REDUCED POWER BROADBAND DIGITAL
SEISMOMETER SYSTEM
MODEL EP300-DT

Date of Shipment March 15, 2004

OPERATION MANUAL
# TABLE OF CONTENTS

Part 1. The EP300-DT Seismometer .................................................................................................................. 3  
1. Introduction................................................................................................................................................ 3  
   i. Orientation ........................................................................................................................................... 3  
   ii. Connection ........................................................................................................................................ 3  
   iii. Power ............................................................................................................................................. 4  
2. Operation of the Instrument .................................................................................................................... 4  
3. Calibration............................................................................................................................................... 4  
4. Transportation ........................................................................................................................................ 4  
5. Seismometer Transfer Function .......................................................................................................... 5  
6. EP300-DT Seismometer Specifications ................................................................................................. 6  

Part 2. The Digitizer .................................................................................................................................... 7  
1. Introduction............................................................................................................................................ 7  
2. Specifications......................................................................................................................................... 7  
3. Time keeping system. ............................................................................................................................ 9  
   i. Electrical specifications. ......................................................................................................................... 9  
   ii. Message format .................................................................................................................................. 9  
4. Communication protocol. ..................................................................................................................... 10  
5. Data layer. ............................................................................................................................................ 10  
   i. Request of channel information - TAG_CHANNEL_INFO_REQ. ......................................................... 12  
   ii. Time information - TAG_TIME_INFO. ................................................................................................ 12  
   iii. Channel information - TAG_CHANNEL_INFO. ............................................................................... 14  
   iv. Communication command - TAG_COMM_COMMAND. ................................................................... 16  
   v. Reset command - TAG_RESET. ........................................................................................................ 18  
   vi. Log message - TAG_LOG. .................................................................................................................. 18  
   vii. Data block. ..................................................................................................................................... 18  
6. Transmission block layer. ....................................................................................................................... 20  
7. SLIP layer. ......................................................................................................................................... 20  

Appendix 2-A: SLIP protocol...................................................................................................................... 21  
APPENDIX 2-B: Working with the tiltmeter ............................................................................................. 27  

Part 3. The ORION System Interconnections .............................................................................................. 28  
1. Instrument Assembly & Connectors Pinouts .................................................................................... 28  
2. The Junction Board. ............................................................................................................................. 29  
   i. The Tiltmeter. .................................................................................................................................... 30  
   ii. Tiltmeter Interface Board.................................................................................................................... 30
Part 1. The EP300-DT Seismometer

1. Introduction.

The EP300-DT is a customized version of the Model EP300 seismometer designed in compliance with the specification of the ORION project.

Unlike traditional seismometers, electromechanical instruments are extremely rugged and do not need to be equipped with arresters or any other special devices for handling, packing, unpacking or transportation. The instruments also do not require mass centering and thus do not have mass position outputs and mass centering inputs. The leveling of a seismometer is intended only for the purpose of the proper orientation of the sensitivity axes. The instrument is fully functional with installation tilts of up to 10°. In order to account for the tilt and be able to rotate sensitivity axes into proper position in those field or ocean-bottom installations where leveling is not possible, model EP300-DT is equipped with a precision bi-axial tiltmeter. The EP300-DT outputs are directly connected to PMD Model 6503 Sensor Digitizer which is described farther below in this Manual. Seismometer, Tiltmeter and Digitizer are placed in the titanium pressure cylinder supplied by the Nautilus GmbH of Bremen, Germany. Three seismic sensors are mounted directly on the base of the cylinder; the tiltmeter with its interface board and the junction blocks and then the digitizer are placed on the top of the seismometer in that order. A photo of the EP300-DT assembly with the cylinder tube removed is shown in the Section Part 3-1 below.

i. Orientation

An arrow and the letter "N" on the upper cover of the titanium cylinder points out to North. The tiltmeter is oriented the same way as the seismometer, with its 'X' axis aligned along the 'N' and 'Y' along the 'E' axes of the seismometer respectively.

ii. Connection.

This ORION project system is equipped with an 8-pin watertight connector whose wiring diagram is shown in the Part 3 of this Manual.

The differential outputs +N, -N; +E, -E; +Z, -Z, are connected to the differential inputs of the data recorder. Each of these outputs can have a maximum swing of ±10V, resulting in the differential swing of ±20V (40V peak-to-peak).
iii. Power

Nominal input voltage is 12 Vdc, with an acceptable range of 9 to 15 Vdc. Nominal current consumption for the EP300-DT seismometer is 14mA. Peak consumption for several seconds immediately after power-on or during strong seismic events may reach up to 100mA; therefore it is necessary to use a power supply capable of producing such current. If an excessive and/or an improper polarity voltage is applied, the built-in resettable fuse will disconnect power. In this event it may be necessary to wait a few minutes until the fuse recovers.

2. Operation of the Instrument

The instrument has a very short settling time and begins operating normally within a few minutes after the power was turned on. However, like in any other seismometer, achieving the normal low noise level of operation may require many hours after the instrument has settled mechanically and the temperature gradients between and within all components come to or near zero. As any other electrochemical sensor, the EP300-DT seismometer requires no adjustment or any other maintenance over the whole life of the instrument.

3. Calibration

In the EP300-DT seismometer CAL_EN (Calibration Enable) inputs of all three seismometer channels are connected together with the common CAL_IN input. A corresponding calibration command sent to the digitizer results in a 3V step applied to the calibration coils of all three sensors. The calibration circuits are such that the output response signal should have the same initial amplitude at the input.

4. Transportation

The seismometer itself is extremely rugged and is difficult to damage during transport. Still, it is recommended to be pad the sensor with a dense foam so that it fits snugly in the packing box to prevent breaking the box in the event of a shock. If the sensor is shaken, a very slight rattling sound may be heard; it is normal and should not be a matter of concern.

---

1 Important notice: since EP300 is a force-balanced instrument, the above current consumption numbers relate to very quiet conditions when seismic signal levels are low; when checked in a laboratory environment, where an inescapable cultural, weather-related, etc. noise is always present, the balancing feedback action will probably increase consumption by up to several milliamperes. In the quiet condition in a vault or especially on the ocean-bottom the above given number will be complied with.
5. Seismometer Transfer Function

Response of the Model EP300 seismometer is velocity-flat in the passband with the sensitivity 2000 V*sec/m; low-frequency cutoff: 2nd order filter; damping 0.707 critical; high-frequency cutoff: 3rd order Butterworth filter. Analytically, the seismometer transfer function can be described by the following expression:

\[ W(jf) = \frac{K \cdot (j \cdot f)^2}{\prod_{n=1}^{5} (j \cdot f - p_n)} \]

Where \( f = \) frequency, Hz; \( j = \) imaginary unit; \( K = 6.74 \cdot 10^7 \) V/(m*sec^2).

Values of poles \( p_n \) are listed in the Table below:

<table>
<thead>
<tr>
<th>n</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.012+0.012j</td>
</tr>
<tr>
<td>2</td>
<td>-0.012-0.012j 3</td>
</tr>
<tr>
<td></td>
<td>-300</td>
</tr>
<tr>
<td>4</td>
<td>-16.8+30.0j</td>
</tr>
<tr>
<td>5</td>
<td>-16.8-30.0j</td>
</tr>
</tbody>
</table>
### 6. EP300-DT Seismometer Specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating principle</td>
<td>Electrochemical motion transducer with force-balancing feedback</td>
</tr>
<tr>
<td>Output analog signals</td>
<td>2 horizontal, 1 vertical; very broad passband, velocity flat response</td>
</tr>
<tr>
<td>Output analog signal swing</td>
<td>40 V p-p differential</td>
</tr>
<tr>
<td>Output digital signal</td>
<td>See &quot;The Digitizer&quot; section below</td>
</tr>
<tr>
<td>Dynamic Range</td>
<td>150 dB @ 1Hz</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>0.0167 - 50 Hz</td>
</tr>
<tr>
<td>Self-noise</td>
<td>Below USGS NLNM in 0.05 - 5Hz range; -170dB @ 0.033Hz. (See noise curves below)</td>
</tr>
<tr>
<td>Generator constant</td>
<td>Standard: 2000 V/m/s; Opt.: 350 - 20,000 V/m/s</td>
</tr>
<tr>
<td>Calibration input</td>
<td>1kΩ; 1V in - 1V out</td>
</tr>
<tr>
<td>Mass Lock</td>
<td>NONE REQUIRED</td>
</tr>
<tr>
<td>Mass Centering</td>
<td>NONE REQUIRED</td>
</tr>
<tr>
<td>Maximum installation tilt</td>
<td>10°</td>
</tr>
<tr>
<td>Mechanical resonances</td>
<td>&gt;150 Hz</td>
</tr>
<tr>
<td>Environmental</td>
<td>Waterproof, submersible to 6000m</td>
</tr>
<tr>
<td>Temperature range</td>
<td>-12 to + 55 °C</td>
</tr>
<tr>
<td>Case diameter</td>
<td>225 mm 538</td>
</tr>
<tr>
<td>Case height</td>
<td>mm</td>
</tr>
<tr>
<td>Weight (Al housing)</td>
<td>~37 kg</td>
</tr>
<tr>
<td>Power - Standard</td>
<td>9 - 15 Vdc; 12 Vdc nominal</td>
</tr>
<tr>
<td>Supply current</td>
<td>14mA</td>
</tr>
</tbody>
</table>

### EP300 Seismometer Noise Curves

---

2 Specification of the Digitizer see farther down in this Manual

3 All three sensors stay fully operational, however they sensitivity axes will rotate in accordance with the tilt.
Part 2. The Digitizer

1 Introduction.

The PMD 6503 Sensor Digitizer is the essential minimum version of the PMD 6500 family of data acquisition products, in that it eliminates both the hard disk and PC. When combined with a PMD, or other seismometer, it becomes a high-resolution digital seismometer. The digital data stream can be connected via an RS-232 or RS-485 serial port and a digital radio or cable link to a central data collection computer. The large SRAM buffer ensures reliable data retrieval even in highly unreliable transmission conditions. The digitizer consumes extremely low power without sacrificing its high resolution, deep anti-aliasing filtering, and wide range of programmable parameters.

This versatile device can adapt to many applications and configurations:
- Mounted within the case of a PMD broadband seismometer.
- As a stand-alone attachment, it can convert the output of any analog sensor into a digital stream.
- It can be readily fit into a benthic sphere or other pressure vessel along with a PMD broadband seismometer for OBS applications.

2 Specifications.

<table>
<thead>
<tr>
<th>DIGITIZER</th>
<th>Converter Type:</th>
<th>24-bit Δ-Σ; 320 kHz Base Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Range:</td>
<td>&gt;137 dB @ 100 sps (rms to FS)</td>
<td></td>
</tr>
<tr>
<td>Data Channels:</td>
<td>3 (4)²; opt. up to 16; Differential or Single-Ended</td>
<td></td>
</tr>
<tr>
<td>Sampling Rates:</td>
<td>0.1, 1, 10, 20, 40, 80, 100, 200, 500, 1000, 2000, 4000 sps</td>
<td></td>
</tr>
<tr>
<td>CMR @ 50, 60 Hz</td>
<td>120dB</td>
<td></td>
</tr>
<tr>
<td>Analog Anti-Aliasing Filter:</td>
<td>&gt;100 dB @ primary sampling rate</td>
<td></td>
</tr>
<tr>
<td>Digital Filter (@ output Nyquist):</td>
<td>&gt;140 dB @ 200 sps (FIR or opt. IIR)</td>
<td></td>
</tr>
<tr>
<td>Programmable Gains:</td>
<td>1,2,4,8,16,32,64</td>
<td></td>
</tr>
<tr>
<td>Differential Input Signal Range:</td>
<td>Programmable: ±2.5, ±20 V</td>
<td></td>
</tr>
<tr>
<td>Input Impedance</td>
<td>±2.5V - 1MΩ; ±20 V - 300kΩ</td>
<td></td>
</tr>
<tr>
<td>Overvoltage Prot.</td>
<td>±40 V</td>
<td></td>
</tr>
<tr>
<td>State-of-Health Channel:</td>
<td>Full 24-bit resolution</td>
<td></td>
</tr>
<tr>
<td>Static RAM Buffer:</td>
<td>Up to 16MB</td>
<td></td>
</tr>
</tbody>
</table>

² Fourth channel may be used as state-of-health channel or function as fully featured data channel

---

PMD SCIENTIFIC/eenetec

105F W. Dudleytown Rd.
Bloomfield, CT 06002
pmdsci@att.net

Tel: Sales: 1-314-454-9977; Support: 1 860-242-8177
Fax: Sales: 1-314-454-9979; Support: 1-860-242-7812
www.eentec.com
### TIMING SYSTEM

<table>
<thead>
<tr>
<th>Type:</th>
<th>PLL controlled, GPS-referenced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. Accuracy (Software Selectable):</td>
<td>&lt;100μsec</td>
</tr>
<tr>
<td>Crystal Frequency Correction Resolution</td>
<td>0.016 ppm</td>
</tr>
<tr>
<td>GPS Duty Cycle (User Selectable):</td>
<td>Once every 18 hrs to achieve &lt;1msec accuracy</td>
</tr>
<tr>
<td>1PPS</td>
<td>Level 5V</td>
</tr>
<tr>
<td>GPS connection:</td>
<td>RS-232; speed 4800; Rx and Tx signals.</td>
</tr>
</tbody>
</table>
PMD factor calibrated: 0.016 ppm

< 1 ppm; Overvoltage protection:

1.5 - 1.5 hours

3.3 V

R x signal

Voltage

COMMUNICATION

Continuous Data Retrieval:

ENVIRONMENTAL

Operating Temperature

Storage Temperature

Size (PCB card stack)

Weight

Continuous

Data Retrieval:

COMMUNICAION

Continuous

Data Retrieval:

ENVIRONMENTAL

Operating

Temperature

Storage

Temperature

Size (PCB card stack)

Weight
6\, on  
- \, figured  
1\, 6  
V  
d  
c \, via  
± \, 6  
0 \, s  
V  
\leq \, 5  
0 \, p  
0 \, m  
W  
( \, 3  
0V:  
reset  
digitizer;  
3.3V:  
resume  
work  
RS232 or RS485 (up to 1km) serial ports; optional internal  
modes  
)  
D  
ed  
p  
e  
P  
MD  
SCIENTIFIC/eentec  
105F W. Dudleytown Rd.
3. Time keeping system.

The digitizer employs a unique time keeping system, with main clock fully synchronized with the clock used for Analog-to-Digital conversion. Therefore, exactly the pre-set number of samples is taken every second and thus the user never needs to re-index data and there are never any gaps in the recordings.

With the GPS-receiver working periodically, as required, the system is capable to maintain time precision better than 100 microseconds.

Without GPS if the system was pre-calibrated at the factory the clock precision can be as high as 0.02 PPM.

The customized time keeping system that was implemented for the Orion Project corrects time just at the moments of synchronization. With 1PPM crystals installed a 1 sample gap (at 100 sps sampling rate) may appear once in 3 hours.

The messages id are to be sent for ~1 minute with periodicity from 0.5 to 1.5 hours. The system evaluates a few messages and uses the average value to reduce error. After successful synchronization the system remains in the "sleep" mode for 0.5 hours and then waits for 1 hour for the next session.

i. Electrical specifications.

Rx signal have RS-232 levels, communication speed 9600;
1 PPS has level 3.3V, square wave with a rising edge corresponding to the beginning of a second.

ii. Message format.

A time message that sent just after the rising edge of a PPS signal has the following format:

TyyddmmHHMMSS\r\n
where
T - this letter "T" is a time message identifier;
yy - two last digits of the year;
mm - month;
dd - day of month;
HH - hour;
MM - minutes; -
SS - seconds;
\r - carriage return 0Dh;
\n - new line symbol 0Ah.
4. **Communication protocol.**

This system is capable of transferring data over RS-485 port at up to 460.8 Kbps rate for a distance of up to 1km (the latter with a slight reduction of the baud rate), which is possible because this is a differential port and hence immune to noise. The current realization supports both full-duplex (4 wires) and half-duplex (2 wires) modes. The same port has an RS-232 interface.

The digitizer has an internal RAM buffer up to 16Mb. The use of a full size buffer could be necessary in case of communication problems.

The communication protocol has 3 layers:
- **Data** - The system's internal format of data storage and transmission command is based on objects (C language STRUCTures). The type of structure is identified by a tag. All data are binary;
- **Block** - Provides a transmitting block with a block length and a one's complement of a one's complement checksum;
- **SLIP** - Transmission protocol over a serial port.

5. **Data layer.**

System internal format of data storage and transmission command is based on objects (C language STRUCTures). The type of structure is identified by a tag.

All data are binary. Data representation in DSP TMS320V5409 is a little tricky. A short integer has the same byte sequence as in IBM-PC. It means that LSByte comes first, MSByte - second. But a long integer number consists of two 2 byte-words, that are placed in reversed order in respect to the IBM-PC format. Thus, if in IBM-PC the byte sequence of bytes is [0123], in this DSP it would be [2301].

All tags contain headers; each header is similar to the header of a data block (see structure BufCom up to aiBuf). It is useful because some tags (log messages for instance) are saved with data blocks in the RAM buffer. Only some tags have structures associated with them. The others contain only a header.

Shown below is a self-explanatory declaration that can be used to interpret tags:

```c
/* Info type definition TAGs. Define the type of transmitted info. */
#define TAG_EMPTY 0 /* Empty message was transmitted. */
#define TAG_CHANNEL_INFO 1 /* ChannelInfo transmitted. */
```
#define TAG_CHANNEL_INFO_REQ 2 /* ChannelInfo request. */
#define TAG_TIME_INFO 3 /* Time keeping system information. */
#define TAG_SYSTEM_RESET 4 /* System reset. */
#define TAG_COMM_COMMAND 5 /* Communication command. */
#define TAG_LOG 6 /* Block contains log messages. */

/* Calculates tag size. */
/* Calculates a tag size. */
#define TAG_SIZE(TAG) \ 
  (((TAG) == TAG_CHANNEL_INFO) ? sizeof(ChannelInfo) \ 
  : (((TAG) == TAG_TIME_INFO) ? sizeof(TimeInfo) \ 
  : (((TAG) == TAG_COMM_COMMAND) \ 
  ? sizeof(CommCommand) : 0)))

Defines a TAG or a Block. */
#define TAG_PROVIDED(P_BUF) \ 
  ((P_BUF)->uNumber > BUFCOM_N)
/* Gives a TAG type. */
#define TAG_TYPE(P_BUF) \ 
  ((P_BUF)->uChan)
/* Produces the repetition number of the TAG(array size). */
#define TAG_NUMBER(P_BUF) \ 
  ((P_BUF)->uNumber-BUFCOM_N-1)
/* Set a tag type and a number of the TAG(array size) in buffer items. */
#define TAG_SET(P_BUF,TAG,NUMBER) \ 
  ((P_BUF)->uChan=(TAG); (P_BUF)->uNumber=(NUMBER)+BUFCOM_N+1)
/* The script defines a tagged message size in 16 bit words(short). */
#define TAG_MESSAGE_SIZE(SIZE) /* SIZE is a size of sended info. */ \ 
  (((SIZE)+sizeof(BufCom)-sizeof(short)*BUFCOM_N-1)/sizeof(short)+1)
/* The short integer in DSP TMS320V5409 has the same byte sequence as */
/* in IBM-PC. It means that LSB is the first, MSB is the second. But */
/* long integer number consist of two words(2 byte), those are swapped */
/* in comparison with IBM-PC. That is if in IBM-PC sequence of bytes */
/* is [0123] then in DSP it is [2301]. The next script swaps them back. */
/* The script swaps the short halves of a long integer. */
#define SWAP_LONG(DEST) \ 
    (((unsigned long)*(unsigned short *)(&DEST))<<16)\ 
    |(*((unsigned short *)(&DEST))+1))
/* The script swaps the short halves of a long integer in place. */
#define SWAP_LONG_IN_PLACE(DEST) (DEST)=SWAP_LONG(DEST)

The tag descriptions below are presented in the same sequence in which they are supposed to be used:

i. Request of channel information - TAG_CHANNEL_INFO_REQ.
   This tag requests that the digitizer send current channel configuration information. It is a simple command that does not have an associated structure. Can be expressed by the script:
   
   TAG_SET(&suBuffer, TAG_CHANNEL_INFO_REQ, 0);
   
   Only 4[TAG_MESSAGE_SIZE(0] 16 bit words need to be sent. The script
   TAG_MESSAGE_SIZE(0) can be used to calculate a message size.
   
   In response, the digitizer will send channel information. Format of this message is similar to
   the channel information command.

ii. Time information - TAG_TIME_INFO.
   Sends to the digitizer initial time and other parameters not related to channel setting. Related structure:
   
   typedef struct { long lCallTime; /* Preferable call time.*/
   short sTrsInitTime; /* TRS init time. */
   short sEpochMsec; /* Part of time (msec) */
   long lEpochSec; /* Current time (sec) */
   long lEpochPrecision; /* Time precision (mksec)*/
   long lOscillator; /* Oscillator frequency. */
   long lFrequency; /* Base frequency. */
   long lCalibrInterval; /* Interval between imp. */
short sCalibrDuration;  /* Duration of impulse. */
short sDummy3;  /* Compatibility with PC*/
/* The script swaps the long intengers in a TimeInfo structure. */
#define TIME_INFO_SWAP(P_TI)
  \S
  SWAP_LONG_IN_PLACE((P_TI)->lCallTime);
  SWAP_LONG_IN_PLACE((P_TI)->lEpochSec);
  SWAP_LONG_IN_PLACE((P_TI)->sEpochMsec);
  SWAP_LONG_IN_PLACE((P_TI)->lEpochPrecision);
  SWAP_LONG_IN_PLACE((P_TI)->lOscillator);
  SWAP_LONG_IN_PLACE((P_TI)->lFrequency);
  SWAP_LONG_IN_PLACE((P_TI)->lCalibrInterval)
where:
  lCallTime  - preferable GPS call time (default 0);
  sTrsInitTime  - Time Reference Source initiate time. In practice, this is the minimal time between TRS calls. In case of an uncontrollable TRS with 1 hour interval between sending a better choice is 1800sec;
  lEpochSec  - initial time in seconds from January 1st, 1970;
  sEpochMsec  - time fraction in milliseconds (default 0);
  lEpochPrecision  - time keeping precision. Unrelated to the uncontrollable TRS and should be set to 1;
  lOscillator  - oscillator frequency (64,000,000);
  lFrequency  - base sampling rate of the ADC converters, default 4000Hz, but may be reduced in systems with more than 6 channels;
  lCalibrInterval  - interval between calibration pulses in seconds (default 0);
  sCalibrDuration  - duration of the calibration pulse in seconds (default 0).

The command can be formed by the following code:

```c
{ TimeInfo suTimeInfo={ 0l, 1800, 0, 0l, 0l, 64000000l, 4000l, 0l, 0l};

  suTimeInfo.lEpochSec=<current_time>;
  TIME_INFO_SWAP(&suTimeInfo);  /* Swap long.*/
  memcpy(suBuffer.aiBuf,&suTimeInfo,sizeof(suTimeInfo));
  TAG_SET(&suBuffer,TAG_TIME_INFO,1);
}
```

It is necessary to send only [TAG_MESSAGE_SIZE(sizeof(suTimeInfo))] 16 bit words.
iii. Channel information - TAG_CHANNEL_INFO.

Sends to the digitizer channel parameters. It is necessary to send this command if the digitizer was not configured before (it would return the channel information message with zero number of channels). Related structure and scripts:

```c
/* Channel flag enumeration. */
typedef enum { CHANNEL_FLAG_ON=1, CHANNEL_FLAG_OFF=0, CHANNEL_FLAG_TO_OFF=-1 } ChannelFlag;

/* ChannelInfo structure definition. This in configuration info, use to */
/* send by central computer. */
typedef struct { short iNumHard; /* Channel number. */
    short iFrequency; /* Channel frequency. */
    short iGain; /* Channel gain. */
    ChannelFlag enFlag; /* Channel flag(ON/OFF...). */
    long lEpochStart; /* Seconds from 70.01.01 0:0*/
    long lEpochStop; /* Seconds from 70.01.01 0:0*/
} ChannelInfo;

/* Channels definitions. */
/* Script to define channels info array. Must be put into the file level. */
#define CHANNEL_INFO_N CHANNEL_N /* Channels number.Das6103.h*/
#define CHANNEL_INFO_DEFINITION()
    ChannelInfo asuChannelInfo[CHANNEL_INFO_N];
    int iChannelInfoN=0
/* Script to set ChannelInfo for one channel. */
#define CHANNEL_INFO_SET(P_CHAN,NUMBER) 
    (P_CHAN)->iNumHard=(NUMBER); (P_CHAN)->iFrequency=125;
    (P_CHAN)->iGain=(1<<8)|1;
    (P_CHAN)->enFlag=CHANNEL_FLAG_OFF;
```

---

PMD SCIENTIFIC/eentec

Bloomfield, CT 06002 Fax: Sales: 1-314-454-9979; Support: 1-860-242-7812
pmdsci@att.net www.eentec.com
(P_CHAN)->lEpochStart=0; (P_CHAN)->lEpochStop=LONG_MAX

/* The script to set a number of input and a gain
#define GAIN_SET(INPUT_N,GAIN) (((INPUN_N)<<8)|(GAIN))
where:

- iNum - Hard channel number (from 0 to 15);
- iFrequancy - word are channel number (from 0 to 15);
- iGain - Hard channel number (from 0 to 15);
el sampling rate. The latter must be selected from the sequence resulted from division of the base sampling rate (4000sps) by 2's and/or 5's. For instance 2000, 400, 125, 100; - number and gain of the input. The digitizer has two inputs for each channel with input spans:

<table>
<thead>
<tr>
<th>Input span Divider</th>
<th>Input Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5V</td>
<td>1</td>
</tr>
<tr>
<td>220.0V</td>
<td>1</td>
</tr>
<tr>
<td>1/8</td>
<td>1</td>
</tr>
</tbody>
</table>

Available gains: 1,2,4,8,16,31,64. Use the following script to set:

- channel flag. Can be On or Off. Use predefined enumeration to set the field:
  suChannelInfo.enFlag=CHANNEL_FLAG_ON;
- channel start time in seconds from January 1st, 1970 (default LONG_MAX);
- channel stop time in seconds from January 1st, 1970 (default LONG_MAX);

A command that starts 3 channels at 100sps, ±2.5V input and gain 16 can be formed by the following code:

```c
{ /* Channels definitions. */
#define CHANNEL_N 3 /* Number of channels in system.*/

    ChannelInfo asuChannelInfo[CHANNEL_INFO_N];
    int iN;

    for( iN=0; iN < CHANNEL_INFO_N; iN++ )
    {
        CHANNEL_INFO_SET(asuChannelInfo+iN,iN+1);
        asuChannelInfo[iN].iFrequency=100;
        asuChannelInfo[iN].iGain=GAIN_SET(2,16);
        asuChannelInfo[iN].enFlag=CHANNEL_FLAG_ON;
    }

    // Other code...
}
```


, sizeof(ChannelInfo) 

} 

; 

} 

TAG_SET(&suBuffer, TAG_CHANNEL_INFO, CHANNEL_N); 

It is necessary to send only 

[TAG_MESSAGE_SIZE(sizeof(ChannelInfo)* 

CHANNEL_N)] 16 bit 

words. 

PMD 

SCIENTIFIC/eentec 

105F W. Dudleytown Rd. 

860-242-8177 Bloomfield, CT 06002 

1-860-242-7812 pmdsci@att.net 

Tel: Sales: 1-314-454-9977; Support: 1 

Fax: Sales: 1-314-454-9979; Support: 

1-860-242-7812 pmdsci@att.net 

www.eentec.com
iv. Communication command - TAG_COMM_COMMAND.

This command can be used to start or stop data transfer and switch digital output lines. The digitizer uses the same tag to send a 'no-more-data' message. Associated structure:

```c
/* Communication command tag enumeration. */
typedef enum { COMM_START=1, COMM_STOP=0, COMM_RESEND=2,
              COMM_OUT_SET=9, COMM_DATA_END=3 } CommCommandTag;
/* Communication command structure. */
typedef struct { short iCommCommandTag;/* Communication Command Tag. */
                 short iDataLevel; /* Level in % when start
                                 the host. */
                 } CommCommand;
```

where:
- **iCommCommandTag** - communication command subtag. Use predefined enumeration to set it;
- **iDataLevel** - data field (can be used differently depending on which command is send).

In all cases it is necessary to send `[TAG_MESSAGE_SIZE(sizeof(CommCommand))]` 16 bit words.

a) Start communication - COMM_START.

This command can be set by the code:

```c
{ CommCommand *psuCommCommand=(CommCommand *)suBuffer.aiBuf;
  // ... set CommCommandTag...
  TAG_SET(&suBuffer,TAG_COMM_COMMAND,1);
}
```

b) Stop communication - COMM_STOP.

**[This option is not used in the ORION system]**

Field **iDataLevel** defines percentage of internal buffer fill at which the digitizer turns on recorder at. The command can be generated by the following code:

```c
{ CommCommand *psuCommCommand=(CommCommand *)suBuffer.aiBuf;
  // ... set CommCommandTag...
  GLOBAL_TO_COMM_COMMAND(&Global,psuCommCommand);
  TAG_SET(&suBuffer,TAG_COMM_COMMAND,1);
}
```
psuCommCommand->iCommCommandTag=COMM_STOP;
psuCommCommand->iDataLevel =95;
GLOBAL_TO_COMM_COMMAND(&Global,psuCommCommand);
}
TAG_SET(&suBuffer,TAG_COMM_COMMAND,1);

c) Re-send last buffer - COMM_RESEND.
Requests to repeat sending of the last sent buffer in case of communication errors. This command can be formed by the following code:
{
    CommCommand *psuCommCommand=(CommCommand *)suBuffer.aiBuf;
    psuCommCommand->iCommCommandTag=COMM_RESEND;
    psuCommCommand->iDataLevel =100;
}
TAG_SET(&suBuffer,TAG_COMM_COMMAND,1);
If there are no errors, just an empty block (zero word-length) is sent back to acknowledge the receipt.

d) Set the output digital lines - COMM_OUT_SET.
There are 3 output digital lines. The 3 least significant bits of the field _iDataLevel_ can be used to change state of these lines. For instance, in order to turn on the inclinometer use the code:

```
define USER_TILT_ON 0x01 /* Turn On/Off(1/0) the tilt-meter.
#define USER_TILT_CHAN 0x02 /* Connect channel X/Y(0/2) to dig. */
```

{
    CommCommand *psuCommCommand=(CommCommand *)suBuffer.aiBuf;
    psuCommCommand->iCommCommandTag=COMM_OUT_SET;
    psuCommCommand->iDataLevel=USER_TILT_ON;
}
TAG_SET(&suBuffer,TAG_COMM_COMMAND,1);

e) No-more-data message - COMM_DATA_END.
The digitizer sends this message when the whole block stored in the internal buffer has been sent. The message can be identified by the following code:

```
if( TAG_PROVIDED(&suBuffer) && TAG_TYPE(&suBuffer) == TAG_COMM_COMMAND && suBuffer.aiBuf[0] == COMM_DATA_END )
```
v. Reset command - TAG_RESET.

The digitizer will restart data acquisition with previously set parameters. It can be implemented with the code:

```
TAG_SET(&suBuffer,TAG_SYSTEM_RESET,0);
```
It is necessary to send only \([\text{TAG\_MESSAGE\_SIZE(0)}]\) 16 bit words.

vi. Log message - TAG_LOG.

This is a block which carries digitizer messages. The DSP is a 16 bit world-oriented processor and each symbol is saved in 2 bytes. One just needs to delete the most significant byte. In all other aspects the log message is a set of lines separated by the new line symbol ('\n\'). The end of the log message is marked with the zero symbol ('\0\'). Log messages can be printed using the code:

```
if( TAG_PROVIDED(&suBuffer) && TAG_TYPE(&suBuffer) == TAG_LOG )
{
    register char *pcB=(char*)suBuffer.aiBuf, *pcE;

    UniToAsc(pcB,pcB,BUFCOM_N); /* Transform TI char. to char */
    printf("%s",pcB);
}
```

vii. Data block.

This Block does not have a tag associated with it. In fact, any message without a tag is a data block. Associated structure and scripts:

```
#define BUFCOM_N 252 /* Number of words in a buffer. */
/* The real predefining name is _TMS320C5XX (_TMS320C5XX__ is in docs.) */
typedef struct {
    #ifdef _TMS320C5XX /* TI packs bits beginning at the most significant.*/
        unsigned uChan : 4;  /* Channel number. */
    #else /* DOS packs bits beginning at the last significand.*/
        unsigned uNumber :12; /* Number of samples in buffer. */
    #else /* DOS packs bits beginning at the last sign-
        significand.*/
        unsigned uNumber :12; /* Number of samples in buffer. */
```

PMD SCIENTIFIC/eentec
105F W. Dudleytown Rd.
Bloomfield, CT 06002
pmdsci@att.net
www.eentec.com
unsigned uChan = 4; /* Channel number. */
#endif /* __TMS320C5XX__ */
short iSecFract; /* Number of fractions of second. */
long lEpochSec; /* Seconds from 1 January 1970 00:00*/
short aiBuf[BUFCOM_N]; /* Buffer space. */
}
} BufCom;
#define SEC_FRACTION 10000 / * Second fraction is a tenth of millisecond */
/* Script to initiate a buffer. */
#define BUFFER_INIT(P_BUFFER) (P_BUFFER)->uChan=(P_BUFFER)->uNumber=(P_BUFFER)->iSecFract=0
where:
  uChan - channel number;
  uNumber - number of samples in the buffer;
  iSecFract - number of fractions of a second;
  lEpochSec - time in seconds from 1 January 1970 00:00;
  aiBuf - space used for packed data.
Recorded data are sent in compressed format. (The decompression c-function is provided.)
Compression concept: Data are written as first differences in 2 or 4 bytes words depending on
the magnitude. Maximum compression factor is 2.
A data block can be processed using the following code:
if(!TAG_PROVIDED(&suBuffer)) {
  #define BUF_N 1024
  long int alBuf[BUF_N]; /* Data */

  /* Time of the first sample can be calculated by: */
  dEpoch=(double)SWAP_LONG(suBuffer.lEpochSec) + (double)suBuffer.iSecFract/SEC_FRACTION;
  /* Data can be unpacked by (look at the attached program): */
  in=ShlDeCode(suBuffer.aiBuf,min(suBuffer.uNumber,BUF_N),alBuffer);
}
6. Transmission block layer.

Transmission block starts with its length in 16-bit words in the form of a short integer number that is followed by a data block (command or message) of up to 2 Kbytes long. Transmission blocks end with a one's complement checksum of a one's complement checksum in form of short integer. The checksum is not included in the block length. The checksum is calculated for whole block including the length. The C-function for calculation of the checksum is provided.

A transmission block can be treated just as:

```c
{ short int aiTbuf[1024+2];

    .... /* Get a block. */
    if( aiTbuf[0] > 0 &&
        && aiTbuf[aiTbuf[0]] == checksum(aiTbuf,2*(aiTbuf[0]+1)) )
    { .... /* Process the block. */ }
}
```

7. SLIP layer.

The standard SLIP (Serial line IP) is used to frame a transmission block on a serial line. The C-function to encode and decode a transmission block in SLIP is provided. This is an example how it could be done:

```c
{ short int aiTbuf[1024+2];

    unsigned char aucSlipBuffer[(1024+2)*2];
    int iN;

    .... /* Prepare a block. */
    if( ( iN=SlipEnCode(SlipBuffer,sizeof(aucSlipBuffer),aiTbuf,aiTbuf[0]+1) )
        <= 0 )
    { printf("SLIP coding is invalid.\n");
      return -1;
    }
    .... /* Send the block. */
    .... /* Get a block. */
    iN=<number of received bytes>;
    if( ( iN=SlipDeCode(aiTbuf,sizeof(aiTbuf),SlipBuffer,iN,NULL) )
        <= 0 )
    { printf("SLIP record is invalid. Data is ignored.\nRet=%d\n",iN);
      return -1;
    }
    .... /* Process the block. */
}
```

A complete description of the SLIP protocol see in the Appendix V.

rfc1055

Press here to go to the top of the rfc 'tree'.

Network Working Group  J. Romkey
Request for Comments: 1055       June 1988

A NONSTANDARD FOR TRANSMISSION OF IP DATAGRAMS OVER SERIAL LINES: SLIP

INTRODUCTION
The TCP/IP protocol family runs over a variety of network media:
IEEE 802.3 (ethernet) and 802.5 (token ring) LAN's, X.25 lines,
satellite links, and serial lines. There are standard encapsulations
for IP packets defined for many of these networks, but there is no
standard for serial lines. SLIP, Serial Line IP, is a currently a de
facto standard, commonly used for point-to-point serial connections
running TCP/IP. It is not an Internet standard. Distribution of
this memo is unlimited.

HISTORY
SLIP has its origins in the 3COM UNET TCP/IP implementation from the
early 1980's. It is merely a packet framing protocol: SLIP defines a
sequence of characters that frame IP packets on a serial line, and
nothing more. It provides no addressing, packet type identification,
error detection/correction or compression mechanisms. Because the
protocol does so little, though, it is usually very easy to
implement.

Around 1984, Rick Adams implemented SLIP for 4.2 Berkeley Unix and
Sun Microsystems workstations and released it to the world. It
quickly caught on as an easy reliable way to connect TCP/IP hosts and
routers with serial lines.

SLIP is commonly used on dedicated serial links and sometimes for
dialup purposes, and is usually used with line speeds between 1200bps
and 19.2Kbps. It is useful for allowing mixes of hosts and routers
to communicate with one another (host-host, host-router and router-
router are all common SLIP network configurations).

AVAILABILITY
SLIP is available for most Berkeley UNIX-based systems. It is
included in the standard 4.3BSD release from Berkeley. SLIP is
available for Ultrix, Sun UNIX and most other Berkeley-derived UNIX
systems. Some terminal concentrators and IBM PC implementations also
support it.

SLIP for Berkeley UNIX is available via anonymous FTP from
uunet.uu.net in pub/sl.shar.Z. Be sure to transfer the file in
binary mode and then run it through the UNIX uncompress program. Take
the resulting file and use it as a shell script for the UNIX /bin/sh
(for instance, /bin/sh sl.sh).

PROTOCOL
The SLIP protocol defines two special characters: END and ESC. END is
octal 300 (decimal 192) and ESC is octal 333 (decimal 219) not to be
confused with the ASCII ESCape character; for the purposes of this
discussion, ESC will indicate the SLIP ESC character. To send a
packet, a SLIP host simply starts sending the data in the packet. If a
data byte is the same code as END character, a two byte sequence of
ESC and octal 334 (decimal 220) is sent instead. If it the same as
an ESC character, an two byte sequence of ESC and octal 335 (decimal
221) is sent instead. When the last byte in the packet has been
sent, an END character is then transmitted.

Phil Karn suggests a simple change to the algorithm, which is to
begin as well as end packets with an END character. This will flush
any erroneous bytes which have been caused by line noise. In the
normal case, the receiver will simply see two back-to-back END
characters, which will generate a bad IP packet. If the SLIP
implementation does not throw away the zero-length IP packet, the IP
implementation certainly will. If there was line noise, the data
received due to it will be discarded without affecting the following
packet.

Because there is no 'standard' SLIP specification, there is no real
defined maximum packet size for SLIP. It is probably best to accept
the maximum packet size used by the Berkeley UNIX SLIP drivers: 1006
bytes including the IP and transport protocol headers (not including
the framing characters). Therefore any new SLIP implementations
should be prepared to accept 1006 byte datagrams and should not send
more than 1006 bytes in a datagram.

DEFICIENCIES
There are several features that many users would like SLIP to provide
which it doesn't. In all fairness, SLIP is just a very simple
protocol designed quite a long time ago when these problems were not
really important issues. The following are commonly perceived
shortcomings in the existing SLIP protocol:

- addressing:
  both computers in a SLIP link need to know each other's IP
  addresses for routing purposes. Also, when using SLIP for
  hosts to dial-up a router, the addressing scheme may be quite
dynamic and the router may need to inform the dialing host of
the host's IP address. SLIP currently provides no mechanism for hosts to communicate addressing information over a SLIP connection.

- type identification:

  SLIP has no type field. Thus, only one protocol can be run over a SLIP connection, so in a configuration of two DEC computers running both TCP/IP and DECnet, there is no hope of having TCP/IP and DECnet share one serial line between them while using SLIP. While SLIP is "Serial Line IP", if a serial line connects two multi-protocol computers, those computers should be able to use more than one protocol over the line.

- error detection/correction:

  Noisy phone lines will corrupt packets in transit. Because the line speed is probably quite low (likely 2400 baud), retransmitting a packet is very expensive. Error detection is not absolutely necessary at the SLIP level because any IP application should detect damaged packets (IP header and UDP and TCP checksums should suffice), although some common applications like NFS usually ignore the checksum and depend on the network media to detect damaged packets. Because it takes so long to retransmit a packet which was corrupted by line noise, it would be efficient if SLIP could provide some sort of simple error correction mechanism of its own.

- compression:

  Because dial-in lines are so slow (usually 2400bps), packet compression would cause large improvements in packet throughput. Usually, streams of packets in a single TCP connection have few changed fields in the IP and TCP headers, so a simple compression algorithms might just send the changed parts of the headers instead of the complete headers.

Some work is being done by various groups to design and implement a successor to SLIP which will address some or all of these problems.
SLIP DRIVERS

The following C language functions send and receive SLIP packets. They depend on two functions, send_char() and recv_char(), which send and receive a single character over the serial line.

/* SLIP special character codes */
#define END 0300 /* indicates end of packet */
#define ESC 0333 /* indicates byte stuffing */
#define ESC_END 0334 /* ESC ESC_END means END data byte */
#define ESC_ESC 0335 /* ESC ESC_ESC means ESC data byte */

/* SEND_PACKET: sends a packet of length "len", starting at location "p" */
void send_packet(p, len)
    char *p;
    int len;
{
    /* send an initial END character to flush out any data that may have accumulated in the receiver due to line noise */
    send_char(END);

    /* for each byte in the packet, send the appropriate character sequence */
    while(len--){
        switch(*p) {
        /* if it's the same code as an END character, we send a special two character code so as not to make the receiver think we sent an END */
            case END:
                send_char(ESC);
                send_char(ESC_END);
                break;

        /* if it's the same code as an ESC character, we send a special two character code so as not to make the receiver think we sent an ESC */
            case ESC:
                send_char(ESC);
                send_char(ESC_ESC);
                break;
        }
/* otherwise, we just send the character */
default:
    send_char(*p);
}
p++;
}
(EN) if more than len bytes
D); are received, the
} packet will
*
/
/* be
RE
CV
PACKET:

recv
* Returns the
terminated
number of bytes
cessives
buffer.
*/

int
packet
recv
inta
into
the
buffer
located
at
"p"
.

* int
len

int
0;

int
recv
packet
(p,
len)

char
at
*p;
bytes

unit

/* reading
we put together a whole packet.

Make sure not to copy them into the packet if we run out of room.

while
{
    if we
    run out of room
    {
        / *
        if it's an END character then we're done
        * /
        char END:
        *
        minor optimization:
        *
        character necessary
        /
        switch
        {
            / *
            if there is no data in the packet,
            ignore it.
            This is
* meant to avoid bothering IP with all
* the empty packets
* duplicate END
* characters which are
* in

PMD
SCIENTIFIC/eentec

105F W.
Dudleytown
n Rd.
Tel: Sales:
1-314-454-
9977;
Support: 1
860-242-81
77
Bloomfield,
CT 06002
Fax: Sales:
1-314-454-
9979;
Support:
1-860-242-
7812
pmdsci@att
.net
www.eente
c.com
* turn sent to try to detect line noise.
 */
    if(received)
        return received;
    else
        break;

    /* if it's the same code as an ESC character, wait
     * and get another character and then figure out *
     * what to store in the packet based on that.
     */
    case ESC:
        c = recv_char();

        /* if "c" is not one of these two, then we
         * have a protocol violation. The best bet
         * seems to be to leave the byte alone and
         * just stuff it into the packet
         */
        switch(c) {
            case ESC_END:
                c = END;
                break;
            case ESC_ESC:
                c = ESC;
                break;
        }

    /* here we fall into the default handler and let
     * it store the character for us
     */
    default:
        if(received < len)
            p[received++] = c;
        
    }
    
}
9. APPENDIX 2-B: Working with the tiltmeter.

1. Start the digitizer with all 4 channels turned On.
2. Turn the tiltmeter ON with axis X connected to the digitizer:
   
   ```
   #define USER_TILT_ON 0x01 /*Turn On(1) the tiltmeter.*/
   #define USER_TILT_CHAN 0x02 /* Connect channel X/Y(0/2)
   to dig. */
   #define USER_TILT_OFF 0x00 /*Turn Off(0) the tiltmeter.*/
   ```
   
   ```c
   { CommCommand *psuCommCommand=(CommCommand
   *)suBuffer.aiBuf;
   psuCommCommand->iCommCommandTag=COMM_OUT_SET;
   psuCommCommand->iDataLevel=USER_TILT_ON;
   }
   TAG_SET(&suBuffer,TAG_COMM_COMMAND,1);
   Send [TAG_MESSAGE_SIZE(sizeof(CommCommand))] 16 bit words (6 words).
   ```

3. Accumulate sufficient data for the tiltmeter X-axis.
4. Switch tiltmeter output to the axis Y:
   ```c
   { CommCommand *psuCommCommand=(CommCommand
   *)suBuffer.aiBuf;
   psuCommCommand->iCommCommandTag=COMM_OUT_SET;
   psuCommCommand->iDataLevel=USER_TILT_ON|USER_TILT_CHAN;
   }
   TAG_SET(&suBuffer,TAG_COMM_COMMAND,1);
   ```

   Send [TAG_MESSAGE_SIZE(sizeof(CommCommand))] 16 bit words (6 words).

5. Accumulate sufficient data for the tiltmeter Y-axis.
6. Turn the tiltmeter OFF:
   ```c
   { CommCommand *psuCommCommand=(CommCommand
   *)suBuffer.aiBuf;
   psuCommCommand->iCommCommandTag=COMM_OUT_SET;
   psuCommCommand->iDataLevel=USER_TILT_OFF;
   }
   TAG_SET(&suBuffer,TAG_COMM_COMMAND,1);
   ```

   Send [TAG_MESSAGE_SIZE(sizeof(CommCommand))] 16 bit words (6 words).

7. Turn off the 4th channel of the digitizer.
   To do it, send the same channel information, that was sent in the item 1, but
   with the Off flag of the 4th channel(suChannelInfo.enFlag=CHANNEL_FLAG_OFF;)

---

PMD SCIENTIFIC/eentec
105F W. Dudleytown Rd.
Bloomfield, CT 06002
pmdsci@att.net
www.eentec.com
Part 3. The ORION System Interconnections

1. Instrument Assembly & Connectors Pinouts

In the ORION configuration EP300-DT assembly is mounted on the bottom lid of the cylindrical titanium pressure vessel\(^5\). This assembly consists of the three seismic sensors and three layers of electronics:

- seismometer power and three amplification/correction/feedback electronics;
- tiltmeter, tiltmeter interface, junction terminals and the intermediate Bendix connector;
- 6503 stack consisting, in turn, of three boards: power, analog (A-to-D converter and input filters and dividers), and digital (DSP, programmable logic, 16MB data RAM, 2MB program RAM, communication chipsets, PLL, etc.) board.

\(^5\) For the pressure vessel outline and dimensions see the attached file "TiHousing.pdf"
An 8-pin pressurized circular connector is screwed into the top lid of the titanium cylinder; a
cable from this connector goes to the intermediate 10-pin Bendix connector mounted on the
Junction layer. connector pinouts are show in the Table below:

<table>
<thead>
<tr>
<th>Lid Connector Pin</th>
<th>Description</th>
<th>Bendix Connector Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+12V Supply</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>0V Supply</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>Tx - DATA</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>Rx - DATA</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>GND (Logic)</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>Tx - Timing</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>1 PPS</td>
<td>G</td>
</tr>
<tr>
<td>8</td>
<td>RESET IN</td>
<td>H</td>
</tr>
</tbody>
</table>

2. The Junction Board.

Photo of the Junction Board is shown below.
All major system components, i.e.
- Seismometer;
- Tiltmeter;
- Digitizer;
- Interface Board

are interconnected via two terminal blocks, TB2 and TB3.

- TB2 connects the Bendix circular connector to the Digitizer via 5 flat cables (actually, cable to the Digital Board connector J5 is reduced to a single wire). It also provides power from this connector and Tiltmeter control commands from the Digitizer to the Interface Board.
- TB2 connects the Analog circular connector to Seismometer Calibration Enable and Calibration Input inputs and to Seismometer Signal Ground; to the buffered Seismometer differential outputs on the Interface Board. Analog connector also provide alternative power to the system.
- TB3 connects seismometer signal outputs directly and controlled and buffered tiltmeter outputs to the signal inputs of the digitizer.

Detailed Junction Board wiring diagram is shown in the attached file "JUNCTION.pdf".
The EP300-DT system uses Applied Geomechanics Model 900 Tiltmeters ("Clinometers"). These sensors were selected so that their full scale tilt (±25°) includes the maximum specified operating tilt angle of the seismometers (±10°), and exhibit excellent resolution: 100mV per degree of tilt or 5592.4 counts/angular minute after digitizing. Tiltmeters data sheets can be found in the attached file "900-Series B-93-10, Rev. C.pdf".

This board is installed next to the tiltmeter. Its complete circuit diagram can be found in the attached file "TILT_CTL.pdf". The circuit contains voltage inverter providing negative power to all operational amplifiers employed. Each tiltmeter output is buffered by a follower- amplifier with a fine offset adjustment. After the system is fully assembled, it is placed on a leveled granite plate and both buffered tiltmeter outputs are set to zero using multiturn potenti- ometers. Analog switches U1 respond to the digitizer commands and turn tiltmeter power on-off and switch digitizer CH.4 input between tiltmeter axes outputs.

All micropower, low-noise amplifiers in the Interface board are Linear Technology LT1078 types.