Lambda Power Supply

IRM software support

Thu, Feb 13, 2003

Lambda makes a ZUP ("Zero-up") series of programmable power supplies that are to be used in the Electron Cooling project here. It is necessary to support them via the IRM serial port. There is provision for supporting up to 31 power supplies via one controller, which is automatically supported by the first power supply in the daisy chain. This note describes a local application that can support this interface.

The connection to the IRM is via RS232 at 9600 Baud. The controller understands certain specific commands that allow a computer to query readings and settings and also to set voltages or currents. It can also provide status and accept digital control. One may think of a PLC-like interface, where a local application understands the special protocol supported by the controller, and it also has a mechanism for accepting setting requests from the underlying system code that can be passed to the hardware via its special serial protocol. This approach was not taken.

One of the factors that must be obeyed is that it requires about 15 ms for the controller to process one command; another command should not be sent without allowing for such a delay. Also, if one queries the controller for information, another command should not be sent until that information has been received. This means there should be a time-out used for responses. One way to handle this is to send no more than one command each 15 Hz cycle.

The hardware mechanism for supporting more than one power supply from a single serial port is as follows: One command selects the target supply number. This value is retained by the addressed power supply, and each of the others places itself in listen mode, from which it awakens only when it hears a command selecting its own power supply number. It is probably a good idea to retain this target number in the local application, in order to save time making unnecessary switches to the same target, each costing an extra cycle. If too many settings are issued at once, we probably would not want to tie up the communications link making only settings and not being aware of changes in readings of other supplies.

Commands for ZUP

Each command is preceded by an ascii ":" and followed by an ascii ";". The following table omits these two bracketing characters.

Commands that need support:

`MDL?`	Get ps model id, as in "Nemic-Lambda ZUP(xxV)-(yyA)"
`REV?`	Get software version, such as "Ver6-33 1.0"

These two commands might be handled internally, but very infrequently.

`DCL`	Clear status registers. Maybe do this "manually."
`ADRnn`	Set target ps address for subsequent communications
`RMT0`	Transition to local mode (return only via front panel button.)
`RMT1`	Transition from latched remote to non-latched remote.
`RMT2`	Transition to latched remote (disabled front panel button)
`RMT?`	Returns remote/local setting: RM1 = remote, RM2 = latched remote.

The above RMT commands may not be used, except to monitor via "RMT?".

`VOLn`	Set output voltage, where allowed range depends upon ps model.
`VOL!`	Returns present programmed output voltage value, id = SV.
`VOL?`	Returns actual output voltage, id = AV.
`CURn`	Set output current, where allowed range depends upon ps model.
`CUR!`	Returns present programmed output current, id = SA.
`CUR?`	Returns actual output current, id = AA.
`OUT1`	Set output on
`OUT0`	Set output off
`OUT?`	Returns output on/off status, as OT1 = on, OT0 = off
`FLD1`	Arm foldback protection
`FLD0`	Release foldback protection
`FLD2`	Cancel foldback protection
`FLD?`	Returns foldback protection status, where FD1 = on, FD0 = off
`OVPn`	Set over voltage protection level in volts, depending on ps model.
`OVP?`	Returns present over voltage protection value, id = OP.
`UVPn`	Sets under voltage protection limit in volts, depending on ps model.
`UVP?`	Returns under voltage protection limit, id = UP.
`AST1`	Set auto-restart on
`AST0`	Set auto-restart off
`AST?`	Returns auto-restart status, where AS1 = on, AS0 = off.

There are quite a few parameters specified, many of which will not be changed at all often. But if they are to be mapped to analog channels, it would be good to update them promptly following a setting.

With so many possible parameters, even if only a few normally need to be accessed, it may be good to have a general method that supports all of them. Then one can install the appropriate references to access those of interest.

It will be necessary to be aware of the model number for each supply that is accessed, since that determines the format of numeric setting values that must be used in the serial protocol. This information may be usable for limiting the values that a user can set, although that can be accomplished by choosing a suitable full scale value in the analog descriptor.

How many different digital settings are there? One is a write-only control bit that clears the status registers. Two more could be used for establishing remote/local state. Beyond that, there is `OT`, `FD` (2 bits), `OP`, `UP`, `AS`. Some of these bits are already mapped in the operational status register: cc/cv, fold, ast, out, srf, srv, srt, alarm. These would take care of OT, FD, AS. The alarm status register shows ovp, otp, a/c-fail, fold, prog. The error codes register shows 5 bits: not used, wrong command, buffer overflow, wrong voltage, wrong current.

What are the analog values related to a supply? Certainly we must include settings and readings of the voltage and current. There is also the over-voltage protection level and the

under-voltage protection limit. If we include a readback of the programmed settings of voltage and current, we have a total of 6 analog channels for each supply. The order may be: AV, SV, AA, SA, OP, UP. (A slightly different order was actually used.)

Two other values that may be of interest can be derived from the model number. The reply to the MDL? query is something like "Nemic-Lambda ZUP(6V-33A)." The two numbers give the rated output voltage and the rated output current, respectively. If we include these two values, we could talk about 8 channels of numeric information per supply.

The STT? command can get all 4 principal numeric values, plus the three status registers, in a single line that may be about 60 characters long, depending on the numeric field widths. This means that such a query will likely require two 15 Hz cycles to complete, at 9600 baud. But the LA can get a call on receipt of the line of text in the reply, so it could send a new query command right away if one is waiting. The number of characters would depend upon knowledge of the baud rate used. At 9600 baud, each character uses about 1 ms.

How can the order of numeric channels relating to one supply be specified? Suppose the LA has a parameter nn that specifies the number of power supplies, ranging from 1–31. Allocate 8 channels for each supply. Let each supply have an internal queue associated with it. After a STT? command completes, the queue is checked for another command, such as a setting or unusual query, and if one is found, it is sent out. After completion of that command, we move on to the next supply in sequence. In the absence of anything waiting in a queue, then, a series of STT? commands will be sent and the subsequent replies parsed into 4 analog values and 3 bytes of status. If there were only 5 supplies, this might take 10 cycles, assuming that STT? commands were sent every other cycle, to repeat the queries. With 31 supplies, it would obviously take longer, so that the update period might be about 4 seconds.

Suppose a supply does not respond. A time-out must be imposed to detect this, and a status bit should be asserted (or cleared) to show that the supply did not respond. The non-responding status bit may be grouped with the other status bits. What about the numeric values from that supply? After some time, they may be considered stale, and they can be cleared to zero, perhaps. But as long as they are in the sequence, they should be queried from time to time. Perhaps the query should be the model number query while in this state. Once a reply is returned from the model number query, they can again be included in the normal sequence. This condition may occur when the local pushbutton is pressed to place the supply into local mode. The only way to return to remote mode is via the pushbutton.

Suppose, when it is time to address a given supply, we first check whether there is MDL information. If there is not, then that must be the first query. Following its completion, the STT? command can be issued. If the MDL info already exists, then check the queue (for that supply) for a possible setting waiting. If there is one, perform it; otherwise, move on to the next supply. The queue can also contain infrequent queries, including the MDL and the over-voltage and under-voltage limits. The REV? command can also be used for this.

When a reply message is received, it can be processed without paying attention to the context, beyond the current power supply number.

Some logic in the 15 Hz cycle activity of the LA is necessary to fill the queues for each supply. The infrequent queries are placed there, including the MDL query, as are setting messages that are gleaned from the message queue that interfaces with the underlying system code.

The formats used by various numeric values depend upon the power supply model. In each case, they are really a kind of fixed point value, where the decimal point occurs in only one position, and every other character in the field is a digit 0–9; no spaces or signs are used. To build these values, it may be easiest to use `CvI`, then insert the decimal point as appropriate, replacing all initial spaces with ascii zero.

When a setting message is found in a message queue, and a similar one is already found in the queue for the target supply, merely replace the earlier one with the new one. This should make knob control more palatable to use.

A data base will be needed for handling all the required logic, including the numeric formatting for each model of power supply. The easy way to handle this might be to create a data file that contains all this information in some structure. The LA would access this data file at initialization. The key to a power supply model is the pair of values for peak voltage and current provided by that power supply model. To specify a numeric format, use the numeric field width and the number of digits past the decimal point. Each field width in use, including the decimal point, ranges from 4–6 and is always at least 2 more than the number of digits following the decimal point, which ranges from 1–3. Only the setting values for output current vary by the peak voltage and peak current ratings for a supply; all other numeric values depend only on the peak voltage rating. It may be advisable to recognize that additional models may be added to the Lambda product line in the future.

One easy structure for the data file would be a simple table of 2 words plus 8 byte-size values. The two words hold the peak voltage and peak current specifications that serve to characterize a given model, and the 8 byte-size values hold the field width and number of decimals for each of the 4 numeric settable parameters, in the order SV, SA, OP, UP. Another scheme can use byte values that specify both width and number of decimals as two 4-bit nibbles. In this way, each model would use only 8 bytes per entry. If there are 15 models, this table is only 120 bytes. The end of the table might be indicated with an 8-byte entry of all zeros. The index to each entry would have no significance, so the table should be easy to edit. After reading in the table, the LA can notice the number of entries in the table and use that loop count in its searches. This would be the entry format:

Field	*Size*	*Meaning*
`pV`	2	peak voltage rating, in volts
`pA`	2	peak current rating, in amps
`fSV`	1	format for setting voltage, as 0xwd, where w=width, d=decimals
`fSA`	1	format for setting current
`fOP`	1	format for setting over-voltage protection level
`fUP`	1	format for setting under-voltage protection limit

LA Parameters

The following list of parameters may be used with the local applicatuion:

```
Prompt     Meaning
ENABLE B   Enable Bit#
SUPPLIES   Number of power supplies to scan
BASE   C   Base Channel# for analog data, 8 channels per supply
BASE   B   Base Bit# for digital data, 4 bytes per supply
```

Returning to digital control, there may be 8 bits of write-only digital control, occupying a single byte. If the corresponding pseudo-address is not installed in the BADDR table, then these controls are not accessible.

Bit#	*Meaning*
7	Clear status registers
6	Output control
5	Auto-restart control
4	(spare)
3	Foldback protection control (2 bits)
2	"
1	Remote/local control (2 bits)
0	"

The format of the status registers in the first 3 of the 4 bytes alloted per supply are:

Bit#	*Meaning*
7	cc/cv mode, 0=constant voltage, 1=constant current
6	fold, 1=foldback protection armed
5	ast, 1=auto-restart on, 0=auto-restart off
4	out, 1=output on, 0=output off
3	srf, 1= foldback protection SRQ enabled
2	srv, 1= over-voltage protection SRQ enabled
1	srt, 1=over-temperature protection SRQ enabled
0	alarm, 1=alarm register bit is set

Bit#	*Meaning*
7	n.u.
6	n.u.
5	n.u.
4	ovp, 1=over-voltage protection tripped
3	otp, 1=over-temperature protection tripped
2	a/c fail, 1=failure at the input voltage supply
1	fold, 1=foldback protection activated
0	prog, 1=programming error occurred

Bit#	*Meaning*
7	n.u.
6	n.u.
5	n.u.
4	n.u.
3	wrong command, 1=unknown string received
2	buffer overflow, 1=overflow in communication buffer
1	wrong voltage, 1=attempt to set supply voltage out of spec limits
0	wrong current, 1=attempt to set supply current out of spec limits

One more bit of status needs to be made available, that which announces a communications failure; the supply did not respond when addressed with a query. The base Bit# for announcing communications failure can target up to 4 bytes, depending on the number of supplies in use. An alternative method could take over one of the unused status bits above.

The commands that are actually needed to be used, aside from those used for digital control, are the following:

`MDL?` or `REV?`, to capture the peak voltage and current ratings
`STT?`, to acquire the usual operating values of AV,SV,AA,SA, and the 3 status bytes.
`VOLn`, to set the output voltage
`CURn`, to set the output current
`OVPn`, to set the over-voltage protection level
`UVPn`, to set the under-voltage protection limit

The digital setting commands needed are:

`ADRnn`, every time it is necessary to switch communications to another supply
`DCL`, to clear status registers
`OUTn`, to set output control on/off
`FLDn`, to set foldback protection on/off
`ASTn`, to set auto-restart control on/off
`RMTn`, to switch the remote/local status

These commands need to be executed separately, even though the bits that cause them are packed together into a single byte.

New planning

After further thinking about how to handle these supplies, here are some brief notes that describe how to simplify the required system support for the ZUP series of supplies.

Planned sequence of analog channels for each power supply:

*Actual voltage
*Actual current
*Over-voltage level
*Under-voltage limit
Set voltage
Set current
Peak output voltage
Peak output current

The asterisk (*) indicates that the actual voltage and actual current channels are controllable, along with the over-voltage and under-voltage values. The set voltage and set current are readings of what the power supply thinks are the current programmed setting values. They are certainly affected by controlling the actual voltage and current channels, but they are not the basis for control.

There is no need for a special analog control type designed to support control of these power supplies. The controllable channels can be dummy channels, where the analog control type field is simply `0x12000000`. The local application monitors changes that occur in the dummy setting values for these channels. When a change is noticed, it arranges to have the new setting promptly passed to the supply at the next convenient time. A subsequent update in the set voltage or set current channel does not affect the setting fields of the actual voltage and actual current channels. As a result, the actual voltage and actual current channel setting field values reflect what the user desires and what the local application should act upon, whereas the set voltage and set current channel readings reflect what the power supply

thinks are its current programmed settings. A user is likely to be interested in the latter settings only when analyzing a problem.

The digital control options can be handled as a set of dummy bits that cause the indicated effect to be delivered to the supply. This scheme is analogous to that used for HLRF systems. A dummy bit is set to "1" to ask that a digital action be taken, and the LA clears the bit when it has accomplished that action. Eight bits may be enough for all digital actions supported by the power supply.

The digital status bytes comprise the status returned by the supply, which consists of one byte, plus two 5-bit fields. It may be useful to assign one of the unused bits of these three status bytes to indicate communications failure with the power supply. A likely candidate for this purpose may be the hi bit of the alarm status byte.

All in all, one has a set of 8 analog channels and 4 digital bytes that make up the software interface to each power supply. This approach means that no system changes are required to implement this kind of power supply support; all the required logic is provided by the local application. In addition, configuring the front end for supporting a set of such power supplies is easy; merely install a single instance—only one serial port is available—of the local application, and also define the set of relevant channels and digital status bits and dummy control bits for each supply. The local application name might be ZUPS, referring to the Zero-UP series of Lambda Power Supplies that use this protocol.

A revised set of control bits, now dummy bits sampled by LA and automatically cleared is:

Bit#	*Meaning*
7	Output ON
6	Output OFF
5	Auto-restart ON
4	Auto-restart OFF
3	Clear status registers
2	Foldback protection CANCEL
1	Foldback protection ARM
0	Foldback protection RELEASE

If another dummy control byte is used, the following controls could be added:

Transition from Remote to Local mode
Transition from latched Remote to non-latched Remote
Transition back to Local or non-latched

These three control actions could be supported by three of the unused bits of, say, the program status byte, in order to keep the 4-byte digital model.

By use of the action control bit approach, none of the particulars of the ZUP protocol for these control actions needs to be forced on the user. The local application merely delivers the appropriate commands as necessary.

Sequencing

How can the LA logic be organized to generate a suitable sequence of RS232-based commands that include many different Lambda power supplies? Earlier there was reference

to a queue of commands to be made for each supply, in which a more recent setting could overwrite an earlier one that has not yet been sent out. Perhaps it can be done differently.

Suppose there is a fixed set of possible commands that can be generated for each supply, so that rather than a queue, there is a list of all possible commands. The act of placing a command into a queue dedicated to a single power supply is then replaced with marking the command for that power supply to be done. When a new setting is seen, by monitoring the setting field of the channel associated with the actual current, for example, and that command is already marked but not yet sent, the program merely overwrites the setting value for that command. (If it is not marked, then set the value anyway and mark it.)

When the sequencing reaches a new power supply, it should check for possible marked commands in some priority order, yet to be determined. If any is found marked, it arranges to send out that command and clear the mark. After that is completed, it sends the usual `STT?` command out, and upon return of the reply data moves on to the next supply.

To back up, when starting to serve a given supply, it should first check whether that supply is "down," meaning that its attempt to communicate failed the last time, say, because no reply data was returned following the last `STT?` command. If the supply is down, the first step should be to send the `MDL?` query to get the peak voltage and peak current specs for the supply. Only if that succeeds in prompting a reply does it then send out the usual `STT?` command. If there is no reply from the `MDL?` command, it just moves on to the next supply in sequence anyway.

With this sequencing logic, there is only one possible command to be done before issuing the usual `STT?` command. This means that the time to sequence through all supplies, even in the case of high activity of setting requests, should be no more than twice as long as it is in the absence of any such setting requests.

In addition, as noted above, there should be logic that occasionally marks commands to be done less frequently, such as querying the over-voltage and under-voltage values, and maybe even requesting the `MDL?` command to be issued. The throttling of such slow requests can be done if no commands are marked, and a counter counts down for that supply. That would mean the specification of time would be in units of the time required for executing the entire sequence, passing through all power supplies. This time depends mostly upon the number of supplies and somewhat on the amount of setting activity.

In the case that the node is "up," there should already be a peak voltage and current value available. But any time no such values exist, that again is a reason to send the `MDL?` command before doing anything else.

Once a supply has been declared "down," and a short period of time has elapsed, the analog values for that supply should probably be cleared, so one is not misled by stale data. By clearing the peak values for the supply, that means that another `MDL?` query will occur before anything else happens to that supply. It is unlikely, but conceivable, that someone changed the supply while it was "down." Following return to "up" status, one should fairly promptly see new values for the various numeric parameters and status, since following a current return of MDL info, the `STT?` command will be sent. It would be good to arrange to query the over-voltage and under-voltage values fairly soon after a supply comes "up," too.

One could adopt an attitude that, when detecting setting changes, special focus is placed on that supply, so that timely updating of other supplies, for which settings are not occurring, languishes somewhat. But that is not the approach of the design described here. The perturbation of response updates is not greatly dependent upon setting activity. It is expected that settings do not normally occur all that often to many supplies at once. If they do, the current design would not optimize the settings to be done as nearly as possible simultaneously. On the other hand, they will likely be sent to all within a second or so, again mostly depending upon the total number of supplies.

Programming details

The sequencing logic for driving the serial port can be used in both the 15 Hz cycle call of the LA as well as a serial input call, which occurs upon receiving a line of text in response to a query command. In the DoCycle routine, there might be the following:

If serial NOT busy, look for command waiting to be sent, send it, set serial busy.
Else if serial busy, and if time-out not exceeded, do nothing.
Else declare current PS down, AdvancePS.

In the DoSerial routine,

Process received serial input
Update data pool with latest values
If last comand was `STT?`, AdvancePS.

Diagnostics

It may be useful to include a diagnostic to record serial activities in a data stream. One might be called "`ZUPSLOG` ". Two kinds of records can be used: one is written whenever a setting command is sent; a second is written every time a reply is received. Each record is 16 bytes long, with the last 8 bytes used for the time-of-day, down to half ms within the current cycle, in keeping with the typical cases of such diagnostics. The first 8 bytes have two forms:

	Field	*Size*	*Meaning*
(1)	psNum	1	One-byte power supply address in range 1–31
	command	3	Three-character ascii command code
	setData	4	BCD data value for setting, using 0xA for decimal point
(2)	psNum	1	One-byte power supply address in range 1–31
	command	3	Three-character ascii command code
	nCRecv	1	#characters received in reply
	eTime	3	Elapsed time from command to reply received, in ms units

The difference between the two forms is that a command that sends a numeric value to a targeted power supply is of the first form. A command that sends no numeric value simply uses the 4-byte field to hold any single digit value sent as an integer (with no decimal point indicator). The second format is used whenever a reply is received. If we use the BCD format for numeric values, then the nCRecv byte can be used to indicate which format is given, since it will be nonzero for any reply, whereas the first byte of the BCD form will be zero, since no numeric values use more than 6 digits. The ms elapsed time values are not expected to use more than one byte. If desired, we could use all BCD format for these four bytes, whic would make it easier to read. That would make the 0xA digit stand out a bit more, as it will be part of each numeric value.

To illustrate how the two formats might look in hexadecimal, here are examples:

```
(1)   0143 5552 0005 A200      ps#1, CUR, 5.200 amps

(1)   014F 5554 0000 0001      ps#1, OUT, 1 (set output on)

(2)   0153 5454 5300 0071      ps#1, STT, 53 chars received, 71 ms
```

The times for type 1 formats indicate when the command started being sent out the serial port. The times for type 2 formats indicate when the reply data has been received. The elapsed time recorded, then, reflects the baud-rate time to send out the command, the time to process the command by the power supply, and the time to receive the reply text. It is assumed that the local application is called soon after the arrival of the CR-LF characters that terminate the reply text.

Data structures

What structure format is suitable for the management of communications with each power supply?

```
typedef struct {
      byte psState;     /* power supply up/down state */
      byte psAdr;       /* power supply address */
      short marks;      /* flags for signaling settings */
      short peakVolt    /* peak voltage value */
      short peakCur;    /* peak current value */

      byte waitPeak;    /* count-down for MDL/REV request */
      byte waitRemote;  /* count-down for remote/local req */
      byte waitOver;    /* count-down for overVolt request */
      byte waitUnder;   /* count-down for underVolt request */
      short version;    /* firmware version# *100 */
      short timeoutCnt; /* #timeouts due to lack of response to query */

      byte4 status;     /* 3 status bytes, 1 control byte */
      long spareL;      /* -- */

      float actVolt;    /* actual values */
      float actCurr;

      float actSetVolt; /* set values */
      float actSetCurr;

      float actSetOver;
      float actSetUnder;

      float setVoltMon; /* last recognized setting values */
      float setCurrMon;

      float setOverMon
      float setUnderMon;
} PSContext;
```

There is an array of these 64-byte structures that is indexed by power supply numbers 1–31.

The values for the marks field are as follows:

	Bit#	*Meaning*
(hi byte)	7	Output ON
	6	Output OFF
	5	Auto-restart ON
	4	Auto-restart OFF
	3	Clear status registers
	2	Foldback protection CANCEL
	1	Foldback protection ARM
	0	Foldback protection RELEASE
(lo byte)	7	Set voltage
	6	Set current
	5	Set over-voltage
	4	Set under-voltage
	3	Get over-voltage
	2	Get under-voltage
	1	Get remote/latched-remote status
	0	(spare)

When readings need to be updated, following processing of the reply data, the following procedure can be used:

```
PROCEDURE SetRealR(chan: Integer; n: Integer; VAR data: Real);
```

The arguments are the starting channel#, the number of consecutive channels targeted, and the array of floats that are to be converted into 16-bit raw values and deposited in the reading fields of successive ADATA table entries.

Digital status

In order to deliver digital status to the data pool, it maybe useful to deliver it to the dummy addresses found in the BADDR table. On the following cycle, then, the usual RBinary Data Access Table entry (0x0405) will update the BBYTE table.

If the BBYTE table entries are targeted instead, the contents of the memory bytes pointed to by the dummy byte addresses will not reflect what is in the BBYTE table, since the BBYTE table is not updated every cycle for all power supply status bytes. The updating only takes place for a given power supply when new status data is received in response to the STT? command. In between such updates, the normal updating of the BBYTE table occurs using values that are held in the dummy bytes. This is why the contents of the dummy bytes need to be set. To be sure of the BADDR entry contents, require that it look like a non-volatile memory address.

Restore

If all of the analog control is handled by dummy channels, how will the settings be passed to the supplies during the Restore operation occurring following system reset? The Restore operation will only deliver the data to the dummy channels. To solve this problem, assume that the local application does all of the settings following initialization. It simply marks all of these setting activities to be performed, using the setting values it obtains from the dummy setting values. The assumption here is that these dummy values always represent the user intended values for the power supplies. If settable values are installed in a

power supply locally, the computer will not know about them, although it will update the set value channels. It will not attempt to "discover" that the user has done something locally and update the dummy setting values to reflect the set values. The user is expected to perform settings remotely through the computer. Without doing this, the restore operation, which can result from resetting the front end or merely restarting the local application, cannot be expected to work correctly and deliver the proper values to the supplies.

Post-implementation notes

What is going on when nothing is happening? Each 15 Hz cycle, the current supply is queried by an STT? command. As soon as a reply is received, an ADR command is sent to select the next power supply in numeric sequence. The reply to the STT? command is expected to take more than one 15 Hz cycle to execute and return a reply at 9600 baud, so the result is that supplies are sequenced every 2 cycles. (Actually, it is almost possible to receive the reply within a single 15 Hz cycle; if that is so, we might sequence through the supplies at a 15 Hz rate. The unknown is the time that the supply requires between reception of the STT? command before it begins its 54-character reply message.)

When a setting is to be performed for the current supply, the appropriate setting message is sent, and on the following 15 Hz cycle, the STT? query is sent, and after receiving a reply, it moves to the next supply. The last command sent to a supply is the query commnd, and no more than one command is sent to that supply before the STT? command is sent. In the extreme case that settings are waiting to be sent to all supplies,the sequencing from supply to reply should take 3 cycles rather than 2.

Installation steps

The ZUPS local application supports up to 31 Lambda power supplies that use the Zero-UP serial (RS232-based) protocol. The supplies are addressed beginning at 1. The number of supplies scanned is given by a parameter. Eight analog channels are allocated for each power supply in sequence, with the meanings of voltage, current, overvoltage protection, undervoltage limit, voltage setting, current setting, peak voltage, and peak current. Four binary bytes are allocated for each supply in sequence, including operational, alarm, and program status bytes and one control byte. A data stream called "ZUPSLOG " should be defined to hold diagnostic records of recent settings that have been made, plus a record of recent data that has been acquired, including the number of characters received and the time between sending the query command and receiving the reply. The serial port baud rate should be set to 9600 baud, the fastest rate supported by the Zero-UP protocol. So, if one has to support 5 supplies, numbered from 1–5, there will be a sequence of 40 analog channels defined, plus 20 bytes (160 bits) defined. Of course, an entry for ZUPS must be placed in the Local Applications table (LATBL) specifying the appropriate parameters, and the LOOPZUPS program downloaded from node0508. The local application also needs access to a data file called "DATAZUPD". The data file contains the specification of numeric formats assumed by the various power supply models in the ZUP series. Only 15 models are known, although the data file can support up to 24. A power supply model is defined by the two values of peak voltage and peak current.