NWRI RSB 00-001

Environment Canada Environnement Canada

Canada

CCIW
FEB 6 2001
LIBRARY

TIME LAPSE VIDEO LOGGER

TECHNICAL DOCUMENTATION

R. Desrosiers and N. Rukavina
National Water Research Institute
Burlington, Ontario L7R 4A6

TD
226
N89
no. 00-001

TIME LAPSE VIDEO LOGGER

TECHNICAL DOCUMENTATION

R. Desrosiers and N. Rukavina
National Water Research Institute
Burlington, Ontario L7R 4A6

NWRI Engineering Technical Report
No. RSB 00-001

Purpose/ Management Perspective

The purpose of this document is to summarize and collate available documents for the Time Lapse Video Logger (TLVL) to make it easier for users of the apparatus to find information for operation and maintenance, for revisions to the system or to aid in the construction of new systems.

Documentation is presented both in printedform and as machine readable files. The latter are found in a CD-R disk which is attached to the inside end folder of the document. Files were created with software commonly used by Engineering at NWRI in the year 2000. Some files such as the Quicktime movies are only available in machine readable form.

NOTICE

Part of this document has re-worked from notes prepared by Mr. Dave Dunfield of Dunfield Development Systems, Nepean, Ontario, Canada under contract KW405-9-5807. (http://www.dunfield.com). This contract allowed the release of technical information and " ...use of the software\firmware\hardware of this system by NWRI or our clients..". The source code is released to fulfill this purpose. The compilers which take this software are not included in this documentation. These may be procurred from Dunfield Systems or the software may be revised by the user to suit the user's preferred compiler.

TABLE OF CONTENTS

1. SYSTEM INTRODUCTION

A1.1 The TLV Controller
A1.2 The HOST computer
A1.3 Operational Overview

A2. INSTALLING THE TLV SYSTEM

A2.1 Installing the controller
A2.2 Installing the TLV host software

A3. USING THE HOST SOFTWARE

A3.1 Manual Control
A3.2 Mission Viewer
A3.3 Mission Editor

B1. ELECTRONIC HARDWARE OVERVIEW

B2. DETAILED DESCRIPTIONS

2.1 Processing Section
2.2 Real Time Clock
2.3 LANC/Control-L Interface
2.4 Input Enable/Inhibit Signals
2.5 Output Control Signals
2.6 Serial Interface

B3. POWER CONSUMPTION

B4. CIRCUITS

B4.1 Microcontroller board circuit schematic
B4.2 Microcontroller board component placement
B4.3 Microcontroller PCB layout (top)
B4.4 Microcontroller PCB layout (bottom)
B4.5 Microcontroller board parts list

B5 INTERCONNECTIONS

B5.1 System wiring interconnection schematic
B5.2 Camera can internal wiring schematic

C1. COMMAND FUNCTION OVERVIEW

C2. DETAILED FUNCTION DESCRIPTIONS

C2.1 Main logic functions
C2.1.1 main() - Main program module
C2.1.2 wait_on_time() - Event processing
C2.1.3 camera_enabled() - Determine if the camera is enabled
C2.1.4 power_on() - Turn power on
C2.1.5 power_off() - Turn power off
C2.1.6 set_next_alarm() - Set time for next event
C2.1.7 process_command() - Host command processing
C2.2 Support functions
C2.2.1 gett() - Get character from serial port with timeout (***)
C2.2.2 geth() - Get a hex value from the serial port
C2.2.3 puth(v) - Write hex value to serial port
C2.2.4 CLfr() - Control-L/LANC frame exchange(***)
C2.2.5 lanc_cmd() - Send Control-L/LANC command
C2.2.6 event_checksum() - Calculate mission data checksum
C2.2.7 set_inhibit_mode() - Set inhibit testing flag
C2.2.8 read_time() - Read time from RTC device (***)
C2.2.9 set_time() - Set the RTC time(***)
C2.2.10 compare_date(date) - Compare complete date/time
C2.2.11 compare_time(s, m, h) - Compare time only
C2.2.12 set_alarm()- Set RTC alarm time (***)
C2.2.13 sleep() - Put processor to sleep until next alarm(***)
C2.2.14 alarm_irq() - RTC Alarm interrupt handler
C2.2.15 serial_irq() - Serial interrupt handler

(***) refers to assembly language functions- other functions programmed in C

C3. SERIAL COMMAND PROTOCOL

C3.1 I - Identify controller
C3.2 N - Power ON
C3.3 F - Power OFF
C3.4 Cxxxx - Execute standard LANC camera command
C3.5 Rxxxx - Execute repeated LANC camera command
C3.6 Oxx - Write control output
C3.7 Q - Query camera
C3.8 T - Query current time/date
C3.9 SssmmhhDDMMYY - Set time/date
C3.10 A - Query next alarm time
C3.11 Bssmmhh - Send next alarm
C3.13 D... - Download mission data
C3.14 Z5A - Restart controller

C4. Common Control-L/LANC command codes
C5. Common Control-L/LANC state codes
C6. Firmware Source code
C7a. PC Host Computer source code- C code

C7b. PC Host Computer source code - 'Help' file source code

D1 MECHANICAL HARDWARE

D2 CAMERA MODULE MECHANICAL SKETCHES

D2.1 Sony TR-7000 slide track
D2.2 Sony TR-7000 camera slide mount
D2.3 Camera can tube
D2.4 Camera housing front plate ring
D2.5 Camera housing rear end plate
D2.6 Camera can front port
D2.7 Camera can overall assembly sketch
D2.8 Camera can parts listing

D3 BATTERY CAN MECHANICAL SKETCHES

D3.1 Battery can rear end plate
D3.2 Battery can main tube
D3.3 Battery can front flange
D3.4 Battery can acrylic tube battery holder
D3.5 Battery can acrylic end plate
D3.6 Battery can front end plate
D3.7 Battery can wire spool, front contact holder and spring
D3.8 Battery can rear contact
D3.9 Battery can overall assembly sketch
D3.10 Battery can parts list

D4 BRACKETRY

D4.1 Mini SeaLite bracket
D4.2 Battery Can bracket
D4.3 Camera can bracket -adapter from T-frame to channel(part A)
D4.4 Camera can bracket - part B
D4.5 Camera can bracket- channel to camera (part C)

E1 PHOTGRAPHS

E2 STILL PHOTOS

E2.1 Photo of system on a frame in the hydraulics laboratory
E2.2 Photo of camera housing
E2.3 Photo of battery can
E2.4 Photo of Mini SeaLite
E2.5 Captured still image of a fish
E2.6 Photo of microcontroller board

E3 MOVIES

E3.1 Quicktime movie of fish in field of view (machine readable file only)
E3.2 Quicktime movie of bubbles in the field of view (machine readable file only)

F ACKNOWLEDGEMENTS

1. INTRODUCTION

A Time lapse video logger (TLVL) was developed initially as part of an effort to explain anomalies in the records of a logger which acoustically measures bottom depth. The system consists of several parts. There are to date three component assemblies mounted in separate submersible housings. These are a camera module which contains a Sony Digital8 camcorder and a microcontroller board, a battery can and a submersible light source.

A part of the system during setup is a surface mounted computer which relays setup information and mission parameters to the microcontroller in the camera housing. Also a video monitor mounted at the surface is used during deployment to help in the adjustment of the measurement field of view and to monitor system activity. The surface computer and monitor are disconnected once the setup is complete.

The advantages of using a digital camcorder are 1) the video tape recorder is no longer the limiting factor in determining the camera resolution and higher resolution images are obtained in comparison to other formats such as Hi8 or even Betacam 2) resolution is not degraded during the creation of multiple copies during the editing process. The advantage of using the Digital8 format in comparison to the DV format is that ordinary 8mm metal tapes may be used.

You will note that many of the commands refer to older cameras. The TLVL controller while specifically designed for Digital8 cameras can also be used with older format digital or analog cameras which follow the Control-L/LANC protocol.

While strictly not part of the time lapse video logger, a laboratory based image analysis system which captures the replayed video record which is transmitted via an IEEE-1394 (Firewire) link from the camcorder to the capture board is also used. This consists of a Pinnacle DV-500 image capture board and Adobe Premiere v.5.1 loaded onto a PC.

This technical note contains information on the system components, circuit schematics, board layout, source code, mechanical sketches and examples of video records. It does not contain a copies of the compilers used to program the C and assembly code. These may be obtained from Dunfield Systems (http://www.dunfield.com) at nominal cost. With slight revisions the code can be reconfigured for other microcontroller compilers. Information is provided in machine readable form using the attached CD-R disk.

A1.1 The TLVL Controller

The TLVL controller is a self- contained control system which operates the camcorder through the Control-L/LANC interface port. The controller has it's own Real Time Clock, and once loaded with mission data, can operate independently, starting and stopping the camera as required by the mission schedule.

In addition to the camera interface, the controller has 7 general purpose control outputs which can switch 500ma of current, and one high power control output (normally used for lighting).

Two input signals can be configured to Enable or Inhibit the camera operation during the programmed ON times.

The controller's real time clock, and mission data memory continue to operate when the controller is powered OFF, allowing the unit to be setup, and then switched off during transport to the mission site. The camera controller may be disconnected from the external battery and mission parameters will still be active since this information is saved in the real time clock module which contains an integral Li battery. Storage will be valid for several years. If the unit is powered on after the programmed mission start time, but before the mission end

time, it will immediately begin processing an event, after which events will occur at the programmed interval.

A1.2 The HOST computer

The host computer (typically a notebook computer) must be connected to the TLVL controller with a serial connection, and running the TLV host interface software in order to download mission parameters. This software may be run on any DOS compatible environment, including:

MS-DOS versions 3.0+
DR-DOS versions 6.0+ (Also called Novell or Caldera DOS)
Windows 3.x
Windows 95/98

The host software provides full manual control of the camera and control outputs, as well as providing an editor for entering and modifying the mission data.

The host computer need only be connected during manual control and for mission data programming. Once deployed, the TLVL controller operates independently. If the host remains connected, or is reconnected during the mission, it can be used to manually control the camera, update the mission data, and to monitor the camera and controller operating states as the mission progresses.

A1.3 Operational Overview

A typical mission would follow this basic procedure:

- Add fresh batteries to the battery can
- Add a fresh tape to the camera with a fully re-charged battery
-Turn camera to "ON" and plug in all internal cables
- Seal housings and insert dummy connectors
- Leak test camera can
- Mount cans on T-frame
- Attach battery, logger, light module cables
- Attach TLVL camera can to a surface mounted host via UW monitor cable and configure Mission data

The TLVL controller then performs these functions:

- Wait until configured mission start time/date has occurred
- While time/date is <= configured mission end time/date :
  - Power on camera and configured external controls (lights)
  - Wait for configured On->Record delay
  - Start camera recording
  - Wait for configured recording ON time
  - Stop camera recording
  - Wait for configured Stop->Off delay
  - Power off the camera and configured external controls
  - Wait till configured Event Interval has elapsed (next event)
- When configured mission end time/date occurs - halt

The mission can also be configured so that camera control can be passed to an external controller such as the Acoustic Logger. If the Mission parameters are configured to allow external control once the mission start time is passed control of the duration and interval pass to the external controller.

At all times that the host is connected via the serial connection, the controller will monitor for, read and execute commands from the host. If desired, the host can be connected to the controller at any time during the mission to monitor the controller and camera state, perform manual operations, or to change the mission parameters.

Configurable Enable/Inhibit signals will allow/prevent operation of the camera during each event's active recording period. These signals are monitored continuously during the processing of an event, and the camera will be started/stopped accordingly.

A configurable "Daily Inhibit" period may be defined which will prevent the camera from operating during a specific period of the day. This option is mainly used to prevent the camera from recording when no natural light source is available, and lights are not being used.

A2. INSTALLING THE TLVL SYSTEM

A2.1 Installing the controller

The TLV controller must be connected to the Camera via the Control-L/LANC micro-stereo connector. Output control and Enable/Inhibit input signals must be connected to the headers provided if they are to be used. For viewing the video information during setup the internal video cable must also be connected.

Once all connections are made, the camera should be powered ON, and placed in camera-record/pause mode (Move the slide power switch to the "CAMERA" position, and set the standby switch to the "STANDBY" position. The camera should power-up, and display the scene it is viewing through the viewfinder (make sure to remove the lens cap).

Once the camera is operating, power ON or restart the TLV controller. If no mission is in progress, the controller will immediately power OFF the camera. The unit is now ready for operation.

NOTE: It is very important to follow this procedure when setting up the TLV controller and camera. If the camera is not left with it's power switch set to CAMERA, and it's standby switch set to STANDBY, the controller will not be able to turn it on when a mission event occurs.

A number of camera features may be important for specific missions but are not required for the microcontroller to perform basic control of the camcorder. For most missions it is desirable to properly set the real time clock (RTC) in the camera (this should not be confused with the Li battery backed up RTC that the microcontroller uses). If this is properly set, the time and date may be accurately viewed on tape playback.

Also the camera may be set to manual or autofocus. Manual focus is preferred in zones of high suspended sediment. If the camera is in autofocus in this situation a disconcerting 'hunting' focusing action is observed whereby the camera attempts to focus on the scene and then on the particulates and then back again.

These and other features are fully described in the Digital Video Camera Recorder Manual.

A2.2 Installing the TLVL host software

To install the host software, insert the distribution diskette into an IBM/PC compatible computer (desktop or notebook), select the drive, and run the INSTALL program. This will invoke the Micro-Installer program which will perform all commands necessary to install the TLV files on your hard drive.

Before using the host software, you must connect the host PC to the TLVL controller. Any of the four standard COM ports on the PC (COM1-COM4) may be used.

A3. USING THE HOST SOFTWARE

The TLVL host software can be run with the command "TLV" (or double-click on TLV.COM from the windows explorer). The following command line options are available:

```
C=n      <= Select COMn port (1-4)     *1
/M       <= Force monochrome display   *2
? or -?  <= Display command line options summary
```

*1 = Software assumes COM1 if this option not given.
*2 = Software detects Mono or Color adapter, this option needed only if you have mono display/color adapter (older laptops).

When the software starts up, it will check the version number and download information size of the attached TLVL controller to verify that the controller matches the host software version. If a mismatch occurs, a warning message will be displayed. This problem should be corrected, as different versions of the controller and host software may not be compatible, and incorrect operation could occur.

A3.1 Manual Control

Any time the host software is not viewing or editing the mission data, it presents a screen offering manual control of the camera and controller. The following commands are available from the screen menu:

A3.1.1 [B]egin Camera Recording

Pressing 'B' causes the camera to begin recording.
This function may not be supported on certain models of cameras (especially older ones), in which case you will need to use the Start/Stop recording function.
NOTE: The camera must be powered ON and STOPped.

A3.1.2 [S]top Camera Recording

Pressing 'S' causes the camera to stop recording.
This function may not be supported on certain models of cameras (especially older ones), in which case you will need to use the Start/Stop recording function.
NOTE: The camera must be powered ON and RECORDing.

A3.1.3 [X] - Start/Stop recording

Pressing 'X' causes the camera to toggle recording ON and OFF.
This function is provided to support older cameras which do not implement the individual [B]egin and [S]top recording commands.
NOTE: The camera must be powered ON

A3.1.4 [C]apture Still Image

Pressing 'C' causes the camera to digitally capture a still image, and record it for several seconds. This function is supported only on certain models of "digital" cameras.
NOTE: The camera must be powered ON and STOPped.

A3.1.5 [D]isplay ON/OFF

Pressing 'D' causes the camera to toggle the on-screen display prompts ON or OFF. When ON, the camera viewfinder prompts can be viewed with remote monitor.

This function may not be supported on certain models of cameras (especially older ones).

NOTE: The camera must be powered ON.

A3.1.6 [L]ocate Record Position

Pressing 'L' causes the camera to execute a Recording Locate function. This plays in reverse for a few seconds, then plays forward until it is back at the starting location. The purpose of this is to allow you to see the last few seconds of previously recorded video on the remote monitor.

NOTE: The camera must be powered ON and STOPped.

A3.1.7 [F/R] - Edit Locate Functions

These functions perform edit positioning on the camera. When 'F' is pressed, the camera will enter Edit-Play-Forward mode. When 'R' is pressed, the camera will enter Edit-Play-Reverse. The camera will continue to play (Forward or Reverse) until the SPACE bar is pressed. During this time, you will be able to see the image previously recorded on the tape.

NOTE: The camera must be powered ON and STOPped.

A3.1.8 [{}] - Zoom Functions

Press ']' to Zoom In slowly.
Press '}' to Zoom In quickly.
Press '[' to Zoom Out slowly.
Press '{' to Zoom Out quickly.

Once activated, the camera will execute the Zoom function (In or Out) until the SPACE bar is pressed. During this time you will see the image on the remote monitor enlarging or shrinking. Fast Zoom functions may not be available on certain camera models.

NOTE: The camera must be powered ON.

A3.1.9 Display [T]ime

Pressing 'T' will display the time in the message window. The time reported from both the PC and the camera control unit is displayed.

A3.1.10 Set [O]utput Signals

Pressing 'O' allows you to set the output control signals at the camera control unit. These signals are normally set by the Control options of the mission configuration, and updated automatically as events occur, however this function may be used to override the outputs at any time. You will be prompted with a Bit Field Menu for the control signal values to be output.

A3.1.11 [?] - Display Camera Status

Pressing '?' displays the camera status/counter values:

Status: aabb Counter: cccccc

aa = TLV controller state
- 00 = Waiting for valid mission data (checksum error)
- 01 = Waiting for mission to start
- 02 = Processing an event - Camera is ON
- 03 - Processing an event - Camera is OFF
- 04 - Waiting for next event
- 05 - Mission has ended

bb = Camera state (see section C5 for a list of camera states)
cc = Camera tape Counter
NOTE: The camera must be powered ON.

A3.1.12 [F2] - Turn Camera Power ON

Pressing F2 executes an event Power-ON sequence. The control outputs are turned on (including Ramp if specified) and the camera is turned ON.

A3.1.13 [F3] - Turn Camera Power OFF

Pressing F3 executes an event Power-OFF sequence. The control outputs are returned to their idle state, and the camera is turned OFF.

A3.1.14 [F4] - Set remote Time

Pressing F4 allows you to set the time on the camera control unit. You will be presented with a Menu offering two choices:

"Set to local PC time" <- Use the time from the PC
"Prompt for specific time" <- Prompt for the time

If you select the "Prompt" option, you will get an Input form allowing you to enter the exact time you wish to set. For your convenience the time value is preset to the current PC time. Usually the easiest procedure is to first set the PC to the correct real time and then use "Set to local PC time".

A3.1.15 [F5] - Set Next Event Time

Pressing F5 allows you to view and alter the time of the next scheduled event on the camera control unit. Normally, the time of the next event is automatically recalculated as each event occurs, using the interval specified in the Mission Configuration. This function allows you to override the time at which the next event will occur. Subsequent events will occur at the specified intervals from this time.

A3.1.16 [F7] TTY Communications Mode

Pressing F7 enters into "TTY" serial communication with the camera control unit. This is mainly for debugging purposes, and should not be used unless you have good reason to communicate directly with the control unit.

A3.1.17 [F8] - Mission Editor/Viewer

Pressing F8 will place you in the mission data viewer. The current mission data is read from the controller, and displayed on the screen. At this time, you may press SPACE to activate the mission editor to update the mission data. More information on the mission viewer/editor is presented in the section entitled "Mission Viewer".

A3.1.18 [F9] - Restart Unit

Pressing F9 causes the camera control unit to restart. This performs a complete re-initialization of the on-board control system. Normally, a restart is only necessary when you have set a new mission starting time after a mission is already in progress. Restarting the unit will cause it to go to sleep until the new start time occurs.

A3.2 Mission Viewer

Pressing F8 from the main "Manual Control" screen will activate the mission viewer. The current mission data is read from the controller and displayed on the screen. At this time, you may press the following keys:

F1 - Help information
F2 - Reload the mission data from the controller
SPACE - Active the "Mission Editor" to update the data
ESC/F10 - Exit the mission Viewer/Editor

A3.2.1 Mission Start Date/Time

This is the date and time at which the camera will begin recording regular video clips. Before this date and time occur, the camera will remain off or in idle (the camera shuts itself off after a few minutes to save power-it will re-activate at a Ctl-L command) . Once this time has passed, the camera will begin recording video clips and the defined Event Interval and Event On Time until the Mission End date and time are over.

A3.2.2 Mission End Date/Time

This is the date and time at which the camera will stop recording regular video clips. Once this date and time have passed, the camera will remain idle until a new Mission Start is programmed, and the controller is restarted.

A3.2.3 Event ON time

This configuration item defines the duration of each video clip. Once the Mission Start date and time have passed, the camera will record regular video clips of this length, at the defined Event Interval. This regular recording will continue until the Mission End date and time pass.

A3.2.4 Event Interval

This configuration item defines the interval time between the start of successive video clips. Once the Mission Start date and time have passed, the camera will record regular video clips of the defined Event ON time duration at this interval. This regular recording will continue until the Mission End date and time have passed.

A3.2.5 Misc. Configuration

This is a Bit Field containing various configuration items which can be toggled ON or OFF:

A3.2.5.1 Enable Control Input

This item when enabled, causes the camera controller to honor the hardware ENABLE INPUT. This means that the ENABLE INPUT must be present in order for the camera to operate. The hardware ENABLE INPUT may be toggled while an event is in progress to start and stop the camera within that event. This mode is used when control of the TLVL is passed to the Acoustic Logger.

A3.2.5.2 Inhibit Control Input

This item when enabled, causes the camera controller to honor the hardware INHIBIT INPUT. This means that the INHIBIT INPUT must NOT be present in order for the camera to operate.

The hardware INHIBIT INPUT may be toggled while an event is in progress to start and stop the camera within that event.

A3.2.5.3 Daily Inhibit Period

This item when set, causes two more fields to appear on the mission configuration menu. These fields control the starting and ending time for an INHIBIT PERIOD. During the INHIBIT PERIOD, the camera will not record. If the INHIBIT PERIOD begins or ends during an active event, the camera will be stopped or started accordingly. If the INHIBIT PERIOD ending time is set earlier than the starting time, the inhibit period will occur between the starting time, and the ending time on the next day.

A3.2.5.4 Single Start/Stop

This item when set causes the TLVL to use a single Start/Stop code for beginning and ending recording. When NOT set, separate Record and Stop commands are issued to the camera. This option is provided to support older cameras which do not support the separate commands. When the separate commands are available, they should be used, as they are a more reliable way of insuring that the camera is doing what is intended.

A3.2.5.5 Debug TTY messages

This option enables text messages to appear on the serial data line from the TLVL controller, which inform the viewer of the internal actions occurring within the TLVL firmware. These messages can be viewed with TTY Mode. This option should not be used in the normal day to day operation of the TLV, as it can cause interference with the serial

communications between the HOST and the TLV controller. It is intended for firmware debugging purposes only.

A3.2.6 Event Control Signals

These fields define the hardware output control signals which will be set by the controller when an event starts and stops.

@OFF <- Defines outputs when the unit is idle.
@ON <- Defines outputs when the unit is recording.
@RAMP <- Defines a power "ramp up" as described below:

The RAMP setting can be used to gently turn on lights. If these entries have different values, the controller will generate a varying duty cycle waveform from 100%-RAMP/0%-ON to 0%-RAMP/100%-ON over approx. 1 second during the power-up sequence. This is used to soft start the external lamp.

A3.2.7 On -> Record Delay Interval

This item specifies a delay (in milliseconds), which the controller waits after powering ON the camera, before starting Recording. This allows the camera time to perform it's own power-on sequence, during which time it may not be ready to accept commands.

A3.2.8 Stop -> Off Delay Interval

This item specifies a delay (in milliseconds), which the controller waits after halting recording command before powering OFF the camera. This allows the camera time to stop the tape transport.

For both the "On -> Record Delay Interval" and the " Stop -> Off Delay Interval" you will note that the camera will log video during a portion of these intervals. The nominal values of 4000 ms are used to ensure that variations due to temperature or battery capacity will not affect the ability of the camera to receive commands while the tape drive is spooling up or down. Some of this delay interval is in effect added to the Event ON time. This will affect the true "Event ON time" and also the effective mission duration. Experimentally determine the appropriate delay intervals for mission conditions and cameras to more accurately set the values if it is necessary to extend mission duration to the maximum.

A3.3 Mission Editor

After you press SPACE on the Mission Viewer, a cursor bar will appear which allows you to select any item for editing. You may:

Press ENTER to make changes to the hilighted item.
Press F2 to reload the values last saved to the unit.
Press F3 to save the changed values to the unit.
Press F4 to clear all of the event fields.
Press F10 or ESCAPE to exit.

NOTE: Changes to the mission will take effect at the next event cycle. If you change the mission start time after a mission has already begun, you should RESTART the unit to force it to recognize the new start time.

B) Electronic Hardware

B1. MICROCONTROLLER BOARD HARDWARE OVERVIEW

The Time Lapse Video Logger (TLVL) microcontroller board consists the following principal hardware components:

- 80C31 Microcontroller
- Serial UART for control/programming (built into 80C31)
- MAX 232 serial driver
- 32K (bytes) of Read-Only memory (27C256)
- Dallas DS12887 Real Time Clock (RTC) which includes an integral Li battery and mission control parameter memory
- bi-directional serial LANC bus for camera control
- switching power supply regulator to convert nominal 12V to 5V power
- 8 parallel output lines for external control
- 2 parallel input lines for Enable/Inhibit control
- buffering of control lines using ULN2003, IRF540 FET, opto-isolated I/O(HCPL-2231)

See Section B4 for detailed drawings and parts lists.

B2. DETAILED DESCRIPTIONS

B2.1 Processing Section

The processing section of the control unit is designed around an Intel 80C31 compatible central processing unit (CPU). We have used the 80C31 and its variants for years and are familiar with it. This device is from the Intel 8051 family, and is identical to an 80C51 except that it does not utilize on-chip program storage. An 80C51 device can be used interchangeably in the TLV controller. Generic 80C31's from other manufacturers are fine. More advanced variants using the basic 80C31 architecture could also use much of the same code but because pinouts are different new printed circuit boards and some re-programming would be required. The required functions for the TLVL can easily be handled by the basic 80C31. The camera can size is determined by the camera and all necessary components fit on a single 100 mm square printed circuit board. In order to make component replacement easier, components were selected to be readily available from electronic distributors located in any major city across Canada (e.g. Active stores, Avnet etc.).

The 80C31 provides an external bus for access to memory and addressable peripherals, and a total of 12 additional parallel I/O lines, all of which are used on the TLVL. In addition, the 80C31 provides a full duplex serial port. Program storage is accomplished with an external 27C256 EPROM, which provides a total of 32K (32768) bytes of non-volatile program memory. Flash memory could be substituted with an appropriate minor revision to the board. The architecture is quite conventional for an 80C31 system. An external 74HC373 latch is provided to de-multiplex the 80C31's combined data/address bus into separate data and address signals which are required by the 27C256 EPROM. The 27C256 is enabled directly from the 80C31's -PSEN signal, which makes it available only to program storage fetches. Since A15 is not connected, the content of the 27C256 appears twice within the 64K program memory address space of the 80C31.The 80C31 in the TLV controller runs at a clock speed of 11.0592 MHz. This is a standard clock speed for the 80C31 (crystals are readily available), which provides for accurate serial speeds at common bit rates, using the 80C31's internal bit rate generator. The 80C31 utilizes interrupts from both the serial port (internal source), and the

DS12887 Real Time Clock (external "wired" interrupt) so that the processor can go into Idle mode between TLV events, which conserves power.

B2.2 Real Time Clock

The TLVL controller uses a Dallas DS12887 Real Time Clock (RTC) device as its primary means of time keeping. The DS12887 also provides 114 bytes of non-volatile RAM which the TLV controller uses to maintain it's configuration and event descriptions. This is retained by a built-in Li battery which will protect system memory for up to several years between power-up from an external source. The TLVL controller does not use more than 50 bytes of the non-volatile RAM, allowing a DS1287 device to be used instead of the DS12887 if desired. Future version of the TLVL controller may utilize this extra resource, and therefore may not be compatible with the older Dallas device. The DS12887 is wired directly to the 80C31's external memory bus, and has it's own address latch, so that only the 8 "ADx" connections are required for address and data. The DS12887 is enabled directly from the 80C31's -RD and -WR signals, which make it available only to external data memory accesses. Since the high-order address lines (A8-A15) are not used with this device, it appears 256 times throughout the 64K external data memory address space. External RAM memory is not used to store data and this poses little problem. The -INT output of the DS12887 is wired directly to the INT0/P3.2 input of the 80C31. This allows the RTC to interrupt the processor, and bring it out of "sleep" mode. More information on the DS12887 device can be obtained from the included DS12887.PDF data sheet.

B2.3 LANC/Control-L Interface

The SONY camcorder is controlled through a single-wire bi-directional serial interface. A "frame" of data is initiated by the camera, and requires that certain bits be "filled" in by the controller to send data to the camera, while other bits are transmitted by the camera and must be read by the controller.

The LANC port of the camera is a *stereo* Micro (mini-mini) connector, and has the following pinout:

pin 1 is the tip, pin 2 is the middle and pin 3 is the pin closes to the flange.

See the camera can internal wiring schematic (section B5.2) for details.

VIEW OF INTERFACE PLUG
1—— serial bus

2—— power(+6-9 VDC @ 100ma)

3—— ground

The TLVL controller does not make use of the camera POWER, however a stereo connector is required to avoid shorting camera power to ground. The serial bus from the camera is connected directly into the 80C31's P3.5 I/O line. To provide some protection

against transients, this signal passes through a 100 Ohm resistor, and has diodes on the CPU side which limit voltages >+5 or <0 by bypassing them to the power/ground rails.

The mini-stereo LANC connector can be bought from Nutech, Parkdale Ave., Hamilton, ON. The connector must be bent by nearly 90 degrees after wiring and sealed with a small amount of silicone rubber (RTV). The cover cannot be used. The bend is required to fit the underwater housing. The pressure against the wall of the housing provides a restraining force since the LANC connector does not have a means of built-in strain relief.

B2.4 Input Enable/Inhibit Signals

The controller supports two binary signals, one which can be used to enable the camera, and one which can be used to inhibit it's operation. These inputs are connected directly to the 80C31's P3.3 and P3.4 I/O lines. To provide some protection against transients, the signals pass through a 1K resistor, and have diodes on the CPU side which limit voltages >+5 or <0 by bypassing them to the power/ground rails.

B2.5 Output Control Signals

The controller provides 8 general purpose output control signals which are generated at the 80C31's P1 out lines. Seven (7) of the output signals are processed through a ULN2003 relay driver IC, which provides 500ma sink for each line, and can control voltages up to 50V. These outputs provide current sink only, and are used to switch external devices through the ground side of their supply. The total current through the ULN2003 should not exceed 2.5 amps. The ULN2003 includes clamp diodes for inductive loads, and a single terminal is provided to connect these to the positive side of the load power supply. For more information on the ULN2003, please refer to the included data sheet.

One (1) of the output signals is used to control a IRL40N power Field Effect Transistor (FET), which can switch loads of up to 20 amps. It is used to control the external submersible 20W lamp. It is triggered directly from line P1.0 with a 22Kohm resistor tied to common to ensure quiescent turn-off. The FET is mounted on the rear camera housing bulkhead to provide heat sinking. Connection between the board and FET is via a Molex connector (Header5).

The external trigger and synchronization lines are optically coupled using HCPL-2231 high-speed low-current opto-couplers.

B2.6 Serial Interface

The TLVL controller uses a serial interface to communicate with the host PC. The serial device is internal to the 80C31, and depending on the application, differing interface circuitry may be desired for the actual connection to the host. To date, a simple RS-232 serial connection has been implemented. This has proven to be sufficient for 20m of cable at the baud rates used. For longer distances it may be worth modifying the board for a RS-422 connection or a fiber-optic interface.

B3. POWER CONSUMPTION

The TLVL controller is designed to require very low supply current levels. Most of the circuitry is built with low power CMOS devices, and when not actively processing an event or a serial command, the 80C31 enters a low-power "Idle" mode. Note that the power drawn by the serial interface load is a significant portion of the total operating power.

The following measurements were made with the prototype TLVL controller operating from a 5v supply:

CPU	SERIAL	Current Measured
Idle	None	11-13ma
Idle	RS-232/U	12-14ma
Idle	RS-232/C	16-18ma
Active	None	21-23ma
Active	RS-232/U	22-24ma
Active	RS-232/C	26-28ma

None = RS-232 module not installed
RS-232/U = RS-232 module installed, but not connected
RS-232/C = RS-232 module installed and connected to a PC

The regulation is provided by a LM2574N switching regulator. This is a relatively low frequency device and provides regulation over a significant voltage range. It is more efficient than a linear regulator even for the modest current requirements outlined above. The efficiency is of the order of 50% over most of the voltage range for the currents we deal with. A linear regulator is between 35 to 40 % efficient over most of the input voltage range.

With an external 20W lamp, it is the lamp which consumes the most power. The rest of the electronics at worst consume about 250 mW from the battery (input to the regulator). In fact it only consumes this when the serial line is connected and active. The normal on power is more like 150 mW. Assuming the TLVL lamp is on for 20s every 20 minutes of 1 minute per hour, the duty cycle is 1/60. Thus the equivalent continuous on power of the lamp is 1/60 x 20 =0.33W or 330mW or about twice that of the remainder of the electronics.

The camera is powered separately using the heavy-duty re-chargeable battery available from the manufacturer.

B4. CIRCUITS

See the following drawings and sketches:

B4.1 Microcontroller board circuit schematic
B4.2 Microcontroller board component placement
B4.3 Microcontroller PCB layout (top)
B4.4 Microcontroller PCB layout (bottom)
B4.5 Microcontroller board parts list

Auxiallilary output(not used)

JP2
HEADER 8

U1
ULN2003

C1 39p
C2 39p
Y1 11.059Mhz
C3 10nF
R1 10K
R2 22K
C8 100n

U2
80C31

JP1
LAMP
HEADER 5

U3A
HCPL-2231
C16 100nF
R2 2K
C13 120p
R3 2K
C14 120p
JP5
4 HEADER
To Acoustic Logger

U5
LM2574N5.0(8)
C12 22nF
L1 330uH
D3 11DQ06
C15 220uF
JP4
HEADER 2
To Battery

D1 1N4004
D2 1N4004
R4 1K
JP3
PD2
To Camera
LANC/ CTL-L

U4
27C256
C9 100n

C10 100n
U7
DS1288

U8
74HC373
C11 100n

C4 10uf
C5 10uf
C6 10uf
C7 10uf
U6
MAX232
U6-2
JP6
HEADER 4
To serial I/O
RXD
TXD

TIMER
Inhibit
Enable

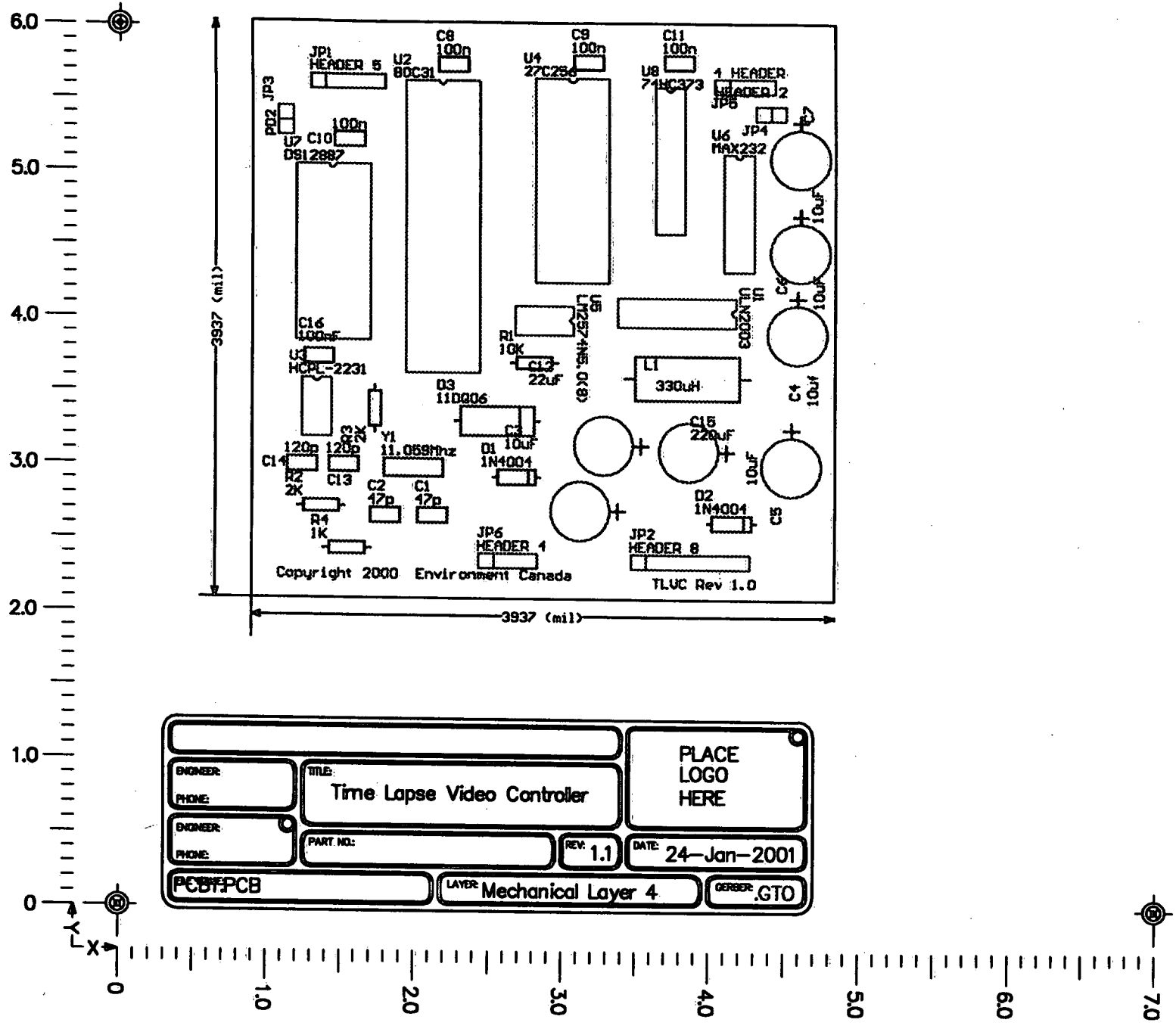
Revision 1:

- Corrected pins 3 & 4 on U3A
--note: pcb traces re-wired to correct ground & output pin

- Added R2
- Aded labels
-removed extraneous junctions
-added necessary junctions
RJD 10 July 2000

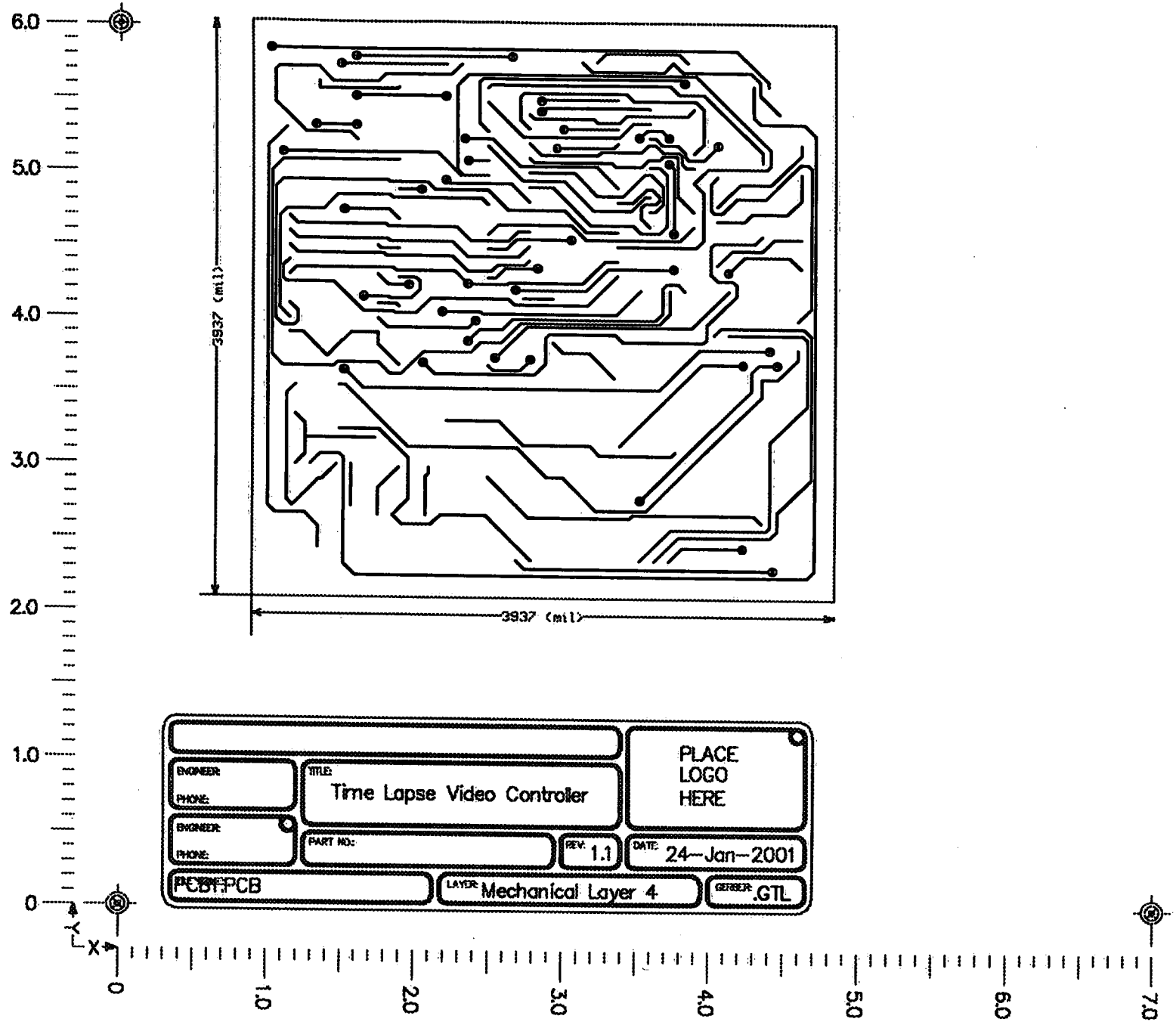
Copyright Environment Canada 2000

Title	Time Lapse Video Logger Control Schematic	
Size: B	Number: EE-2000-01	Revision: 1.1
Date: 23-Jan-2001	Sheet of 1 of 1	
File: C:\My Documents\video logger\tlvlprotel\tlvlschdoc	Drawn By: VB/JWG	

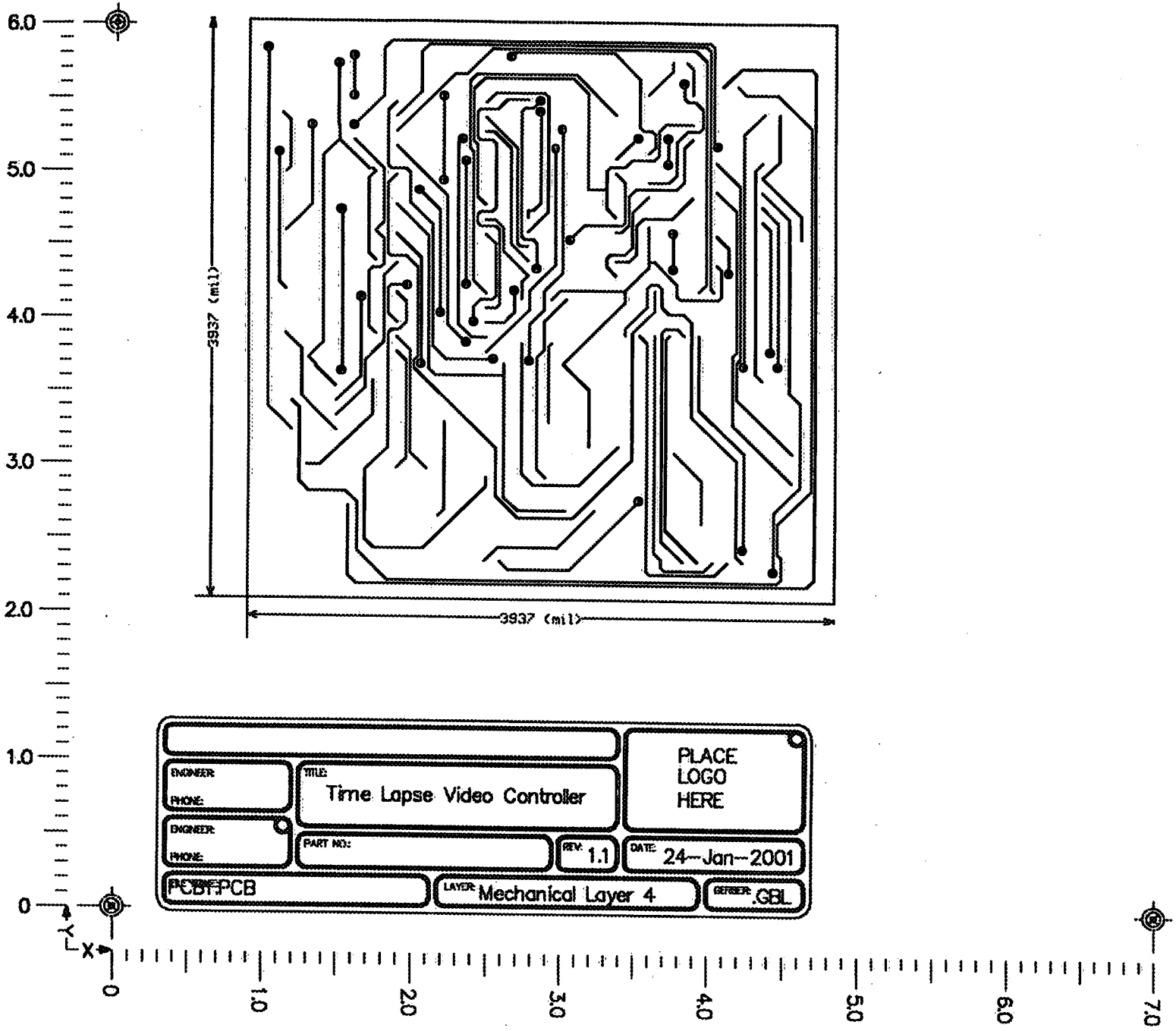


JP1 HEADER 5
U2 80C31
C8 100n
U4 27C256
C9 100n
U8 74HC373
C11 100n
4 HEADER
HEADER 2
JP5
JP4
U6 MAX232
JP3
PD2
U7 DS12887
C10 100n
C16 100nF
U3 HCPL-2231
R3 2K
Y1 11.059Mhz
C14 120p
C13 120p
R2 2K
R4 1K
C2 47p
C1 47p
D3 11DQ06
C3 10uF
D1 1N4004
R1 10K
C12 22uF
U5 LM2574N5.0(8)
U1 ULN2003
L1 330uH
C15 220uF
D2 1N4004
C5 10uF
C4 10uf
C6 10uF
C7 10uF
JP6 HEADER 4
JP2 HEADER 8
Copyright 2000 Environment Canada
TLVC Rev 1.0
3937 (mil)
3937 (mil)
ENGINEER:
PHONE:
TITLE: Time Lapse Video Controller
PLACE LOGO HERE
ENGINEER:
PHONE:
PART NO.:
REV: 1.1
DATE: 24-Jan-2001
PCB1.PCB
LAYER: Mechanical Layer 4
GERBER: .GTO

Comments


6.0
5.0
4.0
3.0
2.0
1.0
0
3937 (mil)
3937 (mil)
ENGINEER:
PHONE:
ENGINEER:
PHONE:
TITLE:
Time Lapse Video Controller
PART NO:
REV: 1.1
DATE: 24-Jan-2001
PLACE
LOGO
HERE
PCB1:PCB
LAYER: Mechanical Layer 4
GERBER: .GTL
Y
X
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

dot


6.0
5.0
4.0
3.0
2.0
1.0
0
3937 (mil)
3937 (mil)
ENGINEER:
PHONE:
TITLE:
Time Lapse Video Controller
PLACE
LOGO
HERE
ENGINEER:
PHONE:
PART NO:
REV: 1.1
DATE: 24--Jan--2001
PCBT.PCB
LAYER: Mechanical Layer 4
GERBER: .GBL
Y
X
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0

bottom

B4.5 Microcontroller board parts list

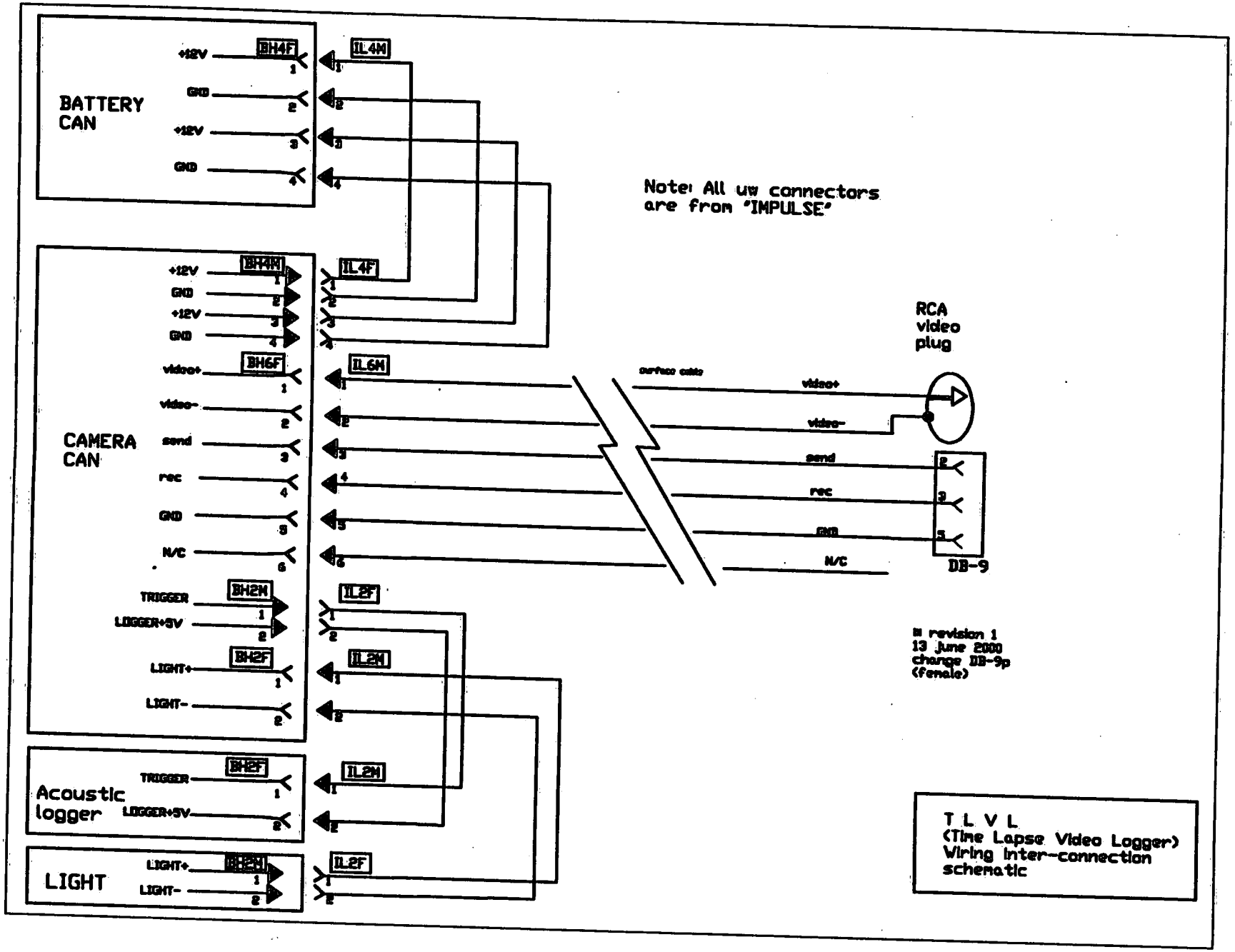
from Protel schematic & files

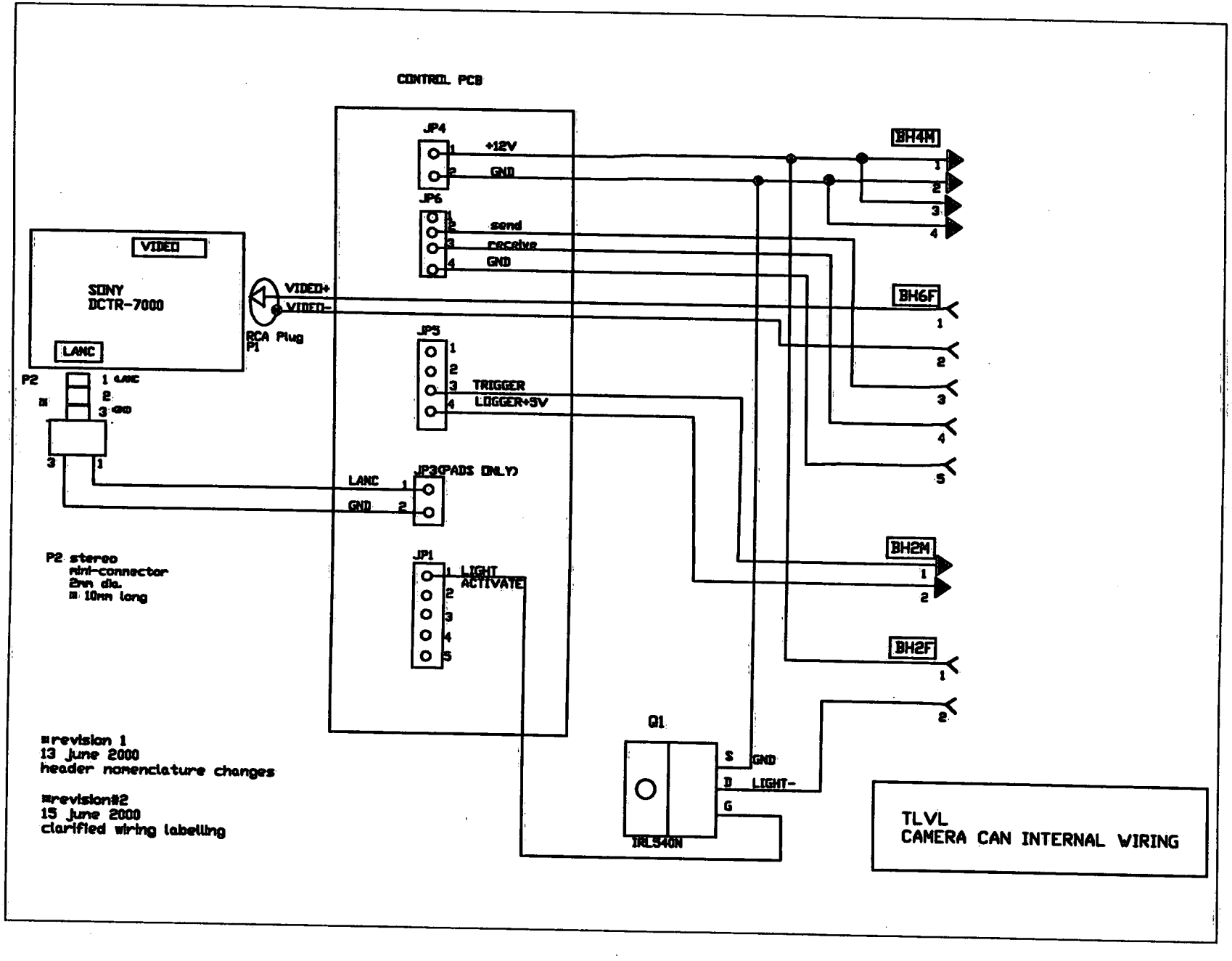
Comment	Footprint	Designators
100n	RAD0.1	C8
100n	RAD0.1	C9
100n	RAD0.1	C10
100n	RAD0.1	C11
100nF	RAD0.1	C16
10K	AXIAL0.4	R1
10uF	RB.2/.4	C3
10uf	RB.2/.4	C4
10uF	RB.2/.4	C5
10uF	RB.2/.4	C6
10uF	RB.2/.4	C7
11.059Mhz	XTAL1	Y1
11DQ06	DIODE0.7	D3
120p	RAD0.1	C13
120p	RAD0.1	C14
1K	AXIAL0.4	R4
1N4004	DIODE0.4	D1
1N4004	DIODE0.4	D2
220uF	RB.2/.4	C15
22uF	RB.2/.4	C12
27C256	DIP28	U4
2K	AXIAL0.4	R2
2K	AXIAL0.4	R3
330uH	AXIAL1.0	L1
4 HEADER	SIP4	JP5
47p	RAD0.1	C1
47p	RAD0.1	C2
74HC373	DIP20	U8
80C31	DIP40	U2
DS12887	DIP24	U7
HCPL-2231	DIP8	U3
HEADER 2	SIP2	JP4
HEADER 4	SIP4	JP6
HEADER 5	SIP5	JP1
HEADER 8	SIP8	JP2
LM2574N5.0(8	DIP8	U5
MAX232	DIP16	U6
PD2	SIP2	JP3
ULN2003	DIP16	U1
IRL540N	FET	on blkhd

B5 INTERCONNECTIONS

See the following sketches:

B5.1 System wiring interconnection schematic
B5.2 Camera can internal wiring schematic


BATTERY CAN
+12V
GND
+12V
GND
BH4F
IL4M
CAMERA CAN
BH4M
IL4F
BH6F
IL6M
video+
video-
send
rec
GND
N/C
BH2M
IL2F
TRIGGER
LOGGER+5V
BH2F
IL2M
LIGHT+
LIGHT-
Acoustic logger
LIGHT
Note: All uw connectors are from "IMPULSE"
surface cable
RCA video plug
DB-9
revision 1
13 June 2000
change DB-9p (female)
T L V L
(Time Lapse Video Logger)
Wiring Inter-connection schematic


CONTROL PCB
JP4
+12V
GND
JP6
send
receive
GND
VIDEO
SONY
DCTR-7000
LANC
VIDEO+
VIDEO-
RCA Plug
P1
P2
LANC
GND
JP5
TRIGGER
LOGGER+5V
JP3 (PADS ONLY)
LANC
GND
JP1
LIGHT ACTIVATE
BH4M
BH6F
BH2M
BH2F
Q1
S
D
G
GND
LIGHT-
IRL540N
P2 stereo mini-connector 2mm dia. 10mm long
revision 1
13 june 2000
header nomenclature changes
revision#2
15 june 2000
clarified wiring labelling
TLVL
CAMERA CAN INTERNAL WIRING

C)Firmware Description

C1. FIRMWARE OVERVIEW

The firmware is the code written for the microcontroller which is embedded in the EPROM. Most of the original source for the embedded code was written in C and is designed to be easily maintainable with well commented code. The half-dozen or so functions which are programmed in assembly are identified with a triple asterisk (***).

The firmware source code is given in the file TLVFIRM.C- printed in section C6. This code can be compiled to a binary image using the "Micro-C 8051 Developers Kit" from Dunfield Development Systems(http://www.dunfield.com). This can be procured at modest cost. With minor modifications this can be compiled using other 80C31 cross-compilers. A description of the functions used is given in the sections which follow.

C2. DETAILED FUNCTION DESCRIPTIONS

This section describes the individual functions in the TLVL controller firmware. Functions which are called, but not described here, are from the Micro-C/51 compiler library, and should be referenced from the Micro-C library documentation.

C2.1 Main logic functions

The functions described in this section implement the main logic of the TLVL controller.

C2.1.1 main() - Main program module

The main() function is where execution of the TLV firmware begins following a reset or software restart. It performs the following functions:

- Initialize the serial port and RTC.
- Validate the programmed mission with a checksum.(See *1 *2)
- Set the parallel output control signals to the @OFF state
- Wait for the mission time/date to arrive(See *2)
- Process an event by:
  - If the mission time has passed, stop processing
  - Record the event starting time
  - Check camera_enabled() and if true:
    - Power up camera and parallel outputs: power_on()
    - Wait the configured On->Record delay
    - Begin recording
  - Wait for the event on time to elapse: wait_on_time()(See *2)
  - Stop the camera recording
  - Wait the configured Stop->Off delay
  - Power OFF the camera and parallel outputs: power_off()
  - Set RTC to wakeup at next event time: set_next_alarm()
  - Go to sleep until RTC alarm goes off: sleep() *2
  - Return to top of loop to process next event
- Mission ended, sleep forever(See *2)

*1 =If the programmed mission fails the validity test on startup or restart, the controller wait for a new mission to be loaded, and does not proceed until the validity test passes.

*2 =During these events, the controller detects pending commands on the serial port, and when one is initiated, it handles it by calling process_command().

C2.1.2 wait_on_time() - Event processing

This function does the active processing during a mission event. It's main purpose is to wait for the configured on-time, and during this interval, handle various conditions:

- Begin with a count of the seconds remaining in the event
- Check for serial command, and process_command() if found
- Wait for one second to expire before proceeding
- If camera_enabled() indicates camera should be ON, but the current camera state is OFF:
  - power_up camera and parallel outputs: power_on()
  - Wait the configured On->Record delay
  - Begin recording
- If camera_enabled() indicates camera should be OFF, but the current camera state is ON:
  - Stop the camera recording
  - Wait the configured Stop->Off delay
  - Power OFF the camera and parallel outputs: power_off()
- continue this loop until the event time expires

C2.1.3 camera_enabled() - Determine if the camera is enabled

Determines if the Enable or Inhibit signals are configured and active, as well as check for the daily inhibit period. This is done continuously throughout the event on time, so that the camera can be started and stopped as required.

- If Enable signal is configured
  - If Enable input is FALSE, return DISABLED
- If Inhibit signal is configured
  - If Inhibit signal is TRUE, return DISABLED
- If a daily_inhibit period is configured
  - If Inhibit_mode != 0 (starting <= ending)
    - If time < starting return ENABLED
    - if time > ending return ENABLED
  - if Inhibit mode == 0 (starting > ending)
    - If time > starting return DISABLED
    - if time < ending return DISABLED
- Return ENABLED if not DISABLED

C2.1.4 power_on() - Turn power on

Turns on the power to the camera, and sets the parallel output signals to the @ON condition. If configured, a RAMP-UP is performed on the @RAMP-@ON signals.

- Pulse LANC control line to turn camera on
- if @RAMP != @ON
  - Perform 64 cycles of ramp-up power waveform
    - Vary duty cycle from 0-100% in 1/16th cycles.

C2.1.5 power_off() - Turn power off

Sets the parallel output signals to the @OFF condition and turns off power to the camera.

- Set parallel outputs to @OFF
- Sent POWER-OFF command to camera

C2.1.6 set_next_alarm() - Set time for next event

Sets the next event time alarm:

- Convert EVT_offtime to h:m:s
- Add off time to event start time, wrapping to next day
- Program the alarm time in the RTC: set_alarm()

C2.1.7 process_command() - Host command processing

This function is called whenever a command start character (ATTN_CHAR) is detected on the serial port. It's purpose is to process commands from the host PC:

- Save machine stack so we can abort/exit cleanly
- Echo the ATTN_CHAR
- Read next character, and check for ATTN_CHAR1
  - If ATTN_CHAR again, back to echo and loop
  - If not ATTN_CHAR1, error exit
- Read command, echo and process(See comment *1)
- If command requires operands, read and echo them(See comment *1)
- Process command, send output to host (See commennt*1)
- At end of command, send ERR_ok in indicate success
- If an error occurred:
  - Send ERR_comms
  - Flush any incoming data
  - Send ERR_ok

*1 =See "Serial Command Protocol" for details of the serial command formats.

C2.2 Support functions

The functions described in this section provide support services to the main logic functions.

C2.2.1 gett() - Get character from serial port with timeout(***)

Gets a single character from the serial port without a timeout, and echo's the character. As this involves bit twiddling, it is written in assembly language.

- Record current RTC second count
- Set time-out counter to initial value
- Test for character, if received:
  - Echo character
  - return with character
- If no character, check for second count changed, if so:
  - Decrement time-out counter, if expired
  - Jump to error handling code at end of process_command()
- Loop waiting for character

C2.2.2 geth() - Get a hex value from the serial port

Gets an ASCII-HEX value from the serial port, and returns it as an 8 bit binary (0-255) quantity.

- Get first character
- Subtract '0' to convert '0'-'9' to binary
- if in the 'A'-'F' range, subtract 7 to complete conversion
- If out of range, Jump to error at end of process_command()
- Get second character
- Subtract '0' to convert '0'-'9' to binary
- if in the 'A'-'F' range, subtract 7 to complete conversion
- If out of range, Jump to error at end of process_command()
- return (first * 16) + second

C2.2.3 puth(v) - Write hex value to serial port

Writes an 8-bit value (0-255) to the serial port as two ASCII-HEX digits.

- Extract first digit (v >> 4), and add '0' to convert
- If in the 'A'-'F' range, add 7 to complete conversion
- Write to serial port
- Extract second digit (v & 15), and add '0' to convert
- If in the 'A'-'F' range, add 7 to complete conversion
- Write to serial port

C2.2.4 CLfr() - Control-L/LANC frame exchange(***)

This function performs a single LANC frame exchange with the camera, both transmitting commands, and receiving status information. This is a very real-time intensive operation with precise timing, therefore this function is written in assembly language:

- Wait for LANC I/O line HIGH for at least 100us
- Wait for start bit / send command TXWORD 0
- Wait for start bit / send command TXWORD 1
- Wait for start bit / receive RXWORD 0
- Wait for start bit / receive RXWORD 1
- Wait for start bit / receive VCRMODE
- Wait for start bit / receive STATUS & GUIDE CODE
- If 4-digit counter indicated:
  - Wait for start bit / Receive LOW COUNTER
  - Wait for start bit / Receive HIGH COUNTER
- If H:M:S counter LOW is indicated:
  - Wait for start bit / receive MM:SS
- if H:M:S counter HIGH is indicated:
  - Wait for start bit / receive HH
- If VCR status is indicated
  - Wait for start bit / receive STATUS

For complete details on the LANC interface handshaking and timing, please refer to the source code comments in TLVFIRM.C.

C2.2.5 lanc_cmd() - Send Control-L/LANC command

Performs a complete LANC command sequence by performing the LANC frame exchange until all counter and status information has been collected or an error occurs.

- Zero received data flags
- Perform LANC frame exchange
- If error indicated, abort
- If not all data received, loop for next

C2.2.6 event_checksum() - Calculate mission data checksum

This function computes the checksum of the mission data. It is used to validate the mission data at startup, and to confirm the validity of mission data uploaded and download to/from the host.

- Add all bytes of mission data to generate a single sum

C2.2.7 set_inhibit_mode() - Set inhibit testing flag

This function simply determines if the daily inhibit starting time is greater then the ending time, and sets the Inhibit_mode flag accordingly. This is necessary so that camera_enabled() can 'know' how to correctly handle the inhibit time period.

- Set global time variable to inhibit start time
- If <= inhibit end time, Inhibit flag = -1, else 0

C2.2.8 read_time() - Read time from RTC device(***)

Reads the time from the RTC. It does so in such as way as to insure that an update has not occurred during the read (which would invalidate the time). As this is a time critical function, it is written in assembly language:

- Poll RTC register A and wait for UIP bit low
- Read ss,mm,hh,DD,MM,YY within 244us

C2.2.9 set_time() - Set the RTC time(***)

Sets the RTC time. It holds the RTC counter chain in reset while doing so, so that update cannot invalidate the partial data already written. As this is a fairly time critical operation, it is written in assembly language:

- Enable SET bit in RTC register B
- Write new ss,mm,hh,DD,MM,YY values
- Clear SET bit in RTC register B

C2.2.10 compare_date(date) - Compare complete date/time

Compares the date/time located in the global time variables with the one at the specified location in mission data.

- Compare year, if same:
 - Compare month, if same:
  - Compare day, if same:
   - Compare hour, if same:
    - Compare minute, if same:
     - Compare second.

return:
-1 = date/time < mission data date/time
0 = date/time == mission data date/time
1 = date/time > mission data date/time

C2.2.11 compare_time(s, m, h) - Compare time only

Compares the time located in the global time variables with the specified hour, minute and second.

- Compare hour, if same:
 - Compare minute, if same:
  - Compare second

return:
-1 = time < specified time
0 = time == specified time
1 = time > specified time

C2.2.12 set_alarm() - Set RTC alarm time(***)

Programs the RTC device alarm with the hour, minute and second located in the global time variables. As this should happen very quickly, and involves writing to external memory (not directly support in C TINY model), this function is written in assembly language.

- Write Second, Minute and Hour to RTC registers
- Set AIE bit in RTC control register B

C2.2.13 sleep() - Put processor to sleep until next alarm(***)

Enables serial and alarm interrupts, then puts the 80C31 processor into low power IDLE mode until the next RTC alarm occurs. As this involves bit twiddling of the 80C31 SFR's, it is written in assembly language.

- Disable 80C31 UART transmitter to prevent interrupt
- Clear any pending TX and RX interrupts from the UART
- Zero the Alarm_flag - no interrupt yet
- Enable serial and Alarm (external-0) interrupts
- Set PCON.0, enable IDLE mode - CPU sleeps
- On wakeup, disable all interrupts
- Re-enable the 80C31 transmitter

C2.2.14 alarm_irq() - RTC Alarm interrupt handler

Handles the RTC alarm interrupt. Clear the interrupt condition from the RTC, and sets the Alarm_flag so that we know an alarm occurred. This interrupt/function will only occur when the CPU is in sleep() mode.

- Read RTC register C to clear interrupt
- Set Alarm_flag to -1

C2.2.15 serial_irq() - Serial interrupt handler

Handles the serial interrupt. Does nothing, just returns. This interrupt/function will only occur when the CPU is in sleep() mode, and once this handler runs, the sleep() function will regain control, and immediately disable serial interrupts. All serial I/O will then occur under programmed polling.

C3. SERIAL COMMAND PROTOCOL

The serial protocol between the host and the TLVL has been kept fairly simple, however the following issues should be known to the reader:

At any point, the TLVL controller may be processing a LANC frame transaction with the camera, which is a very real time sensitive operation. For this reason, serial communications do not use interrupts on a character by character basis. Serial interrupts are used only to "wake up" the TLVL when it's CPU is sleeping during the interval between events.

To avoid overrunning the TLVL controller's uart input buffer, the controller will echo each character as it is received. The PC host must wait for this echo before sending another character. To initiate a command, the PC HOST sends ATTN_CHAR, and waits for the echo. This ATTN_CHAR will sit in the TLVL controller's uart input buffer until the firmware polls it, and calls process_command() to handle the command request. At this time, ATTN_CHAR will be echoed to the PC, informing it that the controller is read to receive the rest of the command sequence.

Normally the controller will respond to ATTN_CHAR fairly quickly, however the delay can be several seconds if you happen to "catch" it during a camera On->Record or Stop->Off sequence. Once ATTN_CHAR is recognized and echoed by the TLV controller, the remaining characters of the command will be processed very quickly.

To reduce the chance that "noise" on the serial line could be mistaken for an ATTN_CHAR, a second ATTN_CHAR1 is required immediately on entering process_command(). If this character is not seen, command processing will be terminated.

Following the sending of ATTN_CHAR1, and receiving its echo, the host will then send the actual command and any operands. In all cases, each character will be echoed by the controller before the next one will be accepted.

Following the execution of a command, the controller will send ERR_ok to signal successful completion of that command. In the event that an error occurs, the controller sends RR_comms, then waits until all data from the PC stops, then sends ERR_ok.

All serial commands consist of a single ASCII character, and all operands consist of one or more byte values, sent as two ASCII-HEX characters for each byte. The individual commands are:

C3.1 I - Identify controller

Sends back a two byte sequence: vvll

vv = Version number of controller firmware
ll = Length of mission data (excluding checksum)

C3.2 N - Power ON

Performs a power-up sequence by calling power_on(). This turns ON the Camera and configured parallel control signals.

C3.3 F - Power OFF

Performs a power-down sequence by calling power_off(). This turns OFF the Camera and configured parallel control signals.

C3.4 Cxxxx - Execute standard LANC camera command

This sends a Control-L/LANC command directly to the camera. This is used by the host to perform most of the camera operation commands (Record, Stop etc.).

C3.5 Rxxxx - Execute repeated LANC camera command

This sends a Control-L/LANC command directly to the camera, and keeps sending it until and ESCAPE character is received from the PC. This is used by the host to perform commands requiring repeated execution (Zoom etc.).

C3.6 Oxx - Write control output

The operand value is written directly to the parallel control output signal latch. This provides an ability to override the control lines at any time.

C3.7 Q - Query camera

Returns the TLV controller and status information. It returns 5 bytes of information to the PC:

xx - TLV controller state(See note below *1)
xx - Camera state as reported by LANC(See note below *2)
xxxxxx - Camera counter as reported by LANC

*1: 00=Mission configuration invalid (checksum error)
01=Waiting for mission start time/date
02=Processing event - Camera is ON
03=Processing event - Camera is OFF
04=Waiting for next event
05=Mission has ended

*2: See SONY "Protocol of Control-L document" for a complete description of the LANC code, or refer to section C5 - Common LANC state codes.

C3.8 T - Query current time/date

Returns the current TLV controller time and date in the following format:
ssmmhhDDMMYY

ss = Second
mm = Minute
hh = Hour
DD = Day
MM = Month
YY = year

C3.9 SssmmhhDDMMYY - Set time/date

Sets the TLV controller time and date. Parameters are the same as the same as 'T' return values described above.

C3.10 A - Query next alarm time

Returns the time of the next TLVL controller event alarm in the following format: ssmmhh

ss = Second
mm = Minute
hh = Hour

C3.11 Bssmmhh - Send next alarm time

Sets the time of the next TLVL controller event alarm. Parameters are the same as 'A' return values described above.

C3.12 U - Upload mission data

Sends the current mission data to the host as a series of byte values. For the exact format of the mission data, refer to the EVT_* #defines in TLVFIRM.C, or the event_struct structure template in TLV.C.

Following the actual mission data a single byte 8-bit checksum of the data is sent.

C3.13 D... - Download mission data

Receives mission data from the host. Format is the same as described under the 'U' command above.

C3.14 Z5A - Restart controller

Restarts and reinitializes the controller. To insure that this does not happen accidentally, the "5A" sequence code is required.

The controller should be restarted when a new mission start time is loaded, or any time that you wish new mission data to be processed immediately.

C4. Common Control-L/LANC command codes
(values are in hex)

-- Camera --	-- VTR --
1127-Begin recording	1128-Double (x2)
1129-Stop recording	1130-Stop
112A-Power OFF	132-Pause
112B-Capture still image	1134-Play
112C-Eject Tape	1136-Rewind
1133-Start/Stop button	1138-FastForward
1139-Capture setup	113A-Record
115E-Alternate Power OFF	1140-Still
1165-Edit reverse	1144-Slow1 (1/10)
1167-Edit forward	1146-Slow2 (1/5)
1169-Record locate	114C-High (Cue)
118E-Zero set memory	1150-Review
119A-Menu ON/OFF	1152-Cue
11A2-Menu Enter	1160-Reverse
11B4-Display ON/OFF	1162-Forward
2135-Zoom in (normal)	1166-Normal (x1)
2137-Zoom out (normal)	1174-Rewind+Play
2139-Zoom in (fast)	118C-Reset Counter
213B-Zoom out (fast)	

C5. Common Control-L/LANC state codes
(values are in hex)

00-Not resp./power off	03-FastForward	74-Video Insert/pause
01-Ejected (Tape out)	33-FF/auto 0 PB	84-Audio Insert/Dub
11-No cassette (DEW)	43-FF/memory stop	94-Audio Insert/pause
21-Eject cass. unload	83-Rewind	06-Playback
31-Eject tape unloaded	A3-RW/TOP auto PB	26-Play x1
02-STOP normal	B3-RW/auto 0 PB	36-Play -x1
12-STOP tape load	C3-RW/memory stop	46-Play/Cue
22-STOP cassette load	04-Recording	56-Play/Review
32-STOP battery down	14-Rec. Pause	66-Play x2/x3
42-STOP unload DEW	24-Timer record	76-Play -x2/-x3
52-STOP EMERGENCY	34-Timer rec. pause	86-Locked Cue/x9
62-STOP tape end	44-A/V insert	96-Locked Review/-x9
72-STOP tape top	54-A/V insert/pause	A6-Frame search queue
92-STOP counter memory	64-Video insert	B6-Frame search review
85-Edit->Forward	95-Edit->Reverse	

C6. FIRMWARE SOURCE CODE

This section contains the source code(both in C and in assembly) for the microcontroller functions described in section C2.

```
* Compile with DDS Micro-C/51 (http://www.dunfield.com)
* Compile command: cc51 TLVFIRM -fop m=T
*
* NOTE: You must make the following modifications to the default
* Micro-C/51 installation files before compiling:
*       8051INT.H                       <- Change INTBASE to 0 (was 2048)
*       LIB51\8051RLPT.ASM  <- Change ORG at beginning to $0000 (was $0800)
*/
#include <8051io.h>                     /* Standard I/O definitions */
#include <8051reg.h>            /* Special Function Registers */
#include <8051bit.h>            /* Bit manipulation macros */
#include <8051int.h>            /* Interrupt wrapper macro */

/*
* Defined constants
*/
/* Hardware I/O ports & pins */
#define LANC_IO                 P3.5            /* LANC interface I/O bit */
#define CONTROL_OUT             P1                      /* Output control port */
#define ENABLE                  (P3&0x08)       /* Enable input line (P3.3) */
#define INHIBIT                 (P3&0x10)       /* P3.4 Inhibit line (P3.4) */

/* Predefined LANC commands */
#define LANC_START 0x1127               /* Begin recording */
#define LANC_STOP   0x1129              /* Stop recording */
#define LANC_SS             0x1133              /* Start/Stop recording */
#define LANC_OFF    0x112A              /* Power OFF */

/* Misc. parameters */
#define CMD_TIMEOUT         2                       /* Timeout for receiving command */
#define ATTN_CHAR   ':'                     /* Attention character for command */
#define ATTN_CHAR1 '~'                      /* Secondary attention character */
#define VERSION                 0x10            /* Version 1.0 */

/* Command responses */
#define ERR_ok                  '.'                     /* Command completed */
#define ERR_cksum       '!'                     /* Checksum error */
#define ERR_comms       '@'                     /* Comms error */

/* Configuration parameters */
#define CONFIG_ESIG 0x01                /* Enable signal */
#define CONFIG_ISIG  0x02               /* Inhibit signal */
#define CONFIG_DINH         0x04                /* Daily inhibit */
#define CONFIG_SS       0x08                /* Single Start/Stop */
#define CONFIG_DBUG         0x80                /* Debug messages */
```

```
/*
 * Event reporting states
 */
#define EVENT_check  0          /* Waiting for valid config */
#define EVENT_start  1          /* Waiting for mission start */
#define EVENT_Con    2          /* Event running - Camera ON */
#define EVENT_Coff   3          /* Event running - Camera OFF */
#define EVENT_sleep  4          /* Sleeping between events */
#define EVENT_end    5          /* Mission has ended */

/*
* Register locations in the real-time clock (DS12887) chip
*/
#define RTC_second    0         /* Second register */
#define RTC_Asecond   1         /* Second alarm register */
#define RTC_minute    2         /* Minute register */
#define RTC_Aminute   3         /* Minute alarm register */
#define RTC_hour      4         /* Hour register */
#define RTC_Ahour     5         /* Hour alarm register */
#define RTC_dow       6         /* Day of week register */
#define RTC_day       7         /* Day register */
#define RTC_month     8         /* Month register */
#define RTC_year      9         /* Year register */
#define RTC_reg_A     10        /* Control register A */
#define RTC_reg_B     11        /* Control register B */
#define RTC_reg_C     12        /* Control register C */
#define RTC_reg_D     13        /* Control register D */
#define RTC_ram       14        /* Beginning of backup RAM */

/*
* Initial values for RTC registers
*/
#define RTC_INIT_A    32        /* 20 Enable Osc. No square wave */
#define RTC_INIT_B    6         /* 06 !SET !PIE !AIE !UIE !SQWE DM, 24 !DSE
*/

/*
* Location of EVENT data in external (RTC) ram
*/
#define EVT_Ssecond   14        /* Second at which mission starts */
#define EVT_Sminute   15        /* Minute at which mission starts */
#define EVT_Shour     16        /* Hour at which mission starts */
#define EVT_Sday      17        /* Day at which mission starts */
#define EVT_Smonth    18        /* Month at which mission starts */
#define EVT_Syear     19        /* Year at which mission starts */
#define EVT_Esecond   20        /* Second at which mission ends */
#define EVT_Eminute   21        /* Minute at which mission ends */
#define EVT_Ehour     22        /* Hour at which mission ends */
#define EVT_Eday      23        /* Day at which mission ends */
#define EVT_Emonth    24        /* Month at which mission ends */
#define EVT_Eyear     25        /* Year at which mission ends */
```

```
#define EVT_ontime          26          /* Interval for event ON (word - sec) */
#define EVT_offtime         28          /* Interval for event OFF (word - 2sec) */
#define EVT_config          30          /* Misc configuration information */
#define EVT_out_A           31          /* Output latch at Idle */
#define EVT_out_B           32          /* Output latch at Transision */
#define EVT_out_C           33          /* Output latch at event end */
#define EVT_delay1          34          /* Delay from Power-ON to Camera start */
#define EVT_delay2          36          /* Delay from Camera stop to Power-off */
#define EVT_ISsecond  38          /* Second of inhibit start time */
#define EVT_ISminute  39          /* Minute of inhibit start time */
#define EVT_IShour          40          /* Hour of inhibit start time */
#define EVT_IEsecond  41          /* Second of inhibit end time */
#define EVT_IEminute  42          /* Minute of inhibit end time */
#define EVT_IEhour          43          /* Hour of inhibit end time */
#define EVT_check           44          /* Checksum to determine valid event */

/*
 * If you wish to remove the debugging option (including the prinff()
 * formatter), change the following to: #define Debug(x)
 *
 * You may also remove this definition, and #include <debug.h>
 * To use the standard Micro-C debug macro.
 */
#define Debug(x) if(peek(EVT_config)&CONFIG_DBUG) printf x;

/*
 * Internal variables for reading/writing RTC time.
 */
register unsigned char
        Second,                     /* Current second */
        Minute,                     /* Current minute */
        Hour,                       /* Current hour */
        Day,                        /* Current day */
        Month,                      /* Current Month */
        Year,                       /* Current year */
        Esecond,                    /* Second event started */
        Eminute,                    /* Minute event started */
        Ehour;                      /* Hour event started */

/*
 * Internal variables for LANC frame exchange
 */
register unsigned char              /* Byte locations */
        RXFLAGS,                    /* Receive portion flags */
        VCRMODE,                    /* Received VCR mode */
        VCRCNT1,                    /* Received VCR Counter */
        VCRCNT2,                    /* Received VCR Counter */
        VCRCNT3;                    /* Received VCR Counter */
register unsigned                   /* Word locations */
        TXWORD,                     /* Transmit code 1 */
        RXWORD;                     /* Received Tuner code 1 */
```

```
/*
 * Misc. internal variables
 */
register unsigned char
        SAVE_SP,                                /* Saved stack pointer for CMD
recovery */
        Alarm_flag,                             /* Alarm has occured */
        Inhibit_mode,                   /* Start<=Stop or Start>Stop */
        Event_state;                    /* Event condition */

/*
 * Indentifier string
 */
static char id_string[] = { "TIme Lapse Video - by Dave Dunfield" };

/*
 * LANC data frame exchange
 *
 * Create separate instances of the handler for each VCR
 * Synchronize to line by waiting for inter-frame gap
 * line must be high for at least 1000 us.
 *
 * Input globals: TXWORD
 * Output globals: RXWORD, RXFLAGS, VCRMODE, VCRSTA*, VCRCNT*
 */
void CLfr(void) asm
{
                MOV     R7,#235         1: 235x4x1.085 = 1020 us
CLfr1   JNB     LANC_IO,CLfr 2: Line is not high - restart
                DJNZ    R7,CLfr1        2: Wait for synchronization
* Output SOURCE/CONTENT code
                MOV     A,TXWORD+1      Prefix code to XMIT
                ACALLCLtx               Output
                MOV     A,TXWORD        Command code to XMIT
                ACALLCLtx               Output
* Receive data from remote device
                ACALLCLrx               SOURCE/CONTENT
                MOV     RXWORD+1,A      Stash it away
                ACALLCLrx               REMOTE CONTROL CODE
                MOV     RXWORD,A        Stash it away
                ACALLCLrx               VCR MODE
                MOV     VCRMODE,A       Stash it away
                ACALLCLrx               VCR STATUS & GUIDE CODE
                ANL     A,#%11110000    Save only guide code
* Save tape counters into their own location
* If 4-digit, save in correct format
                CJNE    A,#$20,CLfr2    Not 4-digit
                ACALLCLrx               Read counter
                MOV     VCRCNT1,A       Stash it away
                ACALLCLrx               Read counter
                MOV     VCRCNT2,A       Stash it away
                ORL     RXFLAGS,#$03    Indicate counter received
```

```
                CLR     A                       Zero ACC
                MOV     VCRCNT3,A               Zero HIGH
                RET
* Record low digits of HMS counter
CLfr2   CJNE    A,#$30,CLfr3    Not HMS low
                ACALLCLrx               Read counter
                MOV     VCRCNT1,A       Stash it away
                ACALLCLrx               Read counter
                MOV     VCRCNT2,A       Stash it away
                ORL     RXFLAGS,#$01 Indicate LOW received
                RET
* Record high digits of HMS counter
CLfr3   CJNE    A,#$40,CLfr4    Not HMS high
                ACALLCLrx               Read counter
                MOV     VCRCNT3,A       Stash it away
                ACALLCLrx               Read counter
                ORL     RXFLAGS,#$02 Indicate HIGH received
                RET
* Read VCR status bytes
CLfr4   ACALLCLrx               Read status
                ACALLCLrx               Read status
                ORL     RXFLAGS,#$04 Indicate status received
                RET
*
* Transmit contents of A by bit-banging
*
CLtx    MOV     R7,#0           Timeout
        MOV     R6,#0           Timeout
CLtx1   JNB     LANC_IO,CLtx2   Found start
        DJNZ    R6,CLtx1        No, keep looking
        DJNZ    R7,CLtx1        Maximum timeout
CLerr   MOV     RXFLAGS,#$FF Indicate error condition
        RET
CLtx2   MOV     R6,#8           1: TX 8 bits
* Skip start bit
        MOV     R7,#44          1: ((R7*2)+8)x1.085 = 104us
        DJNZ    R7,*            2: Wait for time
* Shift out each bit
CLtx3   RRC     A               1: Get low bit into C
        CPL     C               1: Invert
        MOV     LANC_IO,C       2: Write the bit
        MOV     R7,#44          1: ((R7*2)+7)x1.085 = 104us
        DJNZ    R7,*            2: Wait for time
        DJNZ    R6,CLtx3        2: Shift all bits
* Set line high for end
        SETB    LANC_IO         1: Set line high
        RET
*
* Receive a byte from the line by bit-banging
*
CLrx    JNB     LANC_IO,*       Wait for line high
        MOV     R7,#0           Timeout counter
```

```
                MOV     R6,#0           Timeout counter
CLrx1   JNB     LANC_IO,CLrx2   1: Found start
                DJNZ    R6,CLrx1        Keep looking
                DJNZ    R7,CLrx1        Keep looking
                SJMP    CLerr           Report error
* Skip start bit
CLrx2   MOV     R7,#70          1: ((R7*2)+3)x1.085 = 156us
                MOV     R6,#8           1: RX 8 bits
CLrx3   DJNZ    R7,*            2: Wait for time
* Read a bit
                MOV     C,LANC_IO       1: Get the bit
                CPL     C               1: Invert
                RRC     A               1: Shift into ACC
                MOV     R7,#45          1: ((R7*2)+6)x1.085 = 104 us
                DJNZ    R6,CLrx3        2: Do it again
}

/*
 * Transmit a command to the LANC interface
 *
 * Input globals: TXWORD
 */
void lanc_cmd(void)
{
        unsigned i;

        RXFLAGS = 0;                    /* Clear flags */
        i = 10;                         /* Send frame at least 10 times */
        do {
                do {
                        CLfr();         /* Send frame */
                        if(RXFLAGS & 0x80)
                                break; }
                while(--i);
                i = 1; }
        while((RXFLAGS & 0x07) != 0x07);
}

/*
 * Read the real time clock (current time)
 *
 * Output globals: Second, Minute, Hour, Day, Month, Year
 */
void read_time(void) asm
{
                MOV     R0,#RTC_reg_A       Read register A
                MOVX    A,[R0]          Read register
                RLC     A               Test UIP bit
                JC      read_time       Wait for interval
                MOV     R0,#RTC_second  Point at seconds
                MOVX    A,[R0]          Read the value
                MOV     Second,A        Write internal copy
```

```
                MOV     R0,#RTC_minute      Point at minute
                MOVX    A,[R0]              Read the value
                MOV     Minute,A                Write internal copy
                MOV     R0,#RTC_hour        Point at hour
                MOVX    A,[R0]              Read the value
                MOV     Hour,A                  Write internal copy
                MOV     R0,#RTC_day             Point at day
                MOVX    A,[R0]              Read the value
                MOV     Day,A                   Write internal copy
                MOV     R0,#RTC_month           Point at month
                MOVX    A,[R0]              Read the value
                MOV     Month,A                         Write internal copy
                MOV     R0,#RTC_year        Point at year
                MOVX    A,[R0]              Read the value
                MOV     Year,A                  Write internal copy
}

/*
 * Set the real time clock (current time)
 *
 * Input globals: Second, Minute, Hour, Day, Month, Year
 */
void set_time(void) asm
{
                MOV     R1,#RTC_reg_B       Point to control register
                MOV     A,#RTC_INIT_B | $80 Enable SET bit
                MOVX    [R1],A              Write it
                MOV     R0,#RTC_second          Point to second
                MOV     A,Second                Get second value
                MOVX    [R0],A              Write it
                MOV     R0,#RTC_minute          Point to minute
                MOV     A,Minute                Get minute value
                MOVX    [R0],A              Write it
                MOV     R0,#RTC_hour        Point to hour
                MOV     A,Hour                  Get hour value
                MOVX    [R0],A              Write it
                MOV     R0,#RTC_day             Point to day
                MOV     A,Day                   Get day value
                MOVX    [R0],A              Write it
                MOV     R0,#RTC_month           Point to month
                MOV     A,Month                         Get month value
                MOVX    [R0],A              Write it
                MOV     R0,#RTC_year        Point to year
                MOV     A,Year                  Get year value
                MOVX    [R0],A              Write it
                MOV     A,#RTC_INIT_B           Disable SET bit
                MOVX    [R1],A              Write it
}

/*
 * Set RTC alarm expiry time
 *
```

```
 * Input globals: Second, Minute, Hour
 */
void set_alarm(void) asm
{
                MOV     R0,#RTC_Asecond         Point to second alarm
                MOV     A,Second                Get value
                MOVX    [R0],A                  Write it
                MOV     R0,#RTC_Aminute         Point to minute alarm
                MOV     A,Minute                Get value
                MOVX    [R0],A                  Write it
                MOV     R0,#RTC_Ahour           Get hour value
                MOV     A,Hour                  Get value
                MOVX    [R0],A                  Write it
                MOV     R0,#RTC_reg_B           Point to control register
                MOV     A,#RTC_INIT_B | $20     Set AIE bit
                MOVX    [R0],A                  Write the bit
}

/*
 * Calculate checksum of event data
 *
 * Input: RTC event data from 'RTC_ram' to 'EVT_check'
 * Return: 8 bit checksum of event data
 */
unsigned event_checksum(void)
{
        unsigned i;
        unsigned char c;
        c = 0;
        for(i=RTC_ram; i < EVT_check; ++i)
                c += peek(i);
        return c;
}


/*
 * Go to sleep
 *
 * Setup for Serial or Alarm interrupt, then place the
 * processor in idle mode, which will wait for either
 * of these events to occur.
 */
void sleep(void) asm
{
                CLR     SCON.1                  Clear TX interrupt
                CLR     SCON.0                  Clear RX interrupt
                CLR     A                       Get a zero
                MOV     Alarm_flag,A            Zero flag
                MOV     IP,A                    No priority
                MOV     IE,#%10010001           Enable Serial & Alarm ints
                MOV     PCON,#$01               Set IDLE mode
                CLR     IE.7                    Disable interrupts
```

```
                SETB    SCON.1                          Re-enable transmitter
    }

    /*
     * Compare current date/time against a configured date/time
     *
     * Input globals: Year, Month, Day, Hour, Minute, Second
     * Input: date (RTC ram location) to comare with
     * Return:
     *          -1      = current <  date/time
     *                  0       = current == date/time
     *                  1       = current >  date/time
     */
    int compare_date(unsigned char date)
    {
            unsigned char x, x1;

            if((x = Year) == (x1 = peek(date+5)))
                    if((x = Month) == (x1 = peek(date+4)))
                            if((x = Day) == (x1 = peek(date+3)))
                                    if((x = Hour) == (x1 = peek(date+2)))
                                            if((x = Minute) == (x1 = peek(date+1)))
                                                    if((x = Second) == (x1 = peek(date)))
                                                            return 0;

            return (x < x1) ? -1 : 1;
    }

    /*
     * Compare current time against a specified time
     *
     * Input globals: Hour, Minute, Second
     * Input: h, m, s
     * Return:
     *          -1      = current <  time
     *                  0       = current == time
     *                  1       = current >  time
     */
    int compare_time(unsigned char s, unsigned char m, unsigned char h)
    {
            unsigned char x, x1;

            if((x = Hour) == (x1 = peek(h)))
                    if((x = Minute) == (x1 = peek(m)))
                            if((x = Second) == (x1 = peek(s)))
                                    return 0;

            return (x < x1) ? -1 : 1;
    }

    /*
    * Set the daily inhibit mode
```

```
 *
 * Input: Daily inhibit information from RTC CMOS
 * Output globals: Inhibit_mode
 */
void set_inhibit_mode()
{
        Second = peek(EVT_ISsecond);
        Minute = peek(EVT_ISminute);
        Hour   = peek(EVT_IShour);
        Inhibit_mode = (compare_time(EVT_IEsecond, EVT_IEminute, EVT_IEhour) <= 0) ? -1 :
0;
}

/*
 * Determine if the camera should be enabled
 *
 * Input globals: Hour, Minute, Second, Inhibit_mode
 * Return: 0 if camera disabled, !0 if camera enabled
 */
int camera_enabled()
{
        unsigned char config;

        config = peek(EVT_config);

        if(config & CONFIG_ESIG) {                      /* Enable signal */
                if(!ENABLE)
                        return 0; }

        if(config & CONFIG_ISIG) {                      /* Inhibit signal */
                if(INHIBIT)
                        return 0; }

        if(config & CONFIG_DINH) {                      /* Daily inhibit period */
                if(Inhibit_mode) {
                        /*
                         * Inhibit_start is <= Inhibit_end, camera should be ON
                         * when time < inhibit start or > inhibit_end
                         */
                        if(compare_time(EVT_ISsecond, EVT_ISminute, EVT_IShour) < 0)
                                return -1;
                        if(compare_time(EVT_IEsecond, EVT_IEminute, EVT_IEhour) > 0)
                                return -1;
                        return 0; }
                /*
                 * Inhibit_start > Inhibit_end, camera should be OFF when
                 * time > inhibit_start or < inhibit_end
                 */
                if(compare_time(EVT_ISsecond, EVT_ISminute, EVT_IShour) > 0)
                        return 0;
                if(compare_time(EVT_IEsecond, EVT_IEminute, EVT_IEhour) < 0)
                        return 0; }
```

```
            return -1;
    }

    /*
     * Turn on power to the camera, and any lights etc.
     */
    void power_on(void)
    {
            unsigned i, j, k;
            unsigned char p1, p2;

            /* Power on camera by momentary ground of LANC control line */
            clrbit(LANC_IO);
            delay(250);
            setbit(LANC_IO);

            CONTROL_OUT = p1 = peek(EVT_out_B);      /* Get initial ON settings */
            p2 = peek(EVT_out_C);                    /* Get final ON settings */

            if(p1 != p2) {     /* If different, do PWM from initial to final */
                    i = 64;
                    do {
                            CONTROL_OUT = p1;
                            k = i >> 2;
                            for(j=0; j < 16; ++j) {
                                    if(j == k)
                                            CONTROL_OUT = p2;
                                    delay(1); } }
                    while(--i); }
    }

    /*
     * Turn off power to camera and lights
     */
    void power_off(void)
    {
            CONTROL_OUT = peek(EVT_out_A);
            TXWORD = LANC_OFF;
            lanc_cmd();
    }

    /*
     * Get character from serial port with 1-2 second timeout delay
     *
     * Terminate command with error if timeout.
     * Input: Serial data
     * Returns: 0-255 (character received)
     */
    unsigned char gett(void) asm
    {
                    MOV             R0,#RTC_second          Point to seconds
```

```
                MOVX  A,[R0]                    Get current seconds
                MOV           R1,A                    Save for later
                MOV           R2,#CMD_TIMEOUT         Delay for timeout
gett1   JNB           SCON.0,gett2   No key, keep going
                CLR           SCON.0                          Indicate received
                MOV           A,SBUF                  Get the key
                JNB           SCON.1,*                Wait for TX ready
                CLR           SCON.1                          Reset TX
                MOV           SBUF,A                  Echo character
                MOV           B,#0                    Zero high
                RET
gett2   MOVX  A,[R0]                    Read seconds
                XRL           A,R1                    Calculate difference
                JZ            gett1                   Same, keep going
                MOVX  A,[R0]                    Get value back
                MOV           R1,A                    Save it
                DJNZ  R2,gett1                  Wait for expiry
                LJMP  Error                     Report error
}

/*
 * Get HEX value from serial port
 *
 * Terminate command with error if invalid.
 * Input: Serial data
 * Return: 0-255 (byte value)
 */
unsigned geth(void)
{
        unsigned h, l;
        h = gett() - '0';
        if( (h >= ('A'-'0')) && (h <= ('F'-'0')) )
                h -= 7;
        else if(h >= 10)
                Error();
        l = gett() - '0';
        if( (l >= ('A'-'0')) && (l <= ('F'-'0')) )
                l -= 7;
        else if(l >= 10)
                Error();
        return (h << 4) | l;
}

/*
 * Write a value in hex to the serial port
 *
 * Input: character value to write (0-255)
 */
void puth(unsigned char v)
{
        unsigned c;
        c = (v >> 4) + '0';
```

```
        if(c > '9') c += 7;
        putchr(c);
        c = (v & 0x0F) + '0';
        if(c > '9') c += 7;
        putchr(c);
}

/*
 * Process a command from the serial port
 *
 * Input: Serial data
 * Output: Calls functions based on command
 */
void process_command(void)
{
        unsigned i;

        asm " MOV SAVE_SP,SP";

        i = 0;
again:
        putchr(ATTN_CHAR);
        switch(gett()) {
        case ATTN_CHAR :
                goto again;
        default:
                Error();
        case ATTN_CHAR1 : }

        switch(gett()) {
        case ATTN_CHAR:                         /* Ignore repeated ATTN characters */
                goto again;
        case 'N' :                              /* Turn power on */
                power_on();
                break;
        case 'F' :                              /* Turn power off */
                power_off();
                break;
        case 'C' :                              /* Issue a command to the camera */
                TXWORD = (geth() << 8) | geth();
                lanc_cmd();
                break;
        case 'R' :                              /* Issue a repeat command to the camera */
                TXWORD = (geth() << 8) | geth();
                putchr(ERR_ok);
                do {
                        CLfr(); }
                while(chkchr() != 0x1B);
                break;
        case 'Q' :                              /* Query camera */
                TXWORD = 0;
                puth(Event_state);
```

```
                lanc_cmd();
                puth(VCRMODE);
                puth(VCRCNT3);
                puth(VCRCNT2);
                puth(VCRCNT1);
                break;
        case 'T' :                              /* Query current time */
                read_time();
                puth(Second);
                puth(Minute);
                puth(Hour);
                puth(Day);
                puth(Month);
                puth(Year);
                break;
        case 'S' :                              /* Set current time */
                Second = geth();
                Minute = geth();
                Hour = geth();
                Day = geth();
                Month = geth();
                Year = geth();
                set_time();
                break;
        case 'A' :                              /* Send current alarm time */
                puth(peek(RTC_Asecond));
                puth(peek(RTC_Aminute));
                puth(peek(RTC_Ahour));
                break;
        case 'B' :                              /* Set alarm time */
                Second = geth();
                Minute = geth();
                Hour = geth();
                set_alarm();
                break;
        case 'U' :                              /* Upload CMOS data */
                i = RTC_ram;
                do
                        puth(peek(i));
                while(++i <= EVT_check);
                break;
        case 'D' :                              /* Download CMOS data */
                i = RTC_ram;
                do
                        poke(i, geth());
                while(++i <= EVT_check);
                if(event_checksum() != peek(EVT_check))
                        putchr(ERR_cksum);
                set_inhibit_mode();
                break;
        case 'O' :                              /* Write output signals */
                CONTROL_OUT = geth();
```

```
                break;
        case 'T' :                              /* Identify */
                puth(VERSION);
                puth(EVT_check - RTC_ram);
                break;
        case 'Z' :                                      /* Restart */
                if(geth() == 0x5A) {
                        putchr(ERR_ok);
                        asm " LJMP $0000"; }
        default:
                Error(); }

        putchr(ERR_ok);
        return;

        /*
        * Error entry point, restore stack, issue error message & return
        */
        asm "Error: MOV SP,SAVE_SP";
        putchr(ERR_comms);
flush_comms:
        for(i=0; i < 10000; ++i)
                if(chkchr() != -1)
                        goto flush_comms;
        putchr(ERR_ok);
}

/*
* Wait for indicated on-time (from mission data)
* Insure that camera enabled conditions are honored during this time.
*/
void wait_on_time(void)
{
        unsigned seconds_remaining;
        unsigned char last_sec;

        set_inhibit_mode();

        seconds_remaining = peekw(EVT_ontime);
        Debug(("Wait on time: %u, Imode=%u\n", seconds_remaining, Inhibit_mode))

        read_time();
        last_sec = Second;

        do {
                if(chkchr() == ':')                     /* Process pending commands */
                        process_command();

                do {                                            /* Wait for a second to pass */
                        delay(300);
                        read_time(); }
                while(Second == last_sec);
```

```
                if(++last_sec >= 60)            /* Advance to next second */
                        last_sec = 0;

                if(camera_enabled()) {          /* Camera should be ON */
                        if(Event_state != EVENT_Con) {          /* But it's OFF ... */
                                Event_state = EVENT_Con;
                                /* Turn power on and wait startup delay */
                                Debug(("Power on, delay: %u\n", peekw(EVT_delay1)))
                                power_on();
                                delay(peekw(EVT_delay1));

                                /* Start camera */
                                Debug(("Start camera\n"))
                                TXWORD = (peek(EVT_config) & CONFIG_SS) ? LANC_SS :
 LANC_START;
                                lanc_cmd(); } }
                        else {                                          /* Camera should be OFF */
                                if(Event_state != EVENT_Coff) {                 /* But it's ON ... */
                                        Event_state = EVENT_Coff;
                                        /* Stop camera */
                                        Debug(("Stop camera\n"))
                                        TXWORD = (peek(EVT_config) & CONFIG_SS) ? LANC_SS :
 LANC_STOP;
                                        lanc_cmd();

                                        /* Wait for settling delay and power off */
                                        Debug(("Power off, delay %u\n", peekw(EVT_delay2)))
                                        delay(peekw(EVT_delay2));
                                        power_off(); } } }
                while(--seconds_remaining);
}

/*
 * Setup the next alarm using current event start time and mission data.
 */
void set_next_alarm(void)
{
        unsigned t;

        t = peekw(EVT_offtime);

        /* Compute time until next event */
        Hour = t / 1800;
        Minute = (t % 1800) / 30;
        Second = (t % 30) * 2;

        /* Add to event start time */
        if((Second += Esecond) >= 60) {
                Minute += Second / 60;
                Second %= 60; }
```

```
        if((Minute += Eminute) >= 60) {
                Hour += Minute / 60;
                Minute %= 60; }

        if((Hour += Ehour) >= 24)
                Hour %= 24;

        Debug(("Set alarm: %02u:%02u:%02u\n", Hour, Minute, Second))
        set_alarm();
}

/*
 * Main program
 */
main()
{
        CONTROL_OUT = 0;                /* All outputs off till we get going */

        serinit(9600);

        /* Initialize the RTC chip */
        poke(RTC_reg_A, RTC_INIT_A);
        poke(RTC_reg_B, RTC_INIT_B);

        delay(100);

        while(event_checksum() != peek(EVT_check)) {
                Event_state = EVENT_check;
                putchr(ERR_cksum);
                power_off();
                if(getchr() == ATTN_CHAR)
                        process_command(); }
        Debug(("Checksum passed\n"))

        Event_state = EVENT_start;

        CONTROL_OUT = peek(EVT_out_A);

        set_inhibit_mode();

        /* Wait for event start period (current time >= event_time) */
        read_time();
        if(compare_date(EVT_Ssecond) < 0) {
                power_off();
                /* Set initial alarm based on event time */
                Hour = peek(EVT_Shour);
                Minute = peek(EVT_Sminute);
                Second = peek(EVT_Ssecond);
                Debug(("Initial alarm: %02u:%02u:%02u\n", Hour, Minute, Second))
                set_alarm();
                do {
                        sleep();
```

```
                        if(chkchr() == ATTN_CHAR)
                                process_command();

                        read_time(); }
                while(compare_date(EVT_Ssecond) < 0); }
        else {
                set_alarm();
                Alarm_flag = -1; }

        Debug(("First event\n"))

        /* Process events until we are past event end period */
        while(compare_date(EVT_Esecond) <= 0) {
                Debug(("Top of while\n"))
                /* Ignore spurious interrupts */
                if(Alarm_flag || !compare_time(RTC_Asecond, RTC_Aminute, RTC_Ahour)) {

                        Event_state = EVENT_Coff;
                        Ehour = Hour;
                        Eminute = Minute;
                        Esecond = Second;
                        Debug(("Event start: %02u:%02u:%02u\n", Ehour, Eminute, Esecond))

                        if(camera_enabled()) {
                                Event_state = EVENT_Con;
                                /* Turn power on and wait startup delay */
                                Debug(("Power on, delay: %u\n", peekw(EVT_delay1)))
                                power_on();
                                delay(peekw(EVT_delay1));

                                /* Start camera */
                                Debug(("Start camera\n"))
                                TXWORD = (peek(EVT_config) & CONFIG_SS) ? LANC_SS :
LANC_START;
                                lanc_cmd(); }

                        /* Wait for on-time to expire */
                        do
                                wait_on_time();
                        while(!peek(EVT_offtime));

                        /* Stop camera */
                        Debug(("Stop camera\n"))
                        TXWORD = (peek(EVT_config) & CONFIG_SS) ? LANC_SS :
LANC_STOP;
                        lanc_cmd();

                        /* Wait for settling delay and power off */
                        Debug(("Power off, delay %u\n", peekw(EVT_delay2)))
                        delay(peekw(EVT_delay2));
                        power_off();
```

```
                        set_next_alarm(); }

                Event_state = EVENT_sleep;
                Debug(("Sleep\n"))
                delay(50);
                sleep();

                if(chkchr() == ATTN_CHAR)
                        process_command();

                read_time(); }

        /*
         * Mission has ended, just process commands
         */
        Debug(("End of mission\n"))
        Event_state = EVENT_end;
        power_off();
        for(;;) {
                sleep();
                if(chkchr() == ATTN_CHAR)
                        process_command(); }
}

/*
 * Alarm interrupt handler
 */
INTERRUPT(_IE0_) alarm_irq()
{
        peek(RTC_reg_C);                        /* Clear RTC interrupt */
        Alarm_flag = -1;
}

/*
* Serial interrupt handler
*/
INTERRUPT(_SER_) serial_irq()
{
        /* Do nothing */
}
```

C7. PC HOST COMPUTER CODE

This section contains the source code listing for the PC host (executed as TLV.COM). The host software functionality is described in section A3. In simple terms it is used to control the TLVL system during test and deployment and to configure the microcontroller for a mission.

```
/*
 * Time Lapse Video (TLV) controller - HOST interface
 *

* Compile with DDS Micro-C/PC (http://www.dunfield.com)
 * Compile command: cc TLV -fop [DEMO=]
 */

#include <stdio.h>                              /* Standard I/O definitions */
#include <comm.h>                               /* Communications definitions
*/
#include <window.h>                             /* Windowing definitions */
#ifdef DEMO
        #include "comfake.c"                    /* Simulate serial comms */
#endif


/* Version information */
#define VERSION_MA 1                            /* Major version number */
#define VERSION_MI  0                           /* Minor version number */

/* Serial command processing definitions */
#define ATTN_CHAR  ':'                          /* Command wakeup character
*/
#define ATTN_CHAR1 '~'                          /* Secondary attn character */
#define TERM_CHAR  '.'                          /* Command termination
character */
#define ATTN_TIME   36                          /* Time to wait for ATTN (2
sec) */
#define CHAR_TIME   5                           /* Time to wait for char (0.3
sec) */
#define ONOFF_TIME  18*10                       /* Time to wait for Power ON/OFF */
#define DONE_STR    " [Done]"                   /* String displayed when done */
#define CAN_STR             " [Canceled]"       /* String displayed when cancel */

/* Window locations */
#define MAIN_WIN    4                           /* Location of main window */
#define STAT_WIN    20                          /* Location of window split */

/* Help subsystem parameters */
#define LINK_S          ('N'-0x40)              /* Start of a link */
#define LINK_E                  ('O'-0x40)      /* End of a link */
#define TAB_SIZE    4                           /* Spacing of TABs */
```

```
#define LINKS           25                      /* Maximum # links per screen
*/
#define HELP_INDEX  0                           /* Help index screen */
#define HELP_MANUAL     1                               /* Main help screen */
#define HELP_VIEW   2                           /* Help with viewing */
#define HELP_EDIT   3                           /* Help with editing */
#define HELP_BIT    4                           /* Editing a bitfield */
#define HELP_MENU   5                           /* Selecting from a menu */
#define HELP_GETS   6                           /* Inputting a string */

/* Bit definitions in EVT_config */
#define CONFIG_ESIG 0x01                        /* Enable signal */
#define CONFIG_ISIG  0x02                       /* Inhibit signal */
#define CONFIG_DINH         0x04                        /* Daily inhibit */
#define CONFIG_SS   0x08                        /* Start/Stop */

/* Misc. definitions */
#define BIOS_TICK       peekw(0x40,0x6C)        /* Read BIOS tick count */


/*
* Structure of a mission event
*/
struct event_struct {
        unsigned char EVT_Ssecond;              /* Second at which mission starts */
        unsigned char EVT_Sminute;              /* Minute at which mission starts */
        unsigned char EVT_Shour;                /* Hour at which mission starts */
        unsigned char EVT_Sday;                         /* Day at which mission starts */
        unsigned char EVT_Smonth;               /* Month at which mission starts */
        unsigned char EVT_Syear;                /* Year at which mission starts */
        unsigned char EVT_Esecond;              /* Second at which mission ends */
        unsigned char EVT_Eminute;              /* Minute at which mission ends */
        unsigned char EVT_Ehour;                /* Hour at which mission ends */
        unsigned char EVT_Eday;                         /* Day at which mission ends */
        unsigned char EVT_Emonth;               /* Month at which mission ends */
        unsigned char EVT_Eyear;                /* Year at which mission ends */
        unsigned int  EVT_ontime;               /* Interval for event ON (seconds) */
        unsigned int  EVT_offtime;              /* Interval for event OFF (two seconds) */
        unsigned char EVT_config;               /* Misc configuration information */
        unsigned char EVT_out_A;                /* Output latch at Idle */
        unsigned char EVT_out_B;                /* Output latch at Transision */
        unsigned char EVT_out_C;                /* Output latch at event end */
        unsigned int  EVT_delay1;               /* Delay from Power-ON to Camera start */
        unsigned int  EVT_delay2;               /* Delay from Camera stop to Power-off */
        unsigned char EVT_ISsecond;             /* Second for inhibit start */
        unsigned char EVT_ISminute;             /* Minute for inhibit start */
        unsigned char EVT_IShour;               /* Hour for inhibit start */
        unsigned char EVT_IEsecond;             /* Second for inhibit end */
        unsigned char EVT_IEminute;             /* Minute for inhibit end */
        unsigned char EVT_IEhour;               /* Hour for inhibit end */
        unsigned char EVT_check;                /* Checksum to determine valid event */
        } event;
```

```
/*
 * Bit descriptions for bitfield parameters
 */
static char *config_long[] = {              /* Configuration bitfield */
        "Enable control signal",
        "Inhibit control signal",
        "Daily inhibit period",
        "Single Start/Stop (older cameras)",
        0, 0, 0,
        "Debug TTY messages" };
static char *config_short[] = { "ENsig", "INsig", "INtime", "SglSS", 0, 0, 0, "Dbug" };

static char *io_long[] = {                  /* I/O bits */
        "Main lights",
        "Unassigned",
        "Unassigned",
        "Unassigned",
        "Unassigned",
        "Unassigned",
        "Unassigned",
        "Unassigned" };
static char *io_short[] = { "Light", "1", "2", "3", "4", "5", "6", "7" };


/*
 * Screen prompt texts [for prompt()]
 */
static char manual_prompt[] = {                 /* Manual entry screen */
        " F1=Help  F10=Exit" };
static char view_prompt[] = {                   /* Mission viewing screen */
        " F1=Help  F2=Reload  ESC=Exit  SPACE=Edit" };
static char edit_prompt[] = {                   /* Mission editing screen */
        " F1=Help  ESC=Exit  \x18\x19=Move  ENTER=Change  F2=Reload  F3=Save
F4=Clear" };
static char form_prompt[] = {                   /* Form entry */
        " F1=Help  ESC=Quit  ENTER=Next/exit  \x1B\x1A=Fwd/Back-Char
\x19\x18=Fwd/Back-Field" };
static char bitfield_prompt[] = {           /* Bitfield entry */
        " F1=Help  ESC=Exit ENTER=Save  0-7=Toggle bit" };
static char menu_prompt[] = {                   /* Menu selection */
        " F1=Help  ESC=Quit  ENTER=Select  \x19\x18=Move" };
static char help_prompt[] = {                   /* Help screen */
        " ESC=Quit  ENTER=Select  \x1B\x19\x18\x1A=Move" };
static char message_prompt[] = {                /* Message box */
        " F1=Help  ENTER=Continue" };

/*
 * Warning messages [for message_box()]
 */
static char overtime_warning[] = {              /* On time exceeds interval */
```

```
"***WARNING***\n\
On-time + On->Record delay + Stop->Off delay  exceeds event\n\
interval time. This means the next event will have arrived\n\
BEFORE the current event finishes. This will cause the next\n\
event to be delayed by 24 HOURS!" };
static char zero_on_warning[] = {                    /* On time is zero */
"***WARNING***\n\
On-time is zero, which means this event will have the MAXIMUM\n\
ON time of 18:12:16" };
static char interval_warning[] = {                   /* Interval is zero */
"***WARNING***\n\
Event-Interval is zero, which means that this event will NEVER\n\
terminate.  Once the mission starts, the camera will operate\n\
continuously,  however it will still be switched on and off in\n\
response to  Enable/Inhibit signals + the daily-inhibit period\n\
if these options have been configured." };
static char checksum_warning[] = {                   /* Mission checksum failure */
"***WARNING***\n\
The mission data has failed it's validity test. This may be due\n\
to a communications error... Try pressing F2 to reload the data.\n\n\
If the controller does not contain valid mission data, you should\n\
press SPACE to edit, then F4 to clear the data before editing." };

/*
 * Data entry forms [used by wform()]
 */
static char *date_form[] = {                         /* Date/Time entry form */
        33<<8 | 4,
        "\x01\x00\x82Date (dd/mm/yy):",
        "\x15\x00\x82/",
        "\x1A\x00\x82/",
        "\x01\x01\x82Time (hh:mm:ss):",
        "\x15\x01\x82:",
        "\x1A\x01\x82:",
        0 };
static char *time_form[] = {                         /* Time only entry form */
        33<<8 | 3,
        "\x01\x00\x82Time (hh:mm:ss):",
        "\x15\x00\x82:",
        "\x1A\x00\x82:",
        0 };
static char *value_form[] = {                        /* General value entry form */
        21<<8 | 3,
        "\x01\x00\x85Value",
        0 };


/*
 * Selection menus [for wmenu()]
 */
static char *time_menu[] = {                         /* Time selection menu */
        "Set to local host time",
```

```
            "Prompt for specific time",
            0 };
    static char *edit_menu[] = {                    /* Edit exit menu */
            "Save changes before exiting",
            "Quit and lose changes",
            "Continue editing",
            0 };
    static char *restart_menu[] = {                 /* Confirm restart */
            "Do NOT restart (cancel)",
            "Restart remote (confirm)",
            0 };


    /*
    * Command menus for manual control screen
    */
    static unsigned char *manual_cmds1[]={          /* Simple commands */
            "B\x11\x27Begin recording",
            "S\x11\x29Stop recording",
            "X\x11\x33Start/Stop button",
            "C\x11\x2BCapture still image",
            "D\x11\xB4Display ON/OFF",
            "L\x11\x69Record locate",
            "F\x11\x67+edit Forward",
            "R\x11\x65+edit Reverse",
            "]\x21\x35+Zoom in",
            "[\x21\x37+Zoom out",
            "}\x21\x39+Fast Zoom In",
            "{\x21\x3B+Fast Zoom Out",
            0 };
    static char *manual_cmds2[] = {                 /* Special case commands */
            "T  - Display time",
            "O  - Output control",
            "?  - Display unit status",
            "F2 - Power ON",
            "F3 - Power OFF",
            "F4 - Set camera time",
            "F5 - Set next event time",
            "F7 - TTY mode",
            "F8 - Edit mission",
            "F9 - Restart unit",
            0 };


/*
* Video attributes
*/
unsigned char C_color[] = {                         /* Color attributes */
            0x17,                                       /* 0 Main screen */
            0x67,                                       /* 1 Message window */
            0x47,                                       /* 2 Error window */
            0x70,                                       /* 3 Prompt line */
```

```
        REVERSE,                                        /* 4 Pop-up menus */
        0x70,                                           /* 5 Help text */
        0x71 };                                         /* 6 Help links */
unsigned char M_color[] = {                     /* Monochrome attributes */
        NORMAL,                                                 /* 0 Main screen */
        NORMAL,                                                 /* 1 Message window
*/
        REVERSE,                                        /* 2 Error window */
        REVERSE,                                        /* 3 Prompt line */
        REVERSE,                                        /* 4 Pop-up menus */
        NORMAL,                                                 /* 5 Help text */
        0x0F };                                         /* 6 Help links */
unsigned char *color = &C_color;        /* Selects active color scheme */


/*
 * Command line help text
 */
char help_text[] = {
"\nTime Lapse Video (TLV) controller v"#VERSION_MA"."#VERSION_MI"\n\
Copyright 2000 Dave Dunfield.\n\n\
Use:    TLV [options]\n\n\
opts:   C=[1-4]         - Set COM port\n\
        /M              - Force Monochrome display\n\
\n" };


/*
 * Window handles
 */
static struct WINDOW
        *win,                                           /* Main screen */
        *mwin,                                          /* Message window */
        *pwin,                                          /* Prompt window */
        *ewin;                                          /* Error window */


/*
 * Misc. global variables
 */
static char
        Helpfile[100];                          /* Help file location */

static unsigned
        Cbwait = CHAR_TIME;                     /* Time to wait for HEX data */


#include "tlvwin.c"                     /* Use custom windowing library */


/*
* Formatted print to message window
```

```
 *
 * Input: Format string, Variable argument list
 */
void register message(unsigned args)
{
        char buffer[81];

        _format_(nargs() * 2 + &args, buffer);
        w_putc('\n', mwin);
        w_puts(buffer, mwin);
}

/*
 * Formatted print to error window
 *
 * Input: Format string, Variable argument list
 */
void register error(unsigned args)
{
        char buffer[81];

        _format_(nargs() * 2 + &args, buffer);
        w_putc('\n', ewin);
        w_puts(buffer, ewin);
}

/*
 * Set prompt to formatted text
 *
 * Input: Format string, Variable argument list
 */
void register prompt(unsigned args)
{
        char buffer[81];

        _format_(nargs() * 2 + &args, buffer);
        w_gotoxy(0, 0, pwin);
        w_puts(buffer, pwin);
        w_cleow(pwin);
}

/*
 * Simple help system with links
 *
 * Input: Section number to display
 */
void help(unsigned section)
{
        int c, lx, ly, ox, oy;
        unsigned i, j, size, ncount;
        unsigned char buffer[65], *ptr;
        unsigned char xs[LINKS], ys[LINKS], names[LINKS][25], link[LINKS];
```

```
        FILE *fp;

        if(!(fp = fopen(Helpfile, "rb"))) {
                message("Cannot open help file");
                return; }

        prompt(help_prompt);

        /* Locate the section in question */
lookup:
        size = getc(fp);
        size |= getc(fp) << 8;
        if(section--) {
                fseek(fp, 0, size, 1);
                goto lookup; }

        /* Draw the screen */
        wopen(5, 2, 70, 17, WSAVE | WCOPEN | WBOX1 | color[5]);
        wcursor_off();
        i = ncount = ptr = 0;
        while(buffer[i++] = getc(fp));

        /* Make a title */
        memcpy(buffer+50, W_OPEN, 6);
        W_OPEN[1] &= (~WBOX3 >> 8);
        --W_OPEN[3];
        wgotoxy(0,0);
        wprintf(" %s ", buffer);
        memcpy(W_OPEN, buffer+50, 6);

        /* Display help text & record links */
        wgotoxy(0, 0);
        while(i++ < size) switch(c = getc(fp)) {
        case LINK_S :   /* Start link */
                ptr = names[ncount];
                xs[ncount] = W_OPEN->WINcurx;
                ys[ncount] = W_OPEN->WINcury;
                *W_OPEN = color[6];
                break;
        case LINK_E :   /* End link */
                link[ncount++] = getc(fp);
                *ptr = ptr = 0;
                *W_OPEN = color[5];
                ++i;
                break;
        case '\t' :                     /* Horizontal TAB */
                do
                        wputc(' ');
                while(W_OPEN->WINcurx % TAB_SIZE);
                break;
        case 0 :                /* End of line */
                c = '\n';
```

```
        default:
                wputc(c);
                if(ptr)
                        *ptr++ = c; }

        /* Allow user to select field & link to new screen */
        i = section = 0;
        for(;;) {
                wgotoxy(xs[section], ys[section]);
                *W_OPEN = color[6];
                wputs(names[section]);
                wgotoxy(xs[i], ys[i]);
                *W_OPEN = (color[6] << 4) | (color[6] >> 4);
                wputs(names[section = i]);
                *W_OPEN = color[5];
                switch(c = wgetc()) {                   /* User keystroke */
                case _KLA :                                     /* Left arrow - previous field */
                        i = (i ? i : ncount) - 1;
                        break;
                case _KRA :                                     /* Right arrow - next field */
                        i = (i + 1) % ncount;
                        break;
                case _KUA : ox = oy = -1000; goto do_find;
                case _KDA : ox = oy = 1000;
                do_find:
                        size = i;
                        for(j = 0; j < ncount; ++j) {
                                lx = (int)xs[j] - (int)xs[i];
                                ly = (int)ys[j] - (int)ys[i];
                                if(c == _KUA) {
                                        if((ly >= 0) || (ly < oy)) continue; }
                                else {
                                        if((ly <= 0) || (ly > oy)) continue; }
                                if(abs(lx) > abs(ox)) continue;
                                size = j;
                                ox = lx;
                                oy = ly; }
                        i = size;
                        break;
                case '\n' :                                     /* Enter - chain to menu */
                        rewind(fp);
                        section = link[i];
                        wclose();
                        goto lookup;
                case 0x1B:                                      /* Escape exit */
                case _K10 :
                        wclose();
                        fclose(fp);
                        return; } }
}

/*
```

```
 * Display a message box on the screen
 *
 * Input: string to display
 */
void message_box(char *s)
{
        unsigned w, h, c;
        char *ptr;

        /* Determine max height & width of text box */
        ptr = s;
        h = 3;
        c = w = 2;
        while(*ptr) {
                if(*ptr++ == '\n') {
                        ++h;
                        if(c > w)
                                w = c;
                        c = 1; }
                ++c; }

        if(c > w)
                w = c;

        /* Open window and display message */
        prompt(message_prompt);
        wopen(40-(w/2), 10-(h/2), w, h, WSAVE | WCOPEN | WBOX1 | color[4]);
        wcursor_off();
        wputs(s);
        for(;;) switch(wgetc()) {
                case 0x1B :
                case '\n' :
                case _K10 :
                        wclose();
                        return;
                case _K1 :
                        help(HELP_INDEX); }
}

/*
 * Get a character from the serial stream with timeout
 *
 * Input: Time to wait in 1/18 second intervals.
 * Return: Character read (0-255) or -1 for timeout.
 */
int Cgett(unsigned timeout)
{
        unsigned t;
        int c;

        t = BIOS_TICK;
        do {
```

```
                if((c = Ctestc()) != -1)
                        return c; }
            while((BIOS_TICK - t) < timeout);
            return -1;
    }

    /*
     * Wait for a character from the serial stream
     *
     * Input: Character to wait for, Time to wait
     * Return: 0 on success, !0 for timeout.
     */
    int Cwait(int c, unsigned t)
    {
            int c1;

            for(;;) {
                    if((c1 = Cgett(t)) == c)
                            return 0;
                    if(c1 == -1) {
                            error("Serial receive timeout!");
                            return -1; }
                    error("RX expected %02x got %02x", c, c1); }
    }

    /*
     * Get a hex byte from the serial stream
     *
     * Return: 0-255 or -1 for a bad input.
     */
    unsigned Cgetb(void)
    {
            int h, l;
            h = Cgett(Cbwait) - '0';
            Cbwait = CHAR_TIME;
            l = Cgett(CHAR_TIME) - '0';
            if( (h >= ('A'-'0')) && (h <= ('F'-'0')) )
                    h -= 7;
            else if(h >= 10) {
                    error("Bad HEX value: %02x", h + '0');
                    return -1; }
            if( (l >= ('A'-'0')) && (l <= ('F'-'0')) )
                    l -= 7;
            else if(l >= 10) {
                    error("Bad HEX value: %02x", h + '0');
                    return -1; }
            return (h << 4) | l;
    }

    /*
     * Write a hex byte to the serial stream
     *
```

```
 * Input: Value to write (0-255)
 */
void Cputb(unsigned char v)
{
        int c;

        if((c = v >> 4) >= 10)
                c += 7;
        Cputc(c +='0');
        Cwait(c, CHAR_TIME);

        if((c = v & 0x0F) >= 10)
                c += 7;
        Cputc(c += '0');
        Cwait(c, CHAR_TIME);
}

/*
 * Issue a remote command to the unit
 *
 * Input: Format string, Variable argument list
 */
int register command(unsigned args)
{
        char buffer[50], *ptr;
        unsigned ecount;

        _format_(nargs() * 2 + &args, buffer);

        ecount = 0;

restart:
        do {
                while(Ctestc() != -1);
                if(++ecount == 3) {
                        wopen(15, 9, 50, 3, WSAVE | WCOPEN | WBOX1 | color[4]);
                        wputs("Communication error - Please wait."); }
                if(wtstc() == 0x1B) {
                        if(ecount >= 3)
                                wclose();
                        error("Command handshake aborted!");
                        return -1; }
                if(ecount > 3) {
                        wputs("\nAttempting re-sync (Press ESC to abort)");
                        delay(100);
                        while(Ctestc() != -1); }
                Cputc(ATTN_CHAR);
                if(Cwait(ATTN_CHAR, ATTN_TIME))
                        goto restart;
                Cputc(ATTN_CHAR1); }
        while(Cwait(ATTN_CHAR1, CHAR_TIME));
```

```
        if(ecount >= 3)
                wclose();

        for(ptr = buffer; *ptr; ++ptr) {
                Cputc(*ptr);
                Cwait(*ptr, CHAR_TIME); }

        return 0;
}

/*
 * Load the mission event data from the remote unit.
 */
void load_mission_data(void)
{
        unsigned i, l;
        unsigned char c, c1;

        l = (&event.EVT_check) - event;          /* Calculate length */

        c = 0;

        if(!command("U")) {
                for(i=0; i < l; ++i)
                        c += (event[i] = Cgetb());
                c1 = Cgetb();
                Cwait(TERM_CHAR, CHAR_TIME);
                message("Mission data loaded.");
                if(c1 != c) {
                        w_puts(" - CHECKSUM FAILURE!", mwin);
                        message_box(checksum_warning); }
                return; }
        message("Communication error!");
}

/*
 * Save the mission event data to the unit.
 */
void save_mission_data(void)
{
        unsigned i, l;
        unsigned char c;

        l = (&event.EVT_check) - event;
        c = 0;
        if(!command("D")) {
                for(i=0; i < l; ++i) {
                        Cputb(event[i]);
                        c += event[i]; }
                Cputb(c);
                Cwait(TERM_CHAR, CHAR_TIME);
                message("Mission data saved.");
```

```
                return; }
            message("Communication error!");
    }

    /*
     * Edit a date/time entry
     *
     * Input: Pointer to [second minute hour day month year]
     * Return: 0 on success, !0 if canceled.
     */
    int edit_date(unsigned char input[])
    {
            unsigned entry[6], i;
            char flag;
            static unsigned limit[6] = { 59, 59, 23, 31, 12, 99 };

            prompt(form_prompt);

            for(i=0; i < 6; ++i)
                    entry[i] = input[i];

            for(;;) {
                    if(wform(25, 10, WSAVE | WCOPEN | WBOX1 | color[4], date_form,
                     &entry[3], &entry[4], &entry[5], &entry[2], &entry[1], &entry[0]))
                            return -1;

                    for(flag=i=0; i < 6; ++i) {
                            if(entry[i] > limit[i]) {
                                    flag = -1;
                                    entry[i] = input[i]; } }
                    if(!flag) {
                            for(i=0; i < 6; ++i)
                                    input[i] = entry[i];
                            return 0; }
                    message("Entered time is invalid!"); }
    }

    /*
     * Edit a time entry
     *
     * Input: Pointer to [second minute hour]
     * Return: 0 on success, !0 if canceled.
     */
    int edit_time(unsigned char input[])
    {
            unsigned entry[3], i;
            char flag;
            static unsigned limit[3] = { 59, 59, 23 };

            prompt(form_prompt);

            for(i=0; i < 3; ++i)
```

```
                entry[i] = input[i];

        for(;;) {
                if(wform(25, 10, WSAVE | WCOPEN | WBOX1 | color[4], time_form,
                &entry[2], &entry[1], &entry[0]))
                        return -1;

                for(flag=i=0; i < 3; ++i) {
                        if(entry[i] > limit[i]) {
                                flag = -1;
                                entry[i] = input[i]; } }
                if(!flag) {
                        for(i=0; i < 3; ++i)
                                input[i] = entry[i];
                        return 0; }
                message("Entered time is invalid!"); }
}

/*
 * Display bitfield using the short descriptions
 *
 * Input: Bitfield value, Pointer to short descriptions
 */
void display_bitfield(unsigned char b, char *prompt[])
{
        unsigned i;
        char *ptr;

        for(i=0; i < 8; ++i) {
                if(ptr = prompt[i])
                        wprintf(" %s=%c", ptr, ((1 << i) & b) ? 'Y' : 'N'); };
}

/*
 * Edit a bitfield
 *
 * Input: Pointer to bitfield, Pointer to long descriptions
 * Return: 0 on success, !0 if canceled.
 */
int edit_bitfield(unsigned char *field, char *text[])
{
        unsigned i, j;
        unsigned char b, c, *ptr;


        b = *field;
        for(i=j=0; i < 8; ++i)
                if(text[i])
                        ++j;

        wopen(15, 6, 50, j+2, WSAVE | WCOPEN | WBOX1 | color[4]);
```

```
reprompt:
        prompt(bitfield_prompt);

        for(;;) {
                for(i=j=0; i < 8; ++i) {
                        if(ptr = text[i]) {
                                wgotoxy(1, j++);
                                wprintf("%u: [%s] %s", i, (b & (1 << i)) ? "On " : "Off", ptr); } }
                c = wgetc();
                if((c >= '0') && (c <= '7')) {
                        b ^= (1 << (c-'0'));
                        continue; }
                switch(c) {
                case '\n' :
                        *field = b;
                        wclose();
                        return 0;
                case _K1 & 255:
                        help(HELP_BIT);
                        goto reprompt;
                case 0x1B :
                case _K10 & 255:
                        wclose();
                        return -1; } }
}

/*
 * Display the mission data & permit editing
 */
void display_mission(void)
{
        unsigned i, h, m, s, top;
        static unsigned select = 0;
        char save_attr, edit_flag, change_flag, c;

        load_mission_data();
        wclwin();
        wgotoxy(26, 0);
        wputs("** MISSION DATA **");
        save_attr = *W_OPEN;
        edit_flag = change_flag = 0;

redraw:
        prompt(edit_flag ? edit_prompt : view_prompt);

        /* Display the mission data */
        i = 0;
        do {
                wgotoxy(10, i+2);
                if((i == select) && edit_flag)
                        *W_OPEN = (*W_OPEN >> 4) | (*W_OPEN << 4);
                switch(i) {
```

```
                case 0 : wprintf("Mission Start  : %02u:%02u:%02u %02u/%02u/%02u",
                         event.EVT_Shour, event.EVT_Sminute, event.EVT_Ssecond,
                         event.EVT_Sday, event.EVT_Smonth, event.EVT_Syear);
                         break;
                case 1 : wprintf("Mission End    : %02u:%02u:%02u %02u/%02u/%02u",
                                 event.EVT_Ehour, event.EVT_Eminute, event.EVT_Esecond,
                                 event.EVT_Eday, event.EVT_Emonth, event.EVT_Eyear);
                         break;
                case 2 : wprintf("On Time      : %02u:%02u:%02u",
                                 event.EVT_ontime / 3600, (event.EVT_ontime % 3600) / 60,
                                 event.EVT_ontime % 60);
                         break;
                case 3 : wprintf("Event Interval : %02u:%02u:%02u",
                                 event.EVT_offtime / 1800, (event.EVT_offtime % 1800) / 30,
                                 (event.EVT_offtime % 30) * 2);
                         break;
                case 4 :
                         wputs("Config       :");
                         display_bitfield(event.EVT_config, config_short);
                         break;
                case 5 :
                         wputs("Control @Off  :");
                         display_bitfield(event.EVT_out_A, io_short);
                         break;
                case 6 :
                         wputs("Control @Ramp  :");
                         display_bitfield(event.EVT_out_B, io_short);
                         break;
                case 7 :
                         wputs("Control @On   :");
                         display_bitfield(event.EVT_out_C, io_short);
                         break;
                case 8 :
                         wprintf("On->Rec delay  : %-5u", event.EVT_delay1);
                         break;
                case 9 :
                         wprintf("Stop->Off delay: %-5u", event.EVT_delay2);
                         if(!(event.EVT_config & CONFIG_DINH)) {
                                 top = i;
                                 i = -1; }
                         break;
                case 10:
                         wprintf("Inhibit start  : %02u:%02u:%02u",
                                 event.EVT_IShour, event.EVT_ISminute, event.EVT_ISsecond);
                         break;
                case 11:
                         wprintf("Inhibit end    : %02u:%02u:%02u",
                                 event.EVT_IEhour, event.EVT_IEminute, event.EVT_IEsecond);
                         top = i;
                         i = -1; }
                *W_OPEN = save_attr; }
        while(++i);
```

```
        wcleow();

        /* Accept user commands */
        do {
                switch(c = wgetc()) {
                case ' ':                       /* Enable editing */
                        edit_flag = -1;
                        goto redraw;
                case _K1 :                              /* Help request */
                        help(edit_flag ? HELP_EDIT : HELP_VIEW);
                        goto redraw;
                case _K2 :                              /* Reload mission data */
                        load_mission_data();
                        change_flag = 0;
                        goto redraw;
                case 0x1B :                             /* Exit */
                case _K10 :
                        if(change_flag) {
                                i = 0;
                                prompt(menu_prompt);
                                if(wmenu(25, 10, WSAVE | WCOPEN | WBOX1 | color[4],
edit_menu, &i))
                                        goto redraw;
                                switch(i) {
                                default: goto redraw;
                                case 0 :
                                        save_mission_data();
                                case 1 : } }
                        if(!event.EVT_offtime)
                                message_box(interval_warning);
                        else {
                                if(!(h = event.EVT_ontime))
                                        message_box(zero_on_warning);
                                h += (event.EVT_delay1 + 999) / 1000;
                                h += (event.EVT_delay1 + 999) / 1000;
                                h = (h + 1) / 2;
                                if(h >= event.EVT_offtime)
                                        message_box(overtime_warning); }
                        return; }

                /* We only process these commands if editing is enabled */
                if(edit_flag) switch(c) {
                case _KDA :                             /* Move upward */
                        if(++select > top)
                                select = 0;
                        goto redraw;
                case _KUA :                             /* Move downward */
                        if(--select > top)
                                select = top;
                        goto redraw;
                case _K3 :                              /* Save value */
```

```
                        save_mission_data();
                        change_flag = 0;
                        continue;
                case _K4 :                      /* Clear mission data */
                        memset(event, 0, sizeof(event));
                        event.EVT_delay1 = 5000;
                        event.EVT_delay2 = 5000;
                        get_time(&h, &m, &s);
                        event.EVT_Ssecond = s;
                        event.EVT_Sminute = m;
                        event.EVT_Shour  = h;
                        event.EVT_Esecond = event.EVT_Eminute = 59;
                        event.EVT_Ehour  = 23;
                        get_date(&h, &m, &s);
                        event.EVT_Sday  = event.EVT_Eday = h;
                        event.EVT_Smonth = event.EVT_Emonth = m;
                        event.EVT_Syear  = event.EVT_Eyear = s % 100;
                        change_flag = -1;
                        goto redraw; } }
        while((c != '\n') || !edit_flag);

        switch(select) {
        case 0 :        /* Starting time */
                if(!edit_date(&event.EVT_Ssecond))
                        change_flag = -1;
                break;
        case 1 :        /* Ending time */
                if(!edit_date(&event.EVT_Esecond))
                        change_flag = -1;
                break;
        case 2 :        /* On time */
                prompt(form_prompt);
                h = event.EVT_ontime / 3600;
                m = (event.EVT_ontime % 3600) / 60;
                s = event.EVT_ontime % 60;
                if(!wform(25, 10, WSAVE | WCOPEN | WBOX1 | color[4], time_form, &h, &m,
&s)) {
                        event.EVT_ontime = (h * 3600) + (m * 60) + s;
                        change_flag = -1; }
                break;
        case 3 :        /* Interval time */
                prompt(form_prompt);
                h = event.EVT_offtime / 1800;
                m = (event.EVT_offtime % 1800) / 30;
                s = (event.EVT_offtime % 30) * 2;
                if(!wform(25, 10, WSAVE | WCOPEN | WBOX1 | color[4], time_form, &h, &m,
&s)) {
                        if(h >= 24) {
                                h = 24;
                                m = s = 0; }
                        event.EVT_offtime = (h * 1800) + (m * 30) + (s / 2);
                        change_flag = -1; }
```

```
                    break;
                case 4 :            /* Config bits */
                    if(!edit_bitfield(&event.EVT_config, config_long))
                        change_flag = -1;
                    break;
                case 5 :            /* Output idle bits */
                    if(!edit_bitfield(&event.EVT_out_A, io_long))
                        change_flag = -1;
                    break;
                case 6 :            /* Output Ramp bits */
                    if(!edit_bitfield(&event.EVT_out_B, io_long))
                        change_flag = -1;
                    break;
                case 7 :            /* Output ON bits */
                    if(!edit_bitfield(&event.EVT_out_C, io_long))
                        change_flag = -1;
                    break;
                case 8 :            /* On->Record delay */
                    prompt(form_prompt);
                    s = event.EVT_delay1;
                    if(!wform(25, 10, WSAVE | WCOPEN | WBOX1 | color[4], value_form, &s)) {
                        event.EVT_delay1 = s;
                        change_flag = -1; }
                    break;
                case 9 :            /* Stop->Off delay */
                    prompt(form_prompt);
                    s = event.EVT_delay2;
                    if(!wform(25, 10, WSAVE | WCOPEN | WBOX1 | color[4], value_form, &s)) {
                        event.EVT_delay2 = s;
                        change_flag = -1; }
                    break;
                case 10:            /* Inhibit start */
                    if(!edit_time(&event.EVT_ISsecond))
                        change_flag = -1;
                    break;
                case 11:            /* Inhibit end */
                    if(!edit_time(&event.EVT_IEsecond))
                        change_flag = -1;
            } goto redraw;
}

/*
 * Main program
 *
 * Initialize and perform the "Manual Control" functions.
 *
 * Input: Argument count, Pointers to command line arguments
 * Return: 0 (DOS return code)
 */
int main(int argc, char *argv[])
{
        int c;
```

```
        unsigned i, v;
        unsigned char *ptr, td[6];
        unsigned year, month, day, hour, minute, second, d1, d2;
        static unsigned char Outputs = 0;
        static unsigned Com_port = 1;

        /* Determine location of help file */
        ptr = argv[i=v=0];
        while(Helpfile[i++] = *ptr) {
                if(*ptr++ == '\\')
                        v = i; }
        strcpy(Helpfile+v, "TLV.HLP");

        /* Process command line arguments */
        for(i=1; i < argc; ++i) {
                ptr = argv[i];
                switch((toupper(*ptr++) << 8) | toupper(*ptr++)) {
                case 'C=' :                     /* Specify comm port */
                        Com_port = atoi(ptr);
                        if((Com_port < 1) || (Com_port > 4))
                                abort("C= must be 1-4\n");
                        continue;
                case '/M' :                     /* Force monochrome attributes */
                case '-M' :
                        color = M_color;
                        continue;
                default:
                        fprintf(stderr, "Unknown option: %s\n", argv[i]);
                case '?' << 8:
                case '/?' :
                case '-?' :
                        fputs(help_text, stderr);
                        exit(-1); } }

        /* Open comm port and switch to transparent */
        if(Copen(Com_port, _9600, PAR_NO | DATA_8 | STOP_1,
SET_RTS | SET_DTR | OUTPUT_2))
                abort("Cannot open comm port");
        disable();
        Cflags |= TRANSPARENT;
        enable();

        /* Open windows and issue welcome prompts. */
        wopen(0, 0, 1, 1, 0);       /* Open dummy window to set V_BASE */
        wclose();
        if(W_BASE == 0xB000)  /* Change attributes if monochrome */
                color = M_color;
        mwin = wopen(0, STAT_WIN+1, 80, 24-STAT_WIN,
WSAVE | WCOPEN | WSCROLL | WWRAP | color[1]);
        pwin = wopen(0, STAT_WIN, 80, 1, WSAVE | WCOPEN | color[3]);
        ewin = wopen(0, 0, 80, MAIN_WIN,
WSAVE | WCOPEN | WSCROLL | WWRAP | color[2]);
```

```
            wputs("Time Lapse Video (TLV) control host v"#VERSION_MA"."#VERSION_MI"");
            wputs(" - by Dunfield Development Systems\n");
            wputs("Compiled "#__DATE__" at "#__TIME__"");
#ifdef DEMO
            wputs("  (Demo version)");
#endif
            win  = wopen(0, MAIN_WIN, 80, STAT_WIN-MAIN_WIN,
WSAVE | WCOPEN | WSCROLL | WWRAP | color[0]);
            wcursor_off();

            /* Interrogate remote unit to determine version & display */
            command("I");
            d1 = Cgetb();
            d2 = Cgetb();
            Cwait(TERM_CHAR, CHAR_TIME);
            w_printf(mwin, "TLV control unit version is %u.%u", d1 >> 4, d1 & 15);
            if(d1 != ((VERSION_MA<<4) | VERSION_MI))
                  message("*** TLV host version differs from remote");
            if((v = &event.EVT_check - event) != d2)
                  message("*** Mission data size of TLV host(%u) differs from remote(%u)",
                        v, d2);

            /* Display "Manual Control" selections */
redraw:
            wclwin();
            wgotoxy(26, 0);
            wputs("** MANUAL CONTROL **");
            for(i=0; ptr = manual_cmds1[i]; ++i) {
                  wgotoxy(10, i+2);
                  wprintf("%c - %s", *ptr, ptr+3); }
            for(i=0; ptr = manual_cmds2[i]; ++i) {
                  wgotoxy(40, i+2);
                  wputs(ptr); }

reprompt:
            prompt(manual_prompt);

            /* Get a user command and process it */
next_command:
            for(;;) {
                  c = toupper(wgetc());
                  /* For simple commands, just send codes to camera */
                  for(i=0; ptr = manual_cmds1[i]; ++i) {
                        if(*ptr == c) {
                              v = (ptr[1] << 8) | ptr[2];
                              message(ptr+3);
                              if(ptr[3] != '+') {           /* Not a repeat command */
                                    command("C%04x", v);
                                    Cwait(TERM_CHAR, CHAR_TIME);
                                    w_puts(DONE_STR, mwin);
                                    goto next_command; }
                              /* Repeat command, send, then wait for terminate */
```

```
                                command("R%04x", v);
                                Cwait(TERM_CHAR, CHAR_TIME);
                                prompt(" Press SPACE to terminate command...");
                                do
                                        c = wgetc();
                                while((c != ' ') && (c != 0x1B) && (c != _K10));
                                Cputc(0x1B);
                                Cwait(TERM_CHAR, CHAR_TIME);
                                w_puts(DONE_STR, mwin);
                                goto reprompt; } }
                        /* Handle "special case" commands (not simple table lookup) */
                        switch(c) {
                        case '?' :          /* Query Camera */
                                message("Status: ");
                                command("Q");
                                w_printf(mwin, "%02x", Cgetb());
                                Cbwait = 18*3;
                                w_printf(mwin, "%02x", Cgetb());
                                w_printf(mwin, "  Counter: %02x%02x%02x", Cgetb(), Cgetb(), Cgetb());
                                Cwait(TERM_CHAR, CHAR_TIME);
                                w_puts(DONE_STR, mwin);
                                continue;
                        case 'T' :                  /* Display current time */
                                get_date(&day, &month, &year);
                                get_time(&hour, &minute, &second);
                                year %= 100;
                                message("HOST Time: %02u:%02u:%02u %02u/%02u/%02u   ",
                                        hour, minute, second, day, month, year);
                                command("T");
                                second = Cgetb();
                                minute = Cgetb();
                                hour = Cgetb();
                                day = Cgetb();
                                month = Cgetb();
                                year = Cgetb();
                                w_printf(mwin, "CAMERA Time: %02u:%02u:%02u
%02u/%02u/%02u",
                                        hour, minute, second, day, month, year);
                                Cwait(TERM_CHAR, CHAR_TIME);
                                w_puts(DONE_STR, mwin);
                                continue;
                        case 'O' :                  /* Edit outputs */
                                message("Set outputs: ");
                                if(!edit_bitfield(&Outputs, io_long)) {
                                        w_printf(mwin, "%02x", Outputs);
                                        command("O%02x", Outputs);
                                        Cwait(TERM_CHAR, CHAR_TIME);
                                        w_puts(DONE_STR, mwin);
                                        goto reprompt; }
                                w_puts(CAN_STR, mwin);
                                goto reprompt;
                        case _K1 :
```

```
                help(HELP_MANUAL);
                goto reprompt;
            case _K2 :                  /* Power ON */
                message("Power ON");
                command("N");
                Cwait(TERM_CHAR, ONOFF_TIME);
                w_puts(DONE_STR, mwin);
                continue;
            case _K3 :                  /* Power OFF */
                message("Power OFF");
                command("F");
                Cwait(TERM_CHAR, ONOFF_TIME);
                w_puts(DONE_STR, mwin);
                continue;
            case _K4 :                  /* Set time */
                message("Set time");
                v = 0;
                prompt(menu_prompt);
                if(wmenu(25, 10, WSAVE | WCOPEN | WBOX1 | color[4], time_menu,
&v)) {
                    w_puts(CAN_STR, mwin);
                    goto reprompt; }
                get_date(&day, &month, &year);
                get_time(&hour, &minute, &second);
                td[0] = second; td[1] = minute; td[2] = hour;
                td[3] = day; td[4] = month; td[5] = year % 100;
                if(v) {
                    if(edit_date(td)) {
                        w_puts(CAN_STR, mwin);
                        continue; } }
                command("S%02x%02x%02x%02x%02x%02x%02x",
                    td[0], td[1], td[2], td[3], td[4], td[5]);
                Cwait(TERM_CHAR, CHAR_TIME);
                w_puts(DONE_STR, mwin);
                goto reprompt;
            case _K5 :                  /* Set next event time time */
                message("Next Event time");
                command("A");
                td[0] = Cgetb();
                td[1] = Cgetb();
                td[2] = Cgetb();
                Cwait(TERM_CHAR, CHAR_TIME);
                if(!edit_time(td)) {
                    command("B%02x%02x%02x", td[0], td[1], td[2]);
                    w_puts(DONE_STR, mwin);
                    Cwait(TERM_CHAR, CHAR_TIME);
                    goto reprompt; }
                w_puts(CAN_STR, mwin);
                goto reprompt;
            case _K8 :                  /* Mission editor */
                message("Mission Editor");
                display_mission();
```

```
                        w_puts(DONE_STR, mwin);
                        goto redraw;
                case _K9 :              /* Restart unit */
                        message("RESTART");
                        prompt(menu_prompt);
                        v = 0;
                        i = wmenu(25, 10, WSAVE | WCOPEN | WBOX1 | color[4], restart_menu,
&v);
                        if((v == 1) && !i) {
                                command("Z5A");
                                Cwait(TERM_CHAR, CHAR_TIME);
                                w_puts(DONE_STR, mwin);
                                goto reprompt; }
                        w_puts(CAN_STR, mwin);
                        goto reprompt;
                case _K7 :              /* TTY mode */
                        message("TTY Mode");
                        prompt(" TTY: Press F10 to exit.");
                        wcursor_line();
                        wclwin();
                        i = 0;
                        for(;;) {
                                if(c = wtstc()) {
                                        if(c > 0) {
                                                Cputc(c);
                                                continue; }
                                        else if(c == _K10) {
                                                wcursor_off();
                                                w_puts(DONE_STR, mwin);
                                                goto redraw; } }
                                if((c = Ctestc()) >= 0) {
                                        wputc(c);
                                        i = -1; }
                                else if(i) {
                                        wupdatexy();
                                        i = 0; } }
                case _K10 :             /* Exit */
                        wclose();
                        wclose();
                        wclose();
                        wclose();
                        Cclose();
                        return 0;
                case ' ' :              /* Incase he needs space after repeat terminates */
                        Cputc(0x1B); } }
}
```

C7b. PC HOST COMPUTER SOURCE CODE (HELP FILE- TLV.SRC) SOURCE LISTING

This file is the contents of the 'HELP' file used by the TLVL host computer activated when the user presses the function key 'F1'. The contents of this data are in file Tlv.src. and are accesses by the compiled code Tlv.com whose source was given above in section C7a.

The source code is:

| INDEX | Index of help functions

Manual control mode•Manual•
Mission viewer•View•
Mission editing mode•Edit•

Begin Recording•RECORD•
Output•OUTPUT•
Stop Recording•STOP•
Display•STATUS•
Start/Stop button•SS•
Capture still Image•CAPTURE•
OFF•OFF•
On-Screen Display•DISPLAY•
time•STIME•
Record Locate•LOCATE•
time•ALARM•
Edit Locate Functions•EDITL•
Zoom Functions•ZOOM•
mission•VIEW•
Display Time•TIME•

Bit field entry•Bit•
Menu selection•Menu•
Data entry•Input•

Set
Status
Turn Power ON•ON•
Turn Power
Set camera
Set next event
TTY mode•TTY•
View/Edit
Restart•

| MANUAL | Manual Control Mode

You may execute functions by pressing the following keys:

B - Begin Recording•RECORD•
button•SS•
S - Stop Recording•STOP•
time•TIME•
C - Capture still image•CAPTURE•
signals•OUTPUT•
D - Display ON or OFF•DISPLAY•
status•STATUS•
L - Record Locate•LOCATE•
ON•ON•
F - Edit locate (forward)•EDITL•
R - Edit locate (reverse)•EDITL•
] - Zoom In (slow)•ZOOM•
time•ALARM•
[- Zoom Out (slow)•ZOOM•
{ - Zoom In (fast)•ZOOM•
mission•VIEW•
} - Zoom Out (fast)•ZOOM•
Restart•RESTART•

X - Start/Stop
T - Display
O - Set output
? - Display
F2 - Turn Power
F3 - Turn Power OFF•OFF•
F4 - Set camera time•STIME•
F5 - Set next event
F7 - TTY mode•TTY•
F8 - Edit
F9 -

->Index•
| VIEW | Mission Viewer
This display shows the currently configured mission:

Mission Start•	- Time/Date mission will commence
Mission End•	- Time/Date mission will terminate
On time•	- Time camera is on per event
Event Interval•	- Time between start of events
Config•	- Configuration options
Control @OFF•CTRL•	- Control outputs when OFF (inactive)
Control @RAMP•CTRL•	- Control outputs during RAMPing
Control @ON•CTRL•	- Control outputs when ON (active)
On->Rec Delay•OnDel•	- Delay(ms) between Ctrl-output & Recording
Stop->Off Delay•OffDel•	- Delay(ms) between Camera-stop & power off

You may enter the Mission editor•Edit• from this screen by pressing SPACE.
->Index• ->Manual•
| EDIT | Mission Editor

After you press SPACE on the Mission Display•VIEW•, a cursor bar will appear which allows you to select any item for editing. You may:

Press ENTER to make changes to the hilighted item.
Press F2 to reload the values last saved to the unit.
Press F3 to save the changed values to the unit.
Press F4 to clear all of the event fields.
Press F10 or ESCAPE to exit.

NOTE: Changes to the mission will take effect at the next event cycle. If you change the mission start time after a mission has already begun, you should RESTART• the unit to force it to recognize the new start time.

->Index• ->Manual• ->Bit field•Bit• ->Menu Selection•Menu• ->Data Entry•Input•
| BIT | BitField Editor

When editing a bit field, you may:

Press the digit shown to toggle an entry On/Off.
Press ENTER to exit and save the new value.
Press ESCAPE to exit and cancel the change.

->Index• ->Manual• ->Mission Viewer•VIEW• ->Menu Selection•Menu• ->Data Entry•Input•
| MENU | Menu Selection

When selecting an item from a menu, you may:

Use Up and Down arrows to move selection cursor.
Press Home/End to move to First/Last entry.
Press first character of entry to move cursor there.
Press ENTER to exit and proceed with selection.

Press ESCAPE to exit and cancel.

->Index• ->Manual• ->Mission Viewer•VIEW• ->Bit field•Bit• ->Data Entry•Input•
|INPUT|Data Entry

When entering into a screen form, you may:

Use Up and Down arrows to move the entry field.
Use Left and Right arrows to move within the field.
Use DELETE to delete the character under the cursor.
Use BACKSPACE to backup and delete.
Use INSERT to toggle Insert/Overwrite.
Use HOME to go the start of the field.
Use PGUP to clear the entire field.
Use PGDN to clear from the cursor to the end of the field.
Use ENTER to move to the next field, EXIT when on last one.
Use ESCAPE to cancel the form entry.

->Index• ->Manual• ->Mission Viewer•VIEW• ->Bit field•Bit• ->Menu Selection•Menu•
|RECORD|[B]egin Camera Recording

Pressing 'B' from the Manual• Control screen causes the camera
to begin recording.

This function may not be supported on certain models of cameras
(especially older ones), in which case you will need to use the
Start/Stop button•SS• function.

NOTE: The camera must be powered ON• and STOP•ped.

->Index•
|STOP|[S]top Camera Recording

Pressing 'S' from the Manual• Control screen causes the camera
to stop recording.

This function may not be supported on certain models of cameras
(especially older ones), in which case you will need to use the
Start/Stop button•SS• function.

NOTE: The camera must be powered ON• and RECORD•ing.

->Index•
|SS|Start/Stop recording

Pressing 'X' from the Manual• control screen causes the camera
to toggle recording ON and OFF.

This function is provided to support older cameras which do not
implement the individual Begin•RECORD•/Stop• recording commands.

NOTE: The camera must be powered ON•

->Index•
|CAPTURE| [C]apture Still Image

Pressing 'C' from the Manual• Control screen causes the camera to digitally capture a still image, and record it for several seconds.

This function is supported only on certain models of "digital" cameras.

NOTE: The camera must be powered ON• and STOP•ped.

->Index•
|DISPLAY| [D]isplay ON/OFF

Pressing 'D' from the Manual• Control screen causes the camera to toggle the on-screen display prompts ON or OFF. When ON, the camera viewfinder prompts can be viewed with remote monitor.

This function may not be supported on certain models of cameras (especially older ones).

NOTE: The camera must be powered ON•.

->Index•
|LOCATE| [L]ocate Record Position

Pressing 'L' from the Manual• Control screen causes the camera to execute a Recording Locate function. This plays in reverse for a few seconds, then plays forward until it is back at the starting location. The purpose of this is to allow you to see the last few seconds of previously recorded video on the remote monitor.

NOTE: The camera must be powered ON• and STOP•ped.

->Index•
|EDITL| Edit Locate Functions [F/R]

These functions perform edit positioning on the camera. When 'F' is pressed, the camera will enter Edit-Play-Forward mode. When 'R' is pressed, the camera will enter Edit-Play-Reverse.

The camera will continue to play (Forward or Reverse) until the SPACE bar is pressed. During this time, you will be able to see the image previously recorded on the tape.

NOTE: The camera must be powered ON• and STOP•ped.

->Index• ->Manual•
|ZOOM| Zoom Functions [{}]

Press '}' to Zoom In slowly.

Press '}' to Zoom In quickly.
Press '[' to Zoom Out slowly.
Press '{' to Zoom Out quickly.

Once activated, the camera will execute the Zoom function (In or Out) until the SPACE bar is pressed. During this time you will see the image on the remote monitor enlarging or shrinking.

Fast Zoom functions may not be available on certain camera models.

NOTE: The camera must be powered ON•.

->Index• ->Manual•
|TIME|Display [T]ime

Pressing 'T' from the Manual• Control screen will display the time in the message window. The time reported from both the PC and the camera control unit is displayed.

->Index•
|OUTPUT|Set [O]utput Signals

Pressing 'O' from the Manual• Control screen allows you to set the output control signals at the camera control unit. These signals are normally set by the Control•CTRL• options of the mission configuration•VIEW•, and updated automatically as events occur, however this function may be used to override the outputs at any time.

You will be prompted with a Bit field menu•Bit• for the control signal values to be output.

->Index•
|STATUS|Display Camera Status [?]

Pressing '?' from the Manual• Control screen displays the camera status/counter values:

Status: aabb Counter: cccccc

aa = TLV controller state•TSTATE•
bb = Camera state•CSTATE•
cc = Camera tape Counter

NOTE: The camera must be powered ON•.

->Index•
|ON|Turn Camera Power ON [F2]

Pressing F2 from the Manual• Control screen executes an event Power-ON sequence. The control outputs are turned on (including Ramp if specified) and the camera is turned ON. The camera will

remain ON until it is turned OFF•.

->Index•
|OFF|Turn Camera Power OFF [F3]

Pressing F3 from the Manual• Control screen executes an event Power-OFF sequence. The control outputs are returned to their idle state, and the camera is turned OFF. The camera will remain OFF until it is turned ON•.

->Index•
|STIME|Set remote Time [F4]

Pressing F4 from the Manual• Control screen allows you to set the time on the camera control unit. You will be presented with a Menu• offering two choices:

"Set to local PC time"	<- Use the time from the PC
"Prompt for specific time"	<- Prompt for the time

If you select the "Prompt" option, you will get an Input• form allowing you to enter the exact time you wish to set. For your convienence the time value is preset to the current PC time.

->Index•
|ALARM|Set Next Event Time [F5]

Pressing F5 from the Manual• Control screen allows you to view and alter the time of the next scheduled event on the camera control unit. Normally, the time of the next event is automatically recalculated as each event occurs, using the interval specified in the Mission Configuration•VIEW•.

This function allows you to override the time at which the next event will occur. Subsequent events will occur at the specified intervals from this time.

->Index•
|TTY|TTY Communications Mode [F7]

Pressing F7 from the Manual• Control screen enters into "TTY" serial comunication with the camera control unit. This is mainly for debugging purposes, and should not be used unless you have good reason to communicate directly with the control unit.

->Index•
|RESTART|Restart Unit

Pressing F9 from the Manual• Control screen causes the camera control unit to restart. This performs a complete re-initialzation of the on-board control system.

Normally, a restart is only necessary when you have set a new mission starting time after a mission is already in progress. Restarting the unit will cause it to go to sleep until the new start time occurs.

->Index•
| Mission Start | Mission Start Date/Time

This is the date and time at which the camera will begin recording regular video clips. Before this date and time occur, the camera will remain idle. Once this time has passed, the camera will begin recording video clips and the defined event interval• and On time• until the mission end• date and time pass.

->Index• ->Mission Viewer•VIEW•
| Mission End | Mission End Date/Time

This is the date and time at which the camera will stop recording regular video clips. Once this date and time have passed, the camera will remain idle until a new mission start• is programmed.

->Index• ->Mission Viewer•VIEW•
| On time | Event ON time

This configuration item defines the duration of each video clip. Once the mission start• date and time have passed, the camera will record regular video clips of this length, at the defined event interval•. This regular recording will continue until the mission end• date and time pass.

->Index• ->Mission Viewer•VIEW•
| Event Interval | Event Interval

This configuration item defines the interval time between the start of sucessive video clips. Once the mission start• date and time have passed, the camera will record regular video clips of the defined on time• duration at this interval. This regular recording will continue until the mission end• date and time have passed.

->Index• ->Mission Viewer•VIEW•
| Config | Misc. Configuration

This is a Bit field•BIT• containing various configuration items which can be toggled ON or OFF:

Enable Control Signal•ENABLE•
Inhibit Control Signal•INHIBIT•
Daily Inhibit Period•DAILY•
Single Start/Stop•STST•
Debug TTY messages•DEBUG•

->Index• ->Mission Viewer•VIEW•
|ENABLE|Enable Control Input

This item when enabled, causes the camera controller to honor the hardware ENABLE INPUT. This means that the ENABLE INPUT must be present in order for the camera to operate.

The hardware ENABLE INPUT may be toggled while an event is in progress to start and stop the camera within that event.

->Index• ->Config• ->Mission Viewer•VIEW•
|INHIBIT|Inhibit Control Input

This item when enabled, causes the camera controller to honor the hardware INHIBIT INPUT. This means that the INHIBIT INPUT must NOT be present in order for the camera to operate.

The hardare INHIBIT INPUT may be toggled while an event is in progress to start and stop the camera within that event.

->Index• ->Config• ->Mission Viewer•VIEW•
|DAILY|Daily Inhibit Period

This item when set, causes two more fields to appear on the mission configuration menu. These fields control the starting and ending time for an INHIBIT PERIOD. During the INHIBIT PERIOD, the camera will not record.

If the INHIBIT PERIOD begins or ends during an active event, the camera will be stopped or started accordingly.

If the INHIBIT PERIOD ending time is set earlier than the starting time, the inhibit period will occur between the starting time, and the ending time on the next day.

->Index• ->Config• ->Mission Viewer•VIEW•
|STST|Single Start/Stop

This item when set causes the TLV to use a single Start/Stop•SS• code for begining and ending recording. When NOT set, separate RECORD• and STOP• commands are issued to the camera.

This option is provided to support older cameras which do not support the separate commands. When the separate commands are available, they should be used, as they are a more reliable way of insureing that the camera is doing what is intended.

->Index• ->Config• ->Mission Viewer•VIEW•
|DEBUG|Debug TTY messages

This option enables text messages to appear on the serial

data line from the TLV controller, which inform the viewer of the internal actions occuring within the TLV firmware. These messages can be viewed with TTY Mode•TTY•.

This option should not be used in the normal day to day operation of the TLV, as it can cause interference with the serial communications between the HOST and the TLV controller. It is intended for firmware debugging purposes only.

->Index• ->Config• ->Mission Viewer•VIEW•
|CTRL|Event Control Signals

These fields define the hardware output control signals which will be set by the controller when an event starts and stops.

@OFF <- Defines outputs when the unit is idle.
@ON <- Defines outputs when the unit is recording.
@RAMP <- Defines a power "ramp up" as described below:

The RAMP setting can be used to gently turn on lights. If these entries have different values, the controller will generate a varying duty-cycle waveform from 100%-RAMP/0%-ON to 0%-RAMP/100%-ON over approx. 1 second during the power-up sequence.

->Index• ->Mission Viewer•VIEW•
|ONDEL|On -> Record Delay Interval

This item specifies a delay (in milliseconds), which the controller waits after powering ON• the camera, before sending the RECORD• command.

This allows the camera time to perform it's own power-on sequence, during which time it may not be ready to accept commands.

->Index• ->Mission Viewer•VIEW•
|OFFDEL|Stop -> Off Delay Interval

This item specifies a delay (in mulliseconds), which the controller waits after issuing a STOP command before powering OFF the camera.

This allows the camera time to stop the tape transport.

->Index• ->Mission Viewer•VIEW•
|TSTATE|TLV Controller State

This value indicates the current operating state of the TLV controller:

00 = Waiting for valid mission data (checksum error)
01 = Waiting for mission to start

02 = Processing an event - Camera is ON
03 - Processing an event - Camera is OFF
04 - Waiting for next event
05 - Mission has ended

->Status• ->Index•

| CSTATE | Common LANC state codes

00-Not resp./power off	03-FastForward	74-Video Insert/pause	
01-Ejected (Tape out)	33-FF/auto 0 PB	84-Audio Insert/Dub	
11-No cassette (DEW)	43-FF/memory stop	94-Autio Insert/pause	
21-Eject cass. unload	83-Rewind	06-Playback	
31-Eject tape unloaded	A3-RW/TOP auto PB	26-Play x1	
02-STOP normal	B3-RW/auto 0 PB	36-Play -x1	
12-STOP tape load	C3-RW/memory stop	46-Play/Cue	
22-STOP cassette load	04-Recording	56-Play/Review	
32-STOP battery down	14-Rec. Pause	66-Play x2/x3	
42-STOP unload DEW	24-Timer record	76-Play -x2/-x3	
52-STOP EMERGENCY	34-Timer rec. pause	86-Locked Cue/x9	
62-STOP tape end	44-A/V insert	96-Locked Review/-x9	
72-STOP tape top	54-A/V insert/pause	A6-Frame search queue	
92-STOP counter memory	64-Video insert	B6-Frame search review	
85-Edit->Forward	95-Edit->Reverse	->Status• ->Index•	

D1 MECHANICAL HARDWARE

The TLVL mechanics presently consist to three units 1) a camera housing 2) a battery can 3) a miniature submersible light source. Bracketry which hold the units onto T-frames used with the acoustic logger is also described.

The camera sketches were done using Autosketchv2.1. The machine readable files contain drawings in Autosketch format (*.skd) as well as files exported by Autosketch to *.dxf and *.dwg formats for convenience. The accuracy of the conversions was not checked. From past experience the conversions are generally accurate although text labeling, size and fonts may have to be adjusted for the user's program settings.

The TLVL was constructed for shallow water (< 75m use). The housings would have to be re-designed if use at deeper depths is required. In particular the camera housing with it's dome would have to be re-done. The limiting factor here is the dome. Domes capable of deeper use can be readily bought.

The aluminum cans are either painted or anodized and then powder coated. Magnesium sacrificial anodes are used to reduce corrosion.

Mounting hardware such as screws, bolts and the like are not described.

The miniature light source is a "Micro SeaLite", 12V, 20W version from Deep Sea Research, San Diego, CA, USA.

D2 CAMERA MODULE MECHANICAL SKETCHES

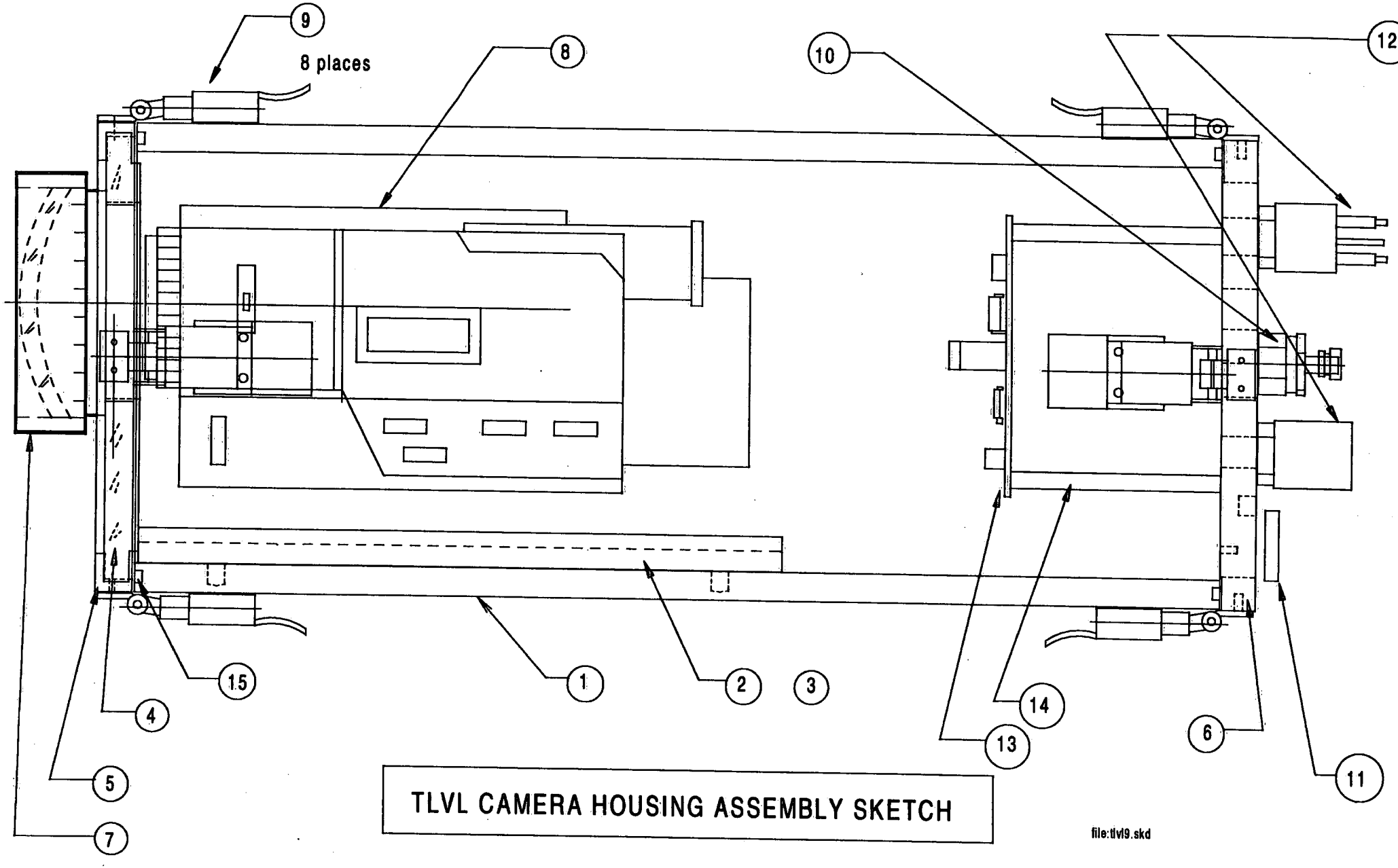
The camera module housing was designed with a view to accommodating a Sony TR-7000 Digital8 camcorder and the microcontroller board. The camera is mounted on a sliding track and is restrained by a pin in the rear and the front port. Modifications to the housing to allow for the latest generation of Digital8 camcorders should be minor. More recent cameras are very similar except they have an LCD screen which can flip out. This is of no use for our enclosed camera and for unattended operation but this feature is now nearly universal. With the LCD screen collapsed, these cameras should fit in the housing without any problem.

The IRL540N FET is mounted on the inside face of the rear bulkhead with a screw and electrical isolation pads.

D2 CAMERA MODULE MECHANICAL SKETCHES

The camera module housing was designed with a view to accomodating a Sony TR-7000 Digital8 camcorder and the microcontroller board. The camera is mounted on a sliding track and is restrained by a pin in the rear and the front port. Modifications to the housing to allow for the latest generation of Digital8 camcorders should be minor. More recent cameras are very similar except they have an LCD screen which can flip out. This is of no use for our enclosed camera and for unattended operation but this feature is now nearly universal.With the LCD screen collapsed, these cameras should fit in the housing without any problem.

The IRL540N FET is mounted on the inside face of the rear bulkhead with a screw and elctrical isolation pads.



TLVL CAMERA HOUSING ASSEMBLY SKETCH

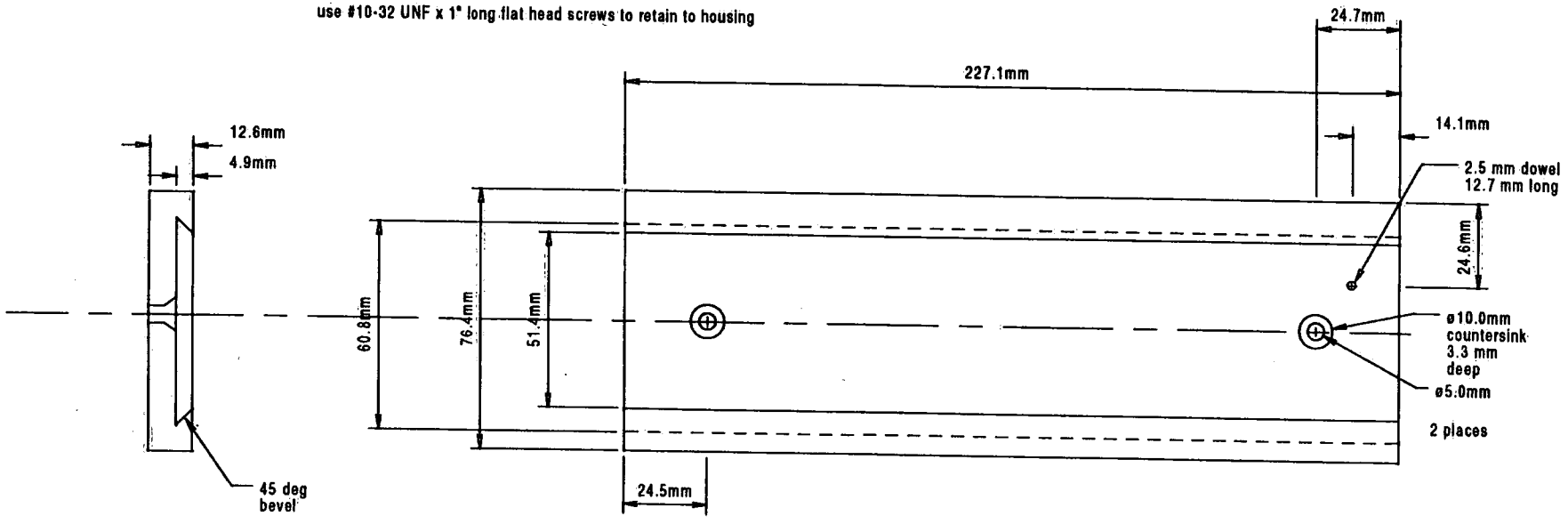
file:tlvl9.skd

drawn by rjd

TLVL CAMERA CAN PARTS LIST
refer to sketch tlvlc9.skd for ID numbers

ID	Part sketch	Description	Mfg or Supplier	Mfg No	Mfg. address
1	tlvlc3.skd	camera housing tube	shop		
2	tlvlc1.skd	SonyTR-7000 slide track	shop		
3	tlvlc2.skd	TR-7000 slide mount	shop		
4	tlvlc6.skd	polycarbonate front port	shop		
5	tlvlc4.skd	port retaining ring	shop		
6	tlvlc5.skd	rear port bulkhead	shop		
7		Ikelite dome for Ikelite 6025.27 housing Note:order dome only	Ikelite Float & Flag		Plains Rd. Burlington
8		Sony digital8 Camcorder	The Sony Store	TR-7000	Fairview St. Burlington
9		Nielsen latch & strike qty 8	Spaenaur	096-807	Waterloo ON
10		Dowty strut valve	Leavens Bros Aviation Supplies		Mississauga
11		25 x 50 x 6 mm thick magnesium sacrificial anode	Ideal metals		Brampton ON
12		Impulse UW connectors including: BH-4-MP BH-6-FS BH-2-MP BH-2-FS incl. dummy connectors	McQuest Marine		Enfield Rd Burlington
13	See elec.	TLVL circuit board	Dynamic Circuits		Stoney Creek ON

14	7 long x 6mm hex standoffs 6-32UNC	Keystone (Electrosonic)	Toronto ON
15	O-ring (nitrile)	Canadian Bearings	Woodward Ave Hamilton



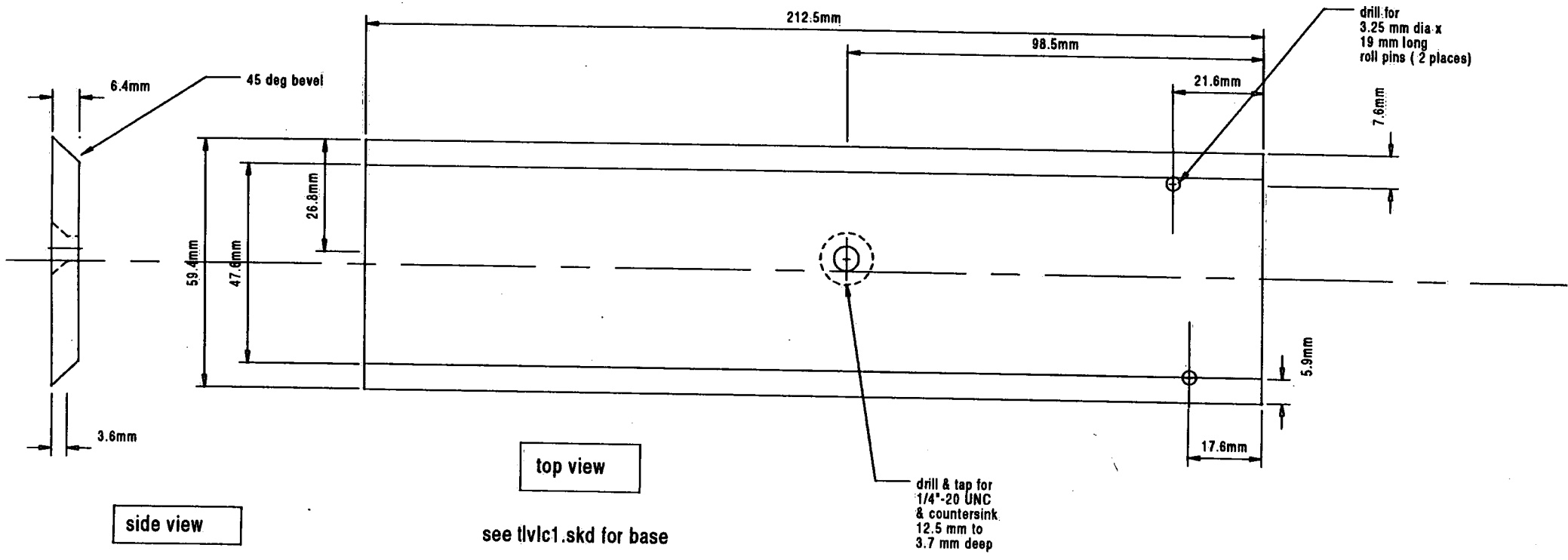
material: AL alloy

TLVL SONY TR-7000 SLIDE TRACK

drawn by rjd

file: tlvlc1.skd

drawn by rjd file tlvlc2.skd

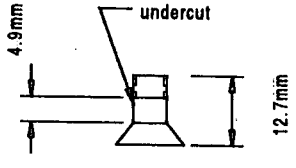


top view

side view

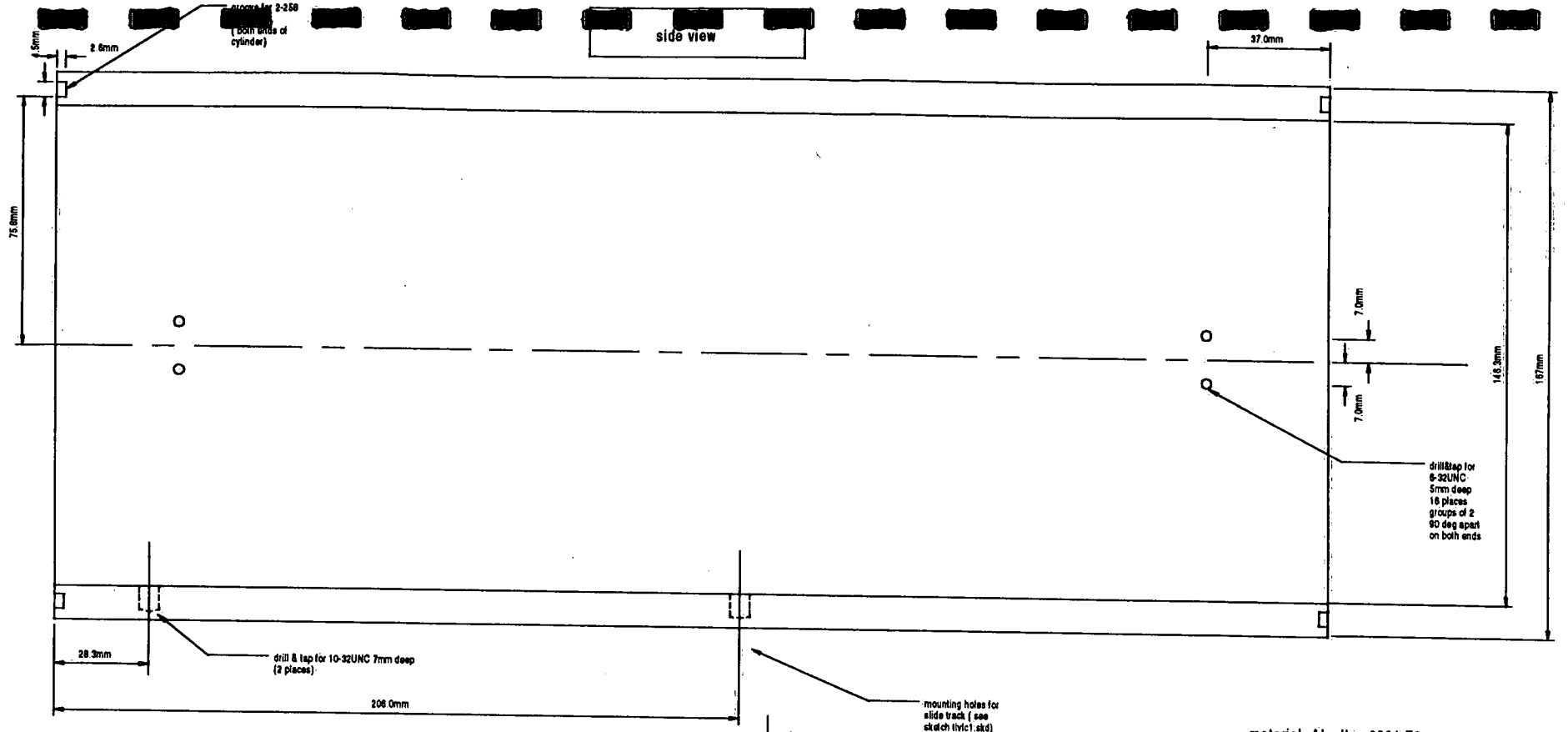
see tlvlc1.skd for base

Sony TR-7000 Camera Slide Mount



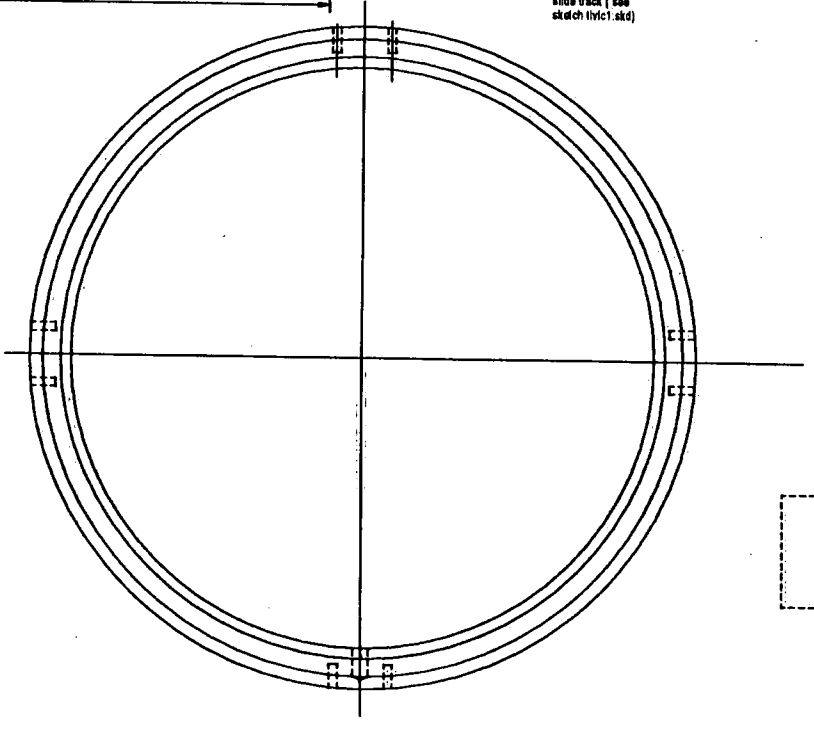
retaining screw

1/4"-20 UNC flat head screw



material: AL alloy 6061-T6
6" schedule 80
pipe

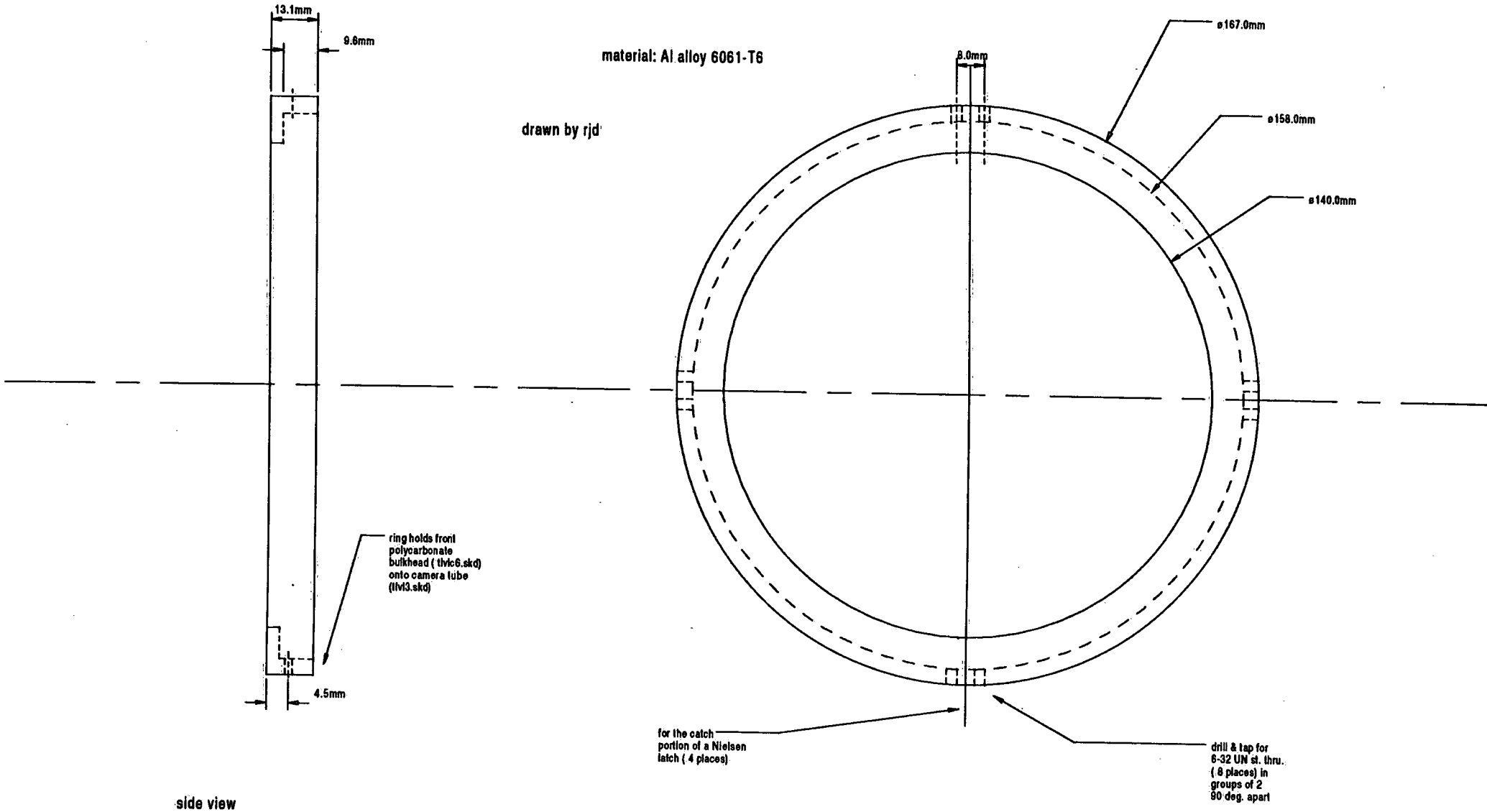
end view



file: tlvlc3.skd

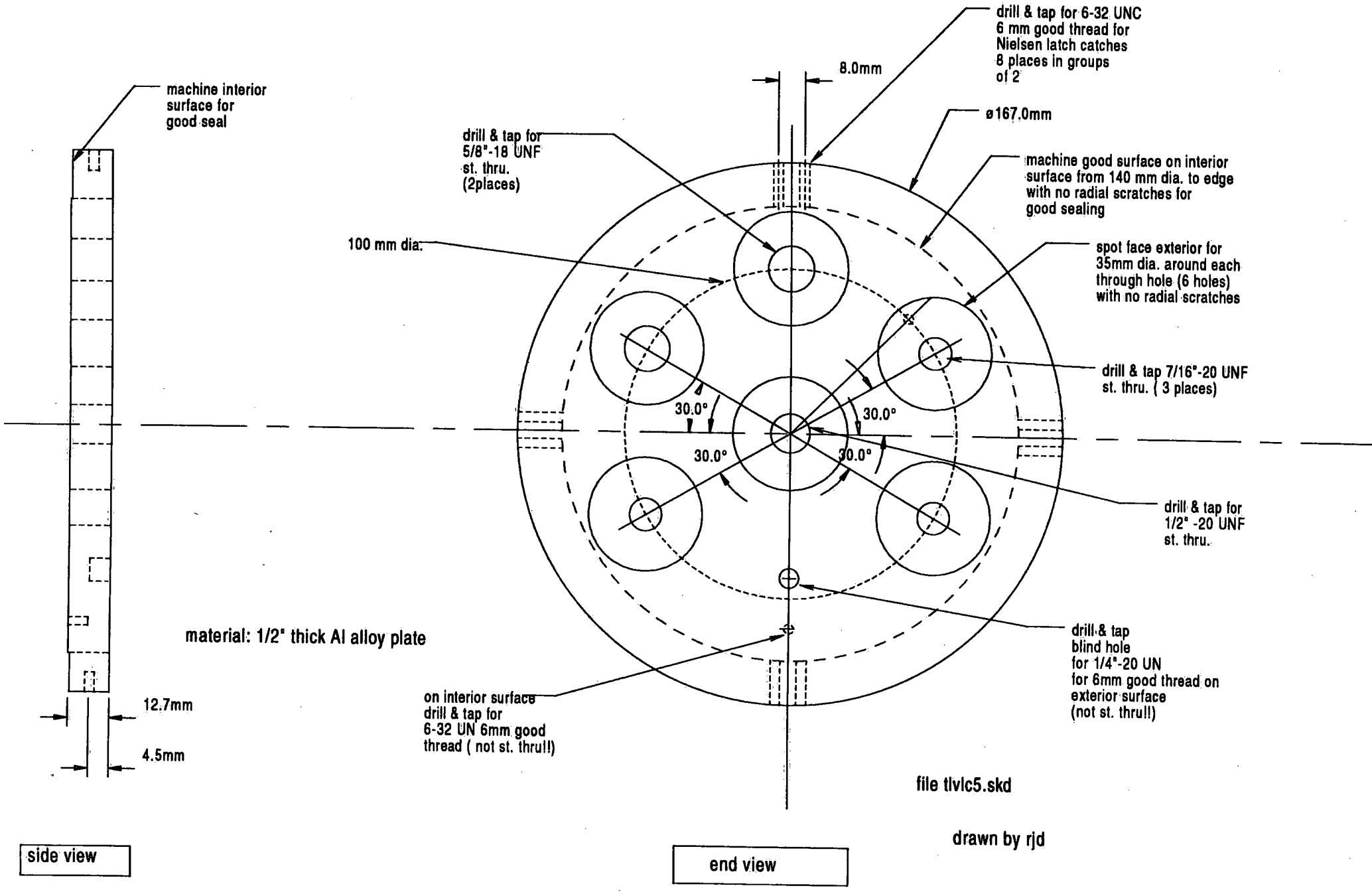
drawn by rjd

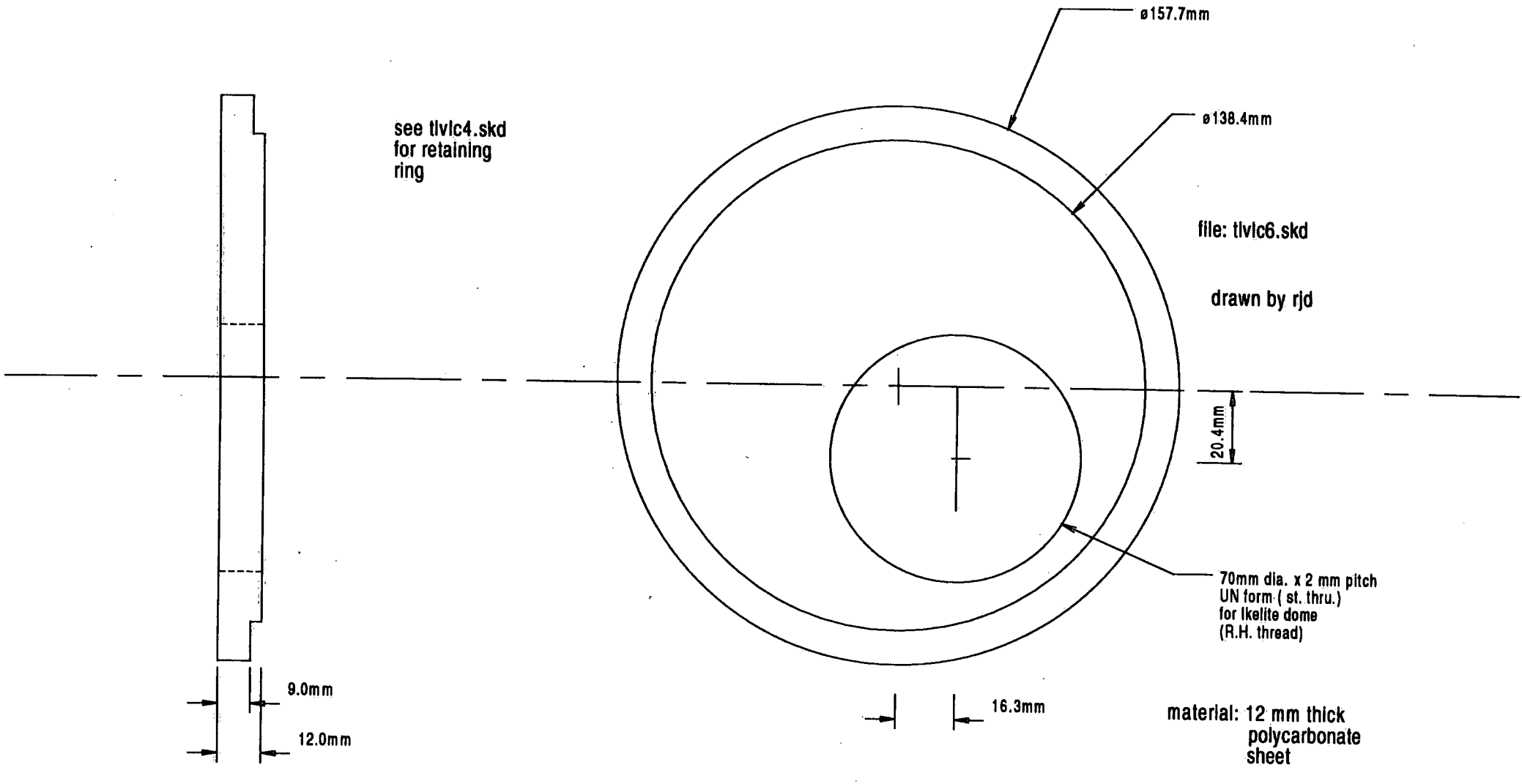
TLVL camera can tube



TLVL HOUSING FRONT PLATE RING

file:tlvl4.skd


machine interior surface for good seal
12.7mm
4.5mm
material: 1/2" thick Al alloy plate
side view
drill & tap for 6-32 UNC 6 mm good thread for Nielsen latch catches 8 places in groups of 2
8.0mm
ø167.0mm
drill & tap for 5/8"-18 UNF st. thru. (2places)
machine good surface on interior surface from 140 mm dia. to edge with no radial scratches for good sealing
100 mm dia.
spot face exterior for 35mm dia. around each through hole (6 holes) with no radial scratches
drill & tap 7/16"-20 UNF st. thru. (3 places)
30.0°
30.0°
30.0°
30.0°
drill & tap for 1/2" -20 UNF st. thru.
drill & tap blind hole for 1/4"-20 UN for 6mm good thread on exterior surface (not st. thru!!)
on interior surface drill & tap for 6-32 UN 6mm good thread (not st. thru!!)
end view
file tlvlc5.skd
drawn by rjd

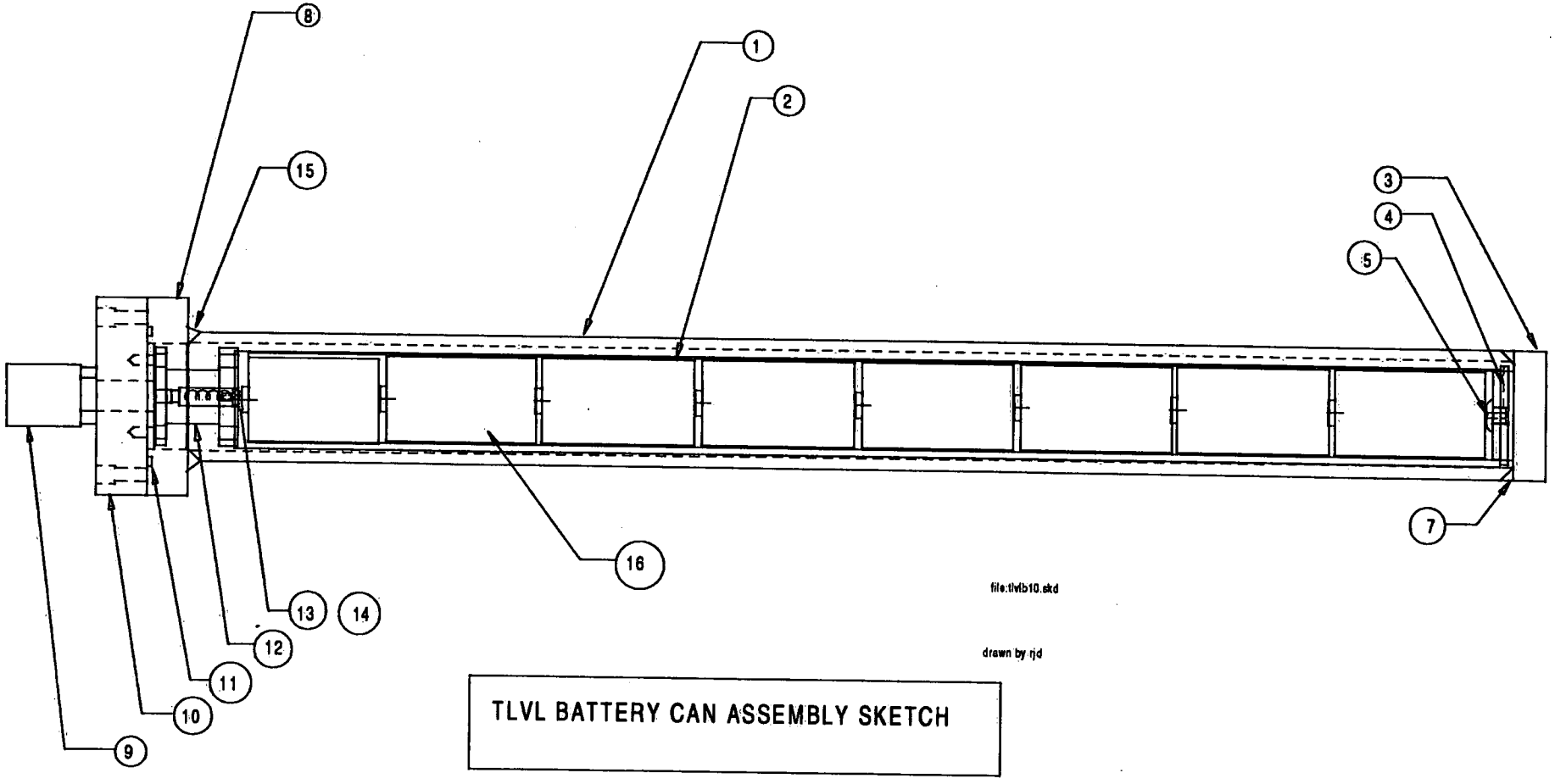


TLVL FRONT PORT

D3 BATTERY CAN MECHANICAL SKETCHES

The battery can comes in two forms an 8 cell version and a 9 cell version. The 8 cell version is for shot missions or missions without the UW lamp. The 9 cell version provides a slightly larger capacity and is suited to longer duration missions with the lamp, particularly under cold water conditions.

The battery can uses multiple Teflon insulated leads wired in parallel to connect from the upper and lower contact to the underwater connector. Excess wire is taken up by the wire spool.



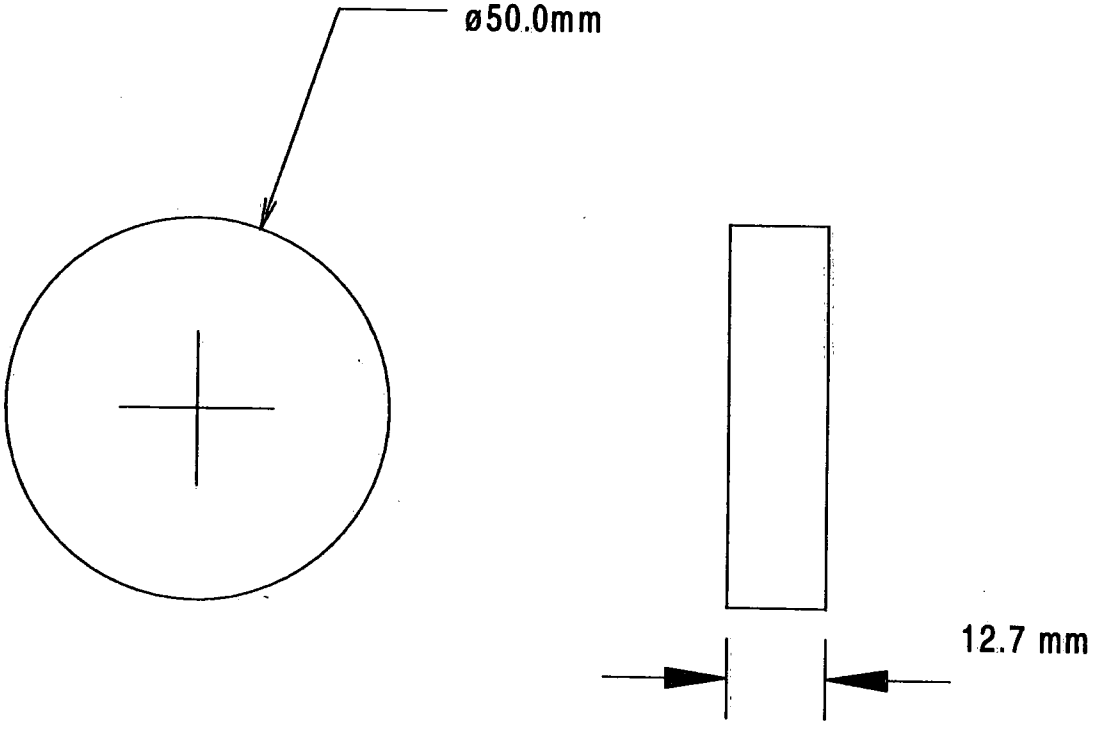
file:tlvlb10.skd

drawn by rjd

TLVL BATTERY CAN ASSEMBLY SKETCH

TLVL BATTERY CAN PARTS
refer to TLVL battery can assembly sketch tlvlb10.skd for ID numbers

ID	Part sketch	Description	Mfg or Supplier	Mfg No
1	tlvlb2.skd	battery can main tube	shop	
2	tlvlb4.skd	acrylic tube battery holder	surplus correr tubes from warehouse	
3	tlvlb1.skd	rear end plate	shop	
4	tlvlb5.skd	acrylic end plate	shop	
5	tlvlb8.skd	rear contact (brass)	shop	
6				
7		weld		
8	tlvlb3.skd	front flange	shop	
9		Impulse 4 pin female uw connector	McQuest Marine Burlington	BH-4-FS
10	tlvlb6.skd	front end plate	shop	
11		O-ring	Canadian Bearings Woodward Ave	
12	tlvlb7.skd	PVC wire spool	shop	
13	tlvlb7.skd	brass front contact	shop	
14	tlvlb7.skd	stainless steel contact spring	shop	
15		weld		
16		Panasonic alkaline D cell	Bluestreak MarkHam	AM-1PA



TLVL Battery Can

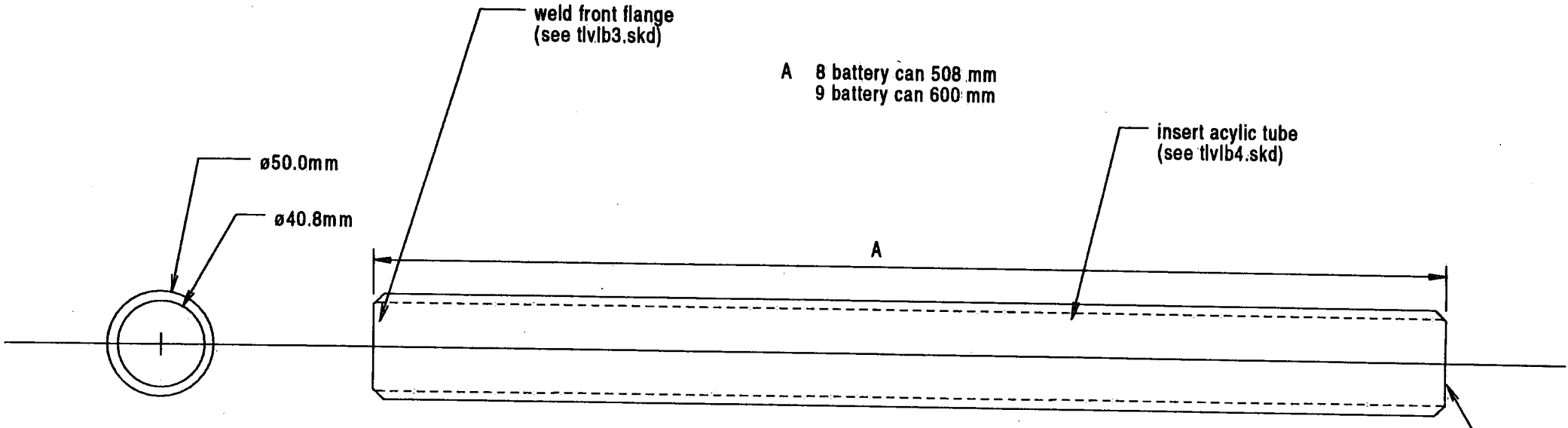
Rear End Plate

material 6061-T6 Al alloy

file: tlvlB1.skd

drawn by rjd

Note: TIG weld to rear of TLVL battery can main tube ("Butt Weld")



Chamfer ends for welds

1) Fillet weld front flange

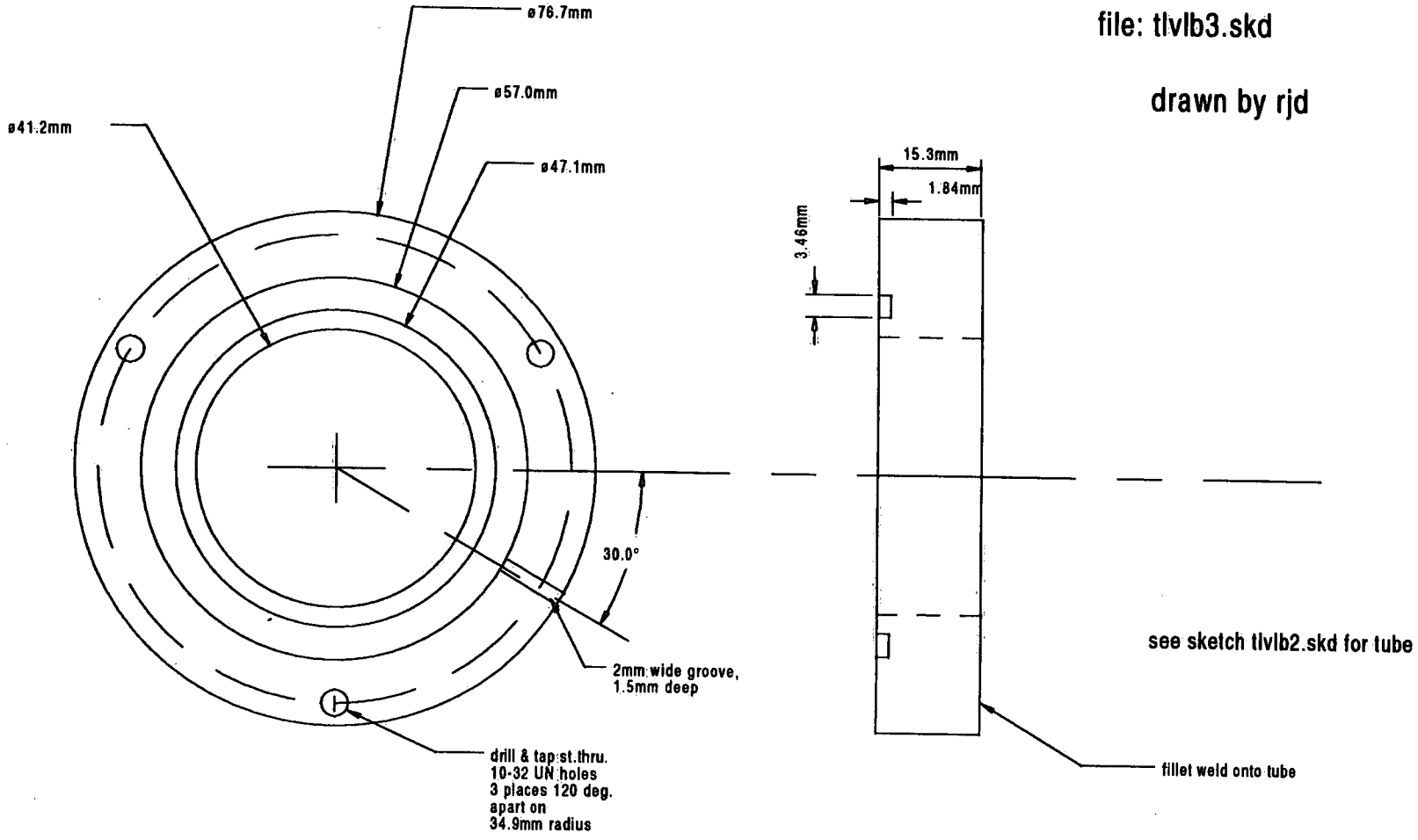
2) Butt weld rear end plate

material 6061-T6 Al alloy

TLVL Battery can main tube

drawn by rjd

tlvlb2.skd



file: tlvlb3.skd

drawn by rjd

see sketch tlvlb2.skd for tube

All dimensions in mm

material 6061-T6 Al

Note: perform final machining on unwelded face after welding and annealing to allow for distortion

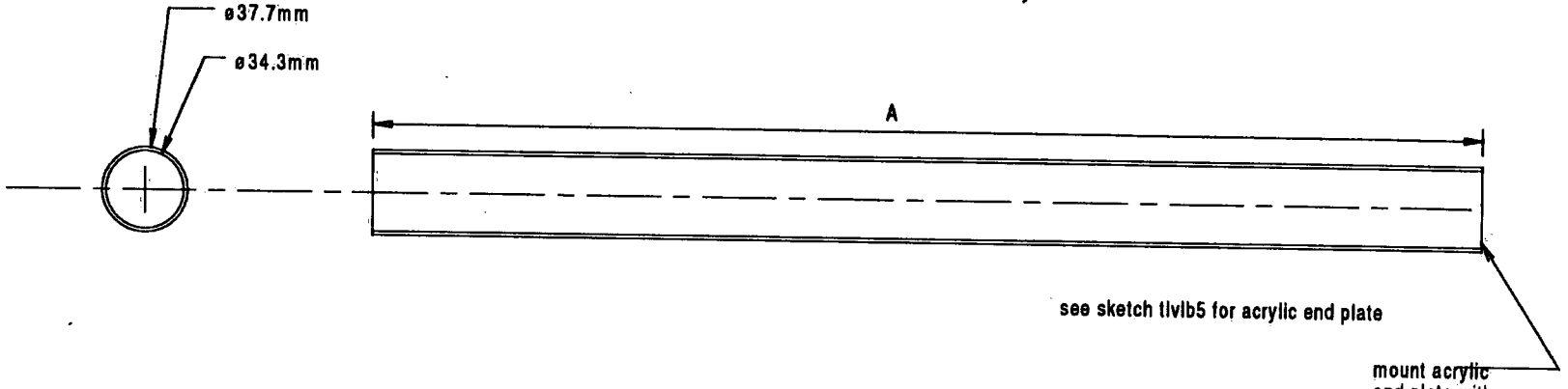
TLVL BATTERY CAN FRONT FLANGE

file: tlvlb4.skd

drawn by rjd

A 485 mm for 8 battery tube

for 9 battery tube

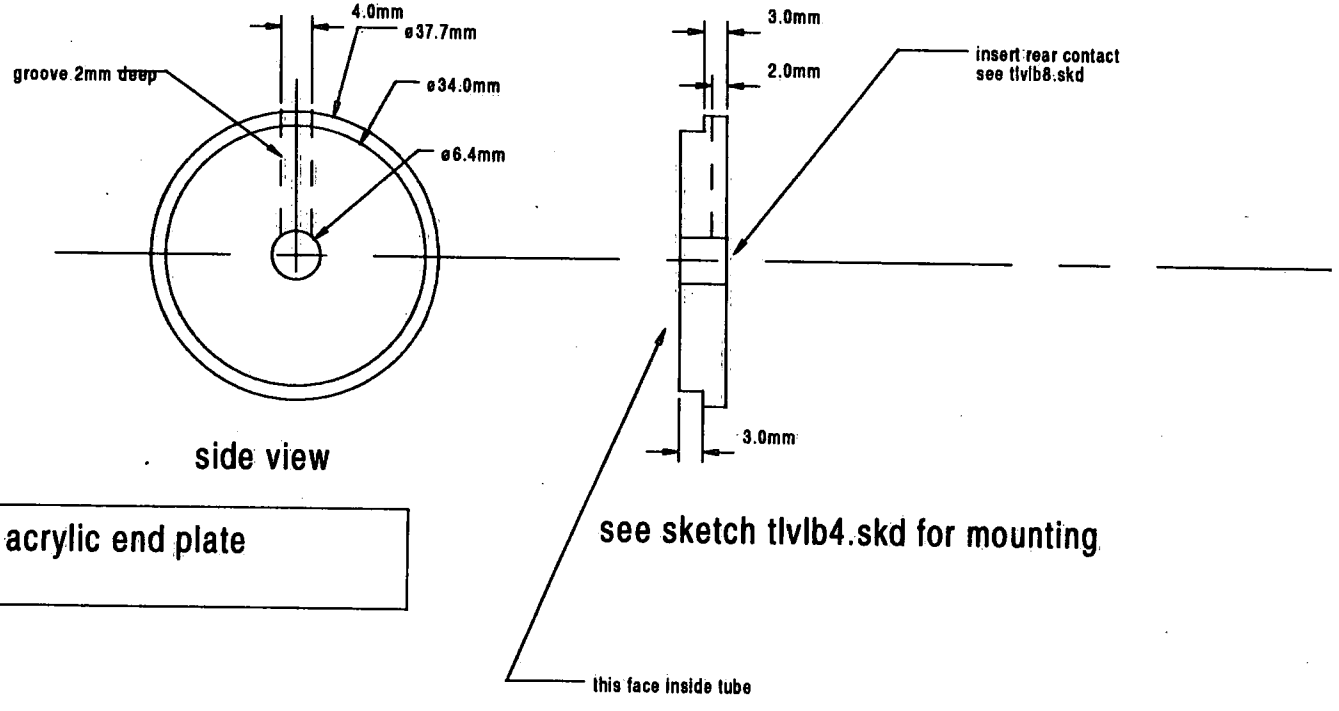


material: acrylic tube

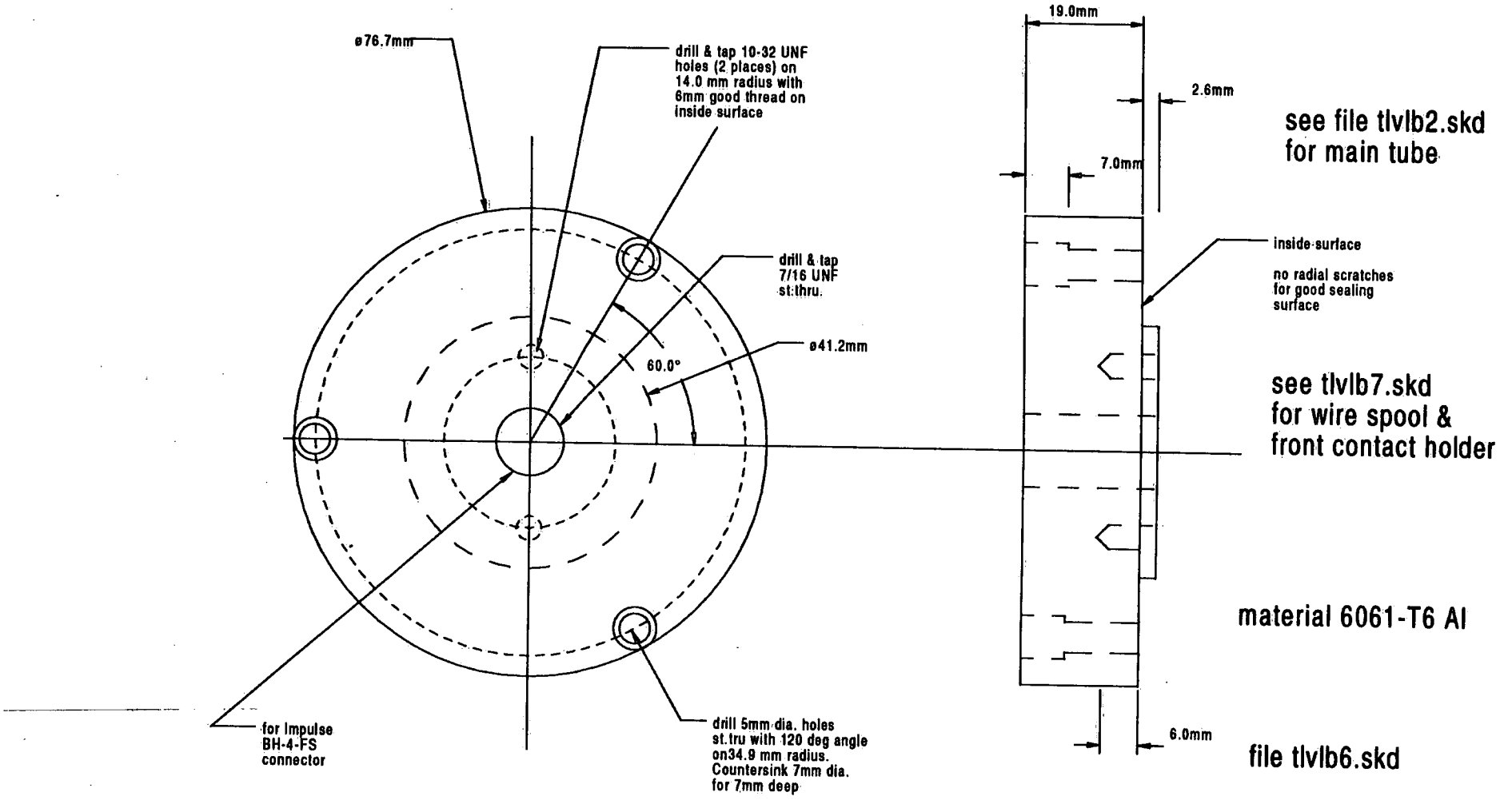
TLVL battery can acrylic tube battery holder

acrylic tubes from re-cycled sediment core tubes

tlvlb5.skd drawn by rjd



acrylic end plate



see file tlvlb2.skd
for main tube

see tlvlb7.skd
for wire spool &
front contact holder

material 6061-T6 Al

file tlvlb6.skd

drawn by rjd

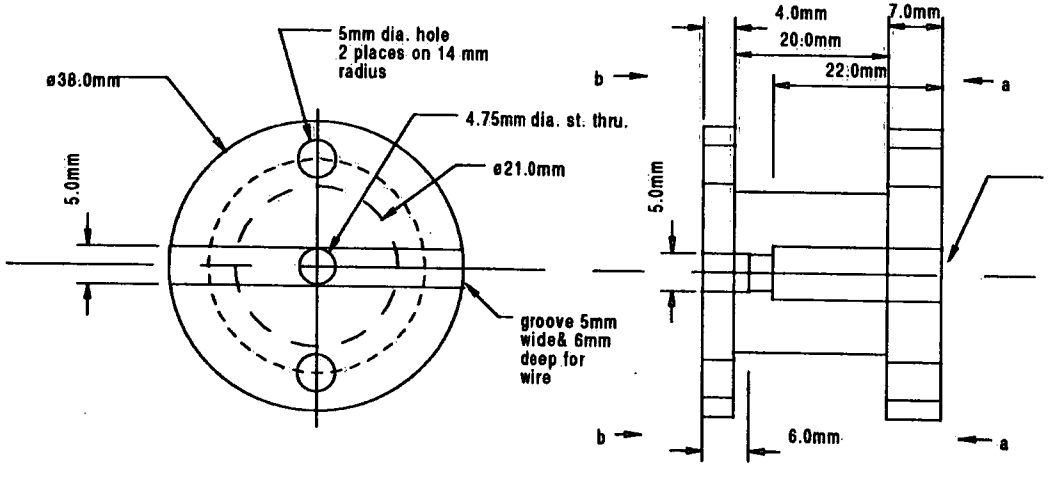
BATTERY CAN FRONT END PLATE

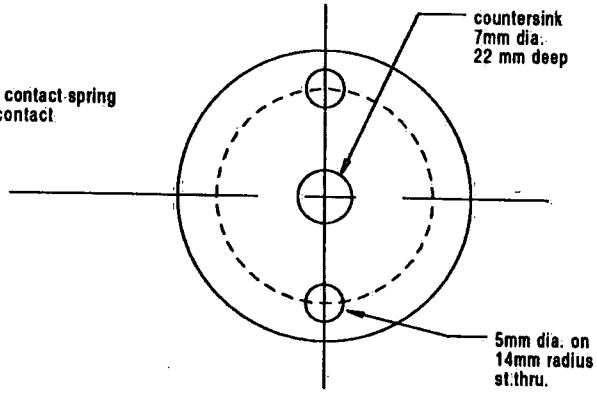
TLVL Battery can wire spool & front contact holder

tlvlb7.skd

drawn by rjd

material PVC





view b-b

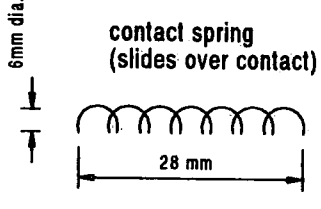
side view

view a-a

spool holds excess wire

attaches to battery can
front plate(tlvlb6.skd)

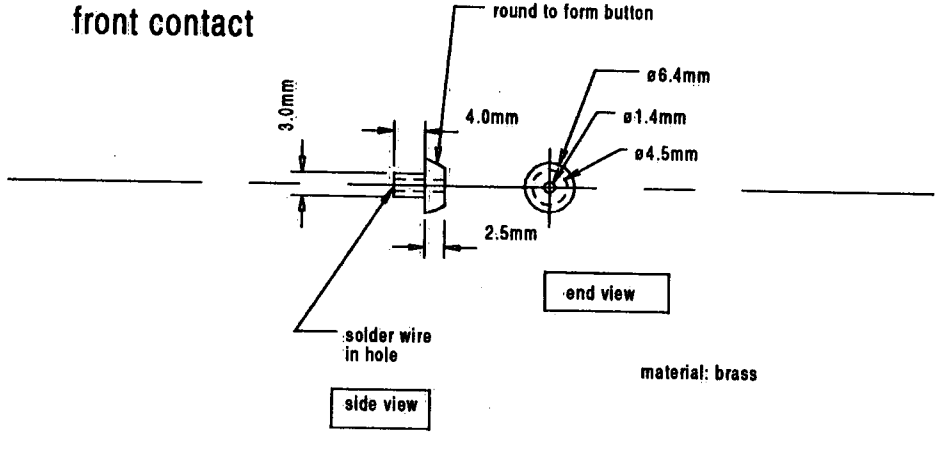
front contact spring



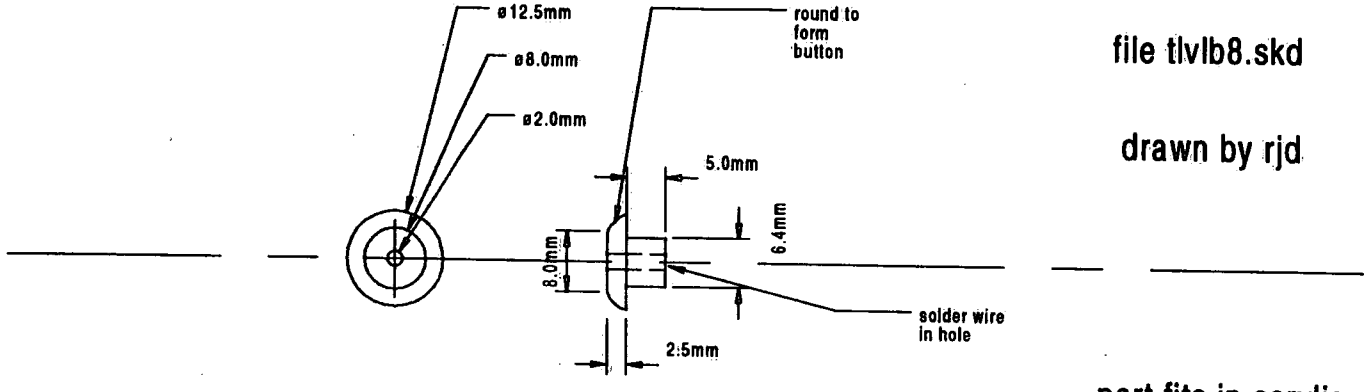
wire 1mm thick
pitch ~ 2mm
material S.S.

4 kg. nom. force
when batteries
installed

front contact



material: brass



file tlvlb8.skd

drawn by rjd

end view

side view

part fits in acrylic end plate tlvlb5.skd

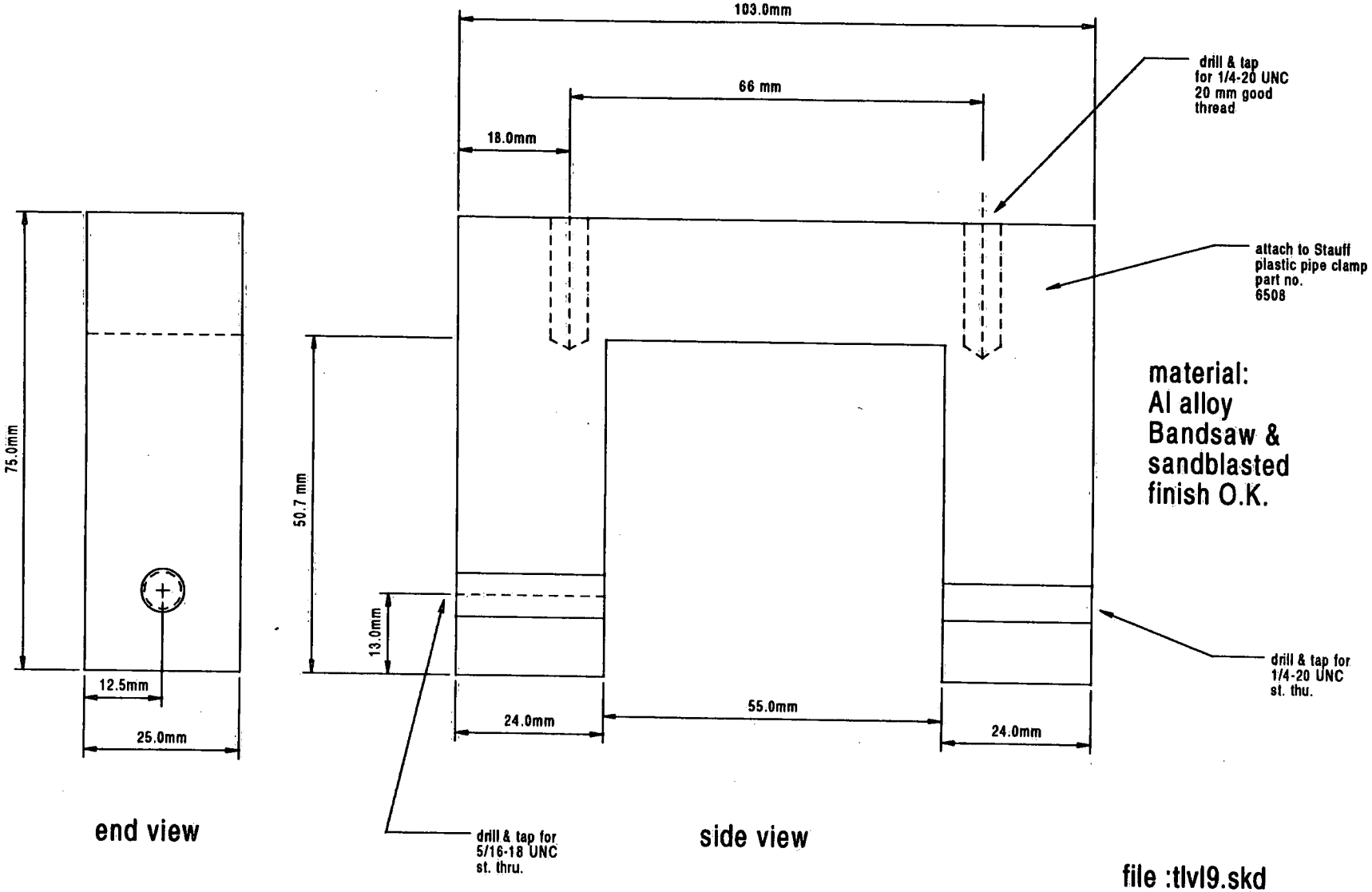
button on inside face of tube

material: brass

TLVL BATTERY CAN REAR CONTACT

D4 BRACKETRY

The brackets shown were designed to mount the sytem components onto a steel T-Frame. The T-frame is made from two inch steel rectangular tubing and fixed to the bottom with railway wheels.

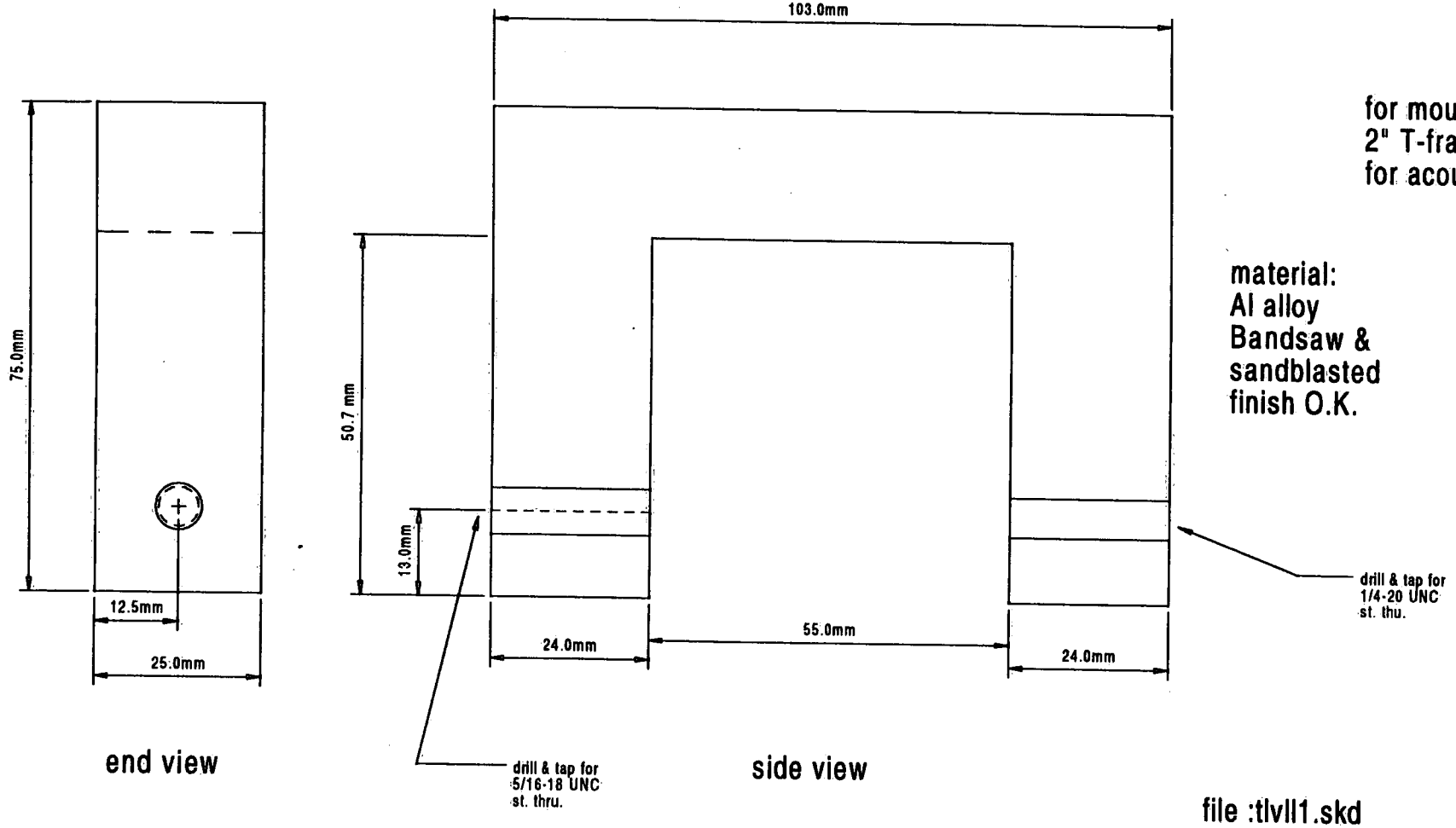


designed to fit on 2"
T-frame used for acoustic logger

file :tlvl9.skd

drawn by rjd

TLVL BATTERY CAN BRACKET (holds battery to T-Frame)



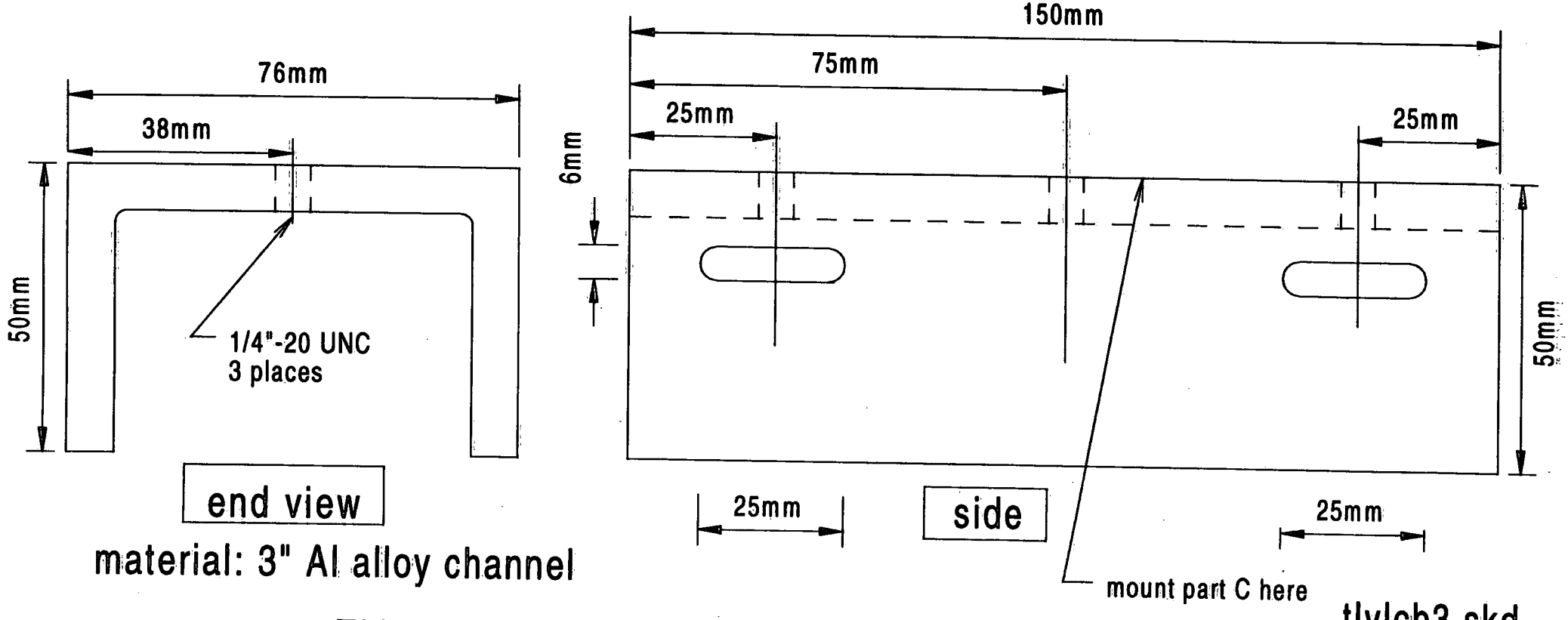
for mounting on
2" T-frame used
for acoustic logger

material:
Al alloy
Bandsaw &
sandblasted
finish O.K.

file :tlvll1.skd

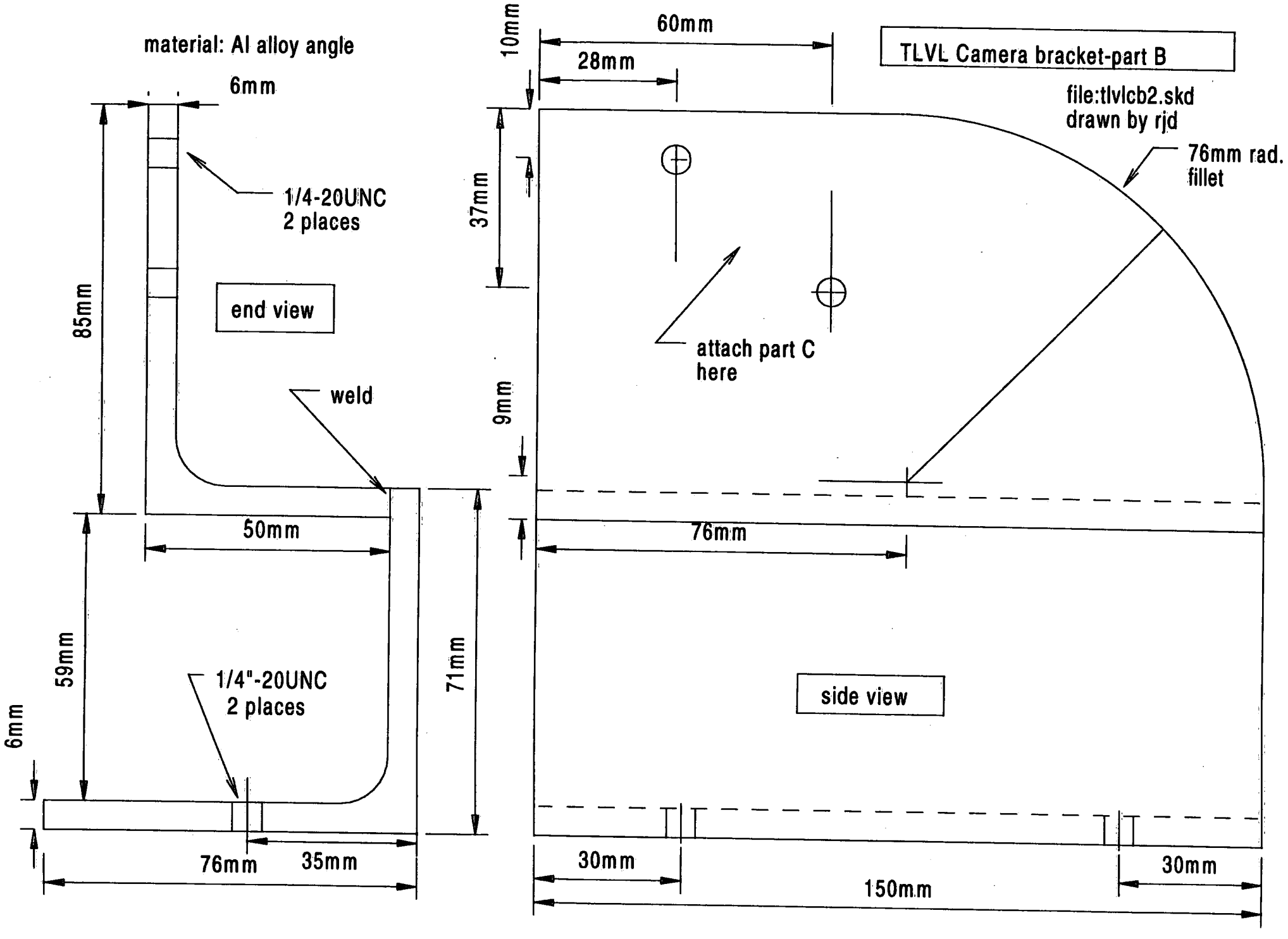
drawn by rjd

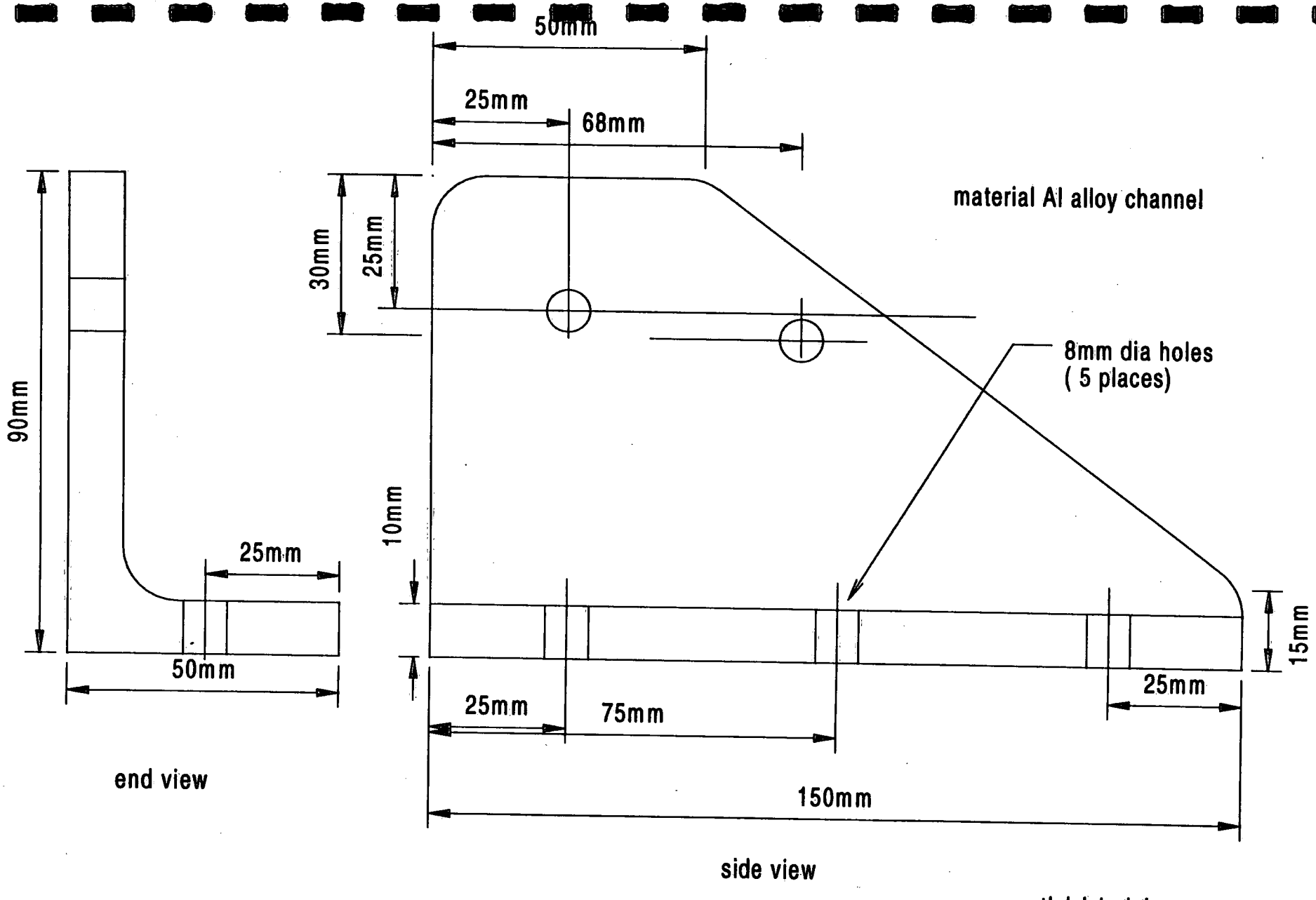
TLVL "Deep- Sea Micro-Sealite" Mounting Bracket



TLVL CAMERA MOUNTING BRACKET- PART A

tlvlcb3.skd
drawn by rjd


material: Al alloy angle
6mm
1/4-20UNC
2 places
85mm
end view
weld
50mm
59mm
1/4"-20UNC
2 places
6mm
71mm
76mm
35mm
10mm
60mm
28mm
TLVL Camera bracket-part B
file:tlvlcb2.skd
drawn by rjd
76mm rad.
fillet
37mm
attach part C
here
9mm
76mm
side view
30mm
150mm
30mm



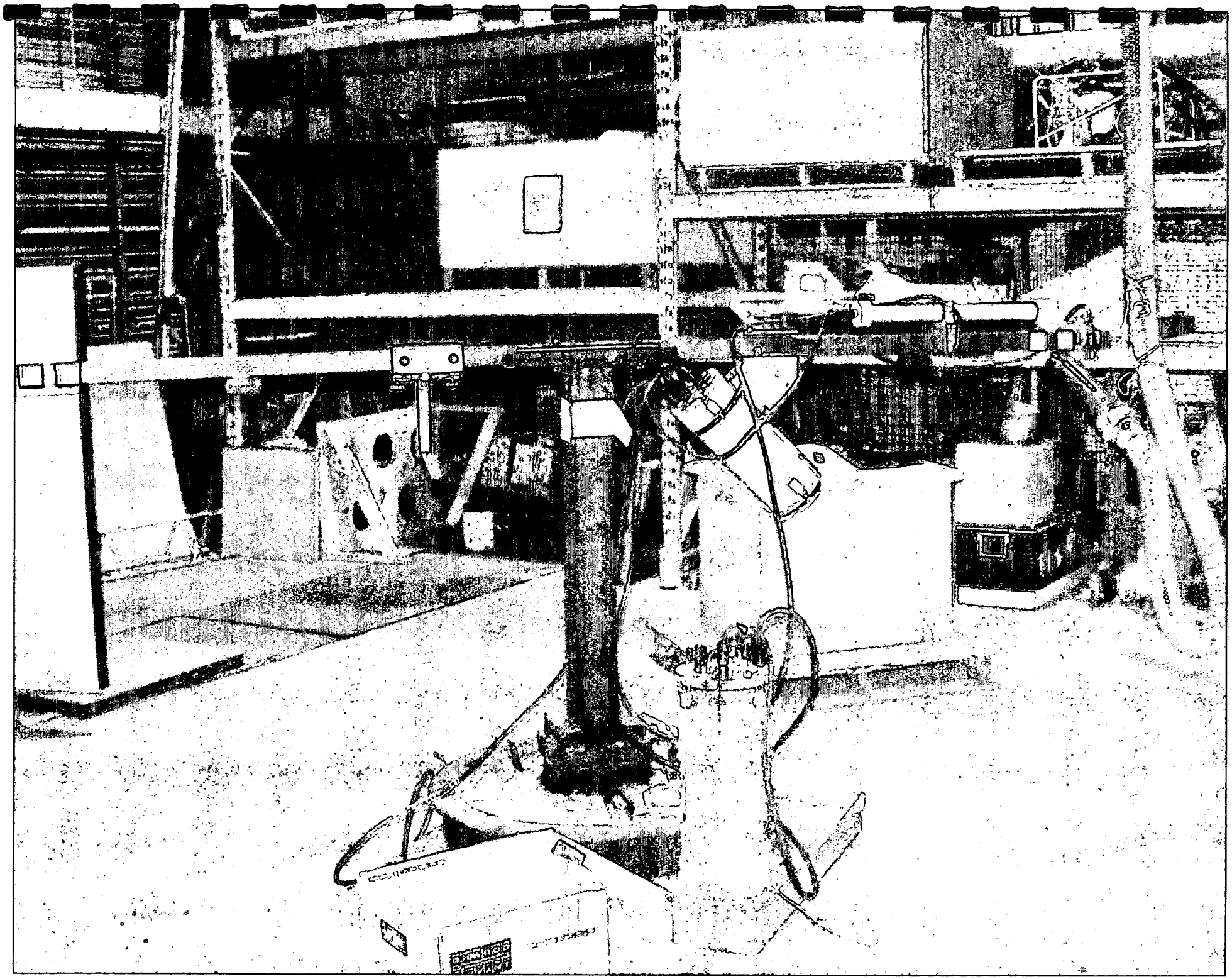
tlvlcb1.skd

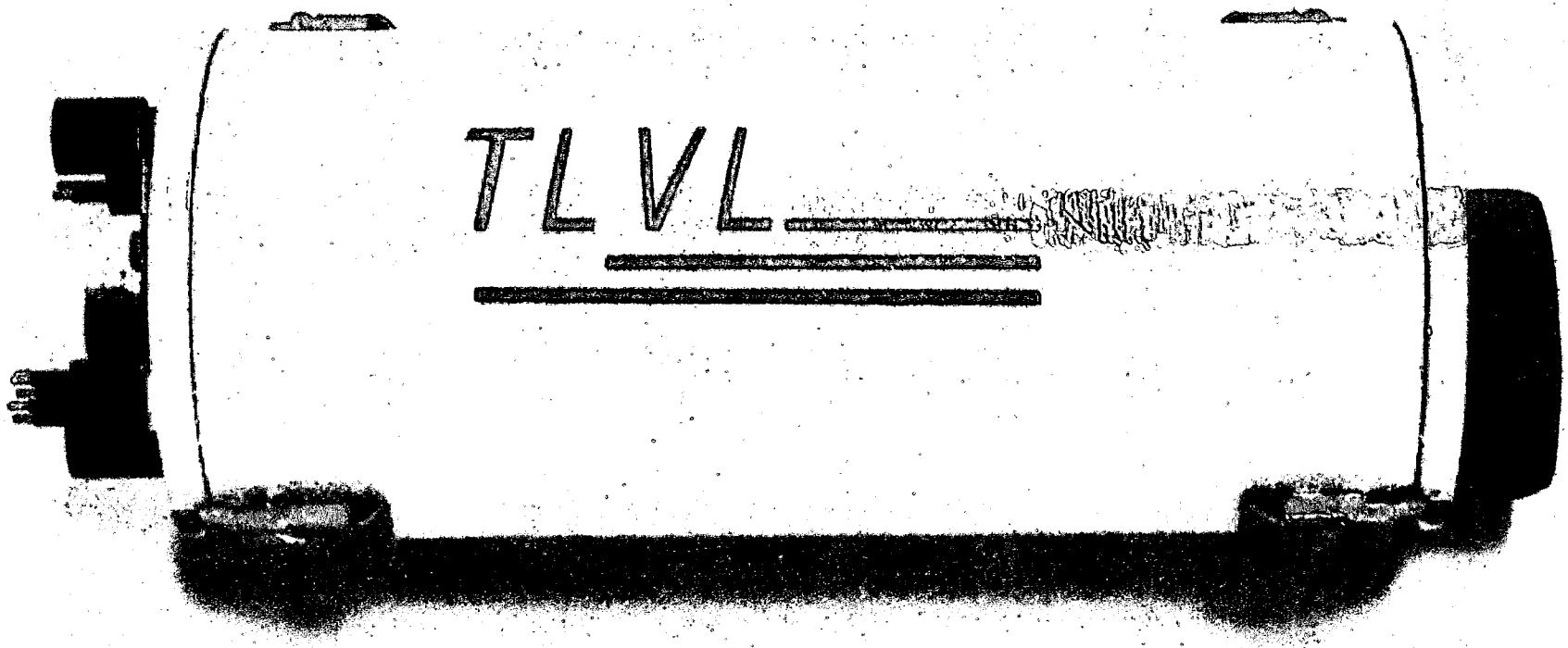
drawn by rjd

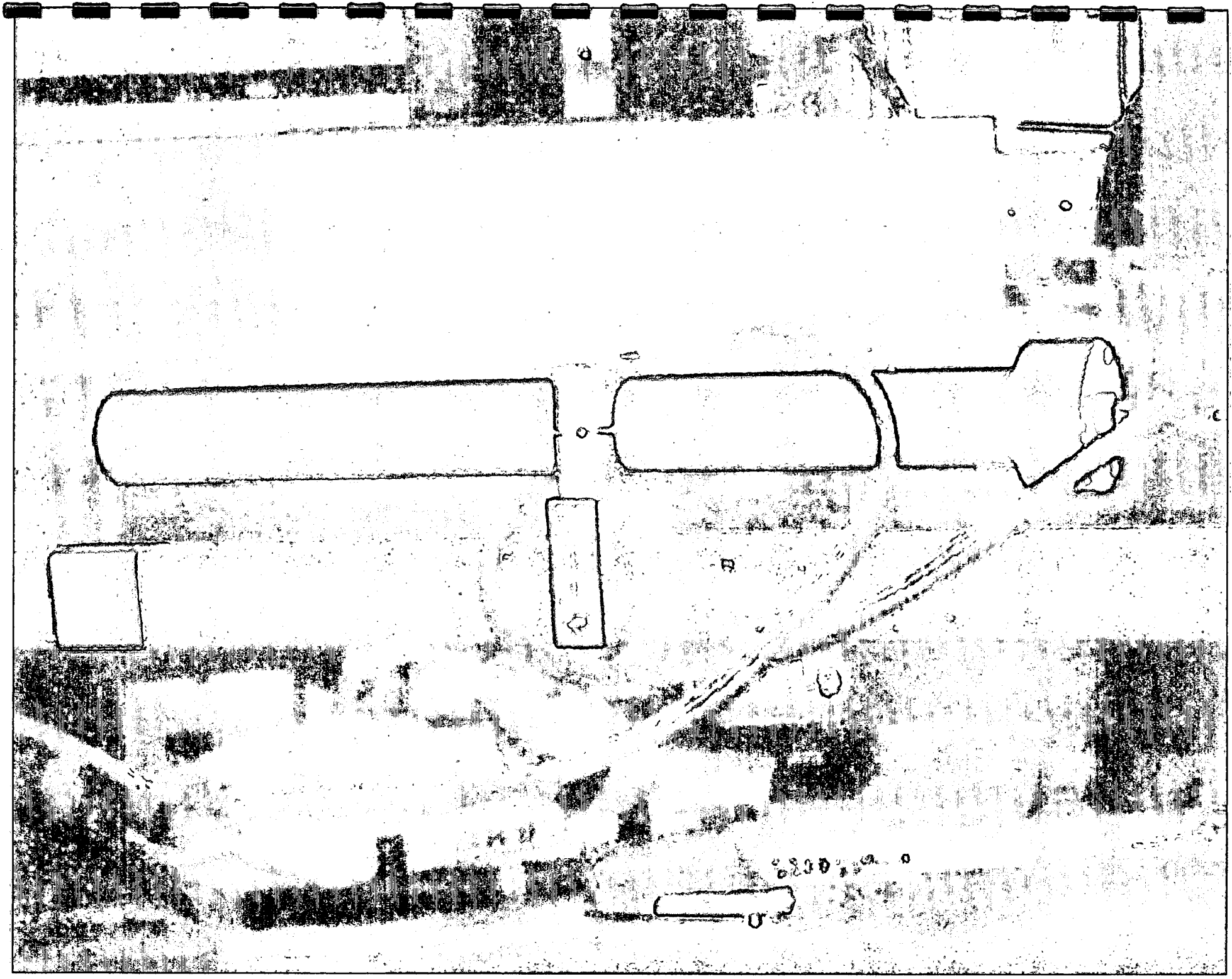
TLVL CAMERA BRACKET - PART C

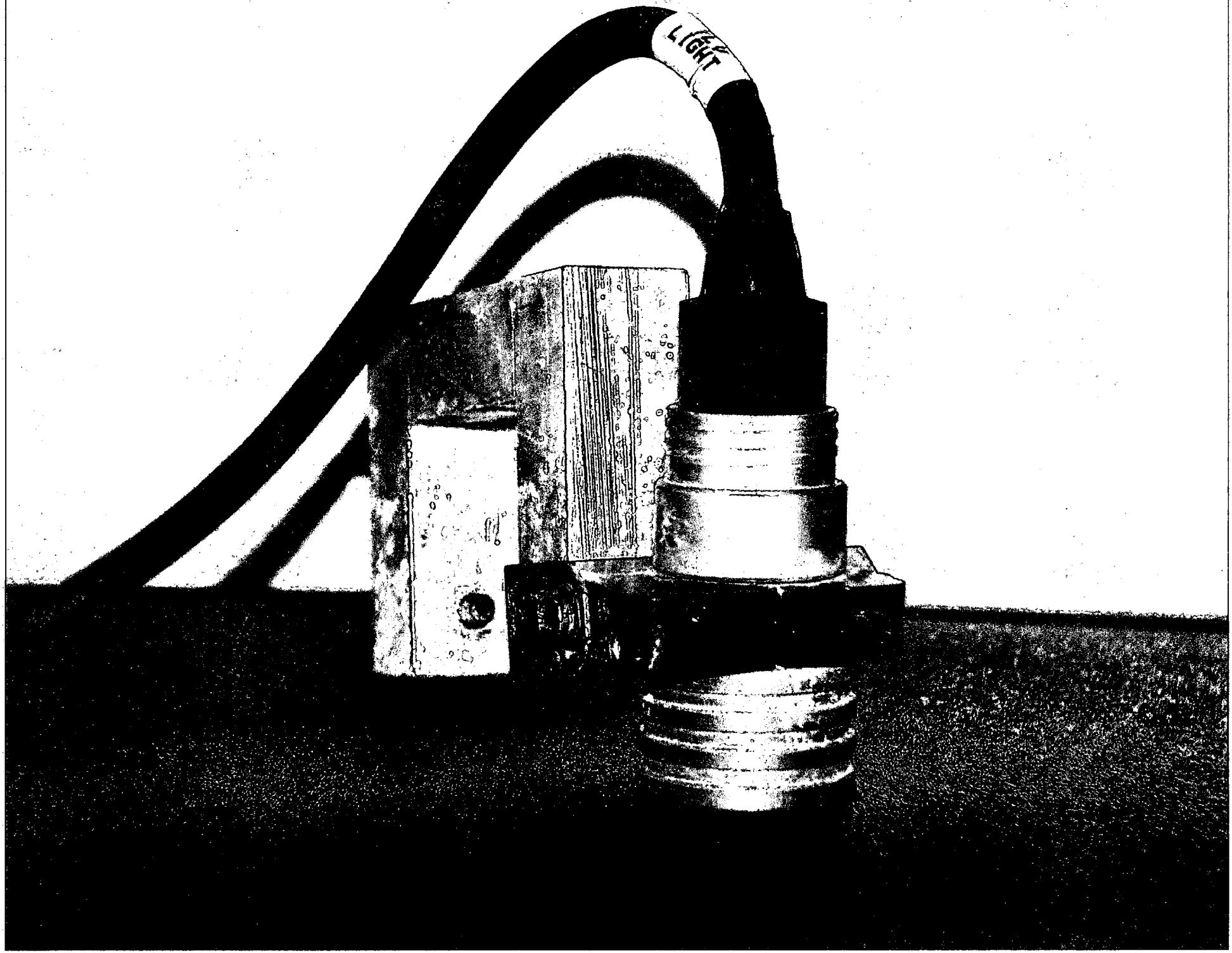
E1 PHOTOGRAPHS

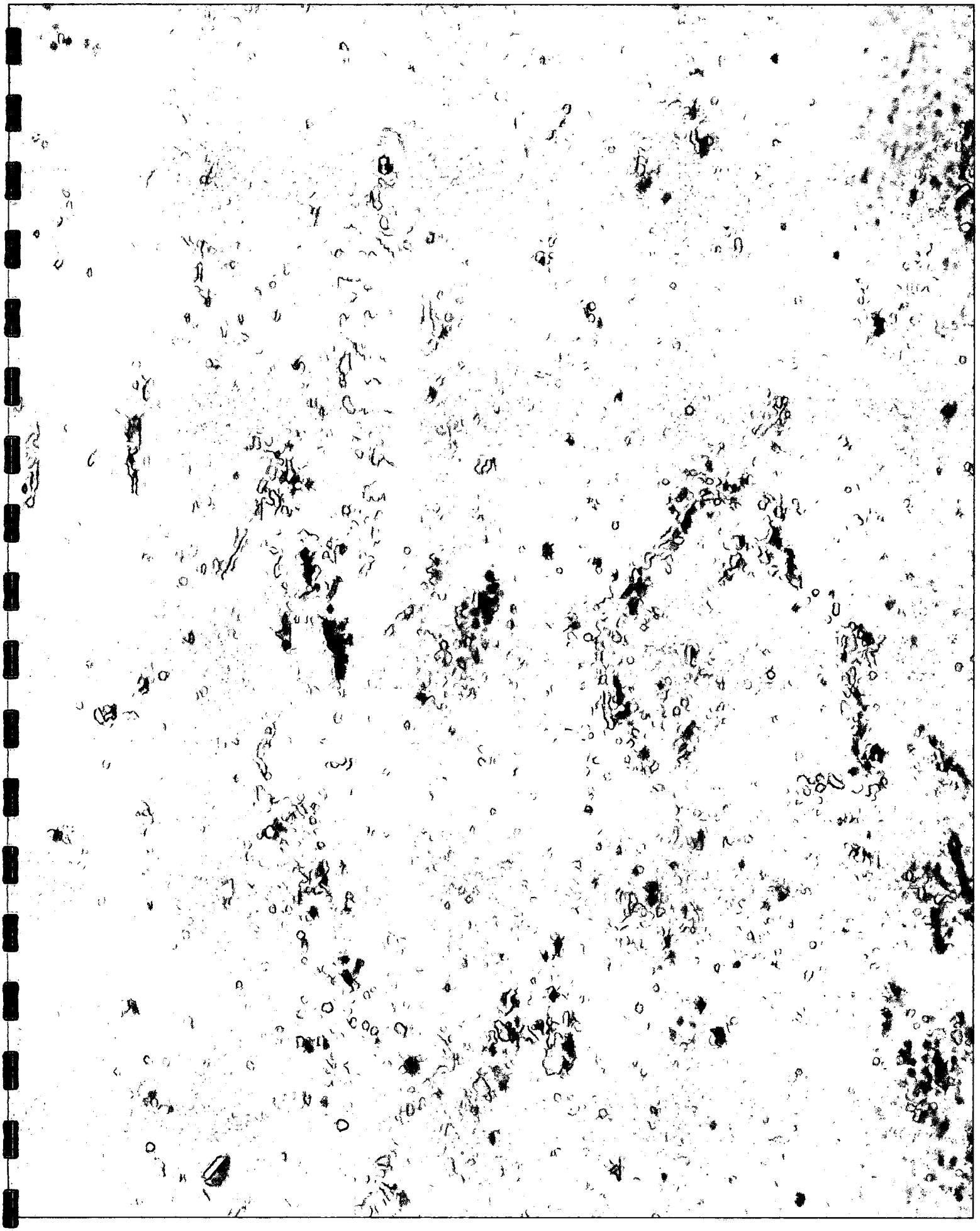
E2 STILL PHOTOS

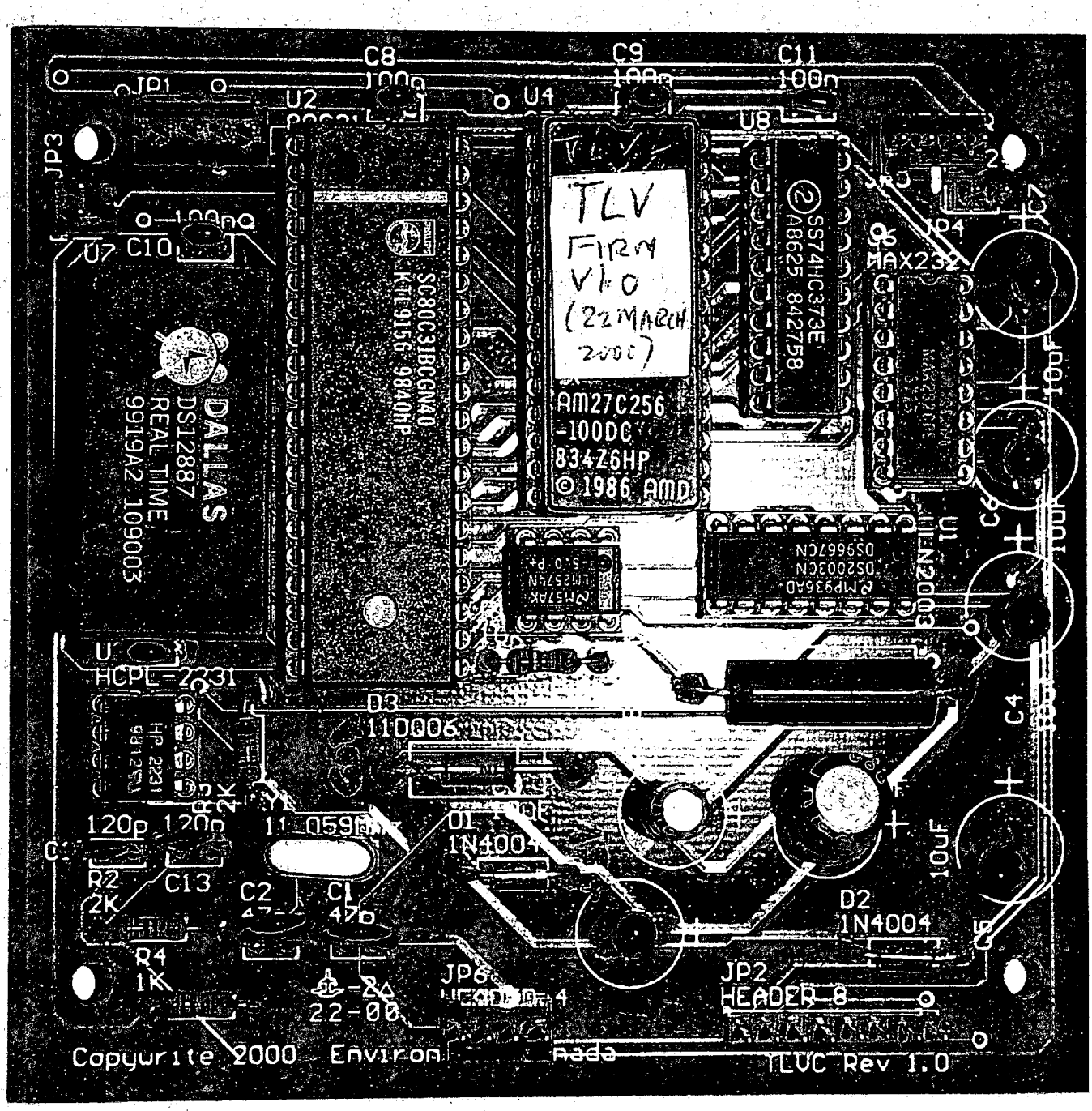



TLVL




LIGHT




TLV
FIRM
V1.0
(22 MARCH
2000)
AM27C256
-100DC
834Z6HP
© 1986 AMD
SC80C31BCGN40
KTL9156 9840HP
SS74HC373E
A8625 842758
MAX232
DALLAS
DS12887
REAL TIME
9919A2 109003
HCPL-2231
D3
110006
D1
1N4004
D2
1N4004
JP2
HEADER 8
JP6
22-00
R2
2K
R4
1K
C13
C2
C1
47p
120p
C8
C9
C11
C10
C4
C6
C7
10uF
Copyurite 2000 Environ
TLVC Rev 1.0

E3 MOVIES

Two Quicktime movies are attached in the CD-R in a single file(TIvlcornwall.mov). They show a fish moving in the field of view in one sequence and show spontaneous bubble formation in the second movie.

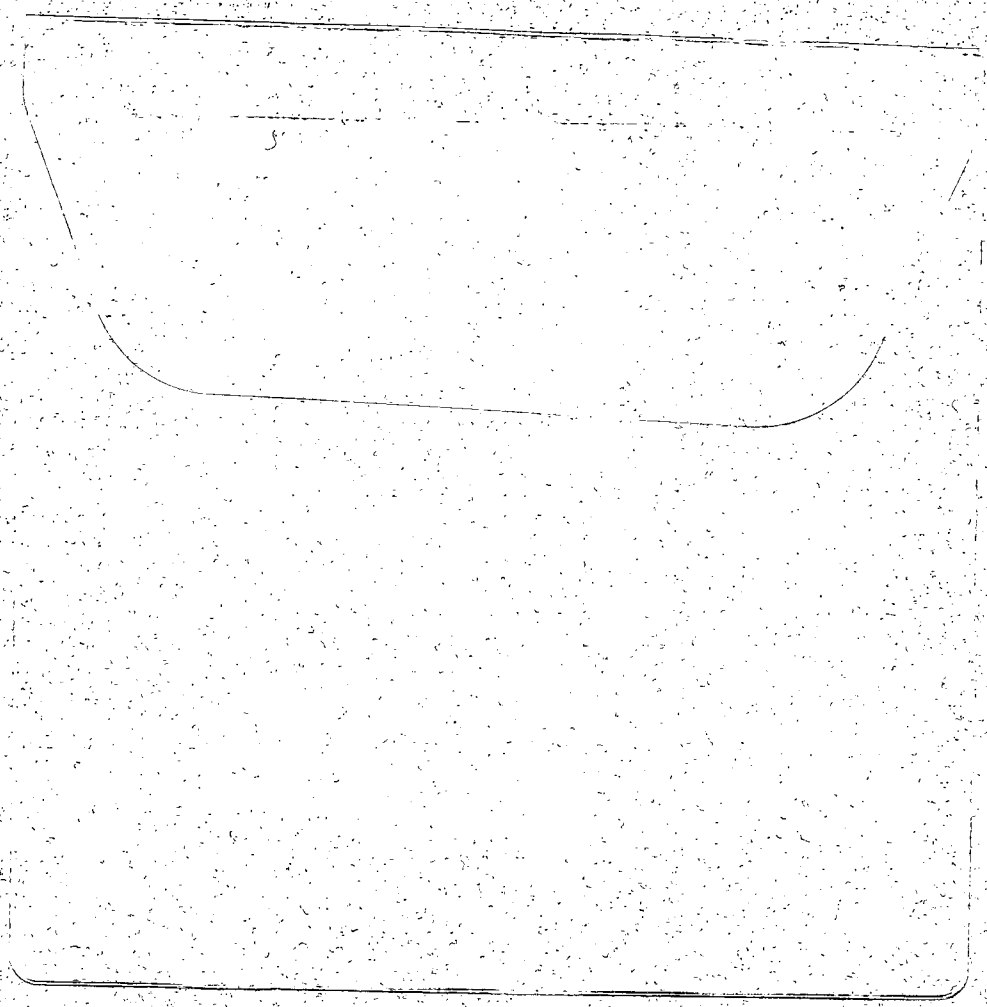
F ACKNOWLDEGEMENTS

We would like to thank a number of people specifically:
D. Dunfield of Dunfield Development Systems, Ottawa(http://www.dunfield.com) for the software code, H. Savile for the conceptualization and fabrication of most of the mechanical housings, J. Gabriele for circuit board layout and electronic troubleshooting, J. Dolanjski for acoustic logger interfacing, B. Trapp for assistance during field work and the preparation of the Quicktime movies, D. Gilroy and T. Breedon for dive support, V. Bun(volunteer) for board and system wiring and drawing the wiring interface sketches.

Additionally D. Heffler BIO, Dartmouth NS discussed his time lapse video logger (based on another camera and processor) with RJD during a visit to BIO.

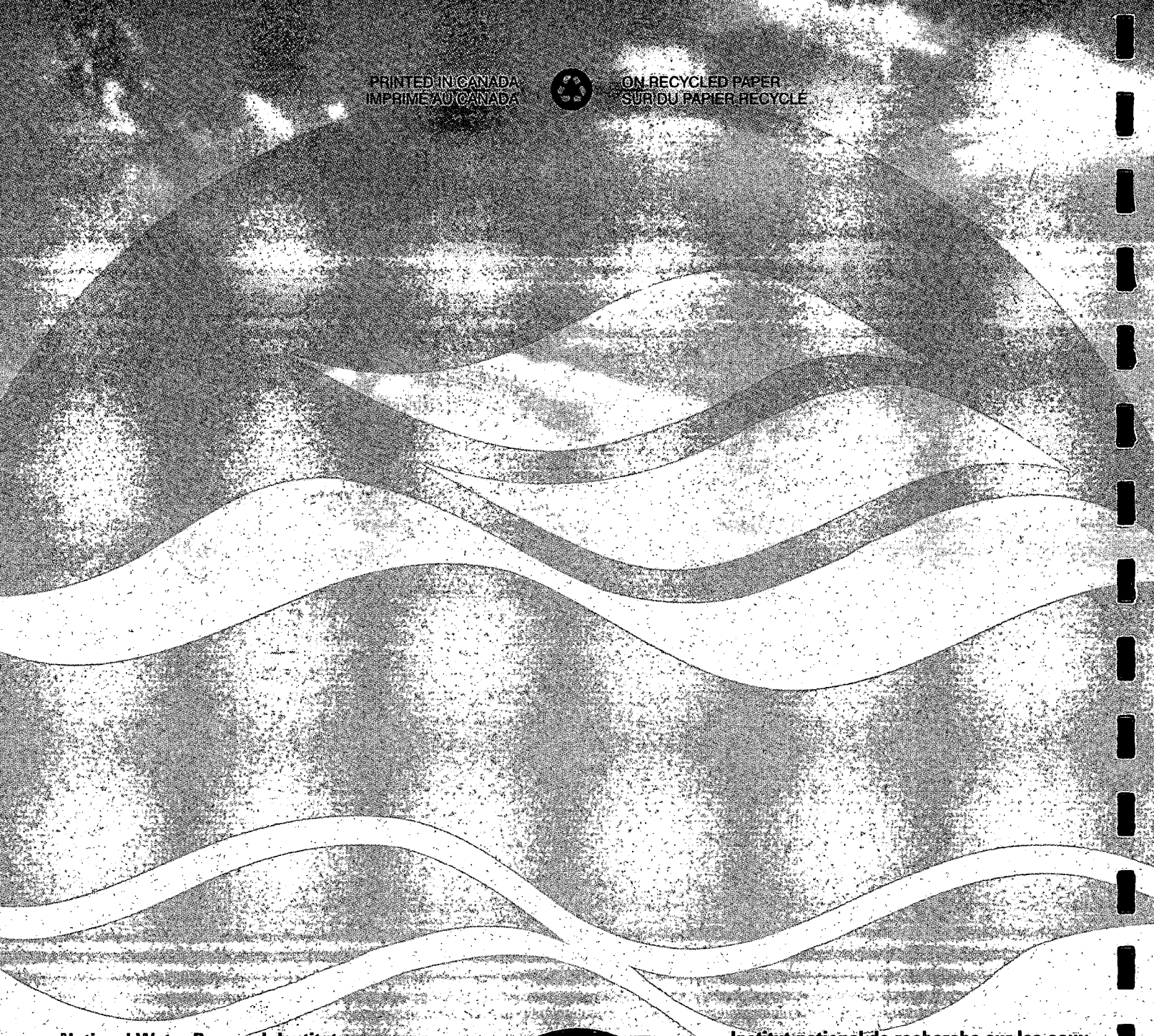
Financing was from the Great Lakes Cleanup Fund.

ENVIRONMENT CANADA LIBRARY, BURLINGTON
3 9055 1016 6282 2



PRINTED IN CANADA
IMPRIMÉ AU CANADA

ON RECYCLED PAPER
SUR DU PAPIER RECYCLÉ



National Water Research Institute
Environment Canada
Canada Centre for Inland Waters
P.O. Box 5050
867 Lakeshore Road
Burlington, Ontario
L7R 4A6 Canada

National Hydrology Research Centre
11 Innovation Boulevard
Saskatoon, Saskatchewan
S7N 3H5 Canada

NATIONAL WATER RESEARCH INSTITUTE
INSTITUT NATIONAL DE RECHERCHE SUR LES EAUX

Institut national de recherche sur les eaux
Environnement Canada
Centre canadien des eaux intérieures
Case postale 5050
867, chemin Lakeshore
Burlington, Ontario
L7R 4A6 Canada

Centre national de recherche en hydrologie
11, boul. Innovation
Saskatoon, Saskatchewan
S7N 3H5 Canada

Environment Canada Environnement Canada

Canada