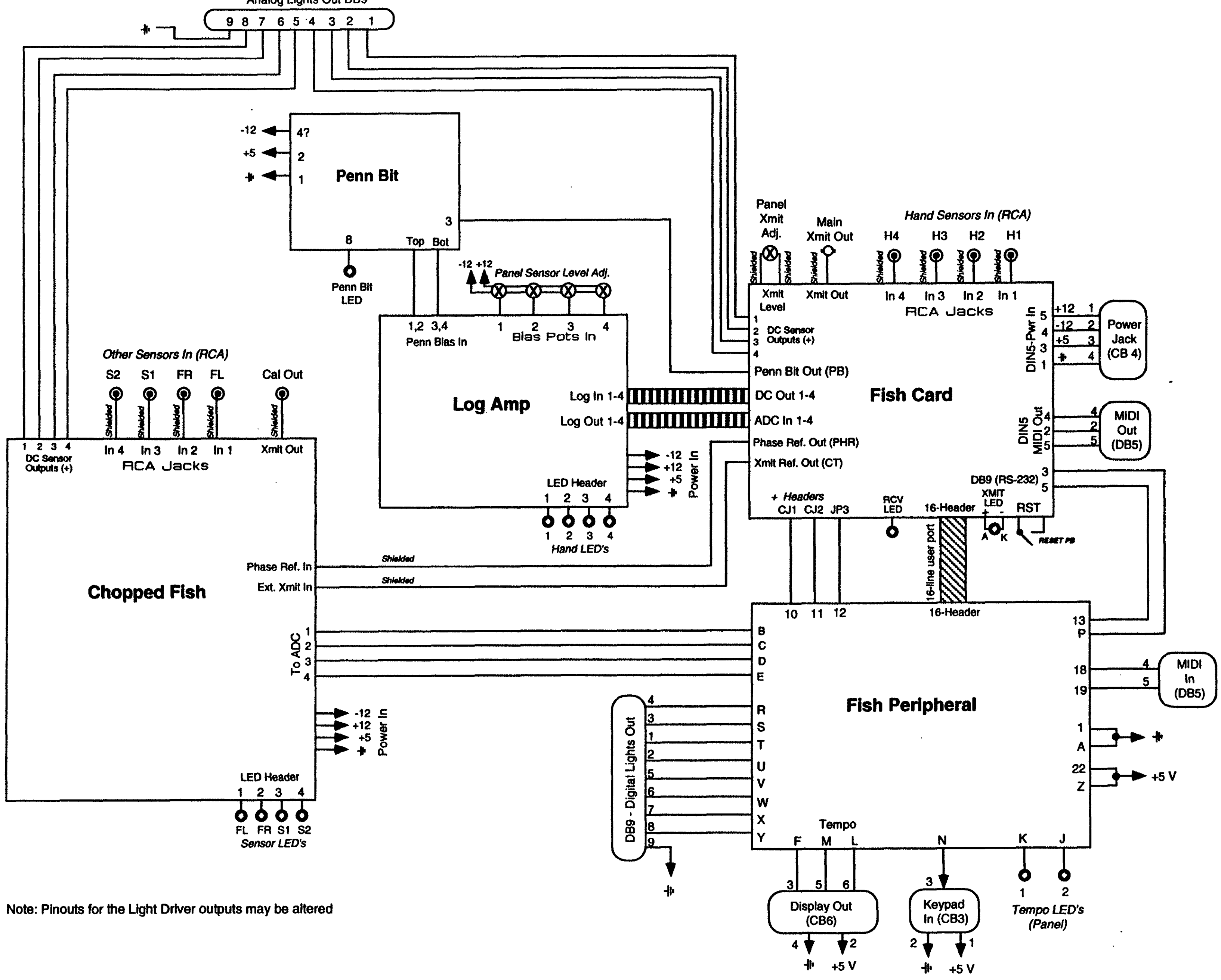


# Wiring Diagrams

# Internal Wirng for the Penn and Teller Grouper

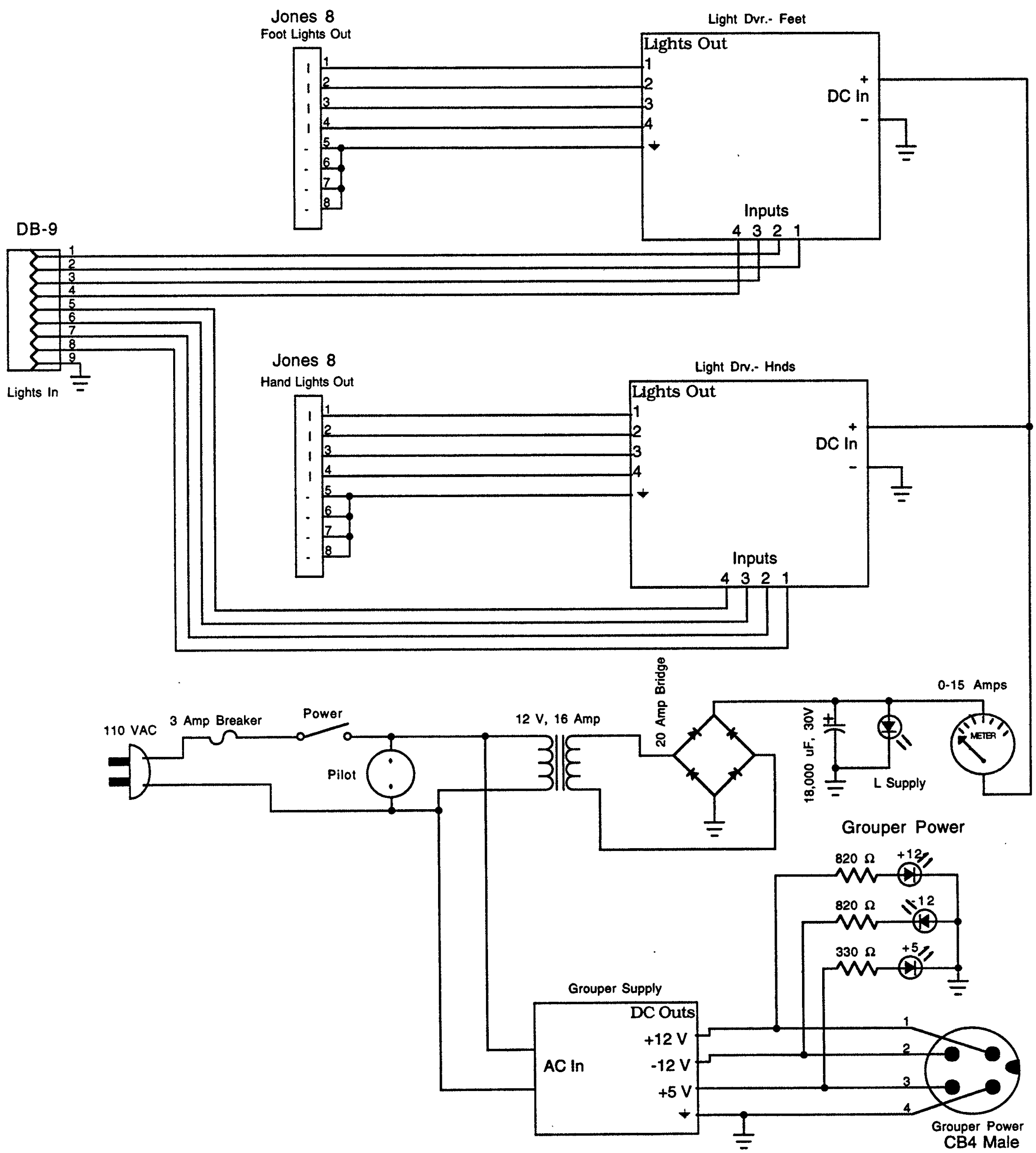
- J. Paradiso 12/94

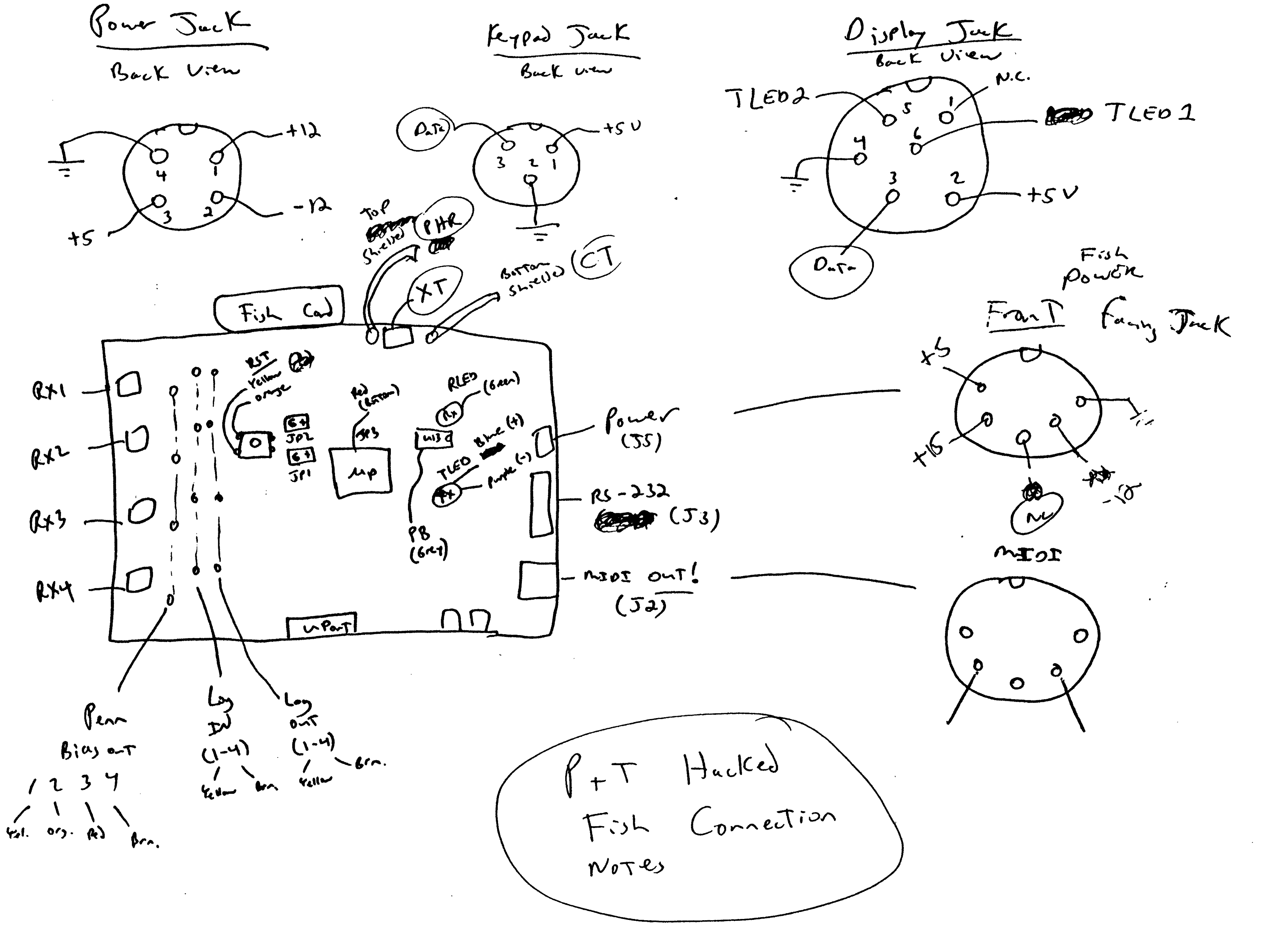


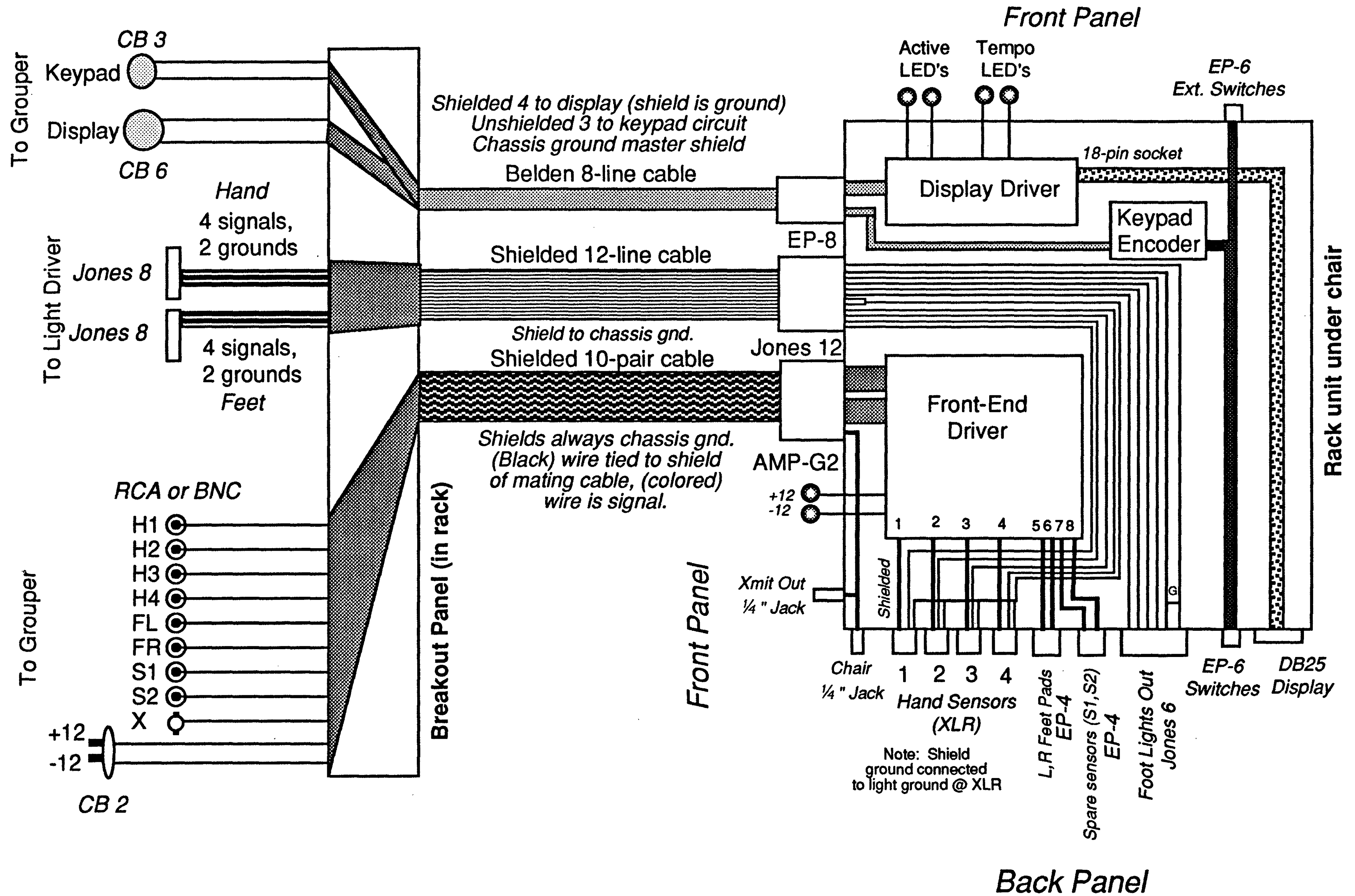

Note: Pinouts for the Light Driver outputs may be altered

# Penn & Teller Light Driver Wiring Diagram

*- J. Paradiso, 12/94*




P+T Hacked Fish Connection Notes
Power Jack
Back View
+12
-12
+5
Keypad Jack
Back view
Data
+5V
CT
Display Jack
Back View
N.C.
TLED2
TLED1
+5V
Data
Fish Card
Top Shield
PHR
XT
Bottom Shield
RST
Yellow
orange
JP2
JP1
Red (Bottom)
JP3
RLED
(Green)
Blue (+)
TLED
purple (-)
PB
(Grey)
Power
(J5)
RS-232
(J3)
MIDI OUT!
(J2)
uPort
RX1
RX2
RX3
RX4
Pan
Bias out
1 2 3 4
Yel. org. Red Brn.
Log IN
(1-4)
Yellow Brn
Log OUT
(1-4)
Yellow Brn.
Fish Power
Front Facing Jack
+5
+15
-15
MIDI



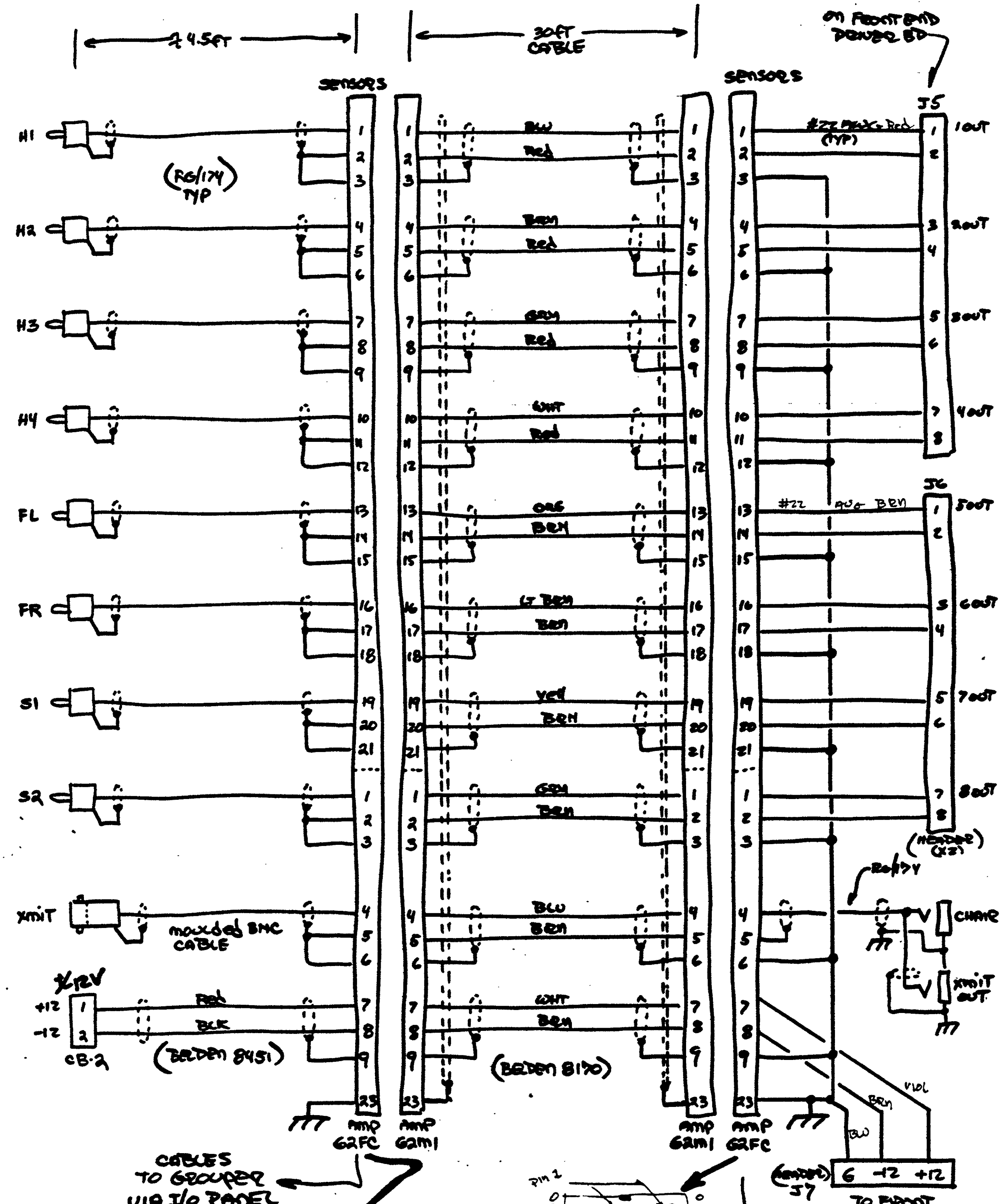
Front Panel
Active LED's
Tempo LED's
EP-6 Ext. Switches
To Grouper
CB 3
Keypad
Display
CB 6
Shielded 4 to display (shield is ground)
Unshielded 3 to keypad circuit
Chassis ground master shield
Belden 8-line cable
18-pin socket
Display Driver
Keypad Encoder
Hand
4 signals,
2 grounds
To Light Driver
Jones 8
Jones 8
4 signals,
2 grounds
Feet
EP-8
Shielded 12-line cable
Shield to chassis gnd.
Shielded 10-pair cable
Jones 12
Front-End Driver
Rack unit under chair
Shields always chassis gnd.
(Black) wire tied to shield
of mating cable, (colored)
wire is signal.
AMP-G2
+12
-12
RCA or BNC
H1
H2
H3
H4
FL
FR
S1
S2
X
To Grouper
+12
-12
CB 2
Breakout Panel (in rack)
Front Panel
Xmit Out
¼ " Jack
Shielded
1 2 3 4 5678
G
Chair
¼ " Jack
1 2 3 4
Hand Sensors
(XLR)
Note: Shield ground connected to light ground @ XLR
L,R Feet Pads
EP-4
Spare sensors (S1,S2)
EP-4
Foot Lights Out
Jones 6
EP-6
Switches
DB25
Display
Back Panel

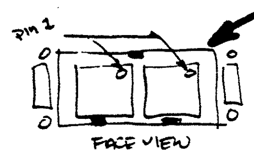
PENN + TELLER CHAIR INTERFACE AND HARNESSING

(of 4)

11/30/94

STUDIO Q ELECTRONICS
26 Raymond Street, Boston, MA 02134

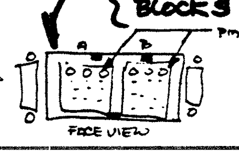



CHAIR INTERFACE BOX

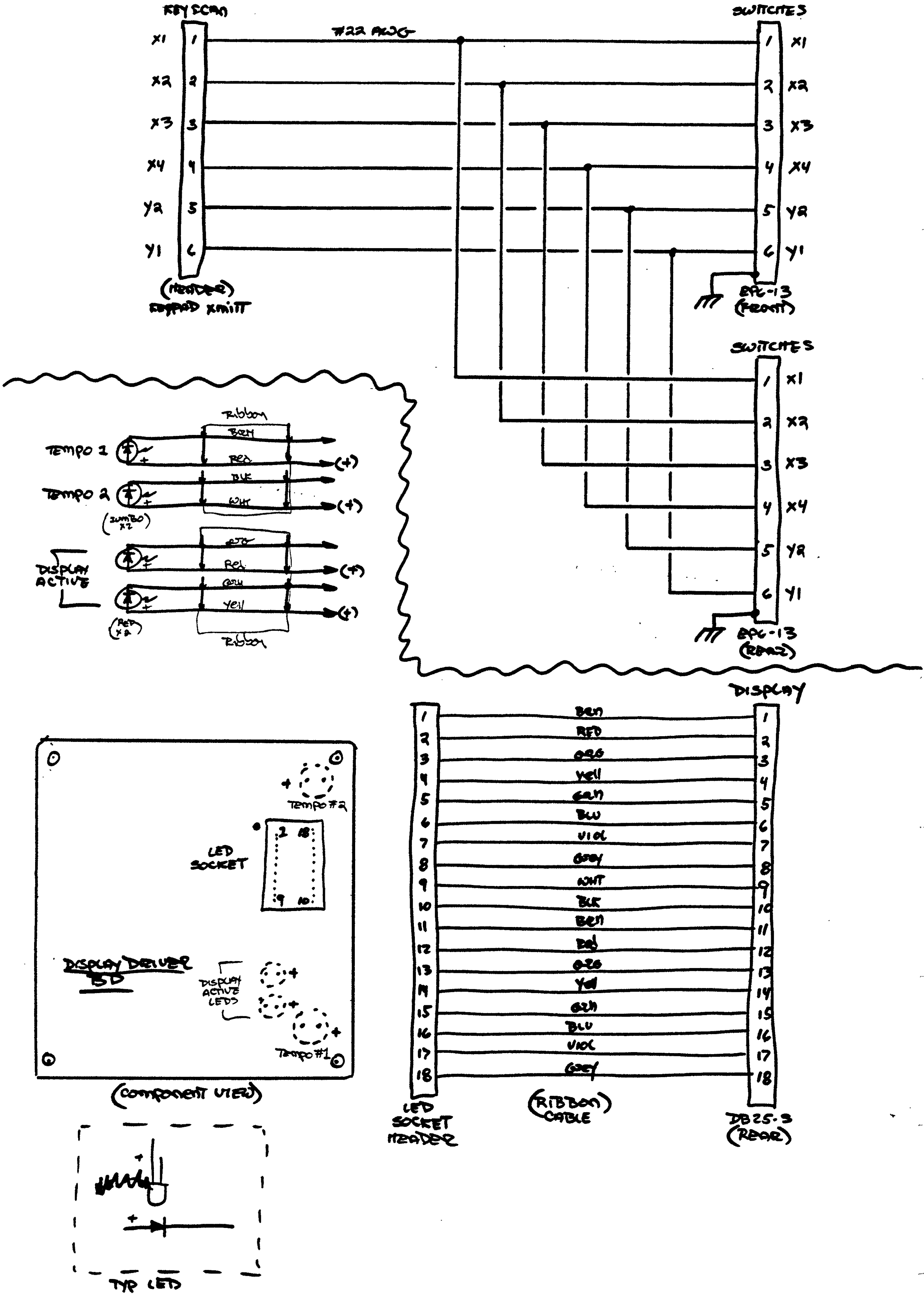
**NOTES:**

- ⏚ = CHASSIS (PANEL) GROUND
- AMP 62FC = { METAL CAGE = P/N 202681-3; LATCH ASSY = P/N 202832-2-00 }
- AMP 62MI = { METAL SHELL = P/N 202617-1 (w CABLE Rubber Grommets); METAL CAGE = P/N 202789-2; BLOCKS (X2) = P/N 202650-2 (23 POSITIONS EA.) }



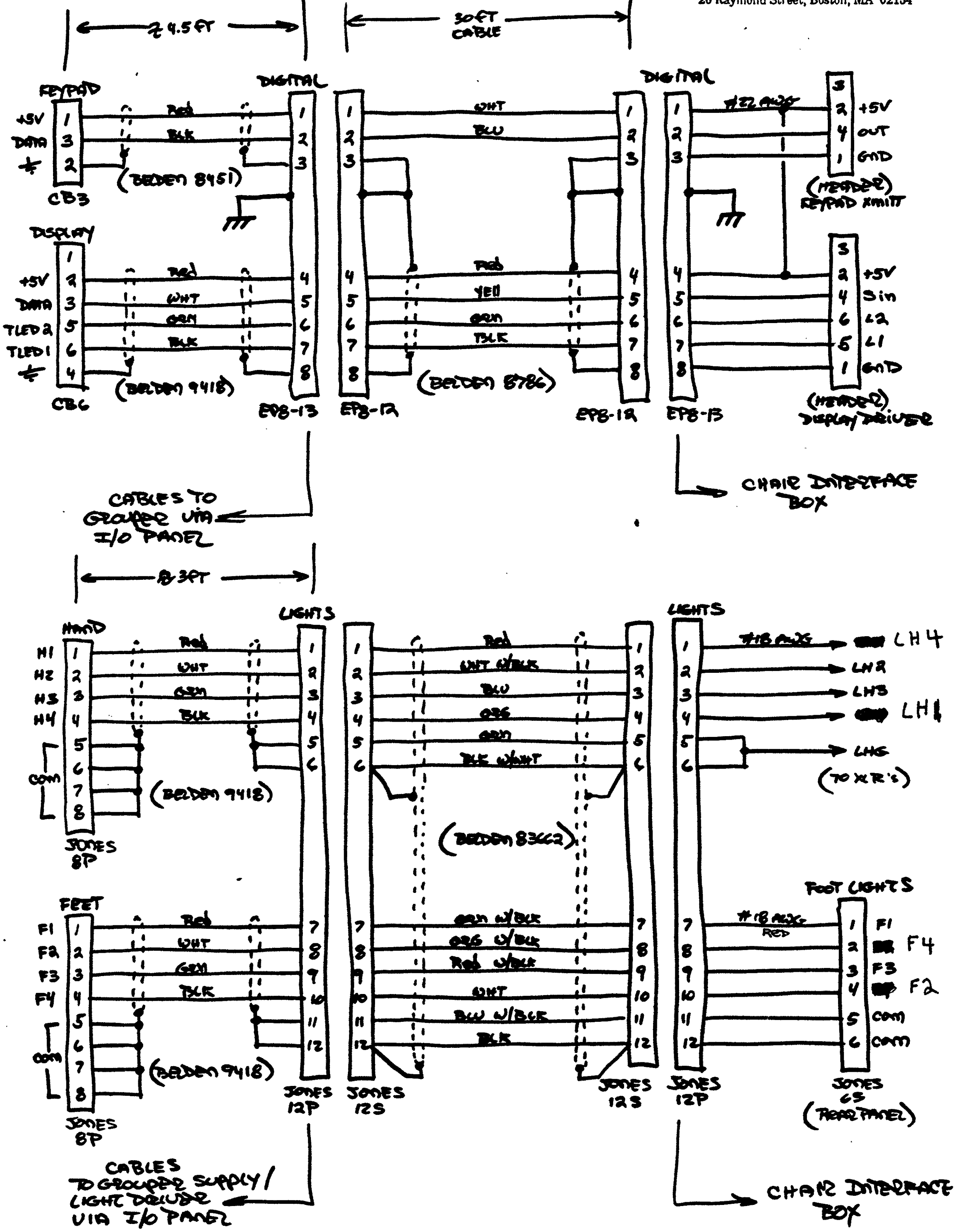
12/1/94

STUDIO Q ELECTRONICS
26 Raymond Street, Boston, MA 02134



12/1/94

STUDIO Q ELECTRONICS
26 Raymond Street, Boston, MA 02134

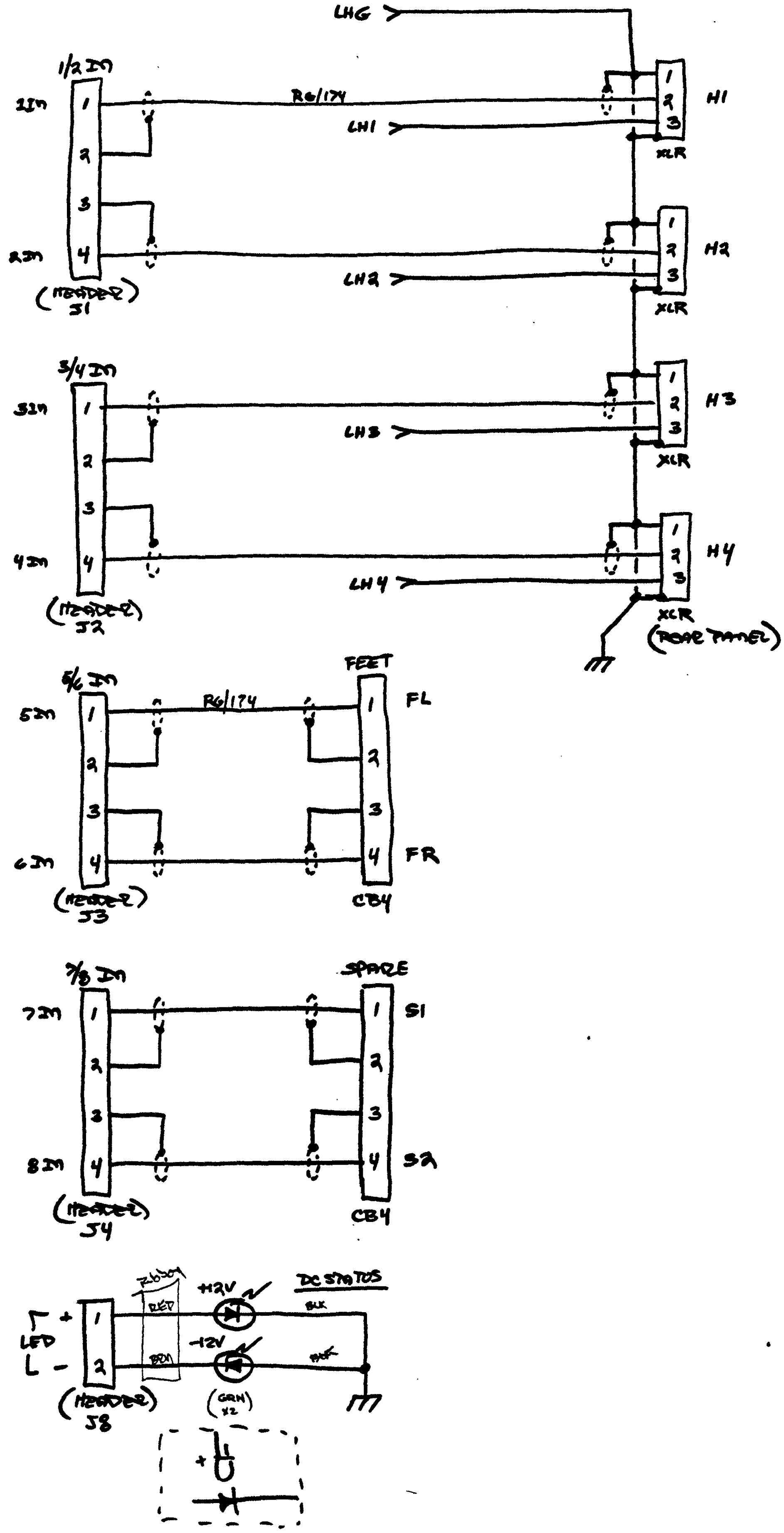



NOTES:

• ⏚ = CHASSIS (PANEL) GROUND

12/1/94

**STUDIO Q ELECTRONICS**
26 Raymond Street, Boston, MA 02134



# Schematic Diagrams

From: Joe Paradiso <joep@media.mit.edu>
Message-Id: <9409120402.AA26395@media.mit.edu>
Subject: Fish Periphal Protocol
To: jrs@media.mit.edu (Joshua R. Smith)
Date: Mon, 12 Sep 1994 00:02:23 -0400 (EDT)
Cc: joep@media.mit.edu (Joe Paradiso), neilg@media.mit.edu (Neil Gershenfeld)
X-Mailer: ELM [version 2.4 PL23]
Content-Type: text
Content-Length: 2595
Status: OR

Josh:

Here's the command structure for the Fish periphal. The added devices hang off the user Port (Port B). One first accesses them by sending an address byte to Port B (with the high bit set to 1), followed by a data byte (with the high bit set to 0). Although it's probably not necessary, put a NOP between sending the address byte and data byte, to insure that the 200 nsec gate pulse that I generate has completely damped. Here are the commands:

1) DAC outputs for light drivers

Address the DAC output by sending a hex 88 (for DAC #1) through hex 8F (for DAC #8). Follow the address with DAC data (7-bits, i.e. 0 - 127).

2) Display Digits

Address the display by sending a hex 84. Send a data BCD byte for the low digit ranging 0 - A (hex), or send a BCD byte for the high digit ranging 10 - 1A (hex). Recall how we discussed implementing this. When the MIDI controller command arrives that addresses the display, break the number into 2 BCD digits. Write the low and high bytes into dedicated locations in RAM. During your event loop, after each 10'th of a second (or so), toggle sending the low and high byte to the display (i.e. send the low byte, wait a tenth second, then send the high byte, wait a tenth second, send the low byte, etc...). This is a simple way to allieviate delay problems in the slow serial link between the Fish Perhiphal and the Holtek receiver at the display.

3) Tempo LED's

Address the tempo LED's by sending a hex 80. Follow this with a data word having the status of each LED in the two low bits (i.e. 0 means both off, 3 means both on, 1 means LED 1 on, 2 means LED 2 on).

4) Reading the pushbutton code

The pushbutton state will appear on Port A, bits 0-2 (the two crystal jump locations, plus the next higher bit). If no switches are down, these bits will be high (i.e. you'll read a 7). If a switch is down, you'll read a binary code ranging 0-6, corresponding to the depressed switch (only one switch will be down at a time). Send MIDI control change commands when the state of a switch changes. Ideally, map a MIDI controller onto each switch, and send a hex F when a switch first goes down, and a 0 when the switch goes up. The switches are debounced already. Remember to mask out the high bits (beyond bit 3) in case there's junk in them; they float (or Tom uses them as outputs).

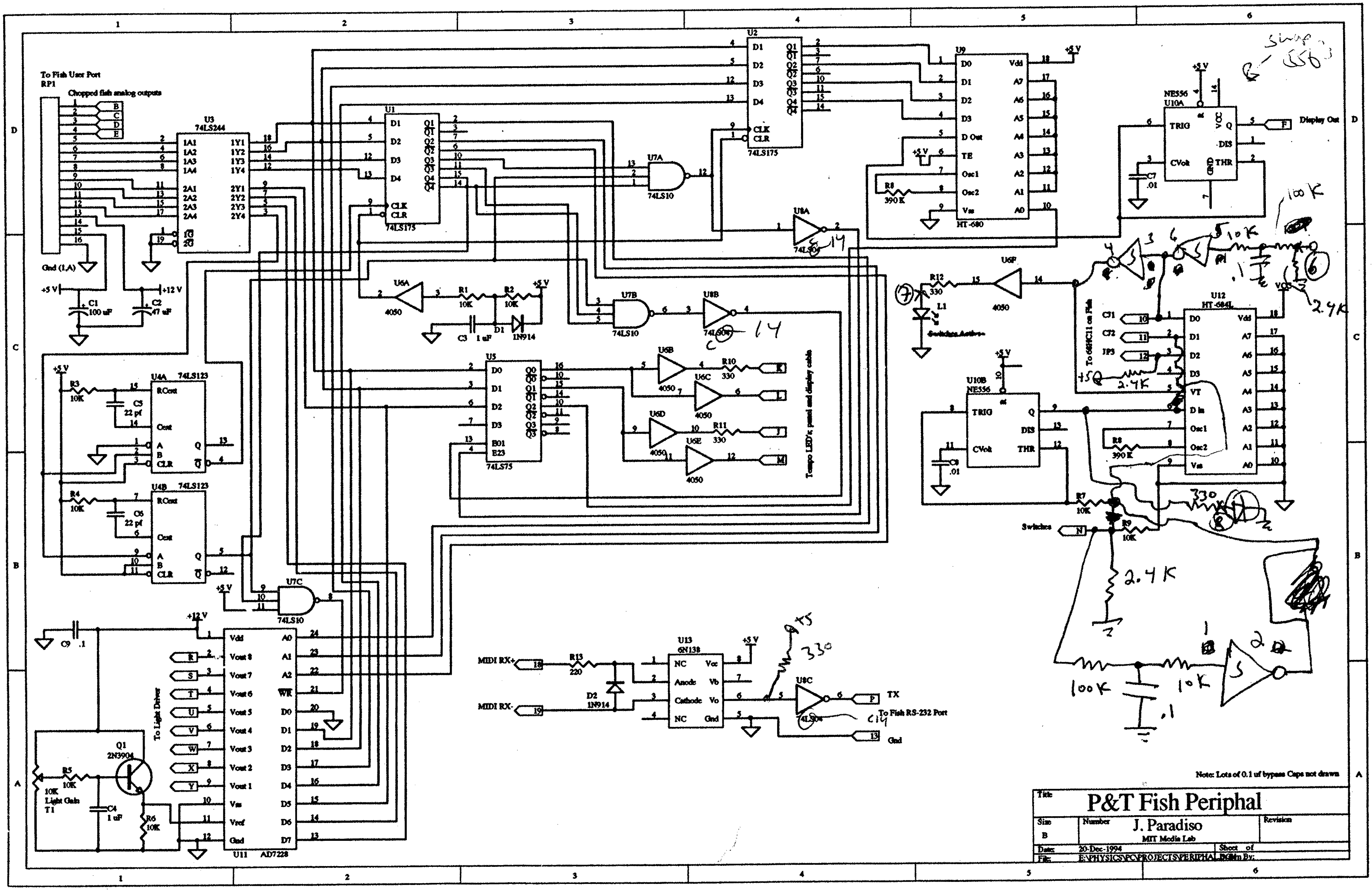
5) The Extra analog channels

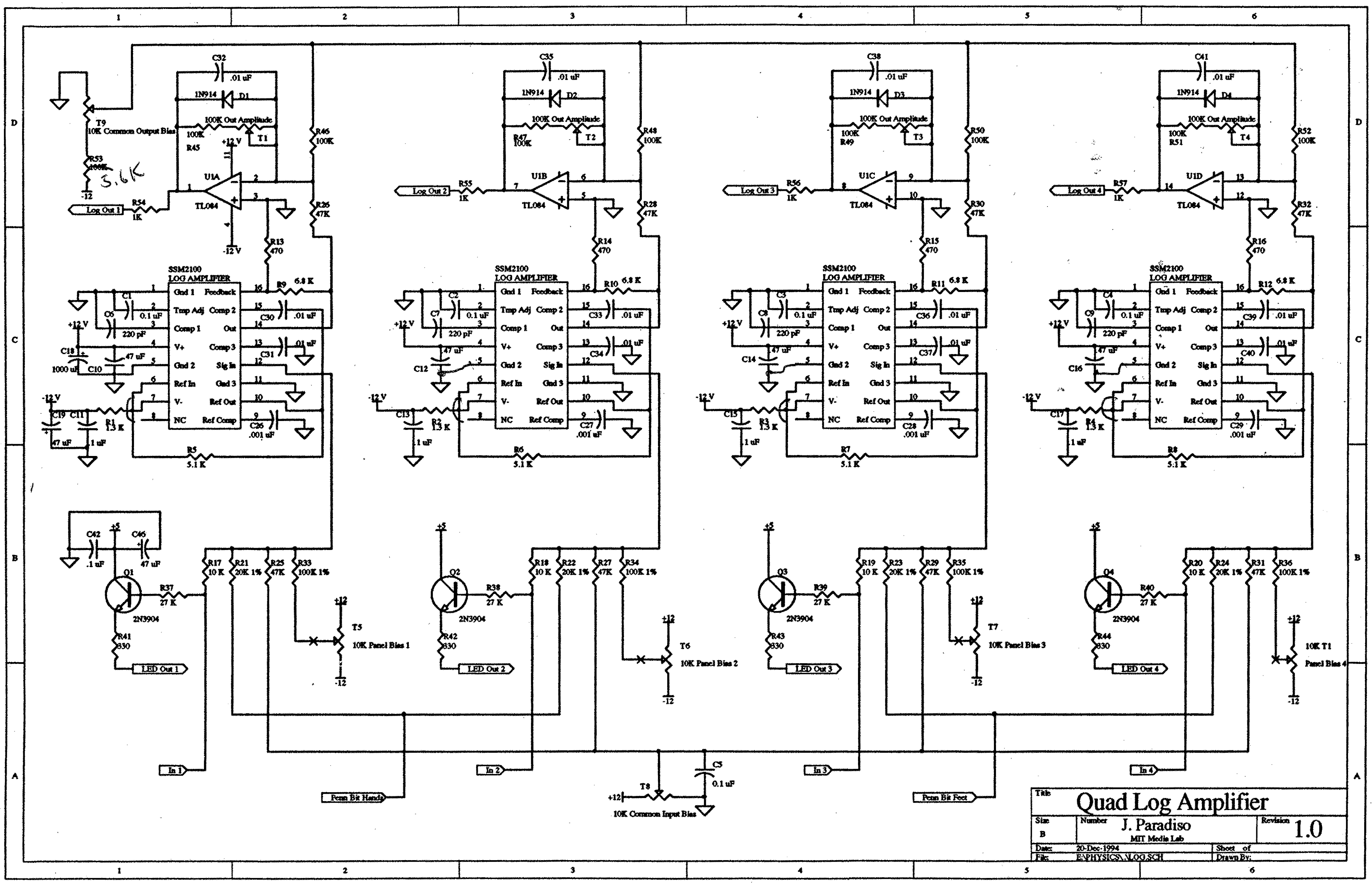
Read the other 4 ADC channels, and treat them just as you do the 4 main fish channels. The auxilary fish card is attached to them.

6) MIDI input

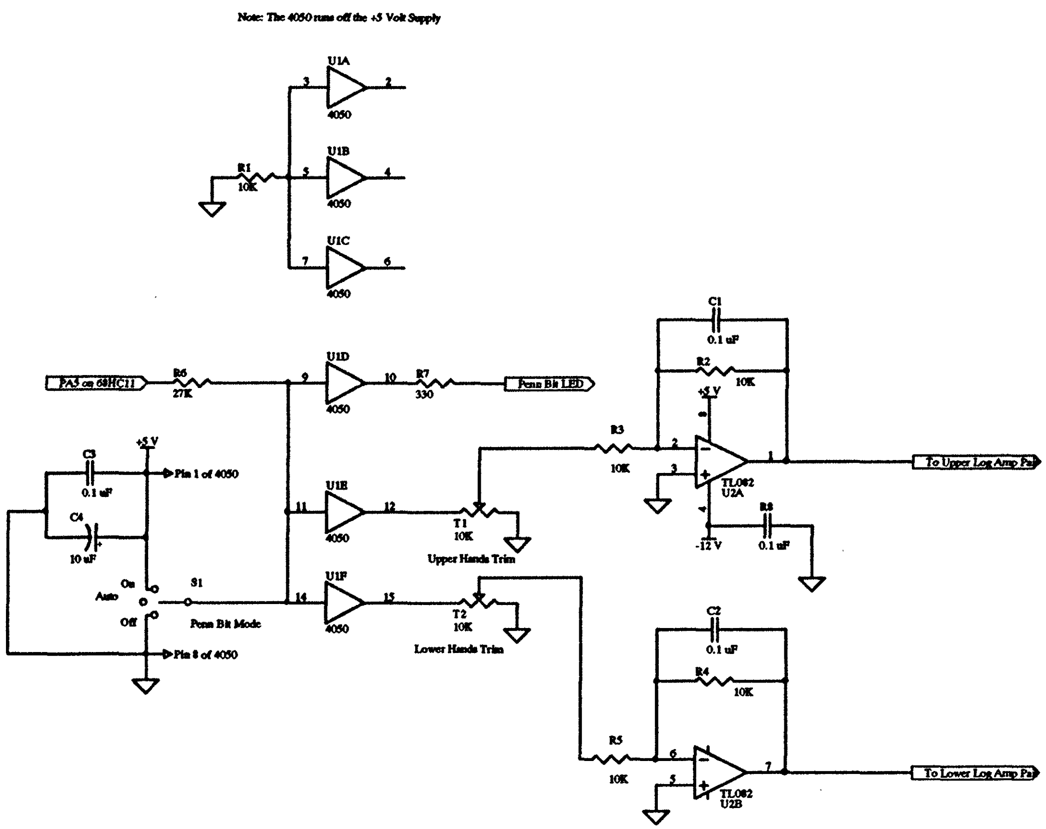
This will appear at the RS-232 serial input.

Enuf -Joe-

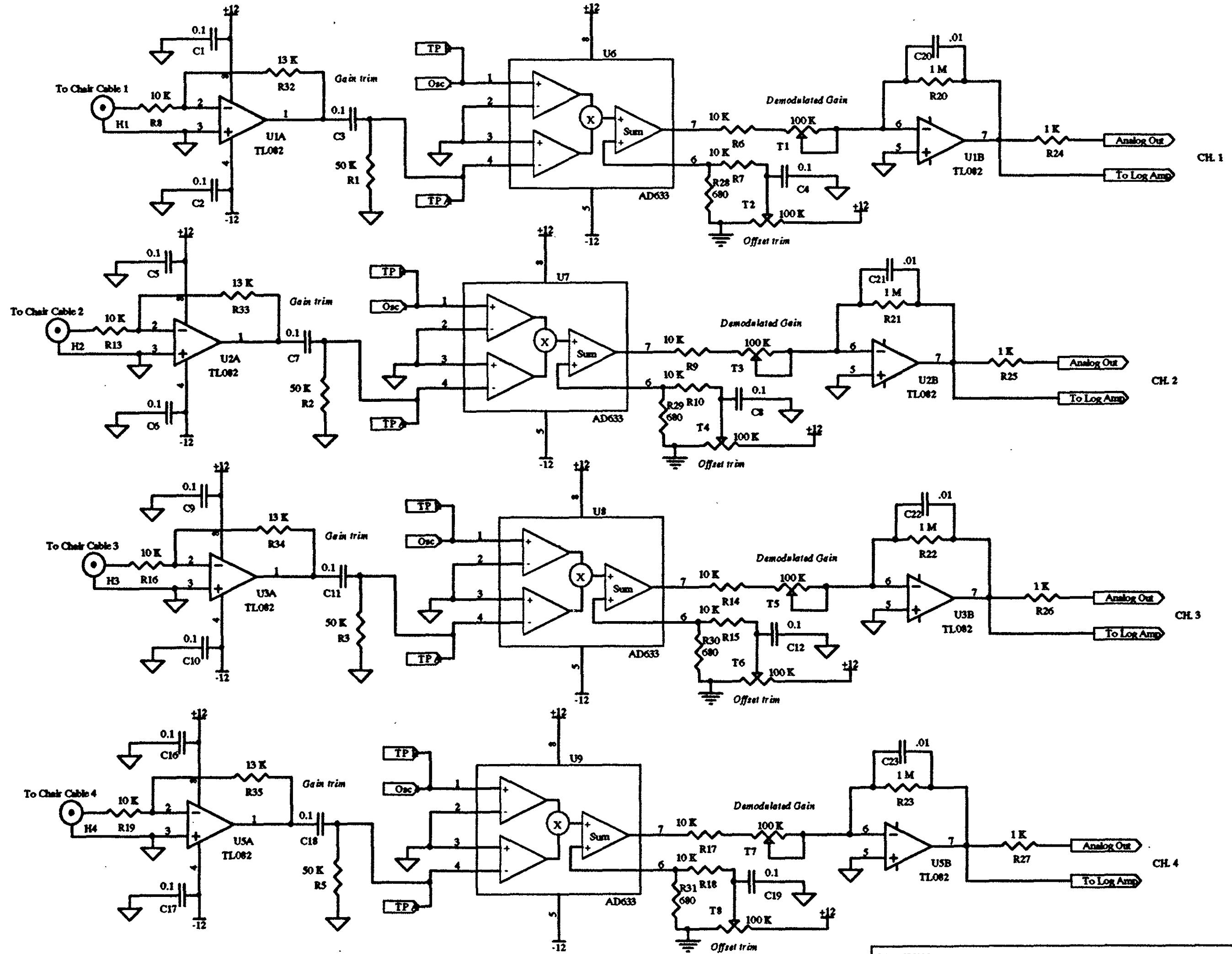

P&T Fish Periphal
J. Paradiso
MIT Media Lab
20-Dec-1994
E:\PHYSICS\PC\PROJECTS\PERIPHAL
Note: Lots of 0.1 uf bypass Caps not drawn
To Fish User Port
RP1
Chopped fish analog outputs
U3 74LS244
U1 74LS175
U2 74LS175
U9 HT-680
NE556 U10A
Display Out
U7A 74LS10
U7B 74LS10
U7C 74LS10
U8A 74LS04
U8B 74LS04
U8C 74LS04
U6A 4050
U6F 4050
U5 74LS75
U4A 74LS123
U4B 74LS123
U12 HT-684L
U10B NE556
U11 AD7228
U13 6N138
Q1 2N3904
To Light Driver
Tempo LED's; panel and display cable
To 68HC11 on Fish
Switches Active
Switches
MIDI RX+
MIDI RX-
To Fish RS-232 Port
TX
Gnd
Light Gain T1



Quad Log Amplifier
J. Paradiso
MIT Media Lab
Revision 1.0
Size B
Date: 20-Dec-1994
File: E:\PHYSICS\...\LOG.SCH



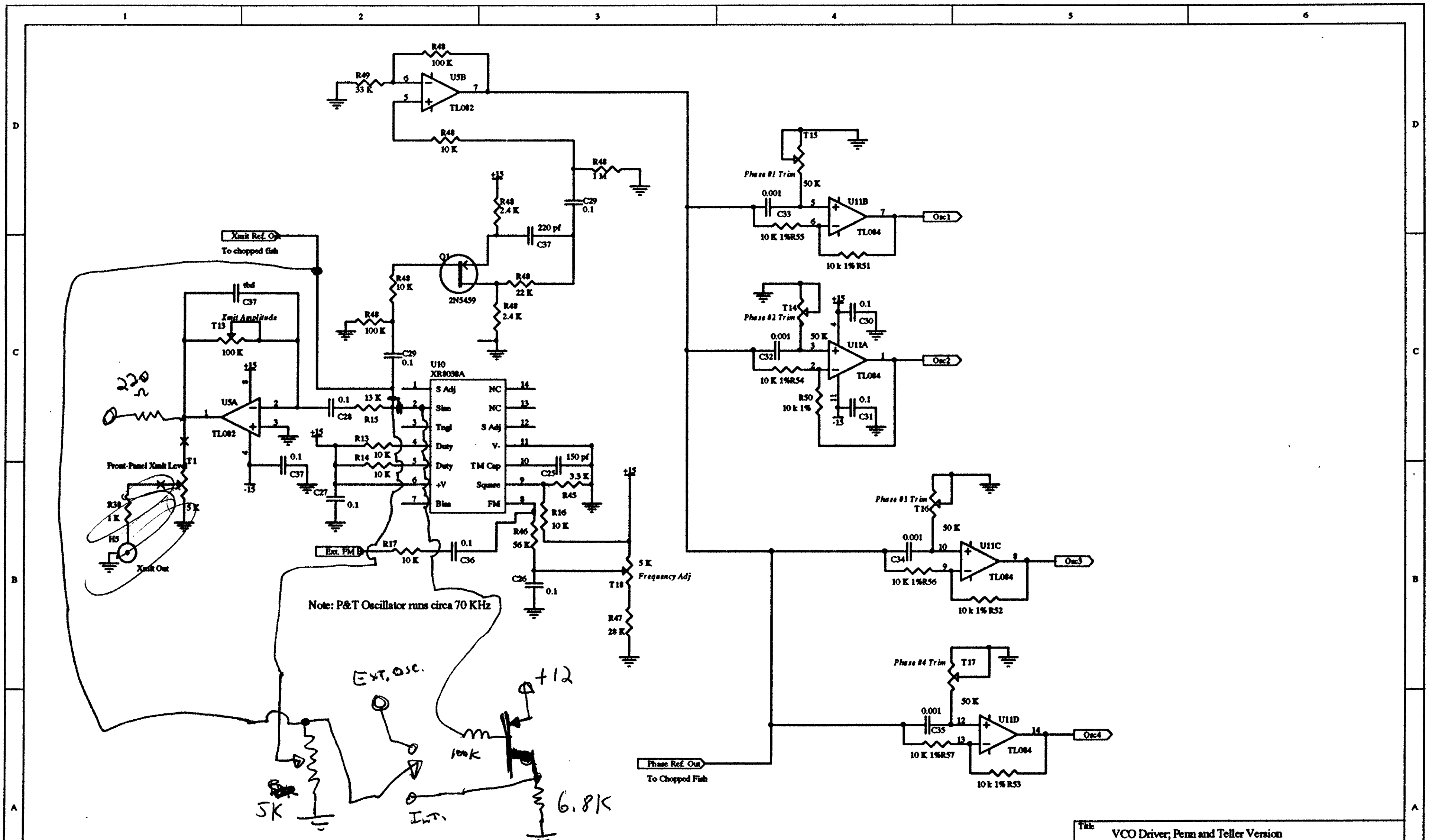

| Title | The Penn Bit | |
|---|---|---|
| Size B | Number J. Paradiso<br>MIT Media Lab | Revision 1.0 |
| Date: 20-Dec-1994 | | Sheet of |
| File: E:\PHYSICS\..\PENNBIT.SCH | | Drawn By: |



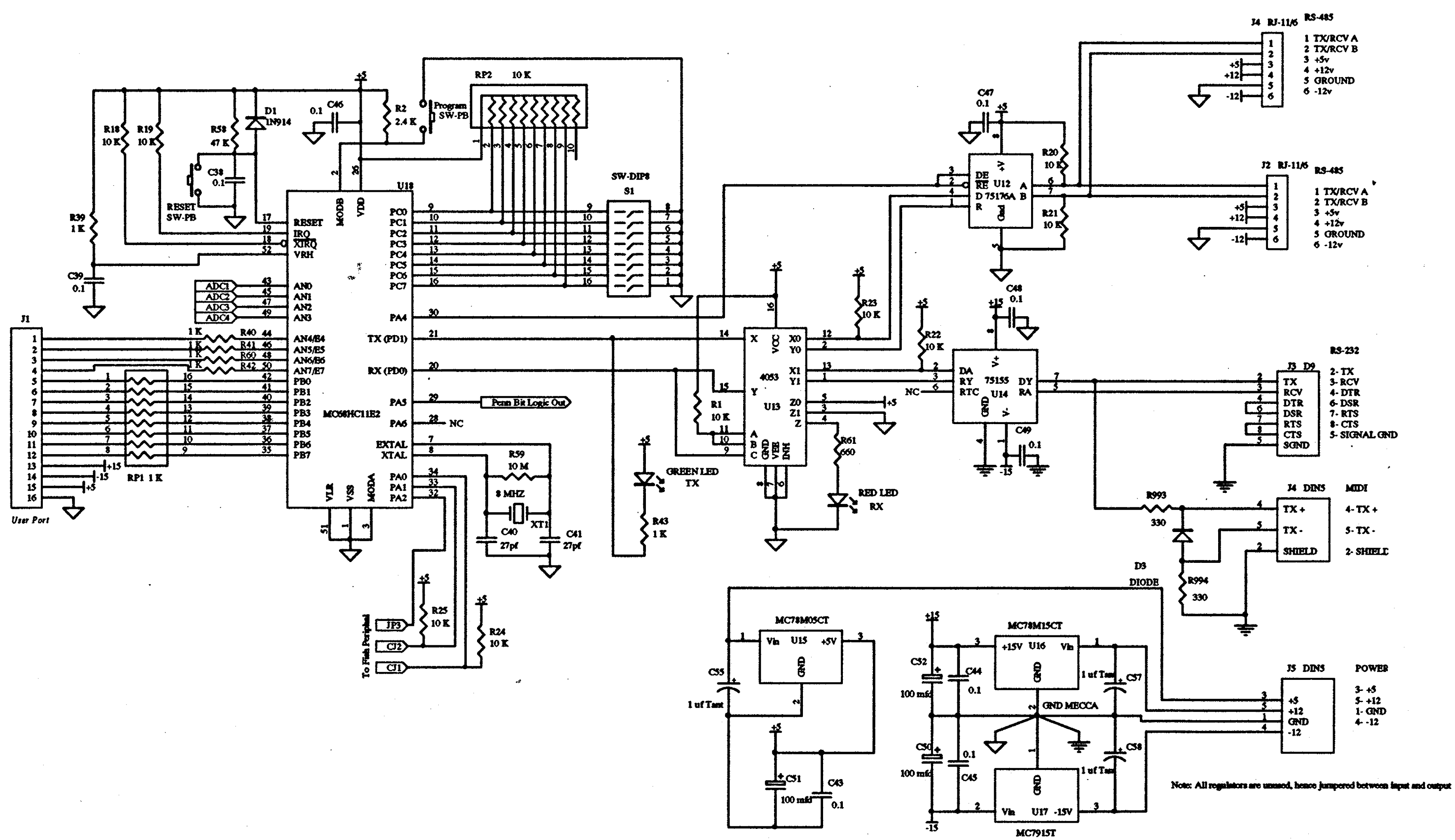

REVISIONS

2- Changed AD633 inputs 1 &4
3- C68 changed from .001 to .01
4- Added R7 and D7
5) Input stage changed for P&T chair driver

| | | |
|---|---|---|
| Title | Synchronous Detector for P&T Fish and Chopped Fish | |
| Size: B | Number | Revision: 4 |
| Date: 20-Dec-1994 | | Sheet of |
| File: E:\PHYSICS\...\CAPRCVPT.SCH | | Drawn By: Joe Paradiso |

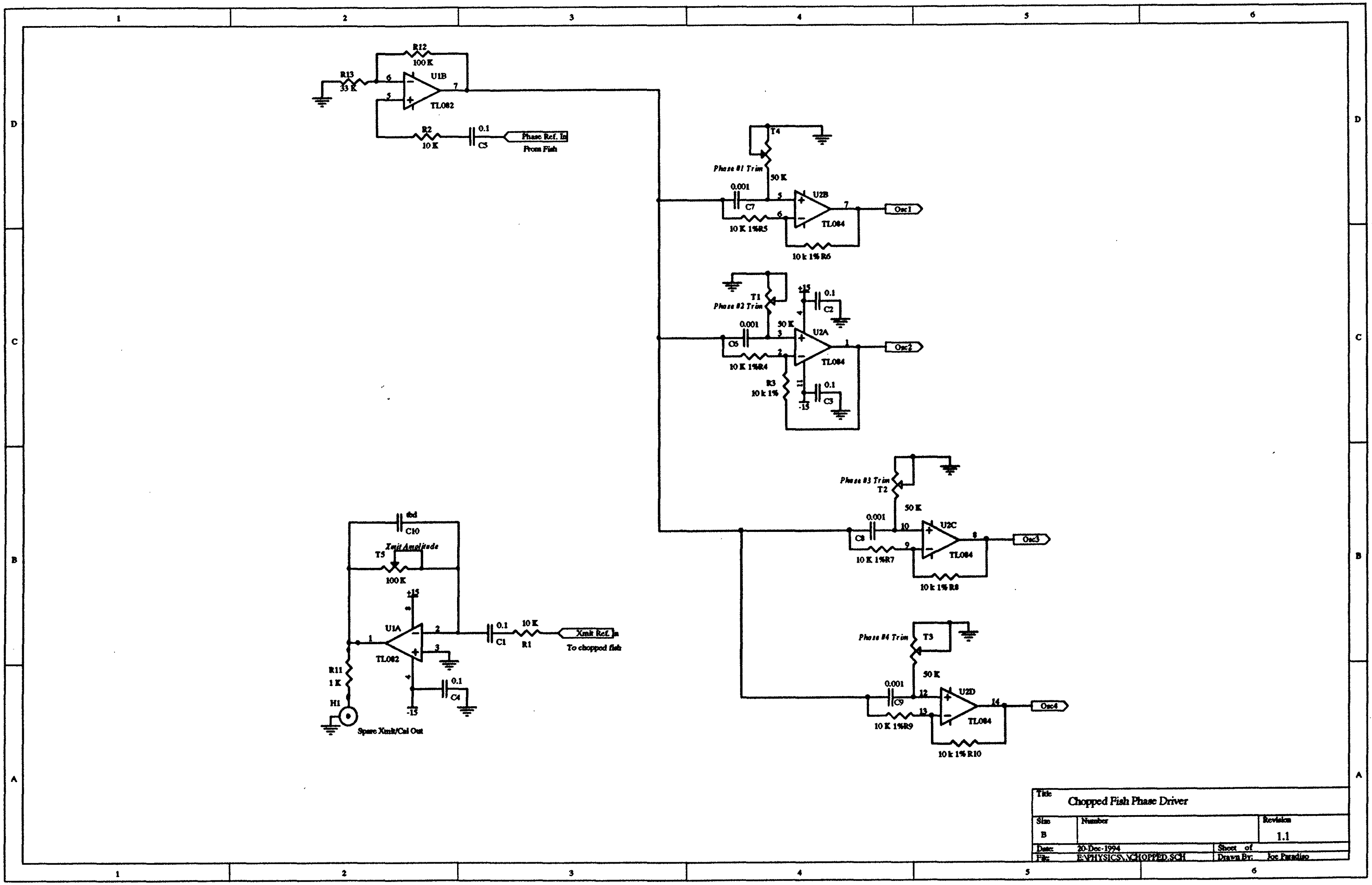



| | | |
|---|---|---|
| Title | VCO Driver; Penn and Teller Version | |
| Size B | Number | Revision 1.1 |
| Date: 20-Dec-1994 | Sheet of | |
| File: E:\PHYSICS\...\OSCPT.SCH | Drawn By: Joe Paradiso | |

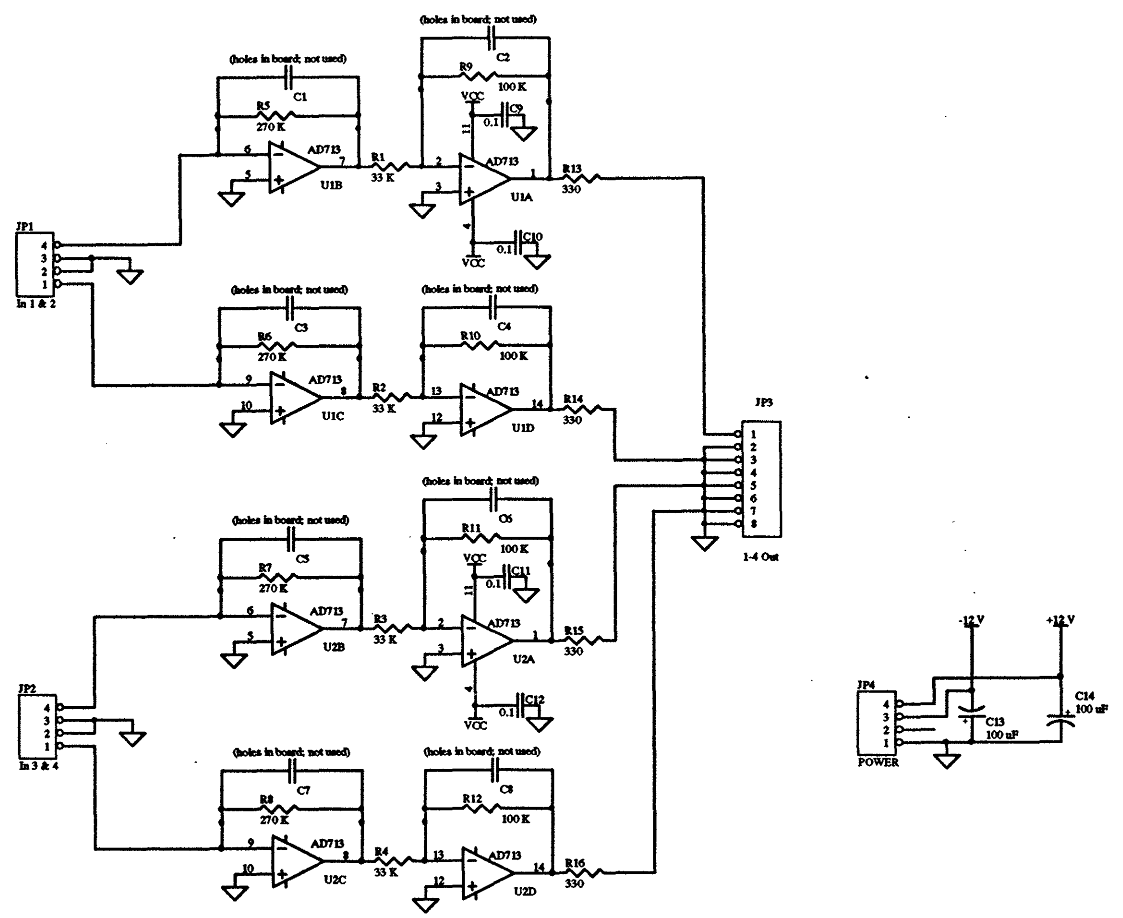



REVISION LIST
1- Midi and power connector added
2- Midi connector & diode wiring changed
3- Leds polarity changed, reset button added, 75155 filter cap removed
4- RS-232 port wiring changed, S1 rotated, 75176 output reversed

| Title | SMALL BOX MICROPROCESSOR, INTERFACE AND POWER SUPPLY<br>Penn and Teller Version | |
|---|---|---|
| Size: B | Number | Revision 4 |
| Date: | 20-Dec-1994 | Sheet of |
| File: | E:\PHYSICS\...\ADCPT.SCH | Drawn By: |

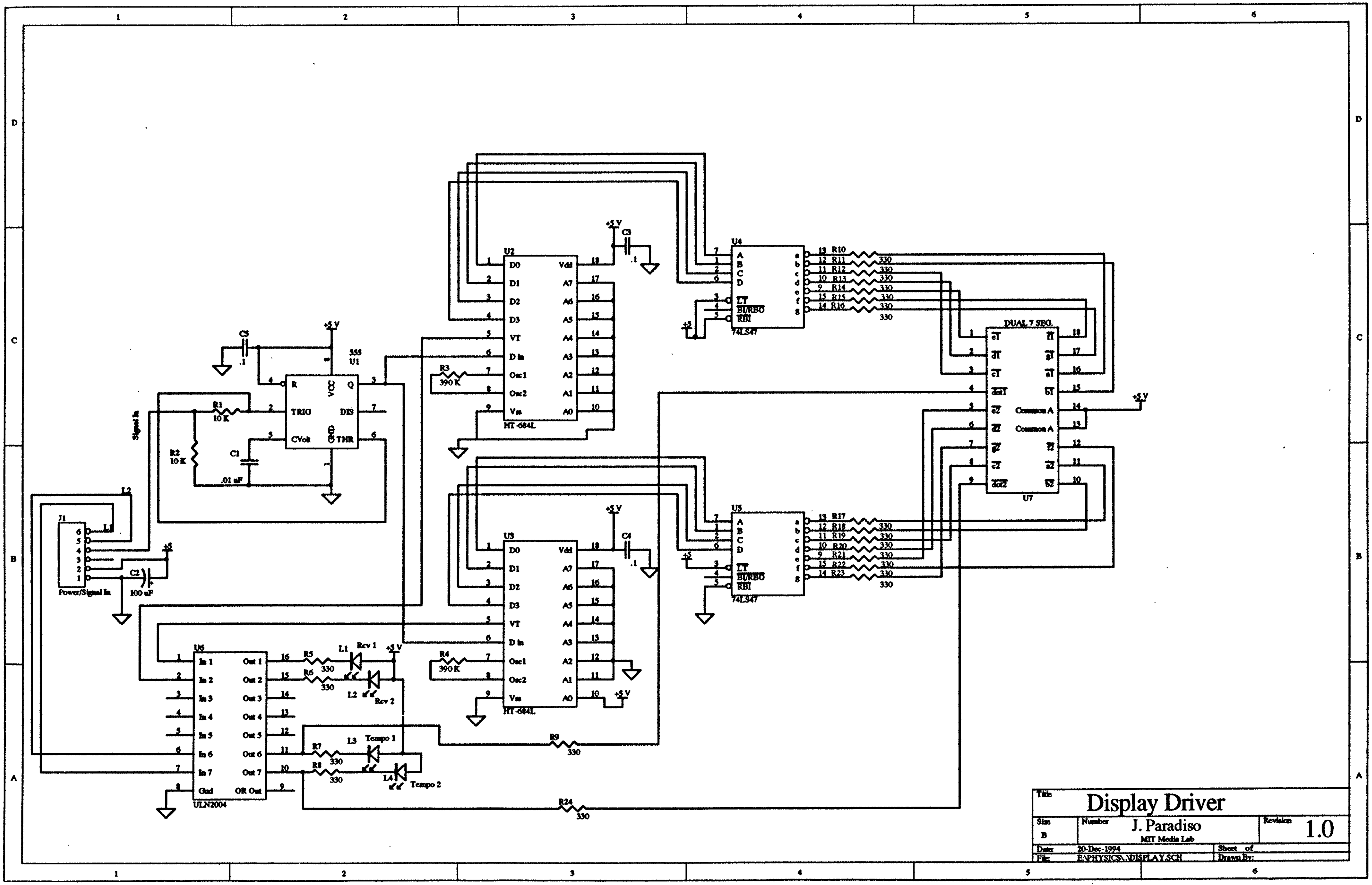


R12
100 K
R13
33 K
U1B
TL082
R2
10 K
0.1
C5
Phase Ref. In
From Fish
Phase #1 Trim
T4
50 K
0.001
C7
10 K 1%R5
U2B
TL084
Osc1
10 k 1% R6
T1
Phase #2 Trim
50 K
0.001
C6
10 K 1%R4
+15
0.1
C2
U2A
TL084
Osc2
R3
10 k 1%
0.1
C3
-15
Phase #3 Trim
T2
50 K
0.001
C8
10 K 1%R7
U2C
TL084
Osc3
10 k 1% R8
Phase #4 Trim
T3
50 K
0.001
C9
10 K 1%R9
U2D
TL084
Osc4
10 k 1% R10
tbd
C10
Xmit Amplitude
T5
100 K
+15
U1A
TL082
0.1
C1
10 K
R1
Xmit Ref.
To chopped fish
0.1
C4
-15
R11
1 K
H1
Spare Xmit/Cal Out
Title
Chopped Fish Phase Driver
Size
B
Number
Revision
1.1
Date: 20-Dec-1994
Sheet of
File: E:\PHYSICS\\CHOPPED.SCH
Drawn By: Joe Paradiso

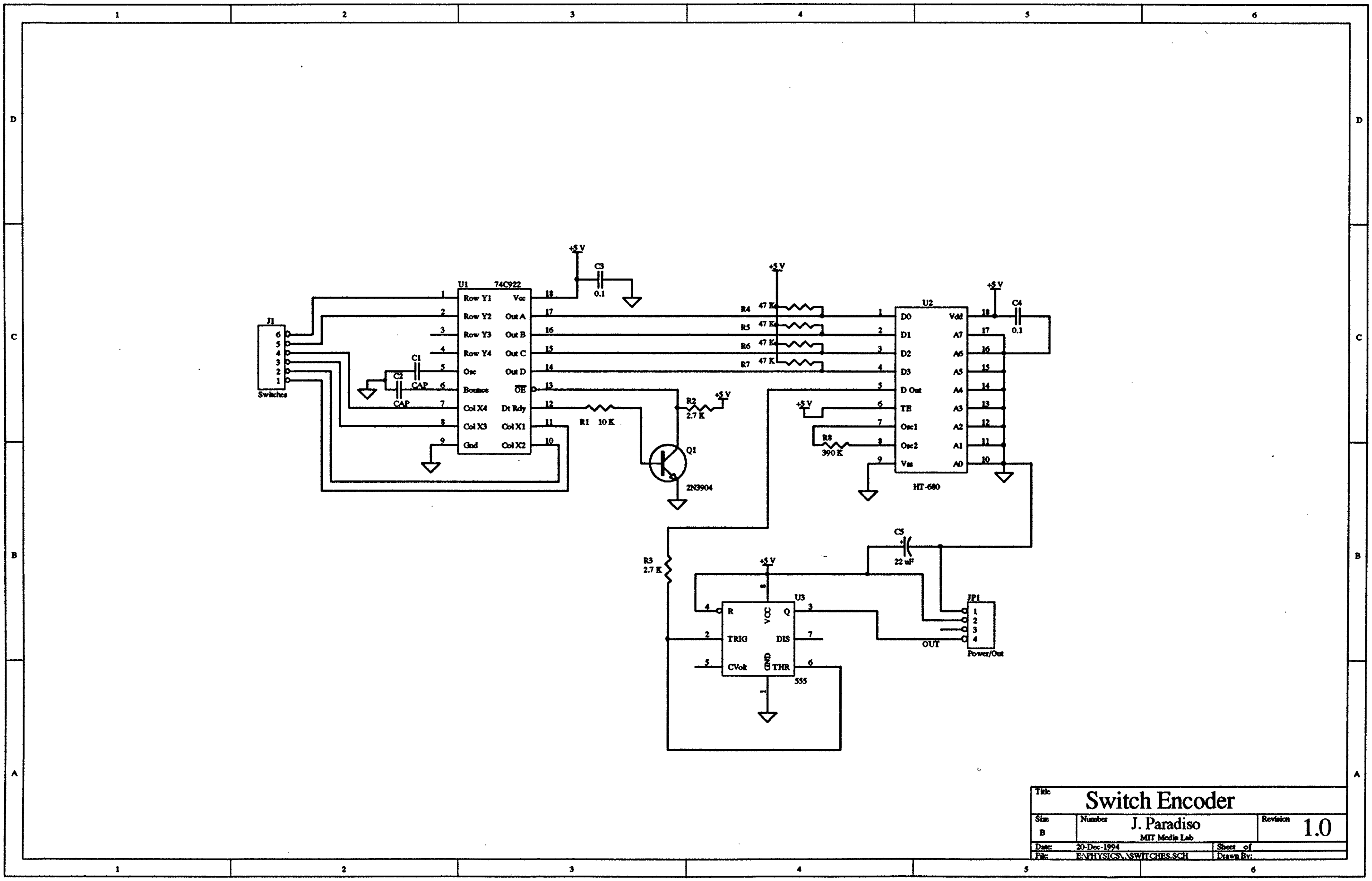


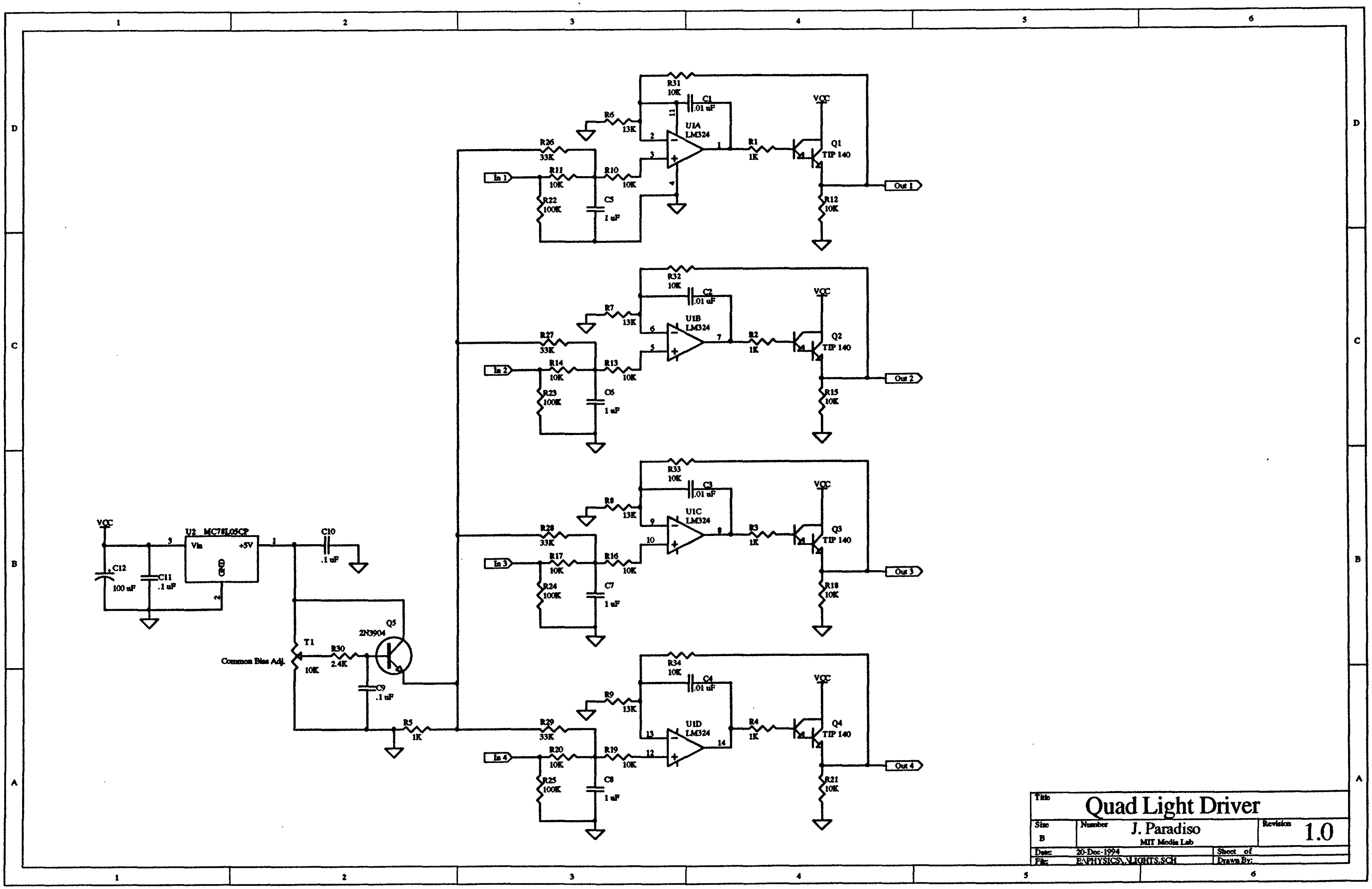

Above circuit repeated for chnls. 5-8

| Title | Octal Front-End Driver | |
|---|---|---|
| Size B | Number J. Paradiso MIT Media Lab | Revision 1.0 |
| Date: 20-Dec-1994 | | Sheet of |
| File: E:\PHYSICS\...\DRIVER.SCH | | Drawn By: |



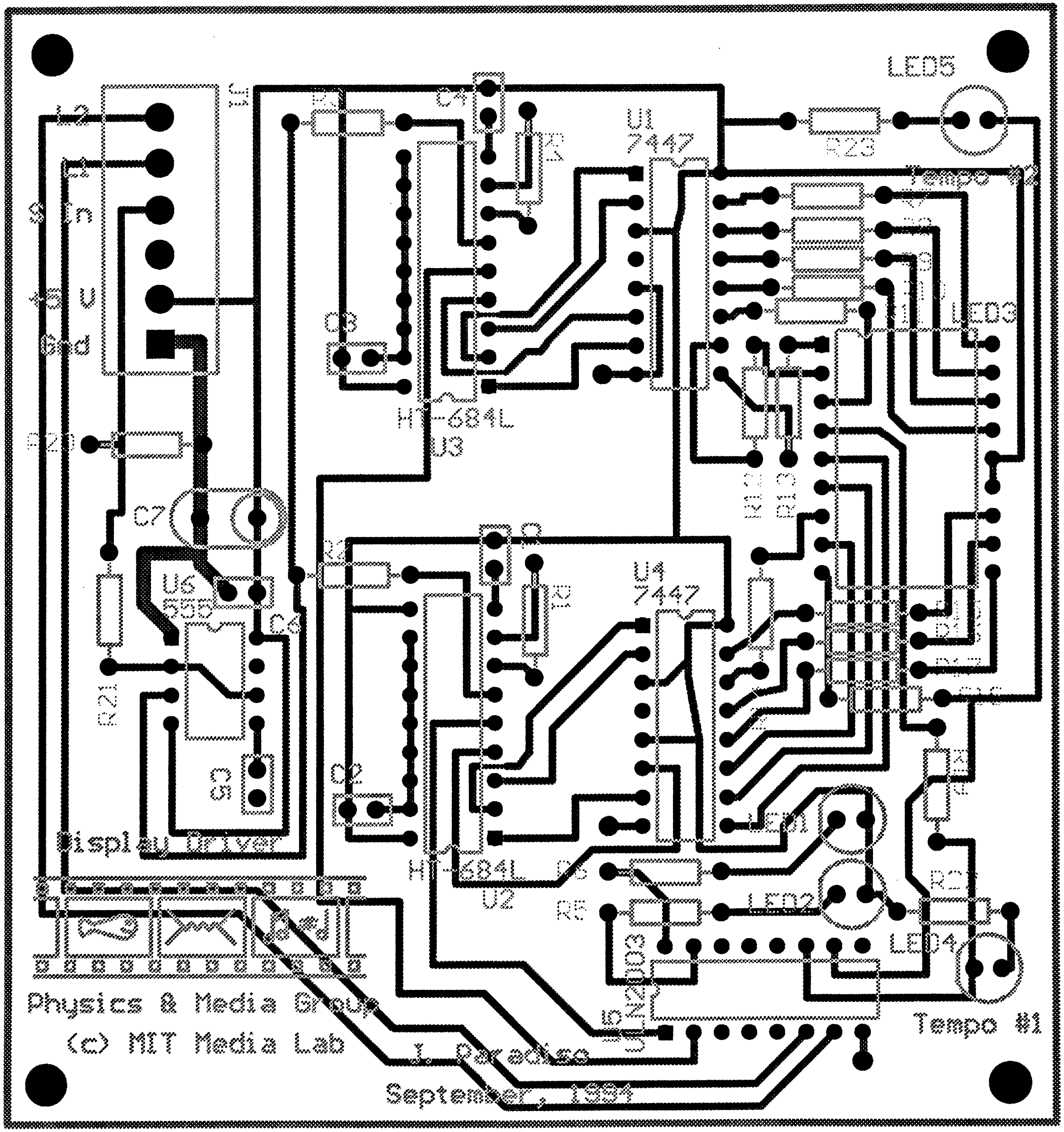
Display Driver
J. Paradiso
MIT Media Lab
Revision 1.0
Size B
Date: 20-Dec-1994
File: E:\PHYSICS\..\DISPLAY.SCH
Sheet of
Drawn By:
555 U1
HT-684L U2
HT-684L U3
74LS47 U4
74LS47 U5
ULN2004 U6
DUAL 7 SEG. U7
Power/Signal In J1
Signal In
Rev 1
Rev 2
Tempo 1
Tempo 2


J1
Switches
U1
74C922
Row Y1
Row Y2
Row Y3
Row Y4
Osc
Bounce
Col X4
Col X3
Gnd
Vcc
Out A
Out B
Out C
Out D
OE
Dt Rdy
Col X1
Col X2
C1
CAP
C2
CAP
C3
0.1
+5 V
R1 10 K
R2
2.7 K
Q1
2N3904
R4 47 K
R5 47 K
R6 47 K
R7 47 K
U2
HT-680
D0
D1
D2
D3
D Out
TE
Osc1
Osc2
Vss
Vdd
A7
A6
A5
A4
A3
A2
A1
A0
C4
0.1
R8
390 K
R3
2.7 K
C5
22 uF
U3
555
R
VCC
Q
TRIG
DIS
CVolt
GND
THR
JP1
Power/Out
OUT
Title
Switch Encoder
Size
B
Number
J. Paradiso
MIT Media Lab
Revision
1.0
Date: 20-Dec-1994
Sheet of
File: E:\PHYSICS\...\SWITCHES.SCH
Drawn By:

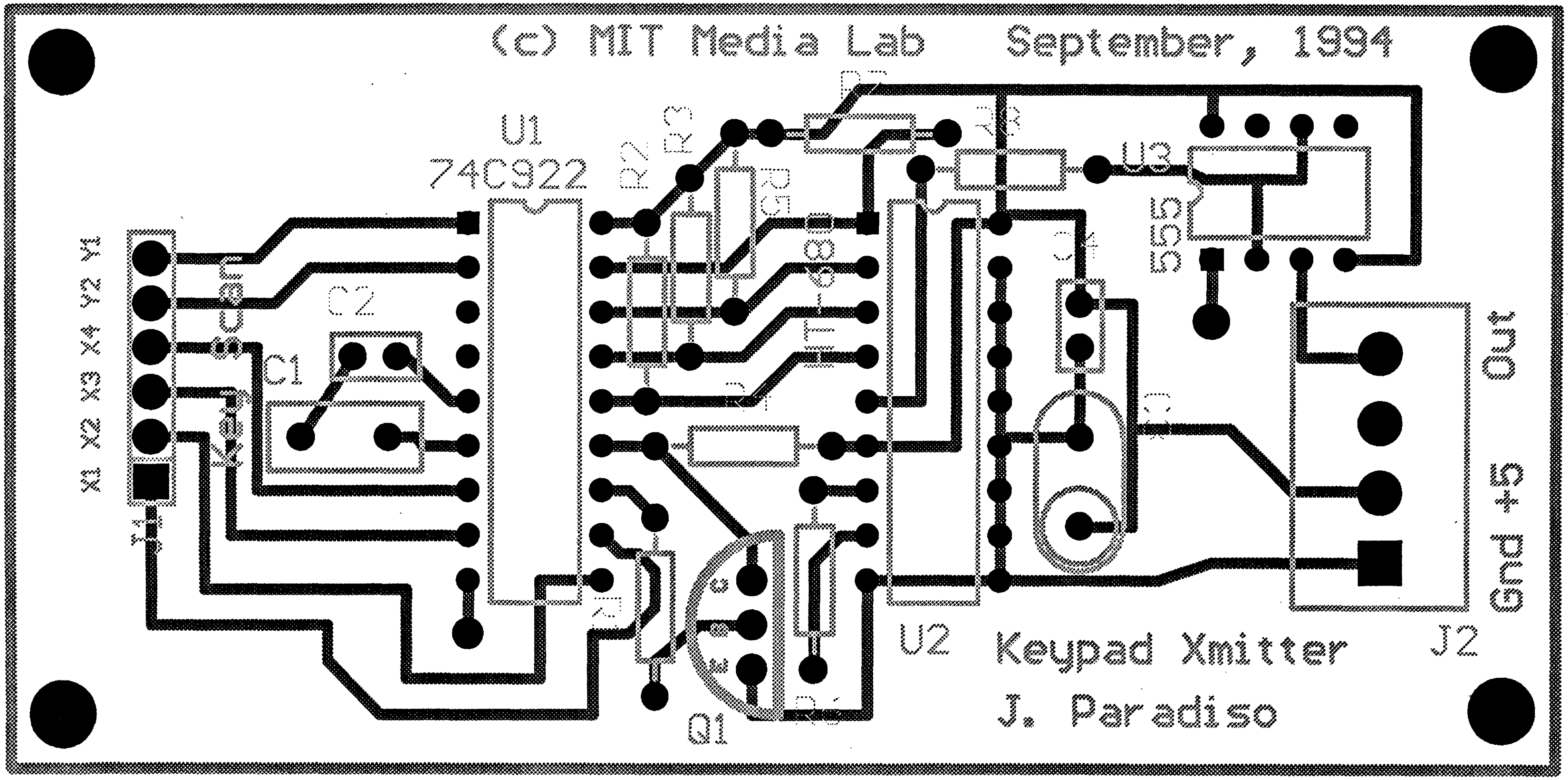


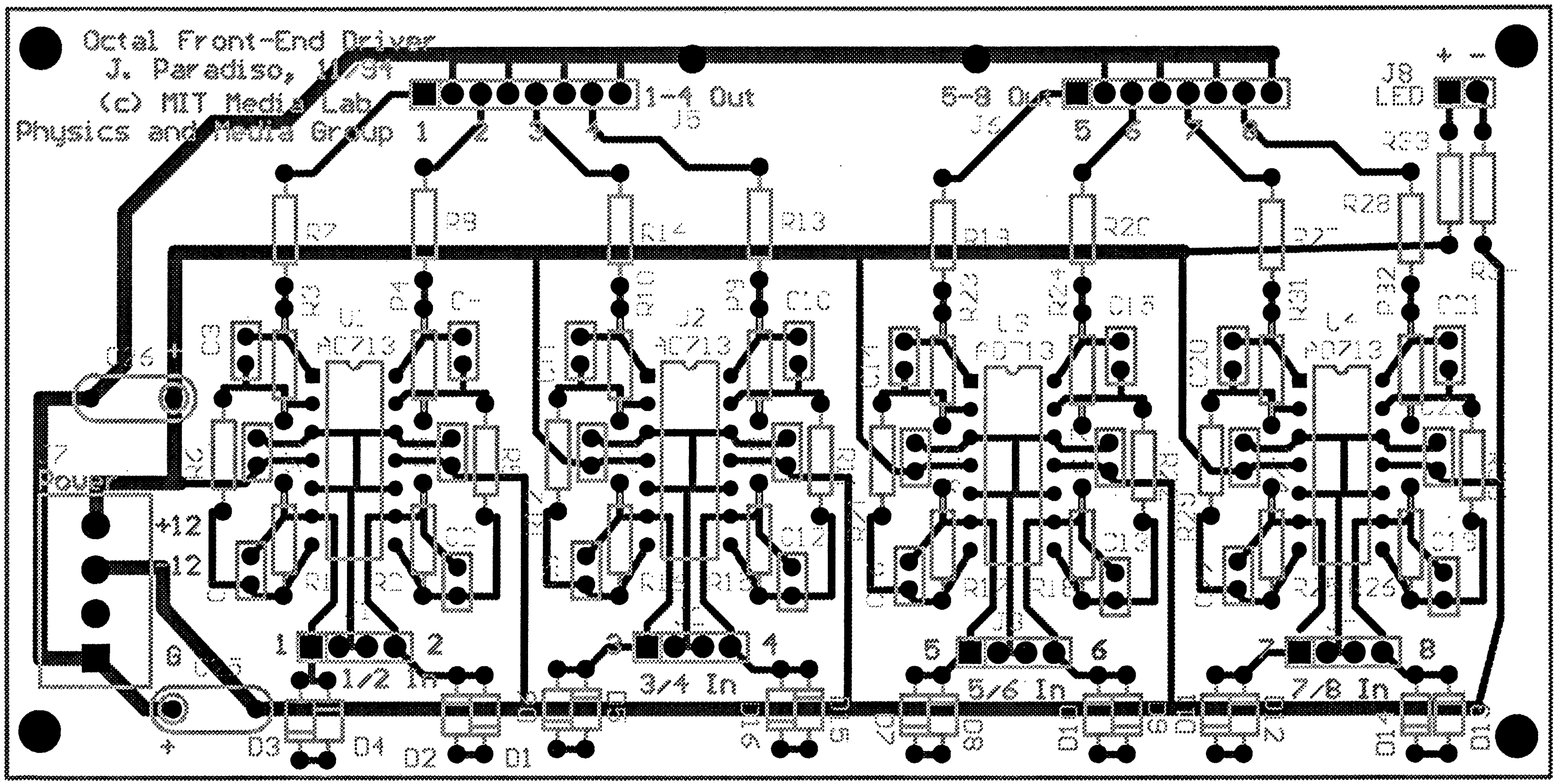
Quad Light Driver
J. Paradiso
MIT Media Lab
Size B
Revision 1.0
Date: 20-Dec-1994
File: E:\PHYSICS\...\LIGHTS.SCH
Sheet of
Drawn By:
U2 MC78L05CP
Vin
+5V
GND
VCC
C12 100 uF
C11 .1 uF
C10 .1 uF
Q5 2N3904
T1 10K
Common Bias Adj.
R30 2.4K
C9 .1 uF
R5 1K
U1A LM324
U1B LM324
U1C LM324
U1D LM324
Q1 TIP 140
Q2 TIP 140
Q3 TIP 140
Q4 TIP 140
In 1
In 2
In 3
In 4
Out 1
Out 2
Out 3
Out 4
R26 33K
R27 33K
R28 33K
R29 33K
R11 10K
R14 10K
R17 10K
R20 10K
R10 10K
R13 10K
R16 10K
R19 10K
R22 100K
R23 100K
R24 100K
R25 100K
C5 1 uF
C6 1 uF
C7 1 uF
C8 1 uF
R6 13K
R7 13K
R8 13K
R9 13K
R31 10K
R32 10K
R33 10K
R34 10K
C1 .01 uF
C2 .01 uF
C3 .01 uF
C4 .01 uF
R1 1K
R2 1K
R3 1K
R4 1K
R12 10K
R15 10K
R18 10K
R21 10K

# PC Layouts

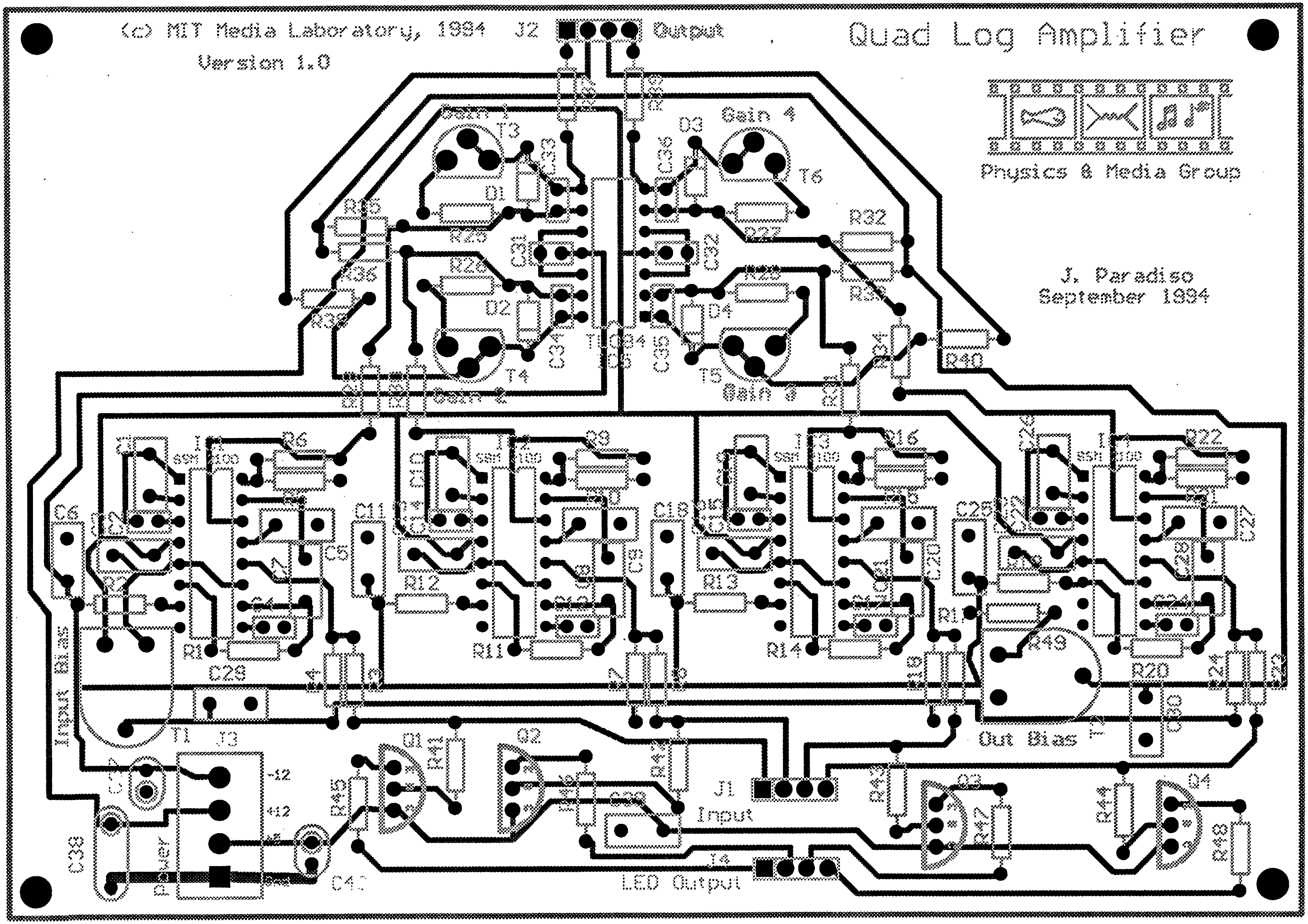


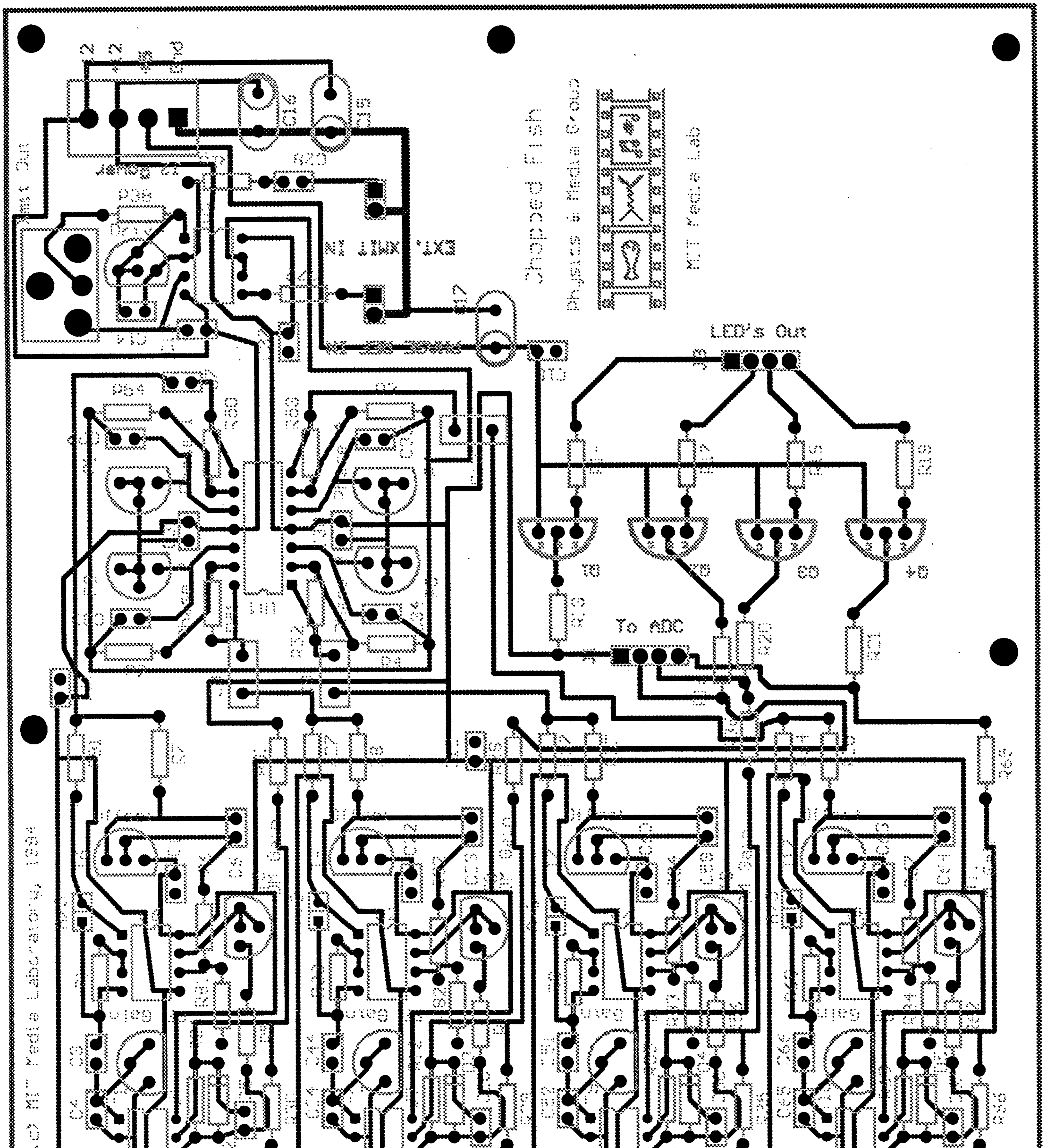
J1
L2
L1
S In
+5 V
Gnd
R20
C7
U6
555
C6
C5
R21
Display Driver
R3
C4
R4
C3
HT-684L
U3
R2
C1
R1
C2
U2
U1
7447
U4
7447
LED5
R23
Tempo #2
R7
R8
R9
R10
R11
LED3
R12
R13
R14
R15
R16
R17
R18
R22
R6
R5
LED1
LED2
LED4
ULN2003
U5
Tempo #1
Physics & Media Group
(c) MIT Media Lab
J. Paradiso
September, 1994

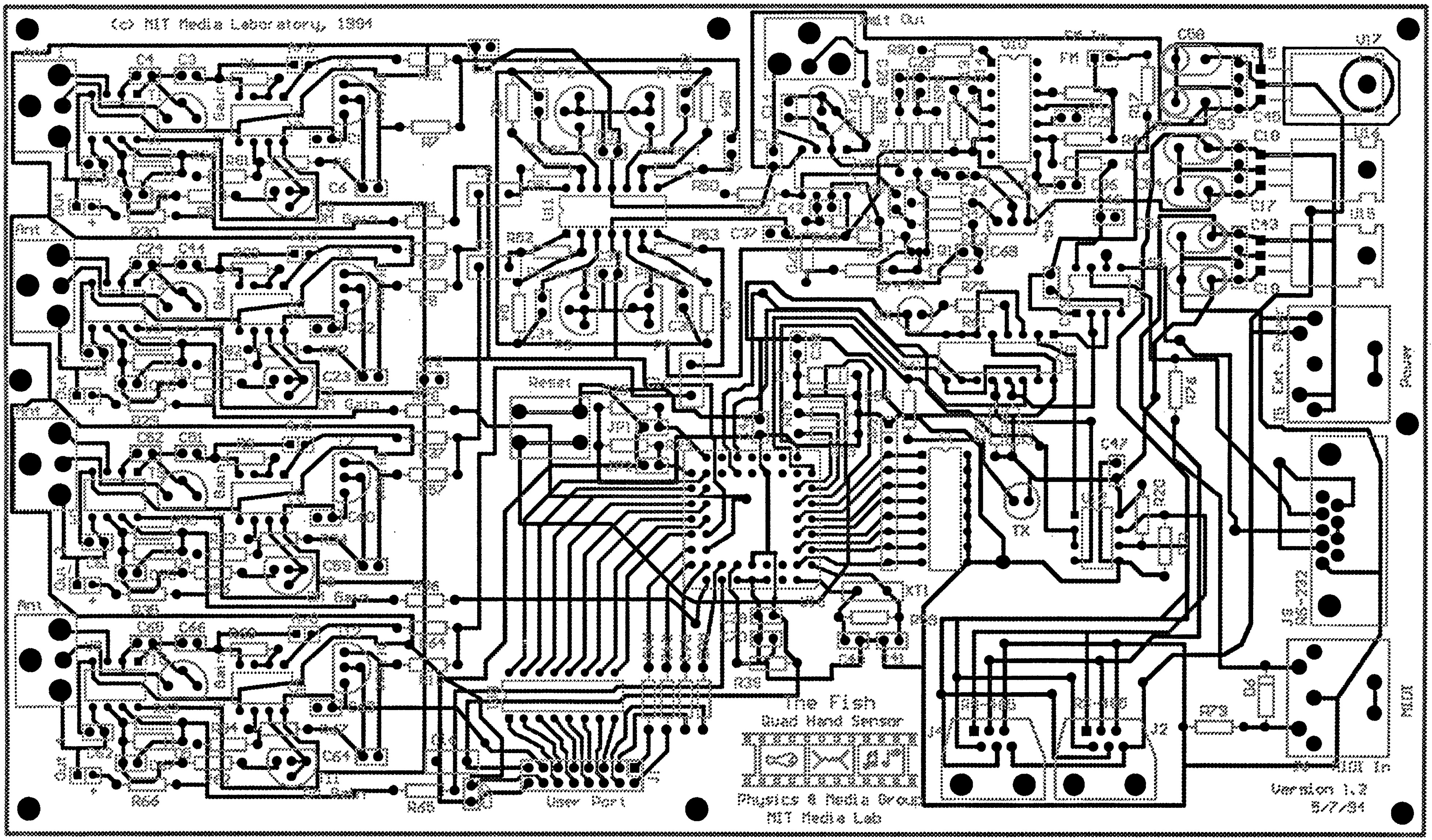

(c) MIT Media Lab September, 1994
U1
74C922
R2
R3
R5
U2
U3
555
C1
C2
Q1
J1
J2
Out
Gnd +5
Y1 Y2 X4 X3 X2 X1
Keypad Xmitter
J. Paradiso

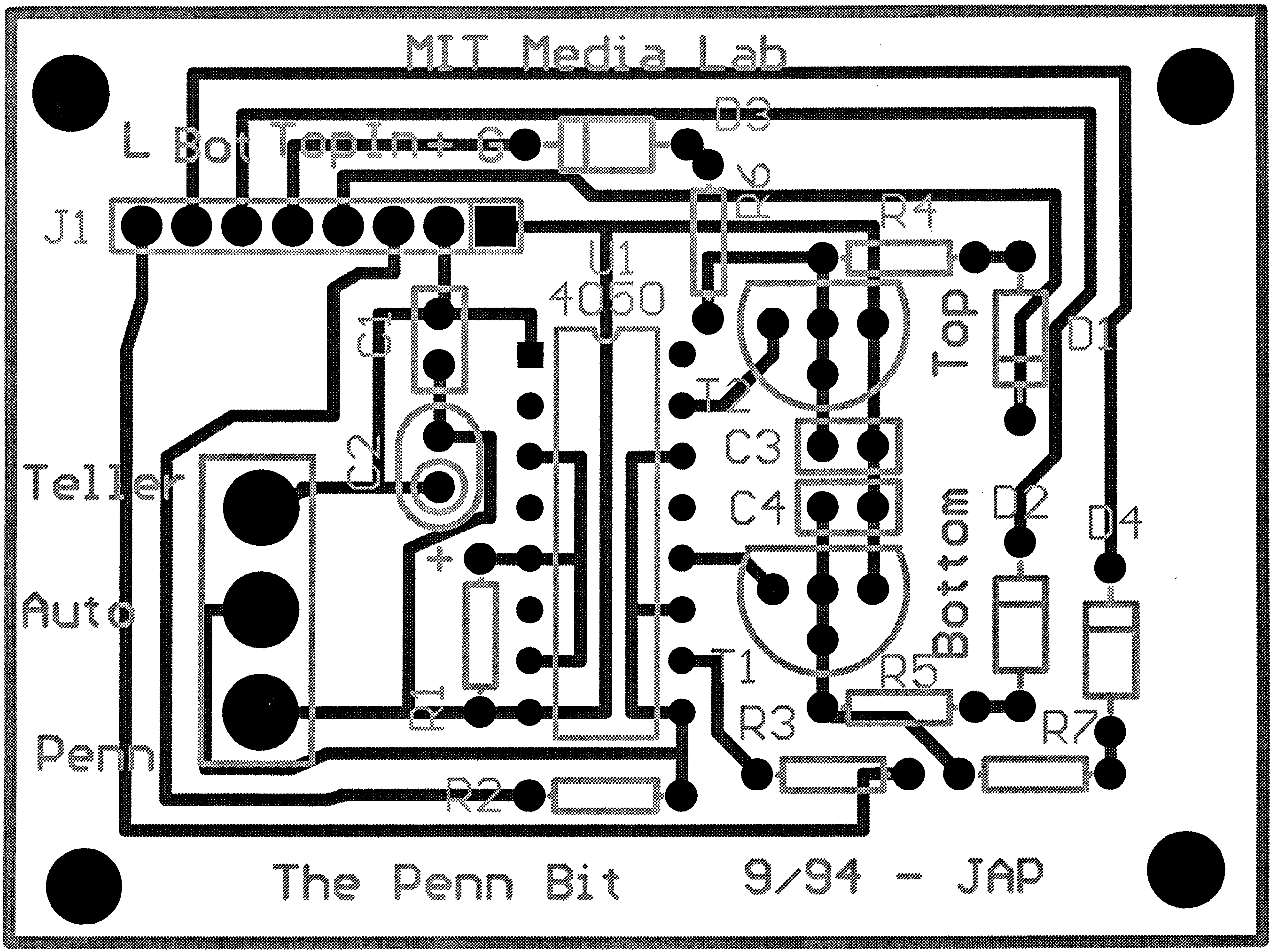


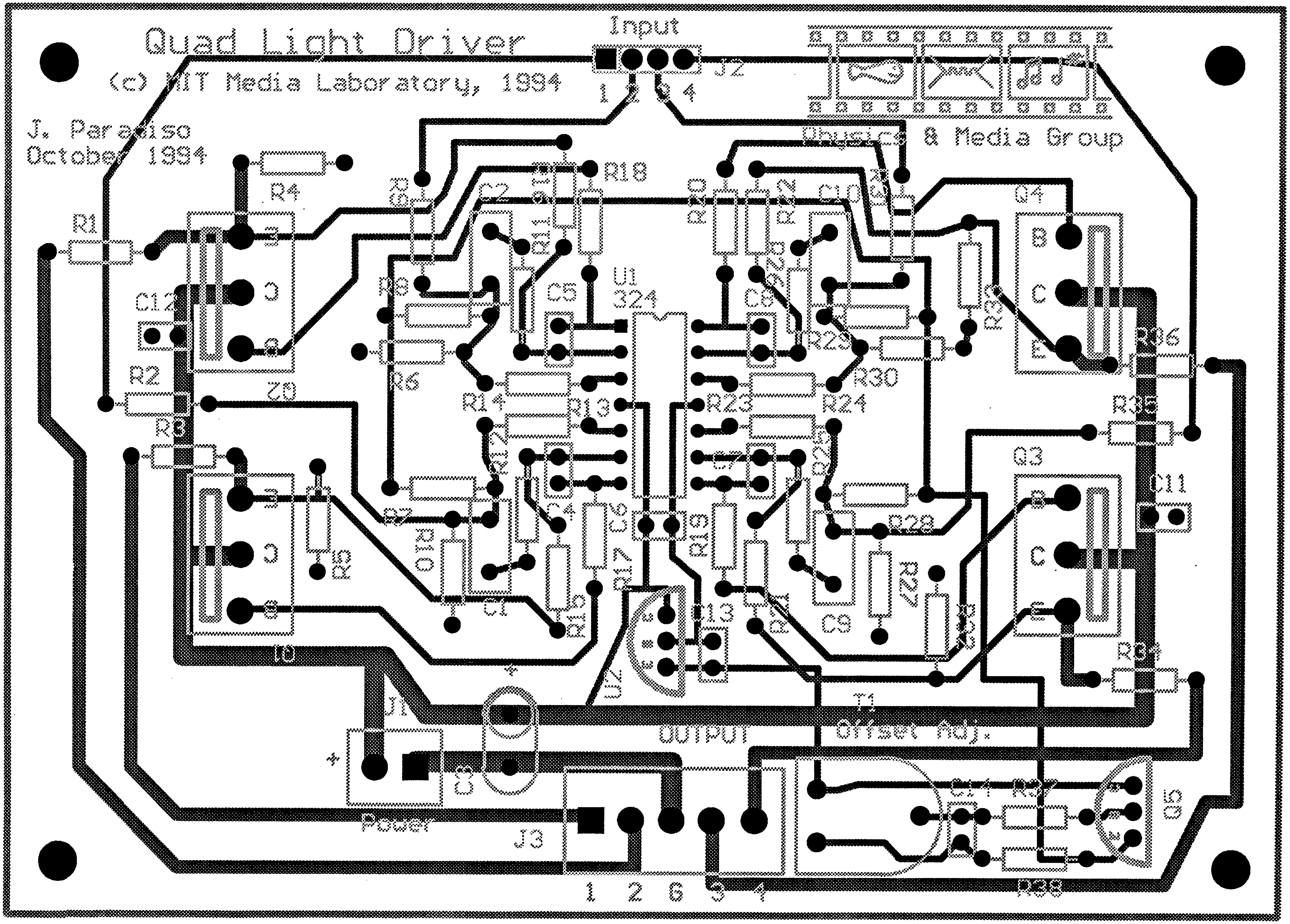
Octal Front-End Driver
J. Paradiso, 11/94
(c) MIT Media Lab
Physics and Media Group
1-4 Out
J5
5-8 Out
J6
J8
LED
1/2 In
3/4 In
5/6 In
7/8 In
+12
-12
Power
U1
AC713
U2
AC713
U3
AD713
U4
AD713
R7
R8
R14
R13
R19
R20
R28
R33
D1
D2
D3
D4


(c) MIT Media Laboratory, 1994
Version 1.0
J2
Output
Quad Log Amplifier
Physics & Media Group
J. Paradiso
September 1994
Gain 1
Gain 2
Gain 3
Gain 4
Input Bias
Out Bias
J1
Input
J3
Power
J4
LED Output
-12
+12


LED's Out
To ADC


(c) MIT Media Laboratory, 1994
The Fish
Quad Hand Sensor
Physics & Media Group
MIT Media Lab
Version 1.2
5/7/94
User Port
Reset
Xmit Out
RS-232
MIDI In
MIDI
Power
Ant 1
Ant 2


MIT Media Lab
L Bot Top In + G
J1
D3
R6
R4
U1
4050
Top
D1
C1
C2
T2
C3
C4
Teller
Auto
Penn
+
R1
Bottom
D2
D4
T1
R5
R3
R7
R2
The Penn Bit
9/94 - JAP


Quad Light Driver
(c) MIT Media Laboratory, 1994
J. Paradiso
October 1994
Input
Physics & Media Group
OUTPUT
Offset Adj.
Power

# Custom Fish Software

(Josh Smith)

From daemon Fri Oct 14 10:25:50 1994
Received: by media.mit.edu (5.57/DA1.0.4.amt)
id AA19192; Fri, 14 Oct 94 10:25:47 -0400
From: Joshua R. Smith <jrs@media.mit.edu>
Message-Id: <9410141425.AA19192@media.mit.edu>
Subject: Re: Codes
To: joep@media.mit.edu (Joe Paradiso)
Date: Fri, 14 Oct 1994 10:25:46 -0400 (EDT)
Cc: pnt-magic@media.mit.edu
In-Reply-To: <9410140346.AA22367@media.mit.edu> from "Joe Paradiso" at Oct 13, 94 11:46:53 pm
X-Mailer: ELM [version 2.4 PL23]
Content-Type: text
Content-Length: 736
Status: OR

>
>
> Josh: could you send out an update of this message with all the latest
> (hopefully final!) controller numbers? Thanx -Joe-
>
>

Joe et al.,
Here is the final info on controller numbers & values for the
P&T Seance Box:

| Numbers | Description | | Values |
|---|---|---|---|
| ======= | ====================== | | ========= |
| 21-28 | raw sensor values from fish | 0-127 | |
| 29-36 | reserved for velocities | | * (velocities no longer sent) |
| 37-42 | pushbuttons | | 0 = released, 127 = pressed |
| 43-50 | lights | | 0-127 |
| 51-52 | tempo LEDs | | 0 = off, 1-127 = on |
| 53 | display | | 0-99 ( > 99 maps to 99 ) |
| 54 | Penn bit | 0 = off, 1-127 = on | |
| 55 | light mode | | 0 = all lights auto |
| | | | 1 = hand lights MIDI controlled |
| | | | 2 = foot lights MIDI controlled |
| | | | 3 = hand & foot lights MIDI |

```
* MIDI velocity
*            23..Check two bits of light auto/MIDI controller value to
*                set hand lights to auto/MIDI separately from feet; took out vel stuff
*            22..Shorten display time out constant; flash lights on initialization
*                fix error in remapping foot sensors to side LEDs
*            21..Add 2 controller numbers to send raw hex to each digit of display
*            20..Switch left & right digits; use lights 7 & 8 in auto mode
*                lengthened time constant for display update with timeout counter;
*                Fish out on MIDI chan 1 (as before); Fish in on 16
*            19..Fix pushbutton controller number problem; on -> $7F, not $0F
*                fix display digit problems (which digit, & timing)
*            18..Running status on output
*            17..Display output every 10th sec
*            16..Light mode (automatic/MIDI controlled) switchable by MIDI
*            15..Working MIDI in...fixed Tempo LEDs
*            14..Test real MIDI in...31.250K BAUD
*            13..Debug version...9600 BAUD
*            12..MIDI in, real parsing...31.250 K BAUD
*            11..MIDI in...ASCII version with debugging output, 9600 BAUD
*             9...read serial port via interrupts...store in circular buffer
*             7...use procedures...fix bank problems...
* Version 6...send both A/D banks
* Version 4...Use timer hardware interrupts
* Version 3...COP stuff removed
* Single chip EEPROM version
        ORG     $0000           ; RAM values
SWITCH  RMB     1
MYADR1  RMB     1
MYADR2  RMB     1
MYADR3  RMB     1
MYADR4  RMB     1
MYADR5  RMB     1
MYADR6  RMB     1
MYADR7  RMB     1
MYADR8  RMB     1
PREV1   RMB     1
PREV2   RMB     1
PREV3   RMB     1
PREV4   RMB     1
PREV5   RMB     1
PREV6   RMB     1
PREV7   RMB     1
PREV8   RMB     1
VEL1    RMB     1
VEL2    RMB     1
VEL3    RMB     1
VEL4    RMB     1
VEL5    RMB     1
VEL6    RMB     1
VEL7    RMB     1
VEL8    RMB     1
Fout    RMB     1       ; Fish out change:    controller value change on ch 1
```

```
Fin      RMB   1       ; Fish in change :    controller value change on ch 16
LMODE    RMB   1       ; Control lights automatically?
LMODEW   RMB   1       ; Working value of LMODE (indicates current bank)
PBOLD    RMB   1       ; Old pushbutton state
PB       RMB   1       ; New pushbutton state
STLED    RMB   1       ; State of Tempo LEDs
STPARS   RMB   1       ; State of MIDI parse automaton.
BUFSZ    RMB   1       ; Current size of buffer
CNUM     RMB   1       ; Controller number
DISLO    RMB   1       ; Low byte of display
DISHI    RMB   1       ; High byte of display
DISFLG   RMB   1       ; Flag indicating which digit of display to write to next
DISCNT   RMB   1       ; Display timer counter
* Two byte values
T1INT    RMB   2       ; Timer 1 interval
T2INT    RMB   2       ; Timer 2 interval
HEAD     RMB   2       ; Head of input buffer... reserve 2 bytes to simplify
TAIL     RMB   2       ; Tail of input buffer... indexing using 16 bit regs X & Y
B1       RMB   1       ; safety margin
BUFLO    RMB   64T     ; Reserve 64 byte buffer
BUFHI    EQU   *
B2       RMB   1       ; safety margin
         ORG   $F800
*        Register defines
REGBAS   EQU   $1000           ; Base addr of register block
PORTA    EQU   REGBAS+$00
PORTC    EQU   REGBAS+$03
PORTB    EQU   REGBAS+$04
OC1M     EQU   REGBAS+$0C
TCNT     EQU   REGBAS+$0E
TOC1     EQU   REGBAS+$16      ; Timer 1 output compare
*                  +$17
TOC2     EQU   REGBAS+$18      ; Timer 2 output compare
*                  +$19
TCTL1    EQU   REGBAS+$20
TMSK1    EQU   REGBAS+$22
TFLG1    EQU   REGBAS+$23
TMSK2    EQU   REGBAS+$24
BAUD     EQU   REGBAS+$2B
SCCR1    EQU   REGBAS+$2C
SCCR2    EQU   REGBAS+$2D
SCSR     EQU   REGBAS+$2E
SCDR     EQU   REGBAS+$2F
ADCTL    EQU   REGBAS+$30      ; A-to-D control
ADR1     EQU   REGBAS+$31      ; A-to-D result
ADR2     EQU   REGBAS+$32      ; A-to-D result
ADR3     EQU   REGBAS+$33      ; A-to-D result
ADR4     EQU   REGBAS+$34      ; A-to-D result
OPTION   EQU   REGBAS+$39
COPRST   EQU   REGBAS+$3A
CONFIG   EQU   REGBAS+$3F
* Controller numbers
```

```
SCODE    EQU     21T              ; Sensors      These 2 are
PBCODE   EQU     37T              ; Pushbuttons  MIDI out.
LCODE    EQU     43T              ; Lights       Last 3 are
TCODE    EQU     51T              ; Tempo LED    MIDI in.
DCODE    EQU     53T              ; Display (2 digit)
PENN     EQU     54T              ; Is Penn in the booth?
MCODE    EQU     55T              ; MIDI controlled light mode? (0 == automatic; 1 == MIDI)
DLCODE   EQU     56T              ; Display (low digit raw hex)
DHCODE   EQU     57T              ; Display (high digit raw hex)
MINCNT   EQU     43T              ; Minimum controller value we must look at
MAXCNT   EQU     57T              ; Max controller value
ACK      EQU     58T              ; Acknowledge... use for MIDI debugging
SCIVECT EQU      $FFD6
INIT
         SEI              ; Disable interrupts during initialization
         LDS     #$00FF   ; Init stack
         LDAA    #$83     ; Turn on A-to-D, set COP to long delay
         STAA    OPTION
         JSR     INITSW   ; Read switches right away
* Initialize serial port
         LDAA    #$20     ; Init SCI 31.250 K baud
         STAA    BAUD
         LDAA    #$00
         STAA    SCCR1
         LDAA    #%00101100       ; Receive interrupt enable, receive enable, xmit enable
         STAA    SCCR2
* Initialize MIDI in buffer
         CLRA             ; Initialize circular queue used to hold MIDI in
         STAA    BUFSZ
         LDAB    #BUFLO   ; Use 2 byte pointers for head & tail
         XGDX             ; When no chars in buffer = HEAD - TAIL = 0
         STX     HEAD
         STX     TAIL
         CLR     STPARS
* Initialize pushbuttons
         LDAA    PORTA
         ANDA    #$07
         STAA    PBOLD
         STAA    PB
         LDX     #$0002
FLASHLP
* Turn lights on, then off
         LDAB    #$88
LOFFLP   STAB    PORTB
         LDAA    #$7F
         STAA    PORTB
         JSR     WAITFF
         STAB    PORTB
         CLR     PORTB
         INCB
         CMPB    #$8F
         BLS     LOFFLP
```

```
* Flash tempo LEDs on
        LDAB    #$80
        STAB    PORTB
        LDAA    #$01
        STAA    PORTB
        JSR     WAITFF
        STAB    PORTB
        LDAA    #$02
        STAA    PORTB
        JSR     WAITFF
        STAB    PORTB
        NOP
        CLR     PORTB
        CLR     STLED    ; Record that state of LEDs is 0
* Flash Penn bit
        LDAA    #$FF
        STAA    PORTA
        JSR     WAITFF
* Initialize Penn bit
        CLR     PORTA    ; Clear Penn bit, etc
        DEX
        BNE     FLASHLP
* Initialize display to 00
        LDAA    #$00
        STAA    DISHI
        STAA    DISLO
        STAA    DISFLG
        STAA    DISCNT
* Initialize MIDI light mode
        CLR     LMODE    ; Lights under automatic control initially
* Initialize timers
        LDAA    TMSK2
        ANDA    #%11111100
        ORAA    #%00000001
        STAA    TMSK2    ; Set 1 timer count = 2 usecs; timer range = .13 s
* Set timer 1
        LDD     TCNT     ; Prepare first timeout
        ADDD    T1INT    ; This value already set by call to INITSW, dipswitch reading routine
        STD     TOC1
        LDAA    #$80     ; Clear any pending OC1F flag
        STAA    TFLG1
* Set timer 2
        LDX     #$FFFF   ; Set interval for 2nd timer to max.
        STX     T2INT
        LDD     TCNT     ; Prepare first timeout
        ADDD    T2INT
        STD     TOC2
        LDAA    #$40     ; Clear any pending OC2F flag
        STAA    TFLG1
        CLI              ; Enable interrupts
*****************************************************************
* Event loop: wait for a dipswitch event or timeout             *
```

```
*****************************************************************
EVENTL
        JSR     READSW  ; Check switches and deal with them if necessary.
        BSR     READAD  ; Read ADC and do any desired filtering.
        JSR     PARSMID ; Parse MIDI
        LDAA    TFLG1   ; Check time out interrupt flags
        BITA    #$80
        BNE     TIME1OUT; Timer 1
        BITA    #$40
        BNE     TIME2OUT; Timer 2
        BRA     EVENTL
*****************************************************************
* Handle timeout event for timer 2
*****************************************************************
TIME2OUT
        LDD     TCNT    ; Prepare next timeout
        ADDD    T2INT
        STD     TOC2
        LDAA    #%01000000
        STAA    TFLG1   ; Clear output compare 2 flag.
        INC     DISCNT  ; Increment timeout counter
        CMPA    #$01    ; Not ready to write to display? set to 1 now...
        BLO     ENDT2   ;  exit
        CLR     DISCNT  ;  else clear timeout counter & then
        LDAA    #$84    ; Send write-to-display command
        STAA    PORTB
        LDAA    DISFLG
        EORA    #$01    ; Toggle display flag
        STAA    DISFLG
        BEQ     T2DISHI
T2DISLO LDAA    DISLO
        BRA     SETDIS
T2DISHI LDAA    DISHI
        ADDA    #$10
SETDIS  STAA    PORTB
ENDT2   BRA     EVENTL
* end timeout handler 2
*****************************************************************
*****************************************************************
* Handle timeout event for timer 1
*****************************************************************
TIME1OUT
        LDD     TCNT    ; Prepare next timeout
        ADDD    T1INT
        STD     TOC1
        LDAA    #%10000000
        STAA    TFLG1   ; Clear output compare 1 flag.
        JSR     SNDCHANS
        BSR     PBHNDLR
        BRA     EVENTL
* end timeout handler 1
*****************************************************************
```

```
*****************************************************************
* Read all 8 AD channels & store results in MYADR1-8
* do any signal processing that must happen every cycle
*****************************************************************
READAD
        LDAA    #$10     ; ADC read in mult mode, chan group 1 ($14= ch2)
        STAA    ADCTL    ; Start read...this gets all channels at once
        LDAA    #$80
ADWAIT1 BITA    ADCTL
        BPL     ADWAIT1 ; Loop until ADC read finished
        LDAA    ADR1
        STAA    MYADR1
        LDAA    ADR2
        STAA    MYADR2
        LDAA    ADR3
        STAA    MYADR3
        LDAA    ADR4
        STAA    MYADR4
        LDAA    #$14     ; ADC read in mult mode, chan group 2 ($10= ch1)
        STAA    ADCTL    ; Start read...this gets all channels at once
        LDAA    #$80
ADWAIT2 BITA    ADCTL
        BPL     ADWAIT2 ; Loop until ADC read finished
        LDAA    ADR1
        STAA    MYADR5
        LDAA    ADR2
        STAA    MYADR6
        LDAA    ADR3
        STAA    MYADR7
        LDAA    ADR4
        STAA    MYADR8
* Now do any data processing that must be done everytime through
        LDX     #$0000
FILTLP  LDAA    MYADR1,X
        LSRA             ; Could just do this at output time, but in general
        STAA    MYADR1,X; want to do filtering everytime through cycle
        INX
        CPX     #$0008
        BLS     FILTLP
        RTS
* end READAD
*****************************************************************
*****************************************************************
* Send pushbutton values
*****************************************************************
PBHNDLR
        LDAA    PORTA
        ANDA    #$07     ; Mask all but lower 3 bits
        STAA    PB
        CMPA    PBOLD    ; Has state of switch changed?
        BEQ     ENDPB    ; If no, leave
        LDAB    PBOLD    ; Check prev PB state
```

```
        CMPB    #$07     ; All off before?
        BEQ     CONTON
        LDAA    Fout     ; Some on before; send switch off
        JSR     OUTSCI
        LDAA    PBOLD    ; Get prev PB state
        ADDA    #PBCODE ; Channel
        JSR     OUTSCI
        LDAA    #$00     ; off
        JSR     OUTSCI
        LDAA    PB       ; Is new state off?
        CMPA    #$07     ; If so, don't send an on
        BEQ     PBMOV
CONTON  LDAA    Fout     ; Send new switch on
        JSR     OUTSCI
        LDAA    PB       ; Get new PB value
        ADDA    #PBCODE  ; Channel
        JSR     OUTSCI
        LDAA    #$7F     ; on
        JSR     OUTSCI
PBMOV   LDAA    PB
        STAA    PBOLD
ENDPB   RTS
* end PBHNDLR
*******************************************************************
*******************************************************************
* Send commands
*
*******************************************************************
SNDCHANS
        LDY     #$0000   ; Calculate all velocities first
SUBLP   LDAA    MYADR1,Y
        SUBA    PREV1,Y
        STAA    VEL1,Y
        INY
        CPY     #$07
        BLS     SUBLP
        LDY     #$0000
CHGLP   LDAA    VEL1,Y   ; If all velocities 0, we will send nothing
        BNE     NONZERO ; found a nonzero vel? Go send a MIDI message
        INY
        CPY     #$07
        BLS     CHGLP
        BRA     SNDRTN   ; Nothing has changed... leave now
NONZERO LDAA    Fout     ; Need a change, so send initial control change cmd
        JSR     OUTSCI   ; on chan#... using running status mode
        LDY     #$0000
        LDAB    #21T     ; Load in lowest controller value
CHLOOP1 LDAA    LMODE    ; Set LMODE indication for this bank
        ANDA    #$01     ; bank 1
        STAA    LMODEW
        JSR     SENDACHAN
        INY
```

```
        INCB              ; Inc and do remaining channels
        CMPB    #25T      ; Low sensor bank
        BLO     CHLOOP1
*                         ; B and Y already set to #25T and #$0004
CHLOOP2 LDAA    LMODE     ; Set LMODE indication for this bank
        ANDA    #$02      ; bank 2
        STAA    LMODEW
        JSR     SENDACHAN
        INY
        INCB              ; Inc and do remaining channels
        CMPB    #29T      ; High sensor bank
        BLO     CHLOOP2
        LDAA    LMODE     ; If high bank is in auto mode, send extra light commands
        ANDA    #$02      ; specifies high bank
        BNE     SNDRTN
CH5     LDAA    #$8C      ; Channel 7 (left foot) -> light 5 (left LED)
        STAA    PORTB     ; (note: channel 6 is back of chair; ch 5 unused)
        LDAA    MYADR7
        STAA    PORTB
CH6     LDAA    #$8D      ; Channel 8 (right foot) -> light 6 (right LED)
        STAA    PORTB
        LDAA    MYADR8
        STAA    PORTB
SNDRTN  RTS
* PRE: A contains LMODE indication for this bank; B contains controller number
SENDACHAN
        LDAA    VEL1,Y
        BEQ     CONT      ; If no change, don't send output
        ADDA    #$40      ; Map -64 -> 0, 0 -> 64, 63 -> 127
        ANDA    #$7F      ; Clear high bit just in case
        STAA    VEL1,Y
RAW     TBA
        JSR     OUTSCI    ; Controler # for raw value
        LDAA    MYADR1,Y
        JSR     OUTSCI    ; Control value
LIGHTS  LDAA    LMODEW    ; Get LMODE for this bank
        BNE     REFRESH   ; Skip this if in MIDI controlled mode
        TBA
        ADDA    #$73      ; (#21 == #$15) + #$73 = #$88
        STAA    PORTB
        LDAA    MYADR1,Y
        STAA    PORTB
REFRESH LDAA    MYADR1,Y
        STAA    PREV1,Y
CONT    RTS
OUTSCI  PSHB
OUTSCIL LDAB    SCSR
        BITB    #$80
        BEQ     OUTSCIL
        STAA    SCDR
        PULB
        RTS
```

```
MOUTSCI PSHB
MOUTSCL LDAB    SCSR
        BITB    #$80
        BEQ     MOUTSCL
        STAA    SCDR
        PULB
        RTS
********************************************************************
* Two entry points for switch reading routine: one conditional
* (READSW) and one unconditional (INITSW).  READSW exits if the
* switch state has not changed.  INITSW doesn't check...this is
* called during initialization to set timer constants.
********************************************************************
READSW  LDAA    PORTC   ; Check switches
        CMPA    SWITCH  ; have they changed?
        BEQ     ENDSW   ; No, leave here
INITSW  JSR     WAIT1MS ; Yes: wait for keys to stop bouncing,
        LDAA    PORTC   ; read switches again, and
        STAA    SWITCH  ; save them for later.
DEVMASK
* MIDI channels are stored in RAM locns so they can be changed
* dynamically (ie by DIP switch changes or MIDI commands).  For
* P&T these are hardcoded.
        LDAA    #$B0    ; Control change on channel 1: Fish MIDI output
        STAA    Fout
        LDAA    #$BF    ; Control change on channel 16:          Fish MIDI input
        STAA    Fin
        LDAA    SWITCH
        ANDA    #%01100000 ; Check out bits 5 and 6
MS1     CMPA    #%00000000
        BNE     MS10
        LDX     #500T
        BRA     SETIME
MS10    CMPA    #%00100000
        BNE     MS20
        LDX     #5000T
        BRA     SETIME
MS20    CMPA    #%01000000
        BNE     MS40
        LDX     #10000T
        BRA     SETIME
MS40    LDX     #20000T ; default
SETIME  STX     T1INT
ENDSW   RTS
WAIT1MS PSHX            ; Save X
        LDX     #$0535  ; 1333 * 6~ * 125ns/~ = 1ms
WAIT1L  DEX             ; loop = 6~
        BNE     WAIT1L
        PULX
        RTS
WAITFF
        PSHX            ; Save X
```

```
            LDX     #$FFFF    ;
WTFFL       DEX               ; loop = 6~
            BNE     WTFFL
            PULX
            RTS
* Pre: HEAD points to next legal place in buf to write to *
*       TAIL points to next place to delete from *
SCIIN
*           LDAA    #20T      ; *DBG
*           JSR     MOUTSCI
            LDX     HEAD      ; Prepare X reg to point to buffer
WAITSCI LDAB        SCSR      ; Read in data from SCI
            ANDB    #$20      ; This read and the one below
            BEQ     WAITSCI ;  of SCDR clear the
            LDAA    SCDR      ;   SCI interrupt flag.
            STAA    0,X       ; Write byte to locn pointed to by HEAD
            BSR     INCHEAD ; Increment HEAD; wraparound & purge if necessary
ENDSCI      RTI
INCHEAD INC         BUFSZ
            INX               ; Increment HEAD pointer
            CPX     #BUFHI    ; Did HEAD wrap around?
            BLS     SAVEHD    ;  if not, continue
WRAP        LDX     #BUFLO    ;  if so, perform wrap HEAD around
SAVEHD      STX     HEAD      ; Save new HEAD value, either incremented or wrapped.
            CPX     TAIL      ; Did head catch tail?
            BNE     INCHEND ;  if not, done with INC
PURGE       LDX     TAIL      ;  if so, increment tail to purge a byte
            INX
            CPX     #BUFHI    ; Did TAIL wrap around?
            BLS     SAVETL    ;  if not, done with INC
            LDX     #BUFLO    ;  if so, wrap TAIL around
SAVETL      STX     TAIL
INCHEND RTS
* This is only used for debugging.
OUTBUF      LDX     #$0000
OUTL        LDAA    BUFLO,X
            JSR     OUTSCI
            INX
            CPX     #32T
            BLE     OUTL
            RTS
* PRE: BUFSZ > 0; X points to next byte to be pulled off
* POST: A holds byte; X points to next byte
PULBYTE DEC         BUFSZ
            LDAA    0,X
            INX               ; Move tail pointer
            CPX     #BUFHI    ; Did tail wrap around?
            BLS     ENDPUL    ;   if not, exit
            LDX     #BUFLO    ;   if so, wrap around
ENDPUL      STX     TAIL
            RTS
*************************************************************
```

```
* Parse MIDI
* accepts the language CCHNG (CNUM CVAL)*
*
*          H   H
*        /------\-------\
* |          |          |
*  v       C |       N |        V
* S0  --> 1  --> 2  --> F3
*              /^\_______________/
*              \_/        e,C
*             C
*
* C: control Change command
* N: controller Number
* V: controller Value
*************************************************************
PARSMID
         SEI                 ; Don't interrupt during this routine!
         LDX      TAIL       ; Point X at tail of MIDI buffer
PMID2    CPX      HEAD       ; If nothing in buffer,   (could check bufsz instead)
         BEQ      ENDPM      ; then leave, else
         LDAA     STPARS
         BEQ      ST0        ; State 0
         CMPA     #$01
         BEQ      ST1
         CMPA     #$02
         BEQ      ST2
         BRA      PMID2      ; should never reach this point
ENDPM
         CLI                 ; re-enable interrupts
         RTS
ST0
         BSR      PULBYTE
         CMPA     Fin        ; A command for us??
         BNE      PMID2      ; If not, go back in same state
         INC      STPARS     ; If so, increment state indicator
         BRA      PMID2      ; and continue parsing
ST1
         BSR      PULBYTE
         CMPA     Fin
         BEQ      PMID2      ; already in state 2 [1 ??]
         BITA     #$80       ; Did we receive data (high bit clear)?
         BNE      GOST0      ;   if not, go back to state 0
         STAA     CNUM       ;   if so, interpret as controller number and save
         INC      STPARS     ;      set state var to 2
         BRA      PMID2
GOST0    CLR      STPARS     ; Reset state
         BRA      PMID2
GOST1    LDAA     #$1         ; Set state to 1
         STAA     STPARS
         BRA      PMID2
ST2
```

```
        BSR     PULBYTE
        CMPA    Fin       ; Got an f?
        BNE     NOTf
        LDAA    #$1       ; If so, go back to state 1
        STAA    STPARS
        BRA     PMID2
NOTf
        BITA    #$80      ; Did we receive data (high bit clear)?
        BNE     GOST0     ;   if not, go back to state 0
*                         ;   if so, interpret as controller value
* Accept State!!
        PSHA              ; A holds controller value
        LDAA    #$1       ; After an accept, we will go back to state 1
        STAA    STPARS
        PULA
        LDAB    CNUM
        CMPB    #MINCNT ; Is controller number < lowest controller?
        BLO     GOST1     ;   if so, go back; else
        CMPB    #MAXCNT ; Is controller number > greatest controller?
        BHI     GOST1     ;   if so, go back
        CMPB    #TCODE    ;   else we are dealing with a valid controller#
        BLO     LIGHTIN   ; less than tempo LED ==> lights
        CMPB    #DCODE    ;
        BLO     TEMPO     ; less than display ==> tempo LEDs
        CMPB    #PENN
        BLO     DISPLAY ; less than penn ==> display
        CMPB    #MCODE
        BLO     PPENN     ; less than mcode ==> penn
        CMPB    #DLCODE
        BLO     MMCODE    ; less than dlcode ==> mcode
        CMPB    #DHCODE
        BLO     DDLCODE   ; less than dhcode ==> dlcode
*                         ; else ddhcode
DDHCODE STAA    DISHI     ; Put controller val straight into high digit
        JMP     PMID2     ; ...don't mess with BCD
DDLCODE STAA    DISLO     ; Put controller val straight into low digit
        JMP     PMID2     ; not BCD
MMCODE  STAA    LMODE
        JMP     PMID2
PPENN
        CMPA    #$00
        BEQ     POFF
PON     LDAA    #$20      ; Turn on Penn bit, PA5
        STAA    PORTA
        JMP     PMID2
POFF    CLR     PORTA     ; Turn off Penn bit
        JMP     PMID2
DISPLAY
        CMPA    #99T
        BHI     SET99
        TAB               ; Transfer controller value to B
        CLRA
```

```
        PSHX                    ; Save X
        LDX     #10T
        IDIV
        PSHB                    ; push remainder
        PSHX                    ; push quotient
        PULA                    ; throw away MSB of quotient
        PULA                    ; get LSB of quotient
        STAA    DISHI           ; save LSB of quotient
        PULA                    ; restore remainder
        STAA    DISLO
        PULX                    ; restore X
        BRA     DISEND
SET99   LDAA    #$09            ; BCD 99
        STAA    DISHI
        STAA    DISLO
DISEND
        JMP     PMID2
TEMPO
*       PSHA                    ; *DBG
*       LDAA    #12T
*       JSR     MOUTSCI
*       PULA
        CMPA    #$00
        BEQ     LEDOFF
LEDON   LDAA    #$01            ; Any non zero value ==> led on
        BRA     WLED
LEDOFF  CLRA                    ; value = 0 ==> led off
WLED    CMPB    #TCODE          ; Which LED was addressed?
        BEQ     LED1
LED2    ASLA                    ; Shift on/off bit up to address LED2
        LDAB    #%11111101 ; Mask off bit 2
        BRA     LEDOUT
LED1    LDAB    #%11111110 ; Mask off bit 1
LEDOUT  ANDB    STLED
        STAB    STLED
        ADDA    STLED           ; We zeroed relevant bit, so won't be any carry
        LDAB    #$80            ; Send Tempo LED command
        STAB    PORTB
        STAA    STLED           ; Save state of LED
        STAA    PORTB           ; Set new state of light
        JMP     PMID2
LIGHTIN
*       PSHA                    ; *DBG
*       LDAA    #13T
*       JSR     OUTSCI
*       PULA
        ADDB    #$5D            ; $5D + $2B (43D) = $88
        STAB    PORTB
        NOP
        STAA    PORTB
        JMP     PMID2
***************************************************************
```

```
* Debugging
DOUTSP  LDAA    #$20
        BRA     DOUTSCI
DOUTCR  LDAA    #$DD     ; a hack...this becomes $0d
DOUTDEC ADDA    #$30
DOUTSCI PSHB
DOUTSCL LDAB    SCSR
        BITB    #$80
        BEQ     DOUTSCL
        STAA    SCDR
        PULB
        RTS
BTOD    CLRA
        LDX     #100T    ; 100 *DECIMAL*
        IDIV
        PSHB             ; push remainder
        PSHX             ; push quotient
        PULA             ; throw away MSB of quotient
        PULA             ; output LSB of quotient
        JSR     DOUTDEC ; trashes B
        CLRA
        PULB
        LDX     #10T
        IDIV
        PSHB             ; push remainder
        PSHX             ; push quotient
        PULA             ; throw away MSB of quotient
        PULA             ; output LSB of quotient
        JSR     DOUTDEC
        PULB             ; restore remainder
        TBA              ; output remainder
        JSR     DOUTDEC
        RTS
*SCI vector
        ORG     $FFD6    ; Point SCI interrupt vector to our SCI input routine
        FDB     SCIIN
* Reset vector
        ORG     $FFFE
        FDB     $F800
```