THE VIRTUAL INSTRUMENT RECORDER

VLA T.R. # 78

BY

PHIL DOOLEY & DAVE WEBER

VLA Technical Report No. 78

VIRTUAL INSTRUMENT RECORDER

Phillip Dooley and David Weber

December, 1999

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS, LabVIEW "G" CODE AND DRAWINGS

1.0 INTRODUCTION:

This manual describes the salient features of the Virtual Instrument Recorder (VIR) system

The VIR System is a maintenance facility that provides graphical time-history charts of user-selected VLA monitor data. The VLA Monitor and Control System (M&C System) gathers monitor data that is indicative of the performance of the VLA system. This real-time data is typically presented to users in a number of computer-driven displays of parameters specific to the VLA hardware function of interest. Examples of these specialized displays are Array Operator Interfaces and System and Module screens. Generalized displays are Data Set MW1 and MW2 screens.

These two types of real-time displays show data as it is gathered but sometimes it can be difficult to compare disparate monitor data, particularly as a function of time. In analyzing VLA system performance it is often useful to group combinations of data channels in a graphical, multiple-channel, time-history format - similar to an analog strip chart recording in which all channels are plotted on a single chart with a common time base. The Virtual Recording System was developed to provide this type of graphic recording capability for up to eight users who can concurrently select and plot data of interest. The VIR System replaces an 8-channel Data Tap-Strip chart recording system at the VLA that was usually restricted to a single user.

It should be noted that for many years the MONPLOT program has provided yeoman data plotting service for VLA monitor data; it plots one channel of analog or digital monitor data. The plots are annotated and can be printed on hard copy. MONPLOT obtains its data from logged monitor data files that have a low, fixed sample rate. For a number of reasons the MONPLOT data sample rate is not conveniently changeable.

The Virtual Recorder system consists of a group of nine networked (via Ethernet) X86 PCs, GARCON, and eight remote (from the VLA) CLIENT PCs. GARCON is the system host and provides monitor data to eight VIR system users via the CLIENT PCs. A hardware interface in GARCON, bridged onto the M&C system's serial monitor data stream, detects the monitor data and causes it to be stored in GARCON's RAM. Each CLIENT computer is equipped with several (presently three) Application Programs. When a user selects an application program, he specifies the data channel addresses (up to eight) to be charted and the application program's control parameters such as scaling, sample rates, chart period, etc. The CLIENT then transmits a list of these eight user-requested data channel addresses to SERVER in GARCON. Using the CLIENT's channel address list, GARCON reads the CLIENT's data from RAM and transmits it to the CLIENT which then provides it to the user's application program.

The VIR System's Application Program screens resemble a control/display panel that includes several multi-channel time history charts. VIR_GRAF is the most general application program; the reader should take a quick look at the VIR_GRAF Front Panel (Section 2.7). Note that users can specify monitor data address, the plotting (data sampling) rate, the chart length, and a time "tick" rate on a marker chart. The chart's data signal level and time axes are annotated. In addition, users can scale and offset the traces, label them (e.g. Channel 1, phase angle, degrees), and switch a data filtering function in or out. Data selection is solely the choice of the users; the VIR imposes no selection constraints[1]. For example, users may elect to view eight channels from a single VLA module or system or, alternatively, view identical channels from eight antennas. The screen images may be printed by color or black ink printers.

[1]It should be noted that there are a few modules in which monitor data is packed in multiple-argument formats; these require an application program specialized to the formats.

The VLA Weather Station and 600 MHz Round Trip Phase application programs are more specialized because their charts are a specific usage of monitor data. The reader should now glance at these two program's control panels which are shown in Sections 2.8 and 2.9, respectively. Note that these three control panels have many common features.

Because the data displays and user's interactions with the system are essentially graphical, user interfaces are coded in LabVIEW "G" ("G" for graphical) code. The VIR software performs two functions: 1) the data routing function performed by SERVER (in GARCON, Section 2.1) to up to eight CLIENTs in remote user's PCs and 2) Application programs that perform data analysis/charting functions (VIR_GRAF, Weather and RT_Phase) in PCs that use data provided to them by their CLIENT.

The VIR system user may find it convenient to refer to the VLA "DCS Manual," (not a VLA Technical Report) which lists the VLA monitor data channel addresses, and associated scaling and tolerance ranges. This is recommended because the VLA M&C system address space is about 37,000 data channels for 32 antennas (DCSs), 6 Data Sets/DCS, and 192 data channels/Data Set. About 9900 data channels are currently implemented.

This Manual's Intended Usage

This manual was written to assist VIR users, facilitate system maintenance and assist those who may want to add additional VIR application programs. The description level is fairly detailed but not exhaustive; the reader is assumed to be somewhat familiar with the VLA system and its functional components. The Appendix contains a brief description of the VLA Monitor and Control System (M&CS) data acquisition functions.

Those who perform VIR system maintenance or add application programs are assumed conversant with X86 class PCs and their configuration, X86 assembly language, "C" and LabVIEW[2] 5.0. This manual is not a tutorial on these tools; see the associated reference manuals for details on their use. LabVIEW was selected for the VIR because its features are convenient for this function: a wide variety of graphic displays and controls, and flexible data plotting features that provide nice, high resolution charts.

LabVIEW Description Conventions

Format Conventions used in LabVIEW program descriptions are: 1) <u>Underlined Items</u> indicate Control Panel controls or indicators, e.g. <u>Channel Entry List</u>; 2) UPPER CASE Items are LabVIEW functions, e.g. INDEX ARRAY or REPLACE ARRAY ELEMENT, etc.; 3) Major programs, VIs and major components are UPPER CASE, e.g. CLIENT, SERVER, GARCON, etc.

VIR System Capabilities

Notable features of the VIR system are:

- The VIR System can concurrently serve up to 8 users with no interactive constraints.
- The user is free to focus on the phenomena of interest with only a minimal understanding of the M&C System operating details. However, the user should understand the M&C System address conventions and MW1 and MW2 usages. (See the Appendix for a brief description of these details.)

[2] LabVIEW is a trademark of the National Instrument Company.

- The VIR system provides data to the application programs at the M&C system bandwidth.

- Application programs can be both general analog charting programs (such as VIR_GRAF) or specialized programs (such as 600 MHz Round Trip Phase or VLA Weather) for VLA systems analysis.

- The VIR_GRAF application enables users to chart any eight analog data channels (addresses 0 through 127_{10}), in any arbitrary combination. The user can scale, offset, apply a filtering function, select either MW1 or MW2 data, annotate any channel's chart trace and select the marker trace tick rate. Each channel's current (plotting) value is displayed in both analog (volts) and digital (hex) formats on Control Panel numeric indicators.

- VIR_GRAF can also chart up to four sets of algebraic sums or differences of two channels, or any combination of direct and sums/differences of the eight channels, within VIR_GRAF's eight-channel capacity.

- The VIR_GRAF and 600 MHz Round Trip Phase applications strip charts are not limited by the specified chart period. When a chart reaches the selected period, charting continues and the time base, chart traces and marker trace scroll in the same manner as a conventional strip chart recording.

- Digital channel data (addresses 128_{10} to 191_{10}) is currently used in two specialized applications, 600 MHz Round Trip Phase and VLA Weather.

- Each VIR application program is an "Encapsulated Application." In Windows terminology it means its operation is virtually independant of the PC's configuration. See "Encapsulated Application" in Norton's "Writing Windows Device Drivers."

- All VIR system programs operate under Windows 95.

VIR System Limitations

- The VIR does not have bulk data storage capabilities to enable replotting a batch of data with different scaling, etc. M&C system monitor data is temporarily stored in GARCON's computer RAM, old values are overwritten by new values as they arrive.

- The VIR_GRAF application program does not chart digital channels (addresses 128_{10} through 191_{10}). Although this is feasible, currently, there is not a general digital charting application program analogous to VIR_GRAF.

- The VIR system cannot influence the operation of the M&CS because it operates autonomously.

- There are a few VLA modules that read out digital monitor data in a packed argument format in which a 24-bit data channel may include both an argument value and discrete mode bits, etc. Specialized application program(s) are required for these cases. Examples of such are L6, L7, ACU and M7; there may be others.

- The VLA Correlator and VLA WYE Monitor Systems are not connected to the M&CS; therefore, the VIR system cannot chart data from these systems.

2.0 VIRTUAL RECORDER SYSTEM DESCRIPTION

This VIR system description has two aspects: 1) A description of the software-hardware facilities that service the Application Programs; 2) A description of the Application Programs that chart the data for the users.

Sections 2.1 through 2.6 describe the first aspect. Section 2.1 describes GARCON. Section 2.2 describes the SERVER-CLIENT communications and Sections 2.3 through 2.5 describe CLIENT, CH_ENTRY and SERVER. Section 2.6 describes SERVER's Monitor Data Retrieval function which reads and stores monitor data messages in the serial monitor data stream. This function is performed by a combination of interface hardware, "C" language programs and assembly language calls to Windows 95[3].

Sections 2.7, 2.8 and 2.9 describe the second aspect which consists of three applications programs, VIR_GRAF, VLA Weather Station and 600 MHz Round-Trip Phase, respectively.

2.1 GARCON Description

GARCON's hardware-software components are shown in Figure 1, the Virtual Instrument Recorder System Block Diagram (next page). GARCON's top block is SERVER, communicating with CLIENTs via Ethernet lines. As noted above, there are two operations performed in GARCON. In amplified form these are:

1) SERVER obtains CLIENT-requested data from GARCON's RAM and ships it to the CLIENTs. SERVER does this by a function call to a Dynamic Linking Library (DLL). The DLL sets an Interrupt (IRQ2F) to a Virtual Device Driver (VXD) that reads the address-designated data from RAM and forwards it to the DLL, which returns it to SERVER for transmission to the CLIENT via the Ethernet network.

2) A Monitor Data Interface (MDI) in GARCON is tapped onto the M&C System serial monitor data stream. It detects each monitor data message in the message stream and stores it in a FIFO memory. At the start of a new VLA cycle, the VLA Track/Hold (T/H) timing discrete signal sets an interrupt (IRQ5) which causes the previous cycle's data in the FIFO memory to be stored in address-designated locations in the RAM for readout by the operations in 1) above.

GARCON is a typical Pentium-class PC. The VIR service function does not require a high speed processor, high-capacity hard drives, a large RAM memory, etc. The monitor data array in RAM requires about 393 Kilobytes, a small fraction of currently available RAM memories. GARCON is equipped with eight Ethernet ports. Since GARCON's equipage is so unremarkable, it is unnecessary to cite configuration specifics.

[3]Windows 95 is a Microsoft trademark.

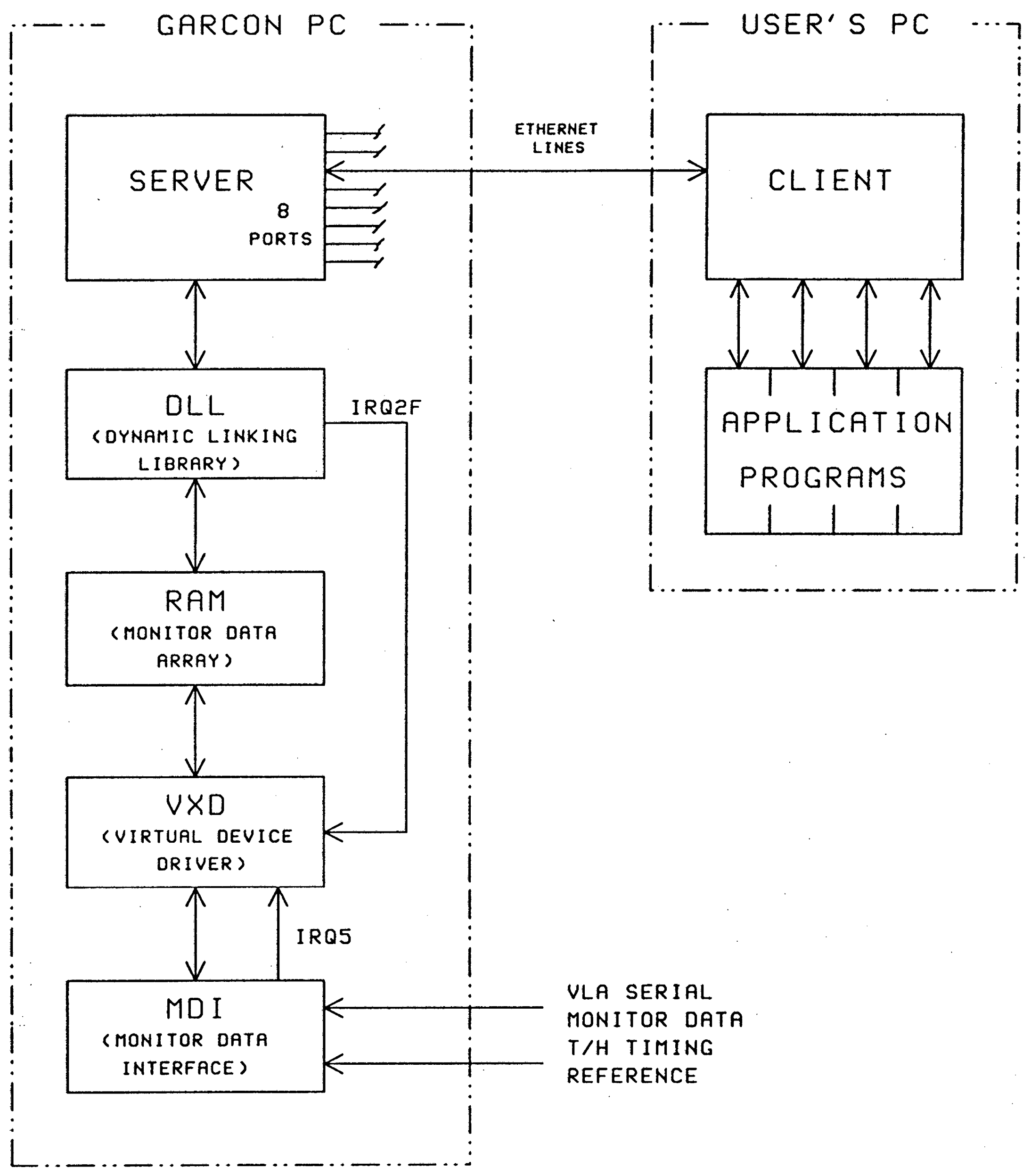


FIGURE 1, VIRTUAL INSTRUMENT RECORDER BLOCK DIAGRAM

2.2 SERVER-CLIENT Communications

A major aspect of the VIR system is the communications between SERVER (in GARCON) and CLIENT (in the user's PC); the two programs provide monitor data to the user's application program. Information is conveyed via time-serial digital messages on Ethernet lines. This section describes the SERVER-CLIENT message formats and protocol, and the relevant Ethernet-Internet Protocol features, but does not describe the details or structure of the communications network.

It may be useful for the reader to refer to the CLIENT and SERVER descriptions in Sections 2.3 and 2.5, respectively.

SERVER can concurrently provide data from any sixty-four (of the approximately 10,000) monitor data channels to eight CLIENTS in any combination of eight sets of data channels per CLIENT. SERVER runs continuously, always listening for a request for data-routing service from any of the eight remote CLIENTS. CLIENT programs are started when a VIR system user wants to examine monitor data. He enters up to eight sets of channel address components into the application program and starts it. The application program puts the address components into its CLIENT which transmits these addresses in a data request message to SERVER. When SERVER reads a data request from a CLIENT, it then continuously routes the requested data to the CLIENT at its maximum execution rate. Upon receipt of a data message, CLIENT provides the data to the application program. When a user stops the application program, CLIENT terminates SERVER's data routing by sending a disconnect message, EOT (\04) to SERVER. SERVER responds to the CLIENT's disconnect by sending an ACK (\06) back to CLIENT. A 60-second timeout in CLIENT will also send an EOT disconnect message to SERVER. In addition, a 60-second timeout in SERVER disconnects the data routing.

SERVER continuously listens for a new data request from inactive CLIENTs; it can do this while it is transmitting data to active CLIENTS. This capability derives from the data-driven character of LabVIEW that permits concurrent program operations and LabVIEW's implementation of the TCP (Telecommunication Protocol) functions, LISTEN, OPEN, CLOSE, READ, and WRITE when isolated by logical or sequence structures.

To initiate data routing from SERVER, a CLIENT first sends a HEADER message to SERVER where HEADER = length of impending DATA REQUEST Message, 4 ascii characters.

CLIENT then sends a DATA REQUEST Message listing each channel's address components: <Channel #1 Address>...<Channel #8 Address><EOM>. This message is transmitted in Spreadsheet String Format which inserts a TAB character between each data character and terminates the channel's address string with an EOL character. The message is terminated by an EOM character. In this list, Channel # refers to the line number in the CLIENT's Channel Entry Table. The <Channel #N Address> format is as follows: $MW+T+DCS_1+T+DCS_2+T+DSA+T+MUX_1+T+MUX_2+T+MUX_3+EOL$, where T denotes TAB and the subscripts denote the first, second and third character of the DCS or MUX components. Leading zeros are omitted which makes the DATA REQUEST a variable length message. EOL and EOM are generated by the ARRAY TO SPREADSHEET STRING function and detected by the SPREADSHEET STRING TO ARRAY function. Since they are intrinsic elements of the Spreadsheet functions, the LabVIEW programmer's graphic code should not attempt to generate or detect EOL or EOM. The user is not required to list eight addresses if he is interested in fewer channels; SERVER does not require eight addresses and will send only the requested data.

SERVER data messages are transmitted in the DATA REQUEST message channel order: <Channel #1 Data>....<Channel #8 Data><EOM>. Each data value is 8 ascii characters, transmitted in Spreadsheet Format: $CH_1+T+CH_2+T+CH_3+T+CH_4+T+CH_5+T+CH_6+T+WCC_1+T+WCC_2+EOL$, where the subscripts 1 through 6 (MSB = 1) designate the six characters in the 24-bit data value and the WCC components designate the waveguide cycle count at the time that the data value appeared in the M&CS data stream tapped by GARCON. The waveguide cycle count feature is not used in any of the application programs and the value is discarded by CLIENTs. Unlike CLIENT, SERVER does not use a HEADER message to designate the number of characters in the data message since the message format is fixed at 8 characters.

The communication links between SERVER and CLIENTs are Ethernet Internet Protocol lines. SERVER and CLIENTs are connected to Ethernet port 18C4H which is an unnasigned function port. SERVER and CLIENT's Ethernet lines are connected to NRAO Ethernet Domain Name Servers that map the Ethernet address into four Internet Protocol (IP) address components. The Domain Name Servers operate in the nrao domain and the nrao symbol is a component of the Ethernet address. SERVER's Ethernet address is garcon.vla.nrao.edu which is mapped into IP address 146.88.60.XX (for edu.nrao.vla.garcon, respectively). The eight CLIENT's Ethernet addresses are clientn.vla.nrao.edu for IP addresses 146.88.60.YY. XX and YY are machine addresses. The Remote IP Address display on the SERVER CONTROL PANEL (see Section 2.5) shows the IP address of the CLIENT being serviced. If more than one CLIENT is being served, the displayed address will constantly change.

2.3 CLIENT

At this point, it is recommended that the reader refer to Section 2.2 which describes the SERVER-CLIENT message format and protocol, and the associated Ethernet-Internet Protocol features.

CLIENT VI resides in the user's PC and is the functional partner of SERVER in GARCON (See the SERVER description in Section 2.5.) Operating in conjunction, they route user-requested monitor data from GARCON to the user's application programs. When starting an application program, the user first enters into CLIENT the address list of the monitor data channels required by the application program. CLIENT then transmits this list to SERVER, receives the designated data from SERVER and makes it available to the application program.

CLIENT is (almost) a general purpose program in that program operations are nearly identical for each VIR application. The "almost and nearly" qualifiers refer to the list of monitor data addresses that CLIENT provides to SERVER; the user provides all address components for the general-purpose VIR-GRAF application but the VLA Weather Station and Round Trip Phase application CLIENTs have preset DSA and MUX addresses because they are fixed and particular to the application. Address presetting reduces the user's "busy work" for these applications. If a user elects to use VIR_GRAF, he must first enter the desired monitor data channel addresses in the CLIENT's CONTROL PANEL Channel Entry List before setting the Control Panel's RUN/STOP switch to RUN.

The CLIENT CONTROL PANEL and BLOCK DIAGRAM follow this text.

CLIENT's request message consists of eight sets of addresses, typified by the VLA Weather Station addresses shown in CLIENT's CONTROL PANEL Channel Entry List. Although the table's ANT and MUX column entries are in decimal and octal formats, respectively, CH_ENTRY VI (a sub VI imbedded in CLIENT) converts these values to Hex characters. Note from the Channel Entry List that the address list message length is variable because the ANT address may be one or two Hex characters and MUX address may be one, two, or three Hex characters. This message is transmitted in string format.

SERVER's data messages always consist of eight sets of six-byte Hex values that are displayed in the Control Panel's Channel Data Display. This message is also transmitted in string format. The values shown in the display are the VLA Weather Station channel data designated by the weather station addresses in the Channel Entry List. Data Channel addresses (MW0/1, ANT, DSA and MUX) are not included in SERVER's data messages and the data order in the serial data message is that of the Channel Entry List. This is also the order that they are presented to the application program and entered in the Channel Data Display. The Channel Data Display values (six-byte, 24 bit) are shown in two formats: 1) a six digit Hex value (DIGITAL column) and 2) two decimal values representing two analog voltages (ANL and ANL+1 columns). The decimal values are derived by converting the upper and lower 12 bit halves of the 24-bit argument; each 12 bit argument is encoded in 2's complement format. The MW0 and MW1 designations refer to time-ordered Data Set monitor words in the serial VLA Monitor Data stream tapped by GARCON. In general, the user should use MW0 because it will (probably) be sampled more frequently by the Data Set. See the Appendix for a detailed description of the differences between MW0 and MW1 and their relationships in the data stream. The Appendix also lists sampling rates of frequently sampled data channels.

It is not necessary to enter null or default addresses in all rows of the Channel Entry Table in the event that an application program does not require all eight of the possible channels or the user is interested in fewer than eight channels. These rows may have been set to default addresses or to addresses set by a previous user; in either event spurious addresses will probably not affect the operation of an application program since they are irrelevant. The user could set the MW, ANT, DSA and MUX entries to 0, which is a valid address (a weather station channel). MW entries greater than 1, ANT entries greater than 31_{10}, DSA entries greater than 5, and MUX entries greater than 377_8 should be avoided because CH_ENTRY (a subVI to CLIENT) will interpret them as erroneous which will halt CLIENT's execution.

To simplify the logic of decoding messages between CLIENT and SERVER, they are transmitted in Spreadsheet format which inserts a TAB character between each data byte and an EOL character at the end of each table row. Each message transmission is prefixed by 4 bytes that tell the recipient (CLIENT or SERVER) the character count of the impending data message. This enables the recipient to cross-check the message for errors. In the event of an error, CLIENT sends an \06 code to tell SERVER to disconnect the data routing service to the CLIENT; setting the CONTROL PANEL RUN switch to STOP also sends an \06 code to SERVER.

VIR System Data Sampling Rate

The system's data sampling rate may be of concern to a user; this is effectively the application program's execution rate which is determined by CLIENT via the CONTROL PANEL's Updates/Min control (described below). This control determines the CLIENT's iteration period. SERVER asynchronously provides CLIENT the requested eight-channel data messages at its natural execution rate.

CLIENT Execution

The CLIENT BLOCK DIAGRAM follows the CONTROL PANEL at the end of this text.

First, a few words about LabVIEW's case and sequence structures. A Multiframe Sequence Structure (MFS) is the LabVIEW way of obtaining control flow within a data flow framework. The number prefixing the brackets identifies the frame number and the numbers inside the brackets indicate the frame range. A Sequence Structure executes frame 0, followed by frame 1, then frame 2, until the last frame executes. This feature enables the programmer to impose sequential program execution when this is the most appropriate tactic for a process. Case, While, For, Boolean, Single Frame and Multiframe Sequence structures may be nested - and this is done in CLIENT. Case or Sequence Structures can also be used to isolate functions that are mutually exclusive; examples of such in CLIENT are TCP OPEN CONNECTION and TCP CLOSE CONNECTION functions which must be separated by some mutually exclusive logic scheme. This is also the case for the TCP READ and TCP WRITE functions. Within an MFS frame, program execution is data driven. Using MFSs also simplifies program compilation and execution because the compiler operates on smaller processes with fewer logical and data dependencies rather than on those required for large, complex, single processes. The BLOCK DIAGRAMs show that Multiframe Sequence Structures are important elements in CLIENT.

After initialization, multiframe sequence structure (MFS) N[0..1] initializes the Channel Update array; MFS N[0..5] controls the TCP connection and message transfer interactions; MFS N[0..2] (within 4[0..5]) recurrently reads the SERVER's data messages; an 8-pass FOR loop inside MFS 0[0..2] converts the data into the format required by the CONTROL PANEL's Channel Data Display and the application program. The details of this sequence are described below.

Note the CONTROL PANEL operating controls and indicators in the Control Panel. There are two controls: the RUN/STOP switch and the Updates/Min digital control. The panel has two string numeric table indicators: Channel Entry List and Channel Data Display which are the user's primary interest. Data is entered into these tables in numeric form. There are another five string indicators, primarily intended for diagnostic purposes: Channels Enabled, RxMsgString, TxMsgLength, TestPointString, and TxMsg. TxMsg is the list of monitor data addresses (shown in the Channel Entry List) that are to be transmitted to SERVER. Two numeric indicators, #RxMsagLength and Rx Digital Data, also provide diagnostic information. The Data Update and ChanUpdates are flashing LED displays which indicate that a data message has been received. The reverse video Client Status shows the TCP connection status. The functions performed by these controls and indicators are described below.

Referring to BLOCK DIAGRAM P3, note that CLIENT is entered from the halt state by setting the CONTROL PANEL'S RUN/STOP switch to RUN; the first program operation is initialization. The INITIALIZATION arrays are: Tx ARRAY (top left), Channel Entry List, Channel Data Display, and ChannUpDate (bottom). Note that the middle two are CONTROL PANEL numeric displays.

After initialization, CLIENT starts the Main While Loop and it remains in this loop until the RUN/STOP switch is set to STOP. Inside the Main While loop is a call to CH_ENTRY, a sub VI (described in Section 2.4) that tests the validity of the addresses in the Channel Entry List as noted in the paragraph above. If the address components are valid, control passes to the Update Period While Loop in the lower left corner of the Main While Loop.

The Update Period While Loop on the lower left of BD P3, controls the iteration period of the Main While loop. The input to this loop is an OR function with two inputs: the RUN switch and the (TCP) CONNECTED state. Note that this loop has a shift register that stores the loop's iteration completion state (Boolean True) and transfers it back to its input in time for the next iteration. Inside this loop is a WAIT UNTIL NEXT ms MULTIPLE function (Metronome icon) driven by a DIVIDE function which divides the constant 60000 by the CONTROL PANEL'S Updates/Min numeric control. The CONTROL PANEL's Updates/Min control is set to 15; the DIVIDE function output is thus 4000. This makes the WAIT UNTIL NEXT ms MULTIPLE period 4000 ms, or 4 seconds which is 15 updates/minute.

The Update Period While Loop output is connected to one input of an AND function; the other input is connected to the RUN/STOP switch. This AND's output is connected to two other functions: the first output is connected to a Boolean Case Structure (BCS, upper region on P3) that encloses three nested case stuctures. This is the normal, error-free operating path, following the establishment of the TCP connection to SERVER after SERVER has been sent the data channel address request list and SERVER has begun transmission of the requested data messages. The logic for this path is: when the Update Period While Loop output is true, RUN is true, and (TCP) CONNECTED is true the AND's True output causes the code inside the BCS to be executed; which is 4[0..5]. The 4[0..5] operations are described below.

The second connection (mentioned in the paragraph above) initiates the sequence that opens connection with SERVER by opening a TCP connection, sending the channel address list (TxMsg) to SERVER and verifying that the length of the first data message length from SERVER is consistent with the character count in the message.

The logic for this path is: the second AND output (mentioned two paragraphs above) is connected to another AND function that has the (TCP) CONNECTED state as its other input. If CONNECTED is False (the TCP connection not having yet been established) the AND function's True output causes the 0[0..1] frame inside the BCS to be executed which flashes the CONTROL PANEL's Data Update LED for 150 ms via the WAIT (ms) function (wrist-watch icon).

Program control on this second path continues to 1[0..1] (BD P4) which has an INITIALIZE ARRAY function. The array's data input is the state of the RUN/STOP switch state - True. The array's outputs are the CONTROL PANEL's Channel Update LEDs. However, they are not flashed at this time; this happens in sequence 0[0..3] described below. This initialization sets the array elements to True. The array's dimension sizes are 8 (i.e., 8 elements for the 8 Channel Update LEDs) and 1 (i.e., a single bit/LED), respectively.

Following the N[0..1] MFS, the program enters the N[0..5] MFS which establishes, maintains and terminates the TCP connection to SERVER. Note that GARCON's Port Address and the TCP Port# enter the N[0..5] sequence via tunnels; GARCON's port brings GARCON's data into the N[00..5] sequence.

The start of the N[0..5] sequence is Frame 0 on BD P6. 0[0..5] encloses a BCS. If the (TCP) CONNECTED state is False (i.e., not connected), the functions in this frame attempt to make a connection to SERVER via the TCP OPEN CONNECTION function. This function is enabled by the TCP Connect ID code. This function has two output states: error and connection established. If established, the SELECT function causes the CONNECTED state (mentioned above and used in many places) to be set True and the CP Client STATUS indicator to show "CONNECTED to VLA-VIR Server."

If the TCP connection cannot be opened, the SIMPLE ERROR HANDLER function causes the Client STATUS indicator to show: "Cannot Open Conn." The CONNECT state is set False and CLIENT reverts to the halt state via 4[0..5] described below.

Frame 1 of the N[0..5] sequence is on BD P7. If the CP's RUN/STOP switch is set to RUN, the SELECT function inputs the TxMsg string into a STRING LENGTH function which determines the string length. The STRING LENGTH FUNCTION's numeric output output is converted to string form by the TO HEXADECIMAL function. The function's width input is 4 so it writes 4 string characters via the TCP WRITE function to tell SERVER the number of characters to expect in the address request (TxMsg) message. (Remember that TxMsg is the list of data channel addresses that CLIENT will request from SERVER; this happens in 2[0..5].) The TCP WRITE function is enabled by the TCP Connect ID.

If the CP's RUN/STOP switch is moved back to STOP, the SELECT function outputs the EOT (\04) instead of CmdMsg. EOT is transmitted to SERVER and is interpreted as the disconnect command; this stops the data routing to CLIENT.

Control next passes to Frame 2 of N[0..5] which is on BD P7. A TCP WRITE function transmits TxMsg to SERVER. This function is enabled by the TCP Connect ID. Program control proceeds to 3[0..5] which is on BD P8.

In 3[0..5] (BD P8), a TCP READ function reads the length of SERVER's data message; this is contained in the 4-character string that precedes the data message. The TCP READ function is enabled by the TCP Connect ID and the Bytes to Read input is set to 4. Its Data Out is connected to a FROM HEXADECIMAL function that converts the hex string to a hex numeric value. This output, a 6-bit integer drives the decimal #RxMsagLength indicator on the CONTROL PANEL. If the message length is 4, control proceeds to Frame 4 of N[0..5]. If the character count is not 4, the function's Error output drives the Error Input of a SIMPLE ERROR function. The Type of Dialog input is set to 0, which specifies a 1 second-based time-out count. An EQUAL function compares the time-out count with 66; when it reaches 66, a Disconnect request state is set and control proceeds to 4[0..5], described below.

Frame 4 of N[0..5] is on BD P3, and as noted in the overview paragraph above, it receives the data messages from SERVER, converts them into DIGITAL, ANL and ANL+1 values for the CP's Channel Data Display table, and makes this data available for the application program. These functions are performed by N[0..2] MFS inside 4[0..5]. Inside 4[0..5] is a False-condition BCS with DISCONNECT as the variable input. If DISCONNECT is False (the TCP is connected), the N[0..2] MFS is executed.

0[0..2] (BD P4) reads #RxMsagLength, the 4 bytes that tell CLIENT the number of bytes in the data message RxMsgString in Spread Sheet format. The TCP READ function's data output (RxMsgString) is connected to a SPREADSHEET STRING TO ARRAY function. This converts RxMsgString to the RxChanTable array signal, which consists of eight sets of hex string values. If there is a TCP read error, the TCP READ's error output drives the SIMPLE ERROR HANDLER function. The handler's code output, a value incrementing once/second, is connected to an EQUAL function with the constant 56 as the other input. At this count, the EQUAL function's output goes true which sets the DISCONNECT state true through an OR function. SERVER can also cause a DISCONNECT state by sending a #RxMsagLength value of 1 (for one character) and an RxMsgString value of 0. When the joint occurence of these two values are detected by two EQUAL functions, an AND function forces a DISCONNECT state through the second input of the OR function mentioned above.

1[0..2] (BD P4) contains an 8-pass FOR loop that converts the RxChanTable array signal (eight sets of hex channel data values from SPREADSHEET STRING TO ARRAY function in 0[0..2]) to eight sets of values that are to be displayed in the DIGITAL, ANL and ANL+1 columns in the CONTROL PANEL's Channel Data Display table.

The FOR loop encloses the N[0..3] sequence which converts each of the eight values in four sucessive steps. The first frame, 0[0..3] (BD P4) generates discretes that drive the ChanUpdates LED array on the CONTROL PANEL; the next three, 1[0..3] through 3[0..3], formulate three components of the CHANNEL DATA array. 2[0..2] sends this composite array to the CONTROL PANEL's DIGITAL, ANL and ANL+1 columns in the Channel Data Display.

In 0[0..3] inside the FOR loop, the RxChanTable array signal (from the SPREADSHEET TO ARRAY function mentioned above) is connected the input of an INDEX ARRAY whose output is the very important composite array CHANNEL DATA, which is accessed by the 1[0..3], 2[0..3] and 3[0..3] frames. Remember that an INDEX ARRAY returns the element of the array pointed to by the index, or indices if the array is two-dimensional, as is the case in all of the N[0..3] arrays. The first index is the FOR loop i; the second is a numeric index, 0 (for the DIGITAL column), 1 (for the ANL column), and 2 (for the ANL+1 column) mentioned in the paragraph above.

The CHANNEL DATA from the INDEX ARRAY is input to five REPLACE ARRAY ELEMENT arrays: two in 0[0..5] and one each in the three subsequent frames. The REPLACE ARRAY ELEMENT arrays replace with the New Element Input the element in the array at index (or indices).

In 0[0..3] there are two REPLACE ARRAY ELEMENT arrays; the first has as its input the CHANNEL DATA array from the INDEX ARRAY mentioned above. The output of the first is connected to a FROM HEX STRING TO A HEX NUMBER function. This function's hex number output is right-shifted 8 bits and connected to the CONTROL PANEL'S Rx Digital Data numeric display. This right shift chops off the WCC count which is not used. (See Section 2.6 for a description of this data). Note that this display shows the eigth (last) DIGITAL value in the Channel Data Display. The HEX STRING TO HEX NUMBER output is masked in an AND function with FF, thus selecting the lower 8 bits of the 24-bit value. This number is then tested by the NOT EQUAL TO ZERO function connected to the New Element Input of the second REPLACE ARRAY ELEMENT array (mentioned above). Thus, if any of the 8 bits are a true ("1"), a 1 is input to the array. This array is the CONTROL PANEL's ChanUpdates LEDs. Therefore, if any of the 8 (of the 24) bits are true, the associated LED is lit.

In 1[0..3] (BD P4) the DIGITAL component of the CHANNEL DATA array is altered in a REPLACE ARRAY ELEMENT. This is done by the TO HEXADECIMAL function, with the 24-bit numeric Rx Digital Data (paragraph above) as its input. This function's hexadecimal string output is connected to the REPLACE ARRAY ELEMENT's New Element Input. This string value is the DIGITAL column's value.

In 2[0..3] (BD P5) the upper 12 bits of the 24-bit numeric Rx Digital Data value is converted to a 2's complement value. The Rx Digital Data is right-shifted 12 bits by the LOGICAL SHIFT function to select the upper 12 bits, then AND'ed with FFF; thus the data is only 12 data bits wide. The AND output is connected to two functions: 1) another AND with a 7FF input; this masks off the MSB. The resultant value is then multiplied by 0.005 in a MULTIPLY function to produce a value ranging from 0 to +10.235, the 2's complement value for the maximum positive voltage. (The value is multiplied by 0.005 because the Data Set's A/D converter scaling is 5 mV/count.) 2) the second connection is to a GREATER? function which compares the AND output with 7FF; if the AND output is greater than 7FF (MSB = 1), the GREATER? output is True which causes the SELECT function to select the (T, top) input. This is the negative value described in the next paragraph.

The 2's complement positive value from the multiplication mentioned above is in turn connected to two functions: the F (bottom) input of the SELECT function and one input of an ADD function. The other input is -10.240 and the sum of -10.240 and the positive value is connected to the T input of the SELECT function. This operation is the equivalent of taking the 1's complement of the positive value and then adding 1 count. If the data value is 7FF or less, the SELECT function outputs the positive value; if the data value is greater than 7FF (i.e., the MSB is a 1, indicating a negative data value), the SELECT function outputs the SUM function's output (i.e., the sum of -10.240 and the input value).

The SELECT function's output is connected to a TO HEXADECIMAL conversion function which converts the hex number to a hexadecimal string. The width inputs are 7 and 3, respectively. The resultant hexadecimal string if the value of the ANL+1 column which is connected to the New Element Input of a REPLACE ELEMENT ARRAY; the array output is the CHANNEL DATA array.

Frame 3[0..3] operations are almost identical to the 2[0..3] operations, the only differences being that the lower 12 bits of the 24 bit value are converted to a 2's complement value. The 1 on the lower array index places the ANL+1 data in the second column position in the CHANNEL DATA array.

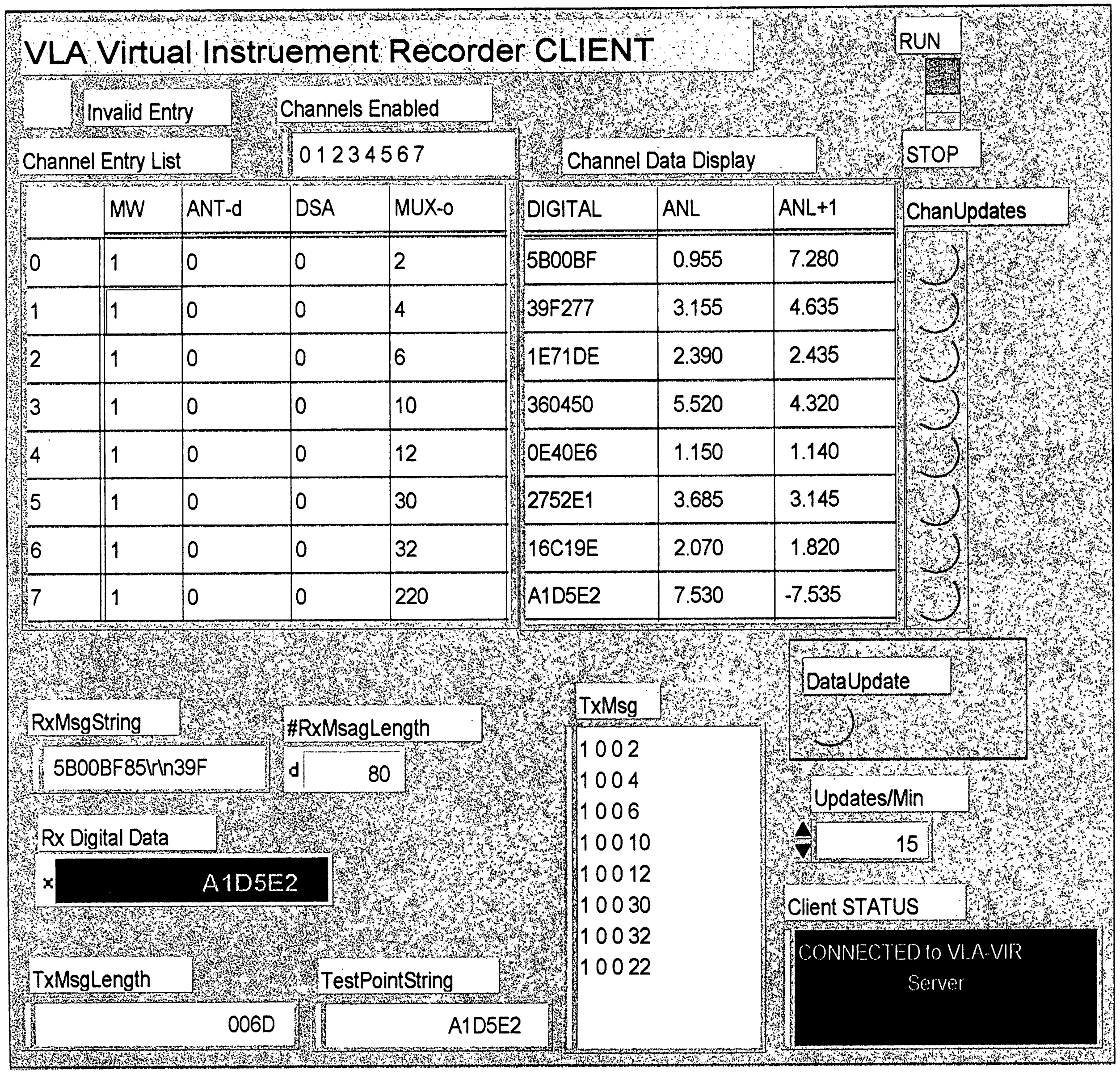
Frame 2[0..2] (BD P3) places the CHANNEL DATA array in the CONTROL PANEL's Channel Data Display. This completes the operations in 4[0..5].

Frame 5[0..5] (BD P8) encloses a BCS that operates when the DISCONNECT state is True. If True, the TCP CLOSE CONNECTION function is activated which tells SERVER to disconnect. The CONTROL PANEL's <u>Client STATUS</u> indicator is set to show DISCONNECTED. After this, control reverts to the halt state.

WEATHER CLIENT

/eather Client.vi
:\LabviewNT\vla_wsa.llb\Weather Client.vi
ast modified on 5/13/96 at 7:17 AM
rinted on 10/15/98 at 11:52 AM

ront Panel



VLA Virtual Instruement Recorder CLIENT

RUN

STOP

Invalid Entry

Channels Enabled: 0 1 2 3 4 5 6 7

Channel Entry List

	MW	ANT-d	DSA	MUX-o
0	1	0	0	2
1	1	0	0	4
2	1	0	0	6
3	1	0	0	10
4	1	0	0	12
5	1	0	0	30
6	1	0	0	32
7	1	0	0	220

Channel Data Display

DIGITAL	ANL	ANL+1
5B00BF	0.955	7.280
39F277	3.155	4.635
1E71DE	2.390	2.435
360450	5.520	4.320
0E40E6	1.150	1.140
2752E1	3.685	3.145
16C19E	2.070	1.820
A1D5E2	7.530	-7.535

ChanUpdates

DataUpdate

RxMsgString: 5B00BF85\r\n39F

#RxMsagLength: d 80

Rx Digital Data: x A1D5E2

TxMsg:

```
1 0 0 2
1 0 0 4
1 0 0 6
1 0 0 10
1 0 0 12
1 0 0 30
1 0 0 32
1 0 0 22
```

Updates/Min: 15

Client STATUS: CONNECTED to VLA-VIR Server

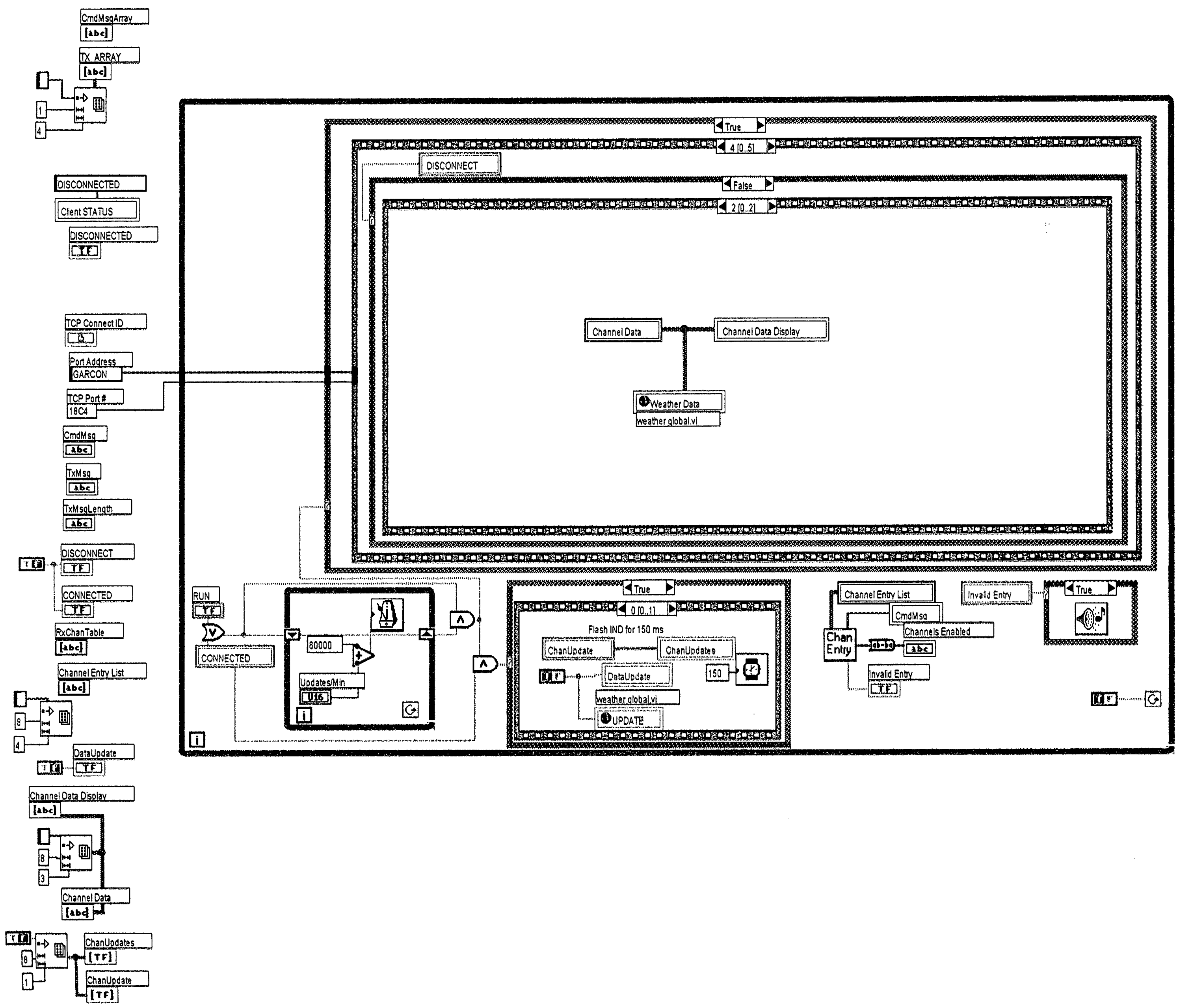
TxMsgLength: 006D

TestPointString: A1D5E2

WEATHER CLIENT

/eather Client.vi
:\LabviewNT\vla_wsa.llb\Weather Client.vi
ast modified on 5/13/96 at 7:17 AM
rinted on 10/15/98 at 11:52 AM

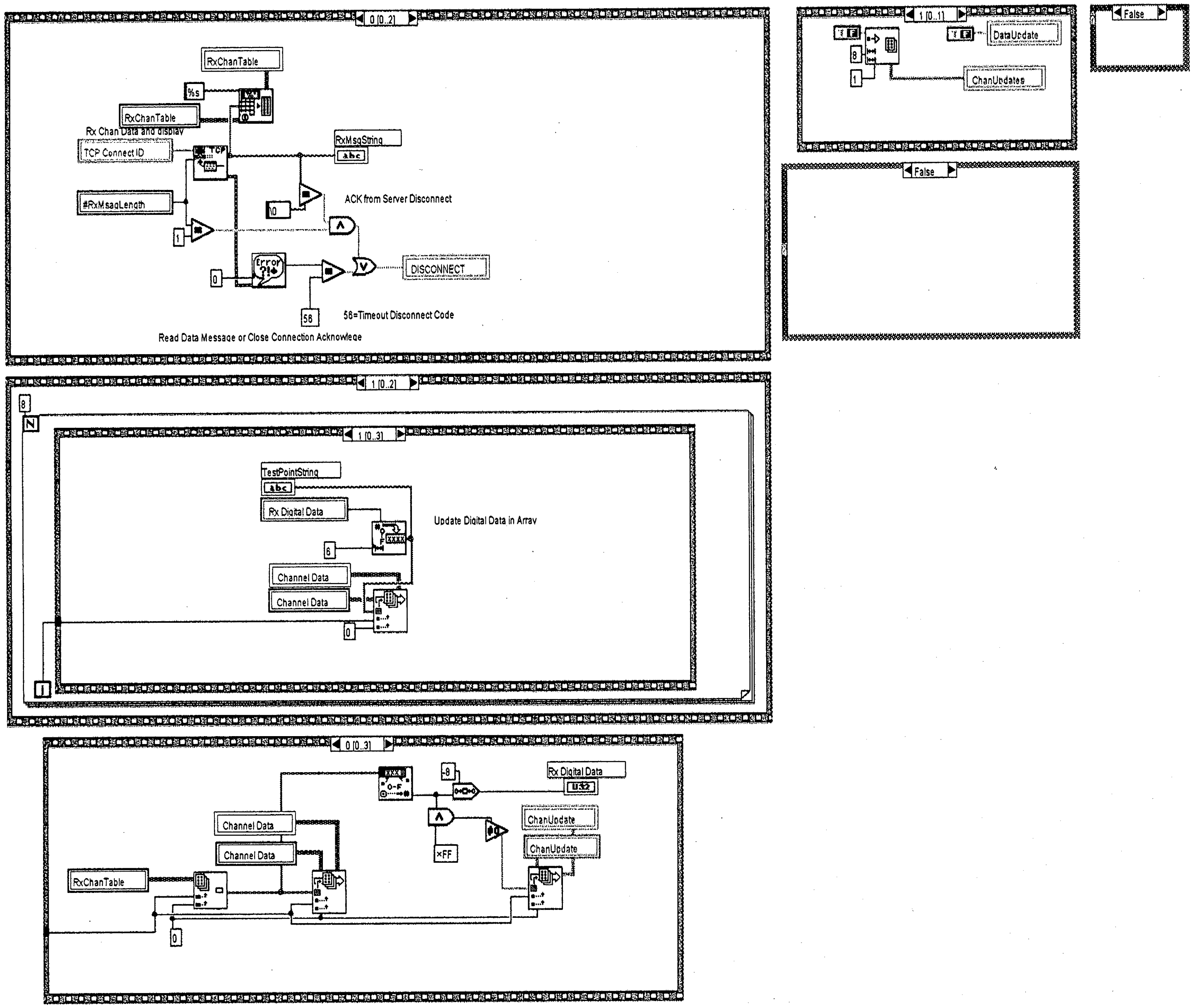
lock Diagram



P

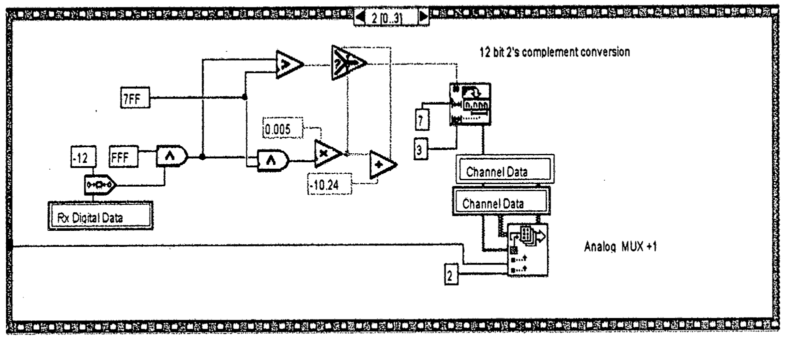
WEATHER CLIENT

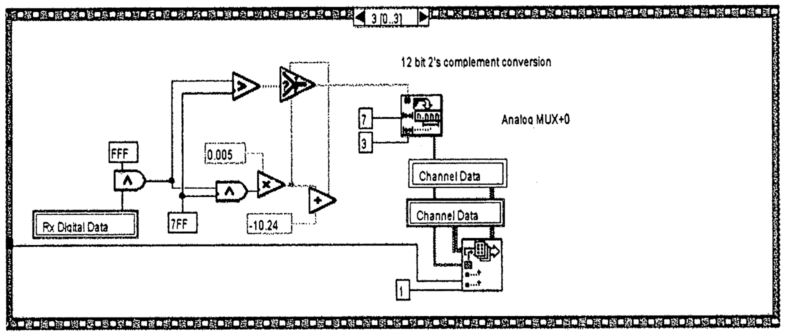
eather Client.vi
\LabviewNT\vla_wsa.llb\Weather Client.vi
ıst modified on 5/13/96 at 7:17 AM
inted on 10/15/98 at 11:52 AM

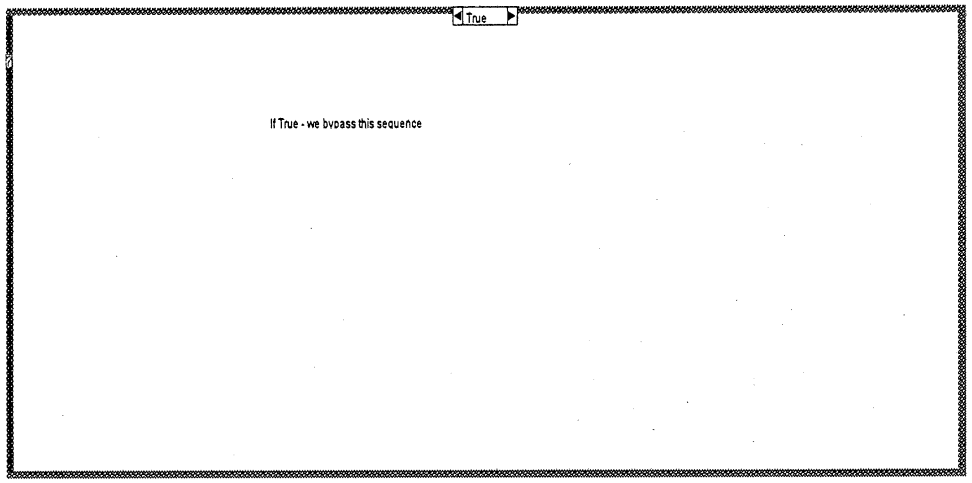


F 5 WEATHER CLIENT

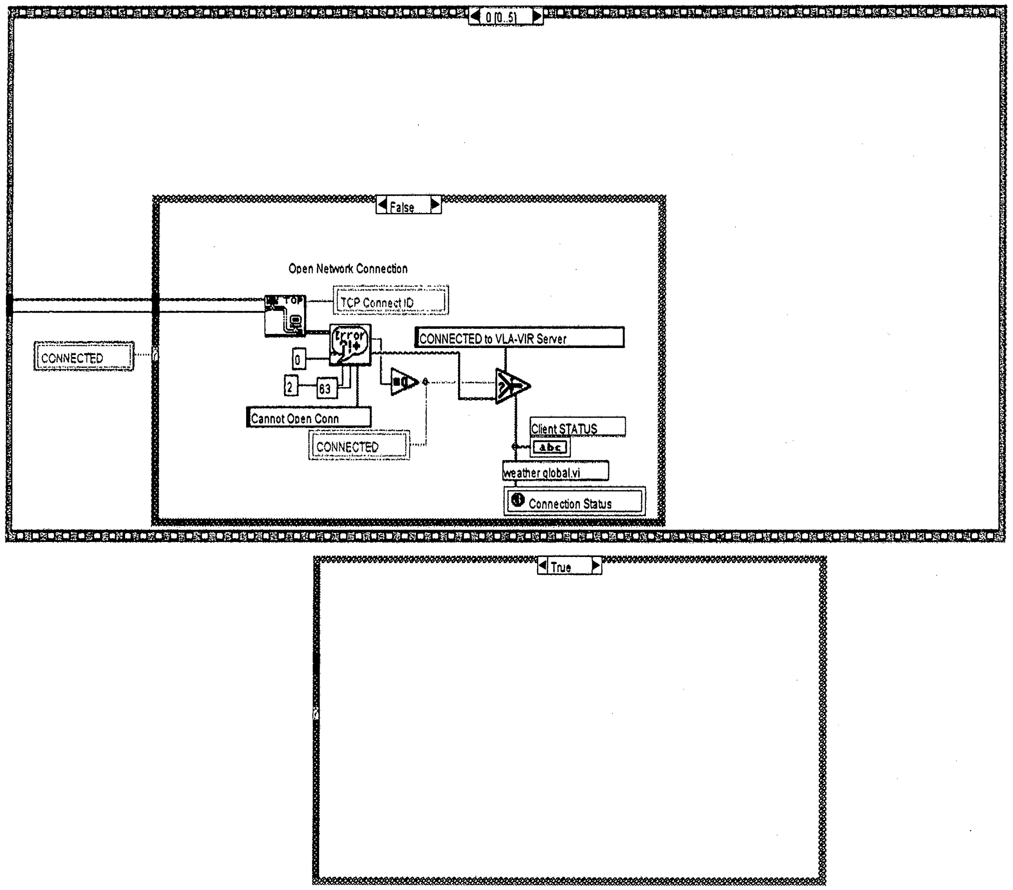
/eather Client.vi
:\LabviewNT\vla_wsa.llb\Weather Client.vi
ast modified on 5/13/96 at 7:17 AM
rinted on 10/15/98 at 11:53 AM







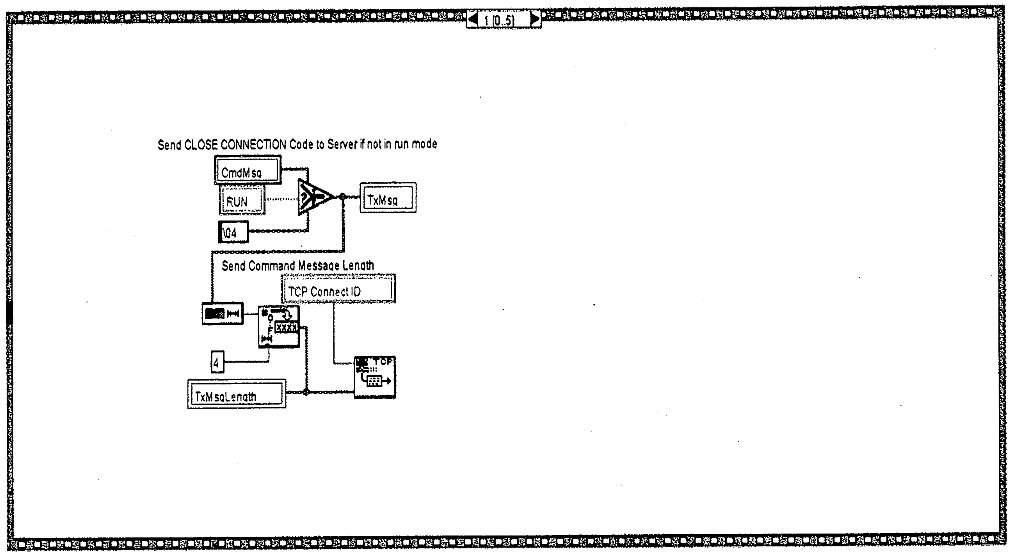
Veather Client.vi
):\LabviewNT\vla_wsa.llb\Weather Client.vi
.ast modified on 5/13/96 at 7:17 AM
'rinted on 10/15/98 at 11:53 AM

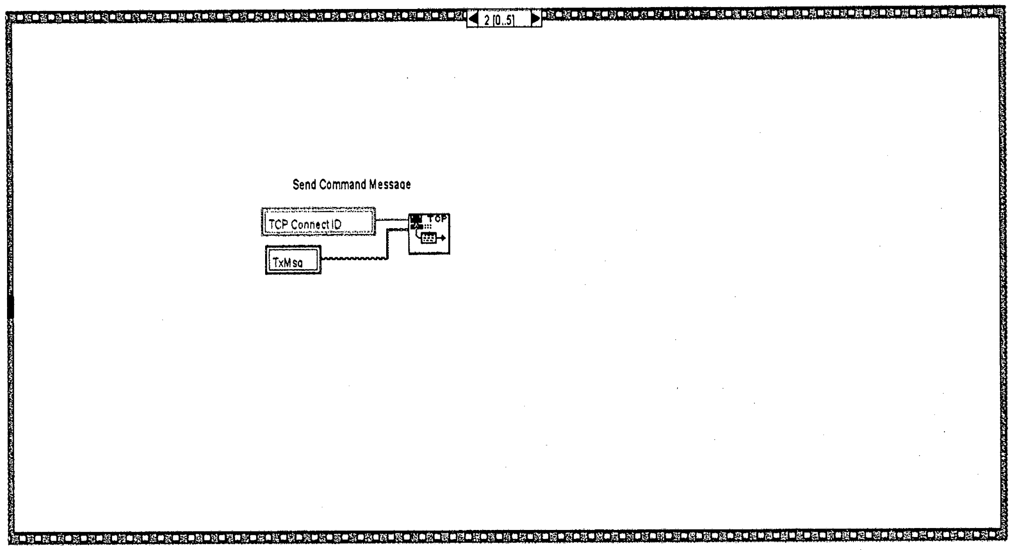


F '

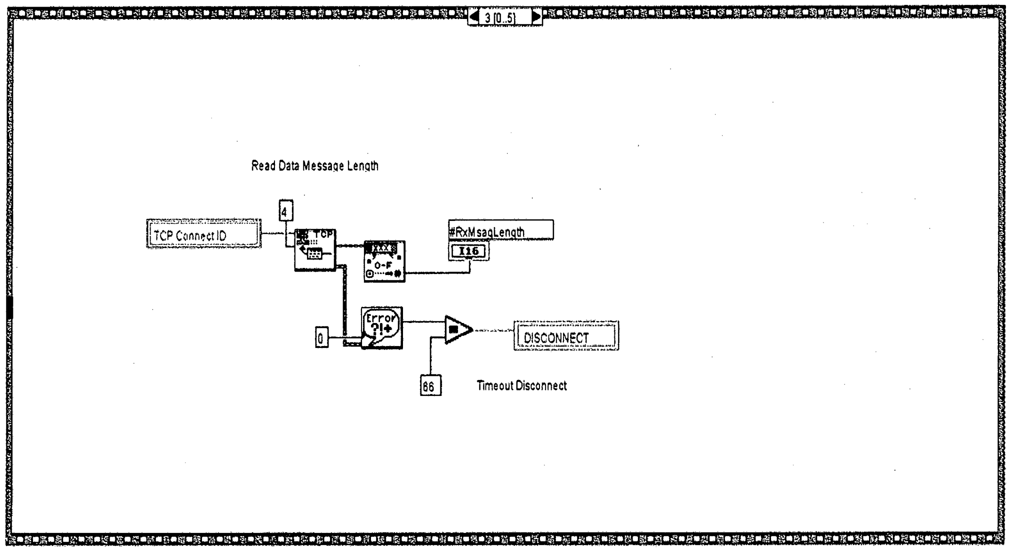
WEATHER CLIENT

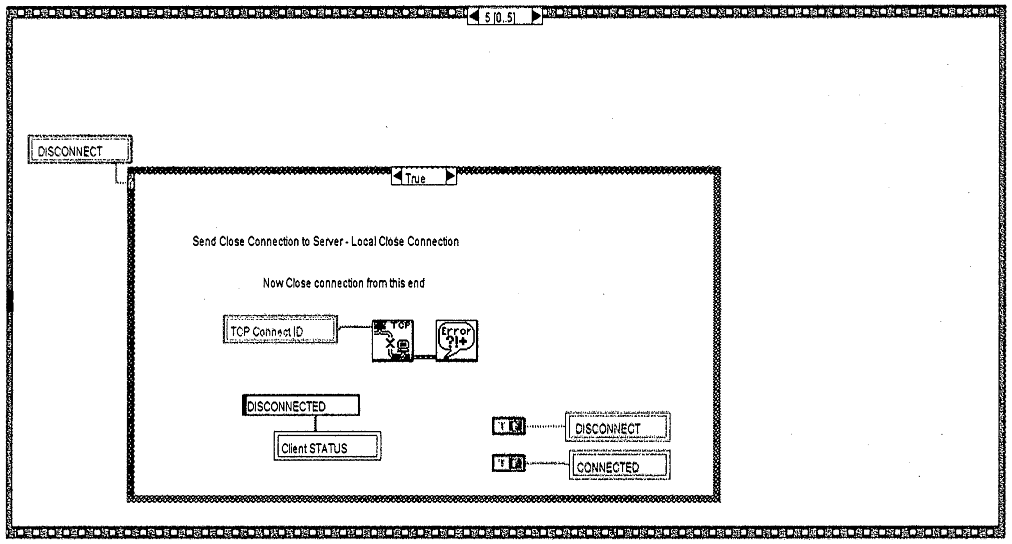
'eather Client.vi
\LabviewNT\vla_wsa.llb\Weather Client.vi
ıst modified on 5/13/96 at 7:17 AM
'inted on 10/15/98 at 11:53 AM


1 [0..5]
Send CLOSE CONNECTION Code to Server if not in run mode
CmdMsg
RUN
\04
TxMsg
Send Command Message Length
TCP Connect ID
4
TxMsgLength
TCP



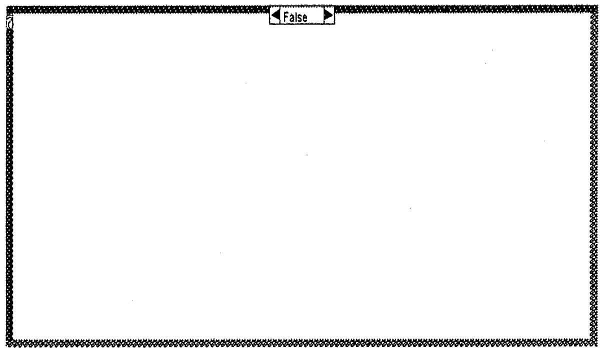
Veather Client.vi
:\LabviewNT\vla_wsa.llb\Weather Client.vi
ast modified on 5/13/96 at 7:17 AM
rinted on 10/15/98 at 11:53 AM

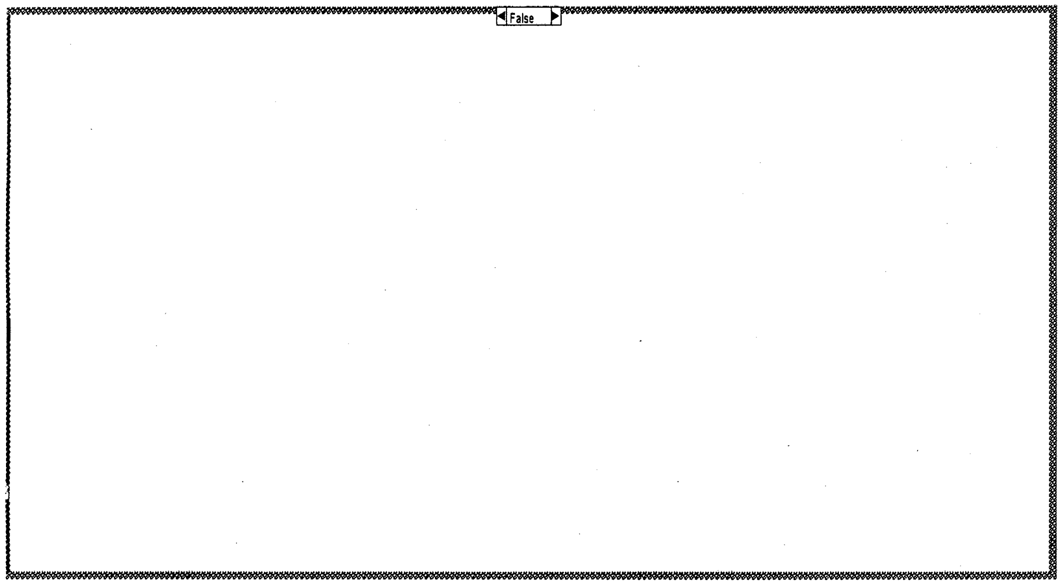




/eather Client.vi
:\LabviewNT\vla_wsa.llb\Weather Client.vi
ast modified on 5/13/96 at 7:17 AM
rinted on 10/15/98 at 11:53 AM

WEATHER CLIENT





2.4 CH_ENTRY

CH_ENTRY.VI is a SubVI embedded in each CLIENT that is called once by the CLIENT when starting the execution of a VIR application program. The application's CLIENT requests that up to eight monitor data channels be routed to the application program. CH_ENTRY tests the range of the address components of the requested data channels to avoid requesting an erroneous or invalid address. The CONTROL PANEL and BLOCK DIAGRAM follows this text.

CH_ENTRY.VI is a fairly general purpose VI and is only slightly particularized for each VIR application. In the case of the Weather CLIENT, this particularization consists of pre-setting the weather station DCS, DS and multiplex addresses; this small fixed set makes the weather appplication more user friendly because it obviates the busy work of looking up addresses. Preset multiplex addresses are used in the Round Trip Phase application because the parameters of interest are only antenna and elevation positions and 600 MHz Round Trip Phase. The VIR_GRAF application is the most general VIR program and must be capable of acquiring data from any VLA M&CS monitor data channel; therefore address pre-setting is not used.

The Channel List Table in the CONTROL PANEL shows eight example sets of address components that are tested by CH_ENTRY. The left column is the table's row index. The column labels designate the data channel's four address components. Note that the first component MW1/MW2 (for Monitor Word 1 or Monitor Word 2) differs from the monitor word address format described in Appendix A. The MW1 and MW2 designations are implicit references for analog data and are not part of the monitor word format. MW1 and MW2 respectively designate the first or second set of 12 bits of data in the monitor data message's 24 bit data field. Antenna Address is actually the DCS address.

Note that in the Channel List Table, Antenna (DCS) Address is a decimal value and the MUX address is in octal format. Octal channel MUX address notation is used here because it's generally used in the VLA system. The reasons are that the M&CS's hardware logic structure is implemented in groups of eight or sixteen channels for which octal usage is convenient. Secondly, the VLA system modules and their documentation use the same notation because of their close linkage with the M&CS. Thirdly, the VLA DCS Manual which lists all monitor and control addresses uses the octal convention.

CH_ENTRY tests that only MW1 or MW2 are requested, that each Antenna Address (i.e. DCS address) is in the range of 0 to 31_{10}, that the DS address is in the range of 0 to 5 and that the MUX address is in the range of 0 to 377_8. Note that this MUX address range also encompasses Command Addresses which range from 300_8 to 377_8. If CH_ENTRY detects an address component out of any of these four ranges, an Invalid Entry signal [T or F] is sent to the CLIENT. **An important point is that in testing the validity of address components, CH_ENTRY does not determine whether or not a requested channel address has actually been implemented in hardware.** For example, CH_ENTRY would permit a user to request data from Antennas 29, 30 or 31 which do not exist. In addition, many MUX addresses have not been implemented. In these cases, SERVER sends a BAD CHANNEL message to the CLIENT. [[

On the CONTROL PANEL, the TxString table is a STRING INDICATOR TABLE (indicated by the vertical bar on the left side) that lists the four sets of address components in the Channel List Table as lines of digits; the first line are the MW1/MW2, Antenna (DCS) Address and DSA digits and the second line are the MUX address digits. Note that in general, the MUX address may be one, two or three digits. This table is a diagnostic convenience because in conjunction with the TxStringLen table, it indicates the size of the character string routed to SERVER. The Channel #'s Enabled table is also a STRING INDICATOR TABLE that shows which channels (i.e. the table row indices) have been enabled in the request message to SERVER. The TxStringLen indicator is another diagnostic STRING INDICATOR TABLE that shows the TxString table character count. Note that the indicated count (88) exceeds the apparent character count for the channels listed; the reason is that the address characters in the requested channel address string routed to SERVER are delimited by a TAB character. This is done to provide an unambigous list to SERVER because the ANT address can be one or two digits and the MUX address can be one, two or three digits. An EOL character delimits each channel's address components but EOL is not counted in the TxStringLen indicator. The use of the TAB and EOL characters make it easy for SERVER to analyze the requested message string to determine the requested channels.

The Invalid Entry LED indicator shows that some invalid or erroneous channel component has been detected. In this event, the user must scan the Channel List Table to identify the error.

The CH_ENTRY block diagram has two regions: an 8-pass FOR Loop and an initialization region outside the loop. The CLIENT inputs the Channel List Table (a string function described above) into the FOR loop and to the array input of an ARRAY TO SPREADSHEET function (at the bottom of the diagram) in the initialization region. This function converts an array of any dimension to spreadsheet string format which is a table in string form; each table character is delimited by the TAB character (09H) and the end of each address component set (i.e., table row) is delimited by an EOL (0AH) character. The function's delimiter input is initialized to the string % character (25H), which is not a "normal" character in the function's spreadsheet string output. In operation, the initialized deliminator value (%) is overwritten by the TAB character. During initialization before the starting the FOR loop execution, the Channel List Table, the TxString table, the Channel #'s Enabled, and TxLen STRING INDICATOR TABLES are initialized to zero. The CP's Invalid Entry LED is also initialized to the FALSE state.

The ARRAY TO SPREADSHEET STRING output is displayed on the TxString table on the CONTROL PANEL as a diagnostic feature. The spreadsheet string is also input to a STRING LENGTH function; the Length output returns the number of characters in the string. This is displayed on the TxStringLen indicator on the CONTROL PANEL.

During each pass of the FOR loop, the loop index i selects a row of address components that are analyzed for range validity. The loop index i is 0 during the first pass and 7 on the last pass; these values correspond to the table's row index. The address components (MW1/2, ANT, DSA and MUX) are each analyzed by an IN RANGE function. If all components are within the prescribed range, AND functions cause the channel address to be added to the Channel #'s Enabled table. If they are outside the prescribed ranges, the Invalid Entry LED is activated on the CONTROL PANEL.

The analysis is done by a RESHAPE ARRAY, four INDEX ARRAYs, four FROM STRING TO INTEGER converter functions (three decimal input, the fourth octal input), four IN RANGE functions, and AND functions that combine the outputs of the IN RANGE functions. The Channel List Table is input to a RESHAPE ARRAY function that changes the dimension of an array according to the dimension size input, the loop index i. The RESHAPE ARRAY function can be used to convert a 2D array into a 1D array which is its usage in this program. In this application, i ranges from 0 to 7 and corresponds to the Channel List Table's row index. The i index to the RESHAPE ARRAY selects one row of the Channel List Table. The RESHAPE ARRAY output drives the array Inputs of four INDEX ARRAY functions. The four INDEX ARRAYS are used to separate the four address components of the table row. The INDEX ARRAY's index inputs (0, 1, 2, 3) select one of the four address component for output. Three FROM DECIMAL string functions convert the MW1/2, ANT and DSA address components to an integer value. The MUX component is converted to an integer value by a FROM OCTAL converter. These four integer values are each tested by IN RANGE functions which return a (Boolean) TRUE if x (middle) input is greater than or equal to lo (lower) input and less than hi (upper) input. If these conditions are not met, the function returns FALSE. The four hi and lo inputs are respectively: 3 and 1 for MW1/2; decimal 32 and 0 for ANT; 6 and 0 for DSA, and Octal 400 and 0 for MUX.

The Channel #'s Enabled table is developed by the SELECT, TO DECIMAL string, and CONCATENATE STRING functions, and the logical sum of the three AND functions mentioned above. The summary AND output drives the s input of a SELECT function. The SELECT function's t input is driven by the TO DECIMAL string function, which has as its input the loop index i. The TO DECIMAL string function outputs a decimal digit string character, converted from the numeric value of i during each pass of the loop. During each pass of the loop, a new string character is input to the CONCATENATE STRINGS function to produce the list shown on the Channel #'s Enabled table on the CONTROL PANEL.

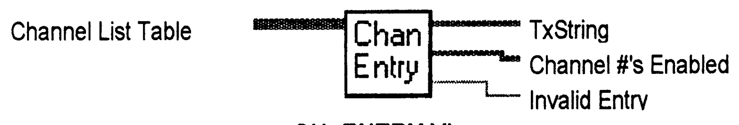
In the event that there are no address components in a table row (i.e., NUL characters), the EMPTY STRING function and AND ARRAY ELEMENTS function return a TRUE to the NOT OR function. If the range analysis or EMPTY STRING outputs are TRUE, the NOT OR output is FALSE. The NOT OR output drives the t and s inputs of a SELECT function which returns the value connected to t or f input, depending on the value of s. If s is FALSE, the function returns the value connected to f.

F 1 Chan Entry

H_ENTRY.VI
\LabviewNT\vla_wsa.llb\CH_ENTRY.VI
ist modified on 5/13/96 at 7:17 AM
inted on 8/4/98 at 2:55 PM

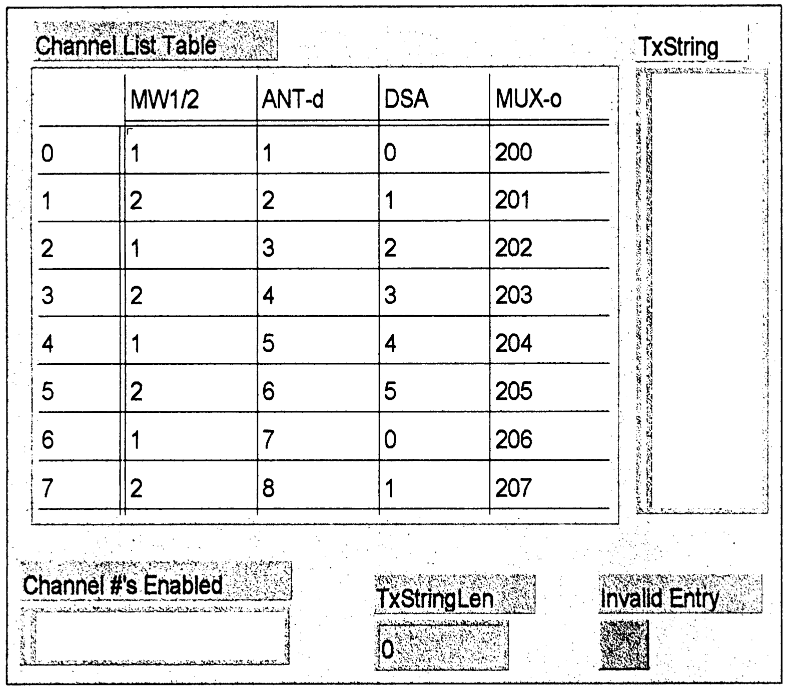
Dave
This embedded into a client VI for passing parameters
[illegible] 8/4/98

onnector Pane



CH_ENTRY.VI

ont Panel



Channel List Table

	MW1/2	ANT-d	DSA	MUX-o
0	1	1	0	200
1	2	2	1	201
2	1	3	2	202
3	2	4	3	203
4	1	5	4	204
5	2	6	5	205
6	1	7	0	206
7	2	8	1	207

TxString

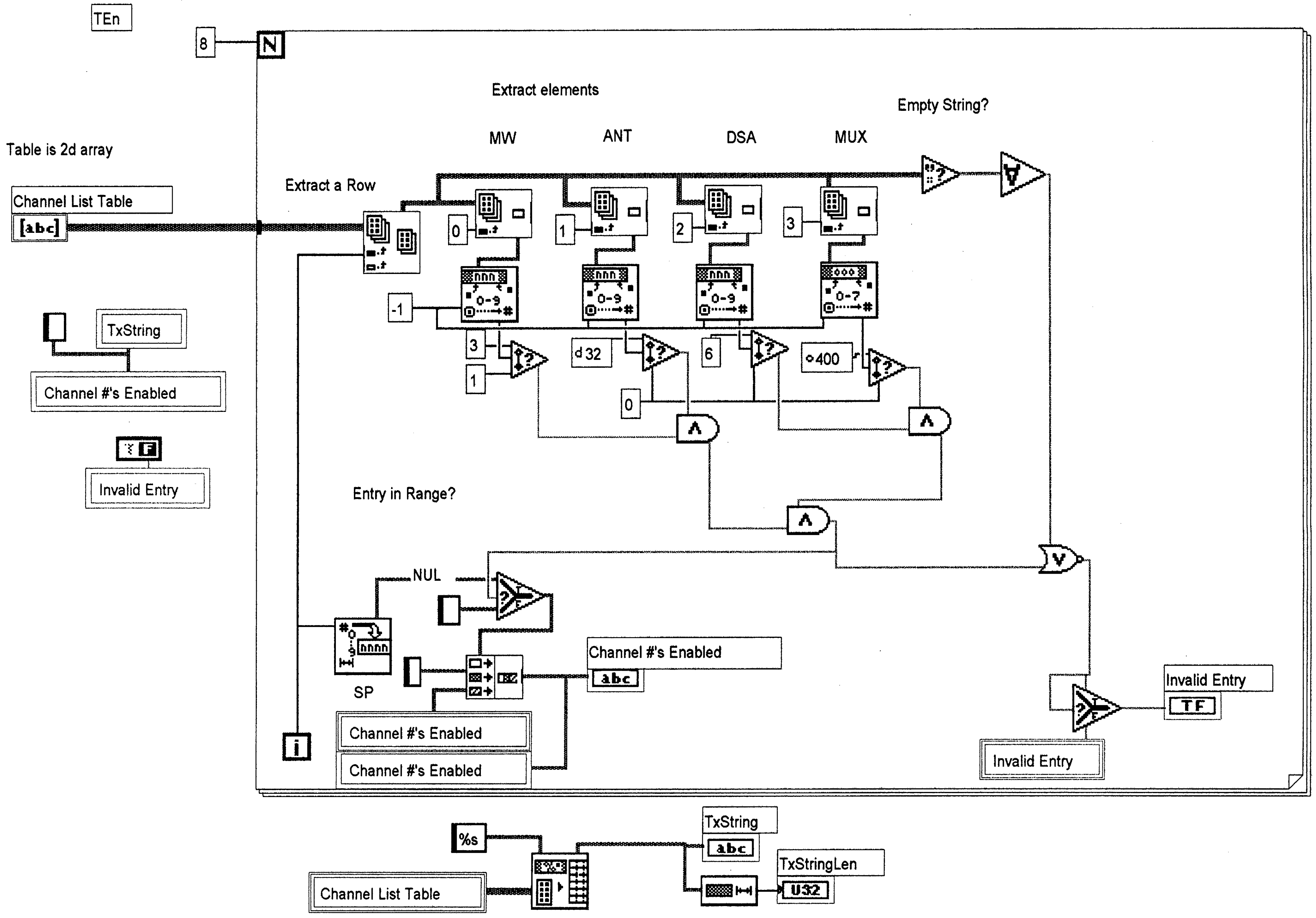
Channel #'s Enabled

TxStringLen
0

Invalid Entry

H_ENTRY.VI
:\LabviewNT\vla_wsa.llb\CH_ENTRY.VI
ast modified on 5/13/96 at 7:17 AM
rinted on 8/4/98 at 2:55 PM

lock Diagram



2.5 SERVER

At this point, it is recommended that the reader refer to Section 2.2 which describes the SERVER-CLIENT message format and protocol and the associated Ethernet-Internet Protocol features.

SERVER VI resides in GARCON and is the functional partner of the CLIENTs in the user's PC; operating in conjunction, they route user-requested monitor data from GARCON to the user's application programs. (See the CLIENT description in Section 2.3.)

The SERVER CONTROL PANEL and BLOCK DIAGRAM follow this text. Although the controls and displays are essential LabVIEW functional elements, the displayed information is primarily useful for diagnostic purposes since SERVER is not normally used as a user interface. Note that the control panel has only two controls: the NetPortNum and the TCP Port ID List. NetPortNum is normally set to Ethernet port 18C4H. The TCP Port ID List control selects two sets of CLIENT information for display: 1) the CLIENT-designated data channel addresses which are displayed in RxMsgTable; and, 2) the associated channel data values which are displayed in TxMsgTable. The selected CLIENT's Remote IP Address is displayed in the Remote IP Address List on the top left of the panel and the TCP Tx Port ID is the currently active CLIENT's Port ID number. Two tables in the lower right portion of the panel display the active CLIENT's Remote IP Addresses and Remote Port Numbers. At the extreme right, a Connection Timeout Counter Array display shows SERVER's timeout count for each of the active CLIENTs. New Command LED indicators flash when SERVER detects a request for service from a CLIENT. The Port ID List Ptr and associated Remote Port Num displays show the state of these variables; the Port ID List Pointer is the control index in the Poll the Connections loop in the Work While Loop (described below). The WaveguideCycleCnt and DataChanNum 0-127 are unused functions.

SERVER Execution

Note that the first page of the SERVER BLOCK DIAGRAM shows that all operations are contained in two concurrently executing, independant While loops within the Main While Loop. Note that these two While loops operate unconditionally; their execution is not dependant upon a variable.

The first is a Listen While Loop (upper part of BD P3) which continuously listens for a TCP Open Connection from a CLIENT via the TCP LISTEN function. A TCP Open Connection is detected when a CLIENT sends a DATA REQUEST Message (described in Section 2.2) to SERVER. The TCP LISTEN function provides the CLIENT's TCP address parameters to the Work While Loop.

The Work While Loop (lower part of BD P3) recurrently outputs the data requested by the CLIENT's DATA REQUEST Message. It obtains this data from GARCON's RAM via the Dynamic Linking Library (DLL, see Section 2.6, SERVER's Monitor Data Retrieval) and outputs it to the CLIENTs at the loop's natural execution rate. (The data from the Monitor and Control Interface Board (see Section 2.9) is stored in the RAM by the Virtual Device Driver, VXD.)

There are several important control states that condition the operations of the two While loops: 1) Listen While Loop: LISTENING, ID IS IN LIST, PORT ID LIST EMPTY and NEW CONNECTION and 2) Work While loop: NEW COMMAND, WORKING, and DISCONNECT. These state's influences are described below.

TCP Functions

A very important aspect of the operation of the two While loops are the program's interactions with the Telecommunication (TCP) Internet Protocol (IP) functions; these are briefly described below. It should be noted that the TCP functions are also LabVIEW Virtual Interfaces.

Note the five TCP Network Connection RefNum icons on the top right of the CONTROL PANEL. These icons resemble a computer monitor and are used when a LabVIEW program opens a network connection in one VI to perform I/O on the network connection in another VI. These SERVER CONTROL PANEL TCP icons are labelled TCP Port ID List, TCP Tx Port ID, TCP Port ID, TCP Open Port ID and TCP Temp. Unlike other LabVIEW Control Panel icons, they are not a control or display in the usual LabVIEW sense because they are not a command source or data sink; they simply designate the program's TCP functions. In contrast, the Block Diagram Refnum connections (enclosing a chopped-corner square) to the Control Panel TCP Refnum icons are literal linkages to the TCP function variables, inputs and outputs. The Refnum is a variable type analagous to the SGL (single-precision floating point) or I8 (8-bit integer).

Some features of these TCP functions deserve special mention and the reader should refer to the BLOCK DIAGRAM for the following list: The TCP LISTEN function, used (only) in the lower left corner of the Listen While Loop (BD P3) creates a listener and waits for an accepted TCP connection at the specified (Input Port). When a listen begins, there must not be a second LISTEN function on the (Input) port. The (Input) PORT is the port that the LISTEN function uses to listen for a connection; in SERVER, it is connected to Ethernet Port 18C4H. The TCP OPEN PORT ID is a network connection reference number that uniquely identifies the TCP connection. It is used as a reference in subsequent calls and is analagous to call number entries in a telephone call log; for example, a telephone call log might have the entries: Call #1 was from XXX; Call #2 was from YYY, etc. In this example, call #N is a call reference and is analagous to OPEN PORT ID. (The LabVIEW V5.0 Help menu calls this function CONNECTION ID.)

The REMOTE IP ADDRESS is the address of the remote machine (i.e, a CLIENT) associated with the TCP connection. (From Section 2.2, remember that the CLIENT's IP addresses are 146.88.60.YY.) REMOTE PORT NUMBER is the port the remote system (a CLIENT) uses for the TCP connection.

The TCP LISTEN has a Timeout input. If a TCP connection (i.e., TCP READ) is not established within the specified time after the LISTEN detects the new connection, TCP LISTEN returns an error to the program on the ERROR OUT function. The wait period is specified in milliseconds. The TCP WRITE also has a timeout input. The TCP CLOSE CONNECTION function does not have a Timeout input.

The TCP READ, WRITE, and CLOSE CONNECTION functions are used only in the Work While Loop. These functions use the TCP address parameters provided by the TCP LISTEN function when it detects an Open Connection; they are: TCP PORT ID, REMOTE IP ADDRESS and REMOTE PORT NUMBER. They are stored in arrays connected to the TCP function's inputs; the array's indices are the TCP PORT ID. The TCP READ and WRITE functions have BYTES TO READ and BYTES WRITTEN connections, respectively, that describe the message length. Data is read out of the TCP READ on the DATA OUT connection and data to be transmitted by the TCP WRITE is connected to the DATA IN connection.

TCP functions have additional inputs and outputs, such as Error In, Error Out, and Port ID Out. (Port ID Out has the same value as the input PORT ID); these are described below in the context of their use in the loops.

Each CLIENT has a timeout count which is incremented in the Listen While Loop. The counters are combined in the Connection Timeout Counter Array.

Initialization

There are a number of variables that must be initialized prior to entry of the Main While Loop. Among these are several important arrays (lower left corner) which contain CLIENT-particuliar TCP functions. These are labelled Network Connections Lists and are TCP address parameter arrays: TCP PORT

ID LIST, REMOTE IP ADDRESS LIST, REMOTE PORT NUMBER LIST and the CONNECTION TIMEOUT COUNTERs. The Indices of all four are initialized to 0. The Element inputs of the Remote Port Number List and Connection Timeout Counter are also set to 0. The Element inputs of the Remote IP Address and TCP Port ID List arrays are initialized to nominal or non-specific values. The Remote IP Address List is initialized to " ", the EMPTY STRING function. The EMPTY STRING function is used because the Remote IP Address List array must not contain a plausible value in 0[0..1] during the first pass of the Listen While Loop. The TCP Port ID List array is initialized to TCP Temp for the same reason, in 0[0..1] it will be searched to see if a new Port ID is already in the Port ID List array.

The Element Inputs of the RxMsgTable and TxMsgTable arrays (that drive the CP's address and data tables) are initialized to a dummy string variable. The RxMsgTable indices are 8 and 8, and the TxMsgTable indices are 8 and 1.

Several other variables are also initialized: the Remote Port Number is initialized to numeric zero and the Remote IP Address is initialized to 000.000.000.000. The Port ID List Pointer (the principle index in the Work While Loop) is initialized to 0. The ACK and EOT functions are also defined by the TO HEXADECIMAL string functions. The control state WORKING, in 0[0..1] in the Work While Loop is intitialized to False and this initialization function has a 150 delay.

The Connection Timeout Counter Array has a second initialization operation: the INITIALIZE ARRAY (two REPLACE ARRAY ELEMENT arrays, lower right side of BD) are initialized to a value of 61 for a period of 61 mS. The index is initialized to 0.

Listen While Loop

The reader should refer to the SERVER BLOCK DIAGRAM (BD P3) and the **TCP Functions** and **Initialization** paragraphs above.

The Listen While Loop's function is to detect new requests for service from inactive CLIENTs and to store the requesting CLIENT's TCP address parameters for use by the Work While Loop. The Ethernet Net Port connection (18C4H) is sensed by the TCP LISTEN function which waits for an accepted TCP connection at its Port input. These detected connections are DATA REQUEST Messages or a disconnect character (EOT) from the CLIENTs. The TCP LISTEN functions outputs are: TCP OPEN PORT ID on the Connection ID output, REMOTE IP ADDRESS on the Remote Address output and REMOTE PORT NUMBER on the Remote Port output. These variables are used in both the Listen While and Work While Loops. The TCP LISTEN function Timeout input is 1000mS; this function becomes significant if the loop's execution time exceeds 1 second.

If a TCP connection has been opened, there are no TCP READ or WRITE errors, and the Timeout period is less than 1 second, the program enters 0[0..1] (BD P3) which sets the NEW CONNECTION state true and starts a 150 mS delay. 1[0..1] also sets NEW CONNECTION true.

Frame 0[0..1], (BD P3) searches the TCP Port ID List array to see if the TCP Open Port ID detected by the TCP LISTEN function (mentioned above) is in the array. This operation is performed by the SEARCH 1D ARRAY function which searches the 1D Array (top input) for the Element (second input) starting at Start Index, (bottom input). The Search Function's output is the index of the element. The Start Index is 0, so the entire array is searched for the presence of the TCP Open Port ID. The NOT EQUAL function on the SEARCH 1D ARRAY output compares the function's output to -1; the output is true if they are unequal. If unequal, the ID IS IN LIST state is set true. The 1D ARRAY SEARCH function's output is also connected to the CP's TEST NUM numeric indicator for diagnostic purposes.

In frame 1[0..1] (BD P4), the Boolean Case Structure tests the state of the ID IS IN LIST variable; if it is false, the three Port Parameters (TCP OPEN PORT ID, REMOTE PORT NUM and REMOTE IP ADDRESS) detected by the LISTEN function are added to the TCP PORT ID LIST, REMOTE PORT NUMBER LIST and REMOTE IP ADDRESS arrays by three BUILD ARRAY functions. The BUILD ARRAY function appends any number of array or element inputs in top-to-bottom order to create an array with appended element(s). The element to be appended is the bottom input; the input array is the top input and the output is the modified array. Since NEW CONNECTION is true, the CONNECTION TIMEOUT COUNTER in that array is set to 0 and appended to the array by a BUILD ARRAY function. Since the connection was just detected, this initialize this counter's value to 0.

Also, in frame 1[0..1], if the ID IS IN LIST variable is false, the Port ID List Ptr (pointer) is incremented; if it rolls over to 0, the PORT ID LIST EMPTY variable is set true.

In Sequence N[0..1], frame 0 (BD P4), the variable LISTENING is set true and the loop waits 100 mS before proceeding to frame 1[0..1].

In this sequence's frame 1[0..1] (BD P3), a For Loop increments the Connection Timeout Counter Array mentioned above. The CLIENT's timeout counts are an important parameter in the Work While Loop. The number of For Loop passes N, is determined by the value of the Port ID List Ptr (pointer) mentioned above. To explain why a For Loop is used to increment the counters, turn back a page or two to SERVER's CONTROL PANEL. The Port ID List Ptr is the row pointer to the three tables of CLIENT IP address parameters and associated timeout count on the lower right. These tables show only the active CLIENT's IP and timeout count parameters. The entries in these tables are top-justified and in the time order of the data requests to SERVER; the first requesting CLIENT's parameters are in the top row, the most recent is at the bottom of the group. Table entries below this group are blank.

The pointer continually sequences through the table; it is the index in the Poll the Connections For Loop. Its value is also displayed in the Port ID List Ptr display just above the Connection Timeout Counter Array table. Using a For Loop to increment only active-CLIENT counters makes the timeout count table's entries logically consistent with the two IP address parameter tables.

Upon completion of the For Loop operation in frame 1[0..1], the Listen While Loop reverts to the TCP LISTEN function for the next loop cycle.

Work While Loop

When this loop starts, the variable WORKING is set true in frame 0[0..1] (BD P4), and a 100 mS delay is started. In 1[0..1] (BD P3), since WORKING is true, the Boolean NEW COMMAND array is initialized to the true state. This array drives the eight New Command LEDs on the CP. The array index is 8, for the 8 CLIENTs.

On BD P3, the Boolean Case Structure (BCS) tests the state of the PORT ID LIST EMPTY variable; if it is false (i.e., the list is not empty) a Poll the Connections For Loop is immediately entered. If the list is empty, control reverts back to the BCS test because none of the CLIENTs have requested service.

Most of the operations in the Work While Loop are done in this Poll the Connections For Loop. The value of the Port ID List Pointer determines the number of passes through the loop.

In 0[0..2] (BD P7) the Port ID List Ptr (pointer) is the Index of the TCP Port ID List Index Array. The value of the Port ID List Ptr returns the element at this index; this is the variable TCP Port ID, the network connection reference number that uniquely identifies the TCP connection established by the CLIENT.

In 1[0..2] (BD P7) a TCP READ function reads the number of bytes in the DATA REQUEST Message. TCP Port ID (e.g. CLIENT's network connection reference number) is the Connection ID input to the function. The Bytes to Read input is set to 4 (remember from Section 2.2 that the HEADER is four characters and indicates the number of bytes in the impending DATA REQUEST Message). The FROM HEXADECIMAL function converts the Hex string value to a numeric value variable, RxMsgLen. This value is tested by the NOT EQUAL TO 0? function and is also displayed on the CP in the numeric RxMsgLen indicator. If the function's output is true, the NEW COMMAND state is set and is input to the element input of a REPLACE ARRAY ELEMENT array which has the Port ID List Ptr as index. What this does is set a true state in the NEW COMMAND array. This array drives the CP's New Command LEDs; therefore the associated CLIENT's LED will flash. If the RxMsgLen is zero, control reverts back to the start of the For Loop.

In 2[0..2] (BD P3), if the NEW COMMAND state is true, the program proceeds to Sequence F, N[0..5]; if false, control reverts back to the start of the For Loop.

In 0[0..5] (BD P4) a TCP READ function reads the contents of the DATA REQUEST Message. Again the TCP Port ID is the function's Connection ID. The Bytes to Read input is RxMxgLen from 1[0..2] and the output value is the string RxMsgString. This string variable contains the address components requested by the CLIENT and will be used to select the data requested by the CLIENT. It is also displayed in the RxMsgString indicator on the CP for diagnostic purposes. Logic functions test this string to see if it is a disconnect code EOT (\04) instead of the DATA REQUEST Message. A STRING LENGTH function determines the length of the RxMsgString; its length output is compared with 1 by an EQUAL function; if true, the string is a single character. The string value is tested by another EQUAL function; if it is equal to \04, the function's output is true. These two EQUAL function outputs are input to an AND function; if its output is true, the RxMsgString is the disconnect code, EOT (\04) and the DISCONNECT state is set true. RxMsgLen is also displayed on the CP for diagnostic purposes. The TCP READ's Error Out connection goes to a SIMPLE ERROR Handler with a 0 on the Error Code input. The function's Error Code output is connected to a NOT = 0 function; in the event of an error, this becomes true.

In 1[0..5] (BD P4) the RxMsgString variable is input to a SPREADSHEET STRING TO ARRAY function which converts the Spreadsheet String to a string array. The function's Format String input is %x, which defines the string components as hexadecimal integers. The function's output is the variable RxMxgTable.

In 2[0..5] (BD P5) an 8-pass For Loop obtains the CLIENT's eight data values from the PC's RAM and formats them for transmission to the CLIENT. During each pass of the loop, in sequence, each data channel's address is input as a function call to the Dynamic Linking Library which reads the designated data from the PC's RAM. The data is then formatted for output to the CLIENT. These operations are performed by three-frame Sequence in the For loop. Upon entry to 2[0..5] a BCS tests the state of DISCONNECT; if DISCONNECT is false, the For loop is entered; if true, control reverts back to the start of the Poll the Connections For Loop at the start of the Work While Loop. This increments the Port ID List Pointer and the loop starts another pass through the loop.

In 0[0..2] (DB P5) the address components are extracted from the RxMsgTable, a string array. Remember that the DATA REQUEST Message lists eight set of address components (see the message format in Section 2.2). During each pass through the For loop, the index points to a set of address components in this string array; the first pass accesses the first set of addresses, the second pass accesses the second set, etc. The extraction code consists of four sets of multi-dimensional Index Arrays with RxMsgTable connected to the array inputs. The loop index selects the channel address; a second index selects components of the channel address. As shown on the Block Diagram, Index value 0 selects the MW 1/2, Index value 1 selects the AntAddr, and so on to the Mux Addr. These numeric variables are then converted to numeric values by a set of string-to-numeric converters. The FROM OCTAL function converts the Mux addr, the FROM DECIMAL function converts the DSA, etc. These four sets of values are displayed in the numeric indicators on the left edge of the CP.

In 1[0..5] (BD P5), these four sets of address components are input to a CODE INTERFACE NODE (CIN) function. This function is a general LabVIEW method for calling C code from the Block Diagram Graphic code. The return value for the CIN returns to the right terminal of the top pair of terminals; in this application it is the DCS DigitalData designated by the four address components and is read from the computer's RAM. Each additional pair of terminals corresponds to a parameter in the functions parameter list. A value is passed to the CIN by wiring to the left terminal of a terminal pair. The corresponding returned value is read on the right terminal (not used in this application). This CIN calls dcsdlla.dll, a C language dynamic linking library program (dcsdlla.c) which calls the function dcsdata(DcsAddr,DataSetAddr,MuxAddr,MwNum) in the DLL. The CIN function parameters are formed into a multidimensional array address. Monitor data that has been stored in a multidimensional array in GARCON's RAM memory by the VXD (Virtual Device Driver) is accessed by calling the dcsdata function. The array dimensions are the arguments DcsAddr, DataSetAddr, etc. in the function call. The function's input and output data types are shown in the input and output terminals. The four address components (AntAddr, DataSetAddr, etc.) are all unsigned 16-bit integers. The output data, DCS DigitalData is unsigned 32-bit integer. The DLL Name, Function and Format String types are string types. CIN labels specify functional properties. The DLL Name label is the program call dcsdlla.dll; the Function Call is dcsdata. The Format String label label specifies the standard LabVIEW string format which consists of four bytes for length information, followed by string data.

The CIN's output, DCS DigitalData is used in next used in 2[0..2] (BD P5) and displayed on the CP in the DCS DigitalData numeric display.

The Return Value Label and its name, Exec Func Template with the numeric argument 0, are a Windows calling convention.

The dcsdlla.c and VXD programs are described in Section 2.6.

The third frame in the For loop is 2[0..2] (BD P3). This frame converts the numeric data (DCS DigitalData) to hexadecimal string format using the TO HEXADECIMAL function which converts the numeric input to a hexadecimal digit string. The 8 on the function's Width input determines the string width so that each data value is eight digits. The converter's output is connected to the New Element input of a REPLACE ARRAY ELEMENT array. This array's variable is TxMsgTable and the eight data values are displayed on the CP's TxMsgTable indicator. The array's first index is the For loop index and the second index is 0, this the WCC value which is not used.

After completion of the operations in the eight-pass For loop described above, the next operation is transmission of this data to the CLIENT; this is done in 3[0..5], BD P6.

In 3[0..5] (BD P6) the eight data values for the CLIENT (TxMxagTable) are input to an ARRAY TO SPREADSHEET STRING function for transmission to the CLIENT. The %x input on the Format String input specifies that the output is to be in hexadecimal integer format. The function's output is connected to the False terminal of a SELECT function. The ACK code (\06) is connected to True terminal. The Select terminal is driven by the DISCONNECT state; if DISCONNECT is true, the ACK code will be output to the CLIENT which is an acknowledgement that SERVER has responded to the Disconnect code, EOT (\04) detected in Frame 0[0..5]. If DISCONNECT is false, the spreadsheet format data is passed by the SELECT function. This output is TxDCS Data String which is displayed on the CP in the numeric indicator with the same name.

Since transmissions between SERVER and CLIENT are always preceded by a four-byte value that specifies the length of the impending message, TxDCS Data Sting is input to a STRING LENGTH function which returns the value. This value is next input to a TO HEXADECIMAL function which converts the value to a hexadecimal string. The function's Width input is 4 which designates the width of the output string. The conversion function's output is Tx DCS Data Len which is displayed on the CP in the indicator with that name and input to a TCP WRITE function. The TCP Port ID designates the CLIENT's port number.

In 4[0..5] (BD P6) the TxDCS Data String (or ACK) from 3[0..5] is input to a TCP WRITE function for transmission to the CLIENT. The CLIENT's TCP Port ID designates the CLIENT's port number.

In 5[0..5] there two Binary Case Structures (BCS) that determine whether SERVER will continue data routing to a CLIENT or disconnect this service. Remember that the DISCONNECT state can be set true for either of two reasons: the receipt of a disconnect command, ACK (\06) from a CLIENT, or a connection time-out count that exceeds 60 seconds. The two BCS's are on BD P3 and BD P6, respectively, and perform identical operations. Sequence Diagram SH 4 briefly outlines the operations of the two BCSs. Refer to 5[0..5] of Sequence F. The BCS on BD P3 is operative when the CLIENT sends a disconnect (ACK) to SERVER; the BCS on BD P6 is operative when there has been a Connection Timeout Count greater than 60 seconds. The CLIENT's PC could have had some failure such as power loss; this would probably inhibit the CLIENT's ability to transmit the disconnect code to SERVER. For this reason, a timeout count in SERVER is the best way to decide that data transmission to a CLIENT should be terminated.

This disconnect operation involves removing the CLIENT's address parameters from the TCP Port ID List, the Remote IP Address List, the Remote Port Number List and the timeout count from the Connection Timeout Counter Array. The Work While Loop's index is the Port ID List Pointer; this points to a row position in these CP tables. Remember that a CLIENT's address parameters are not assigned to fixed locations in these arrays; the positions are determined by the time order of the various CLIENT's requests for data routing service or disconnection of this service. Since a CLIENT's address parameters may be in any row, the pointer value is not a direct reference to the CLIENT's address parameters in these arrays. For this reason, the Port ID List array is searched to determine the disconnecting CLIENT's Port ID Number position in the array. The CLIENT's other address parameters are at the same position in these other arrays.

The mechanics of this process involves locating the Port ID Number in the Port ID List array, splitting the array into two parts so that the Port ID Number to be eliminated is at the bottom of the upper part. It is then reshaped to remove the Port ID Number at the bottom and then the two parts are recombined. A SEARCH 1D ARRAY function searches for the element in the TCP Port ID List array that matches the to-be-removed TCP Port ID at its Element Input. The search function's output is the pointer to this element. Code elements for removal of the designated Port ID Number from the Port ID List are replicated for the three other arrays and the SEARCH function's element index also points to the CLIENT's other address parameters in the Remote IP Address, Remote Port Number and Connection Timeout Counter arrays.

This index is incremented and input to the Start Index input of four SPLIT 1D ARRAY functions. This function divides the array at the index and returns two portions: the top output is the first subarray and the bottom output is the second subarray. Since the index is incremented, the to-be-removed Port ID Number is at the bottom of the top (first) subarray. The top subarray is input to two array functions: the first is an ARRAY SIZE function which returns the number of elements in the array. This value is decremented and input to the Dimension Size input of a RESHAPE ARRAY function connected to the top output of the SPLIT 1D ARRAY function. The output of the RESHAPE ARRAY function is thus the top array with the designated Port ID Number removed. The bottom output of the SPLIT 1D ARRAY function and the modified array out of the RESHAPE ARRAY are recombined in a BUILD ARRAY function which appends elements in top-to-bottom order to create a single array. The modified array is input to the Array input and the bottom array is connected to the Element input.

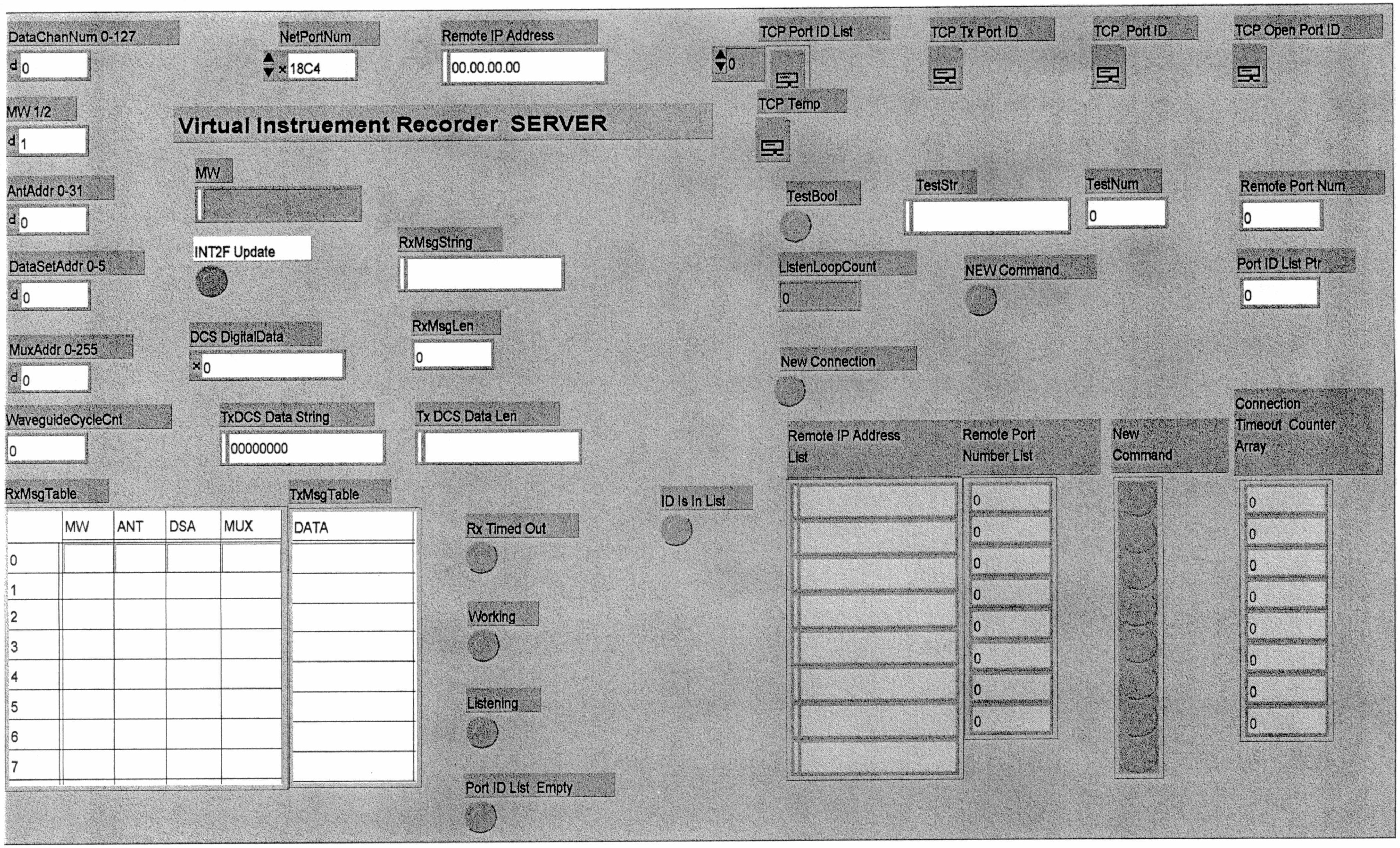
After removing the address and timeout count parameters from the arrays, the CLIENT is disconnected from SERVER by the TCP CLOSE CONNECTION function. The TCP Port ID number is connected to the Connection ID input.

In the two BCSs, in addition to the operation of removing the address and timeout count parameters from the arrays as described above, the Port ID List Ptr (pointer) is decremented and tested to see if it is 0. If it is 0, the CP's Port ID List Empty LED is activated.

The two BCSs differ in the way that they are activated; the BCS on BD P3 responds to the disconnect code (ACK) from a CLIENT case. The BCS on BD P6 implements the disconnect based upon the connection timeout count case. Referring to BD P6 and SD SH 4, note that this code is the alternate case when DISCONNECT is false on P3. Inside this false case, is an INDEX ARRAY for the Connection Timeout Counter Array. The Port ID List Ptr is the index input which causes the array to return the CLIENT's timeout count. This is compared with 60 by a GREATER function; if the function's output is true, an inner BCS is entered and the CLIENT address parameter and timeout count removal operations described above are performed. If the timeout count is less than 60, the loop control reverts back to the start of the Poll the Connections For Loop on BD P3.

F ? DATA SERVER

IR Multiple Connect Server1.vi
:\GARCON\Virservr.llb\VIR Multiple Connect Server1.vi
ast modified on 1/24/96 at 9:54 AM
rinted on 8/3/98 at 2:28 PM



Virtual Instruement Recorder SERVER

DataChanNum 0-127: 0
MW 1/2: 1
AntAddr 0-31: 0
DataSetAddr 0-5: 0
MuxAddr 0-255: 0
WaveguideCycleCnt: 0
NetPortNum: 18C4
Remote IP Address: 00.00.00.00
MW:
INT2F Update
RxMsgString:
DCS DigitalData: 0
RxMsgLen: 0
TxDCS Data String: 00000000
Tx DCS Data Len:
TCP Port ID List: 0
TCP Tx Port ID
TCP Port ID
TCP Open Port ID
TCP Temp
TestBool
TestStr:
TestNum: 0
Remote Port Num: 0
ListenLoopCount: 0
NEW Command
Port ID List Ptr: 0
New Connection
ID Is In List
Rx Timed Out
Working
Listening
Port ID List Empty

RxMsgTable

	MW	ANT	DSA	MUX
0				
1				
2				
3				
4				
5				
6				
7				

TxMsgTable

DATA

Remote IP Address List	Remote Port Number List	New Command	Connection Timeout Counter Array
	0		0
	0		0
	0		0
	0		0
	0		0
	0		0
	0		0
	0		0

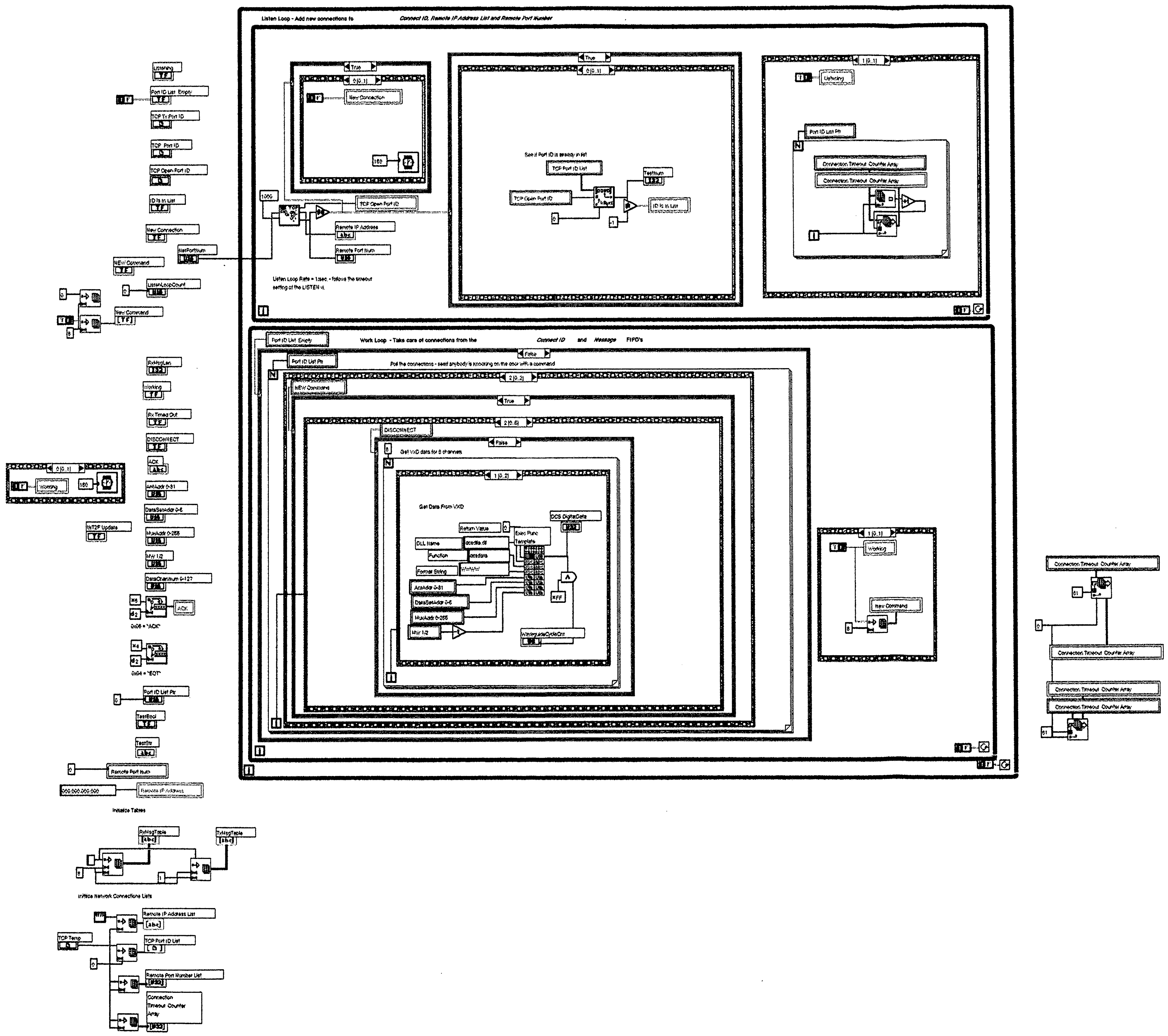
R Multiple Connect Server1.vi

:\GARCON\Labview\Vir_serv.llb\VIR Multiple Connect Server1.vi

ast modified on 6/27/96 at 11:34 AM

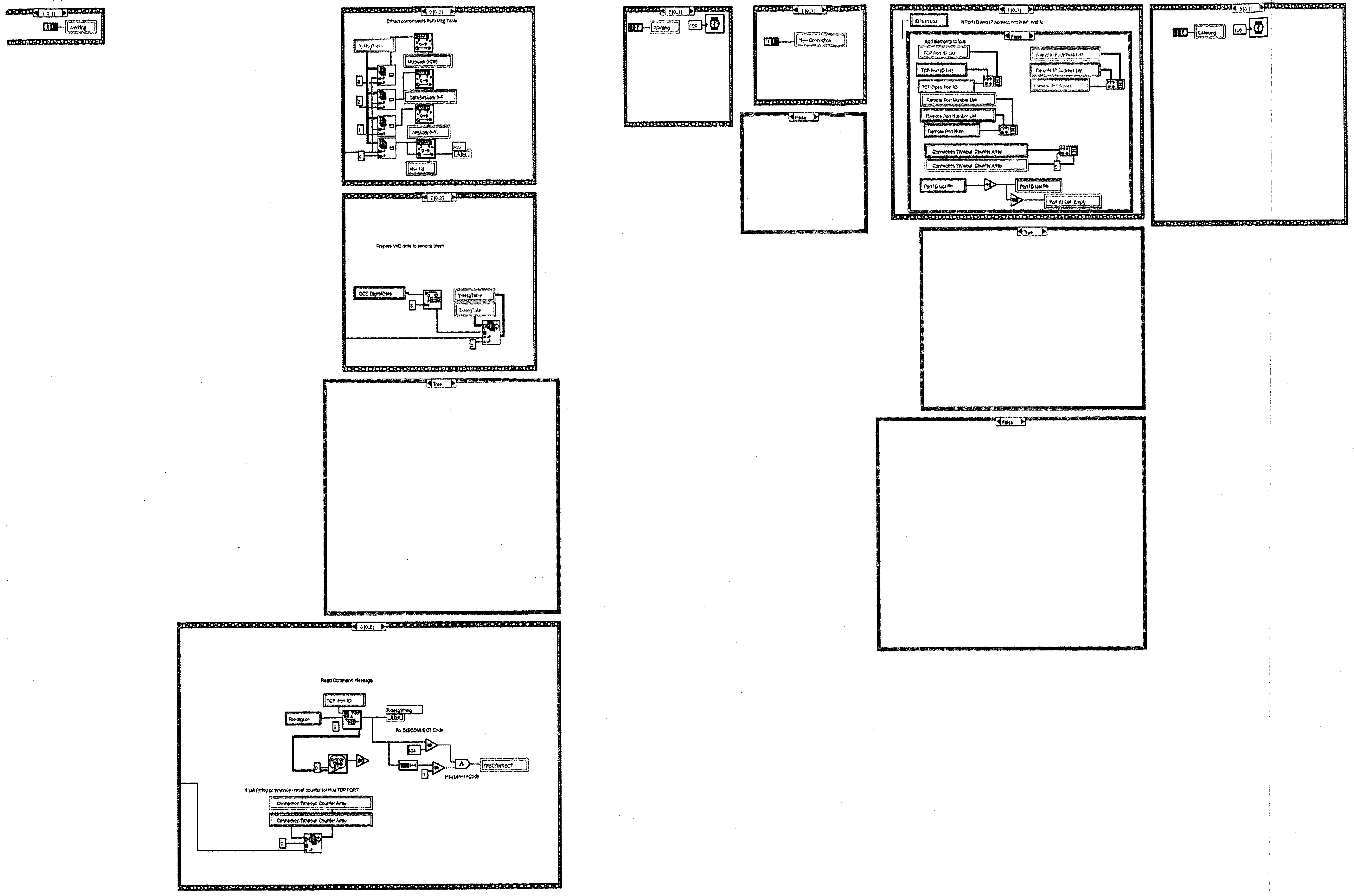
inted on 5/3/99 at 3:00 PM

lock Diagram



R Multiple Connect Server1.vi
:\GARCON\Labview\Vir_serv.llb\VIR Multiple Connect Server1.vi
st modified on 6/27/96 at 11:34 AM
inted on 5/3/99 at 3:00 PM

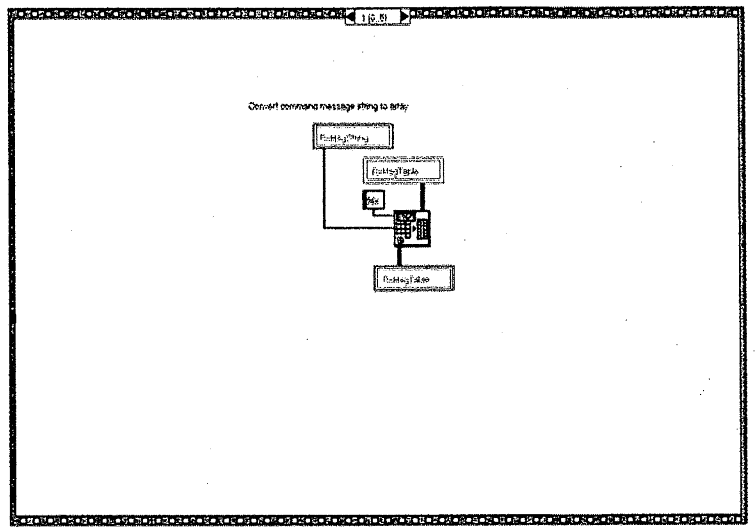
DATA
SERVER

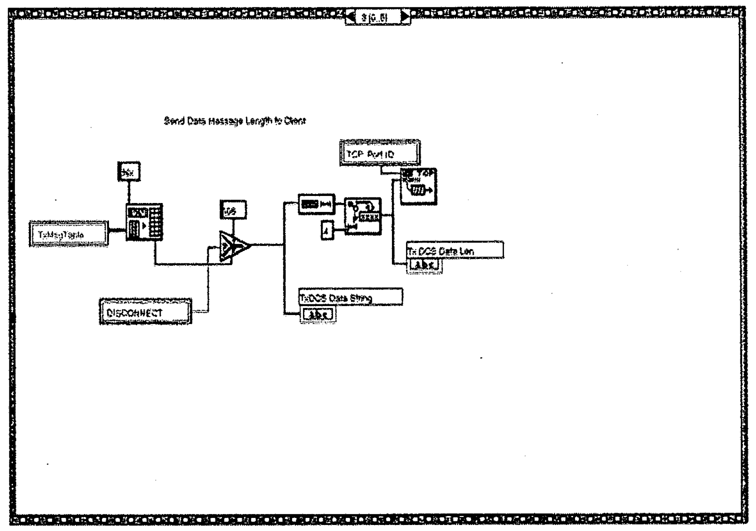


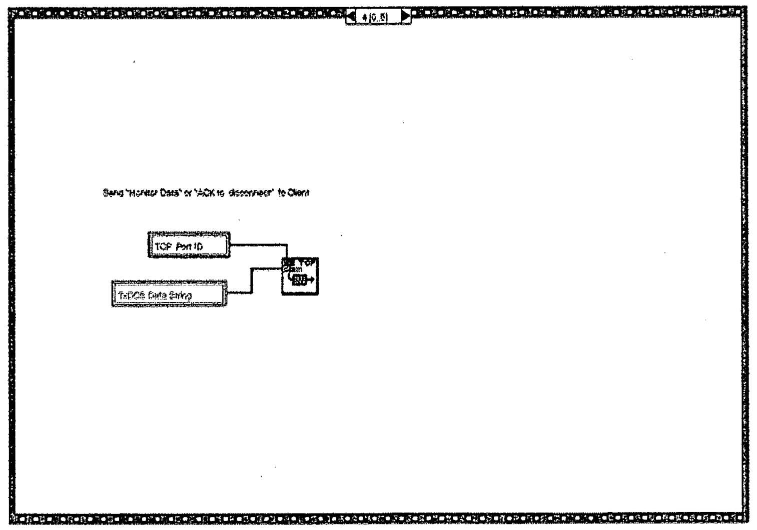
Extract components from Msg Table
Prepare VxD data to send to client
Add elements to lists
If Port ID and IP address not in list, add to
Connection Timeout Counter Array
Read Command Message
TCP Port ID
Rx DISCONNECT Code
DISCONNECT
If still fixing commands - reset counter for that TCP PORT

DATA SERVER

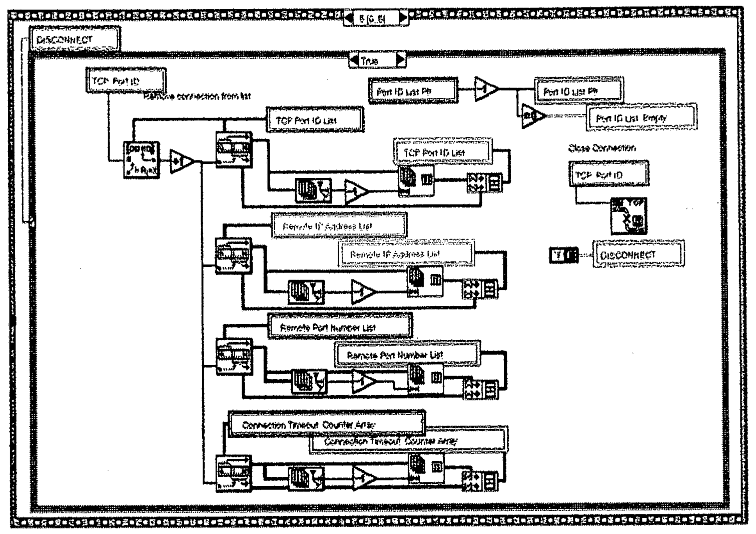
IR Multiple Connect Server1.vi
:\GARCON\Labview\Vir_serv.llb\VIR Multiple Connect Server1.vi
ast modified on 6/27/96 at 11:34 AM
inted on 5/3/99 at 3:01 PM

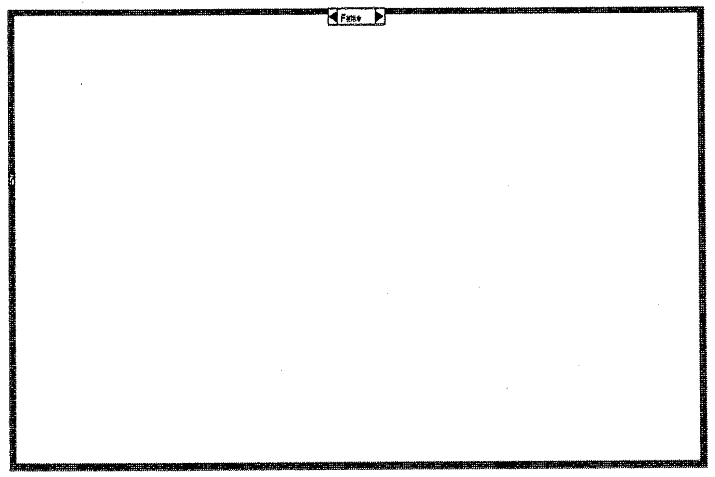


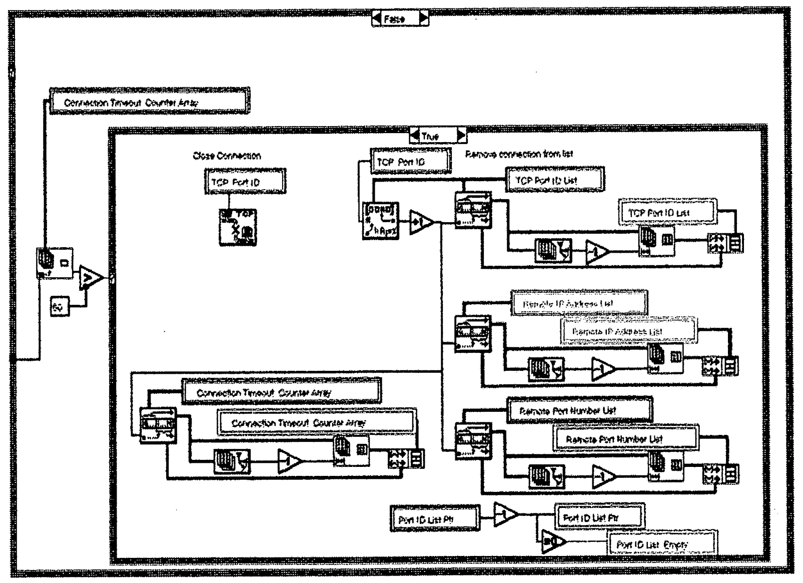




R Multiple Connect Server1.vi
:\GARCON\Labview\Vir_serv.llb\VIR Multiple Connect Server1.vi
ast modified on 6/27/98 at 11:34 AM
inted on 5/3/99 at 3:01 PM

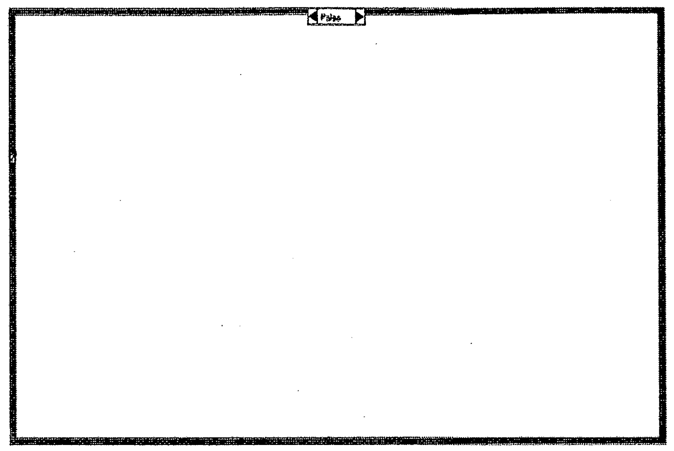


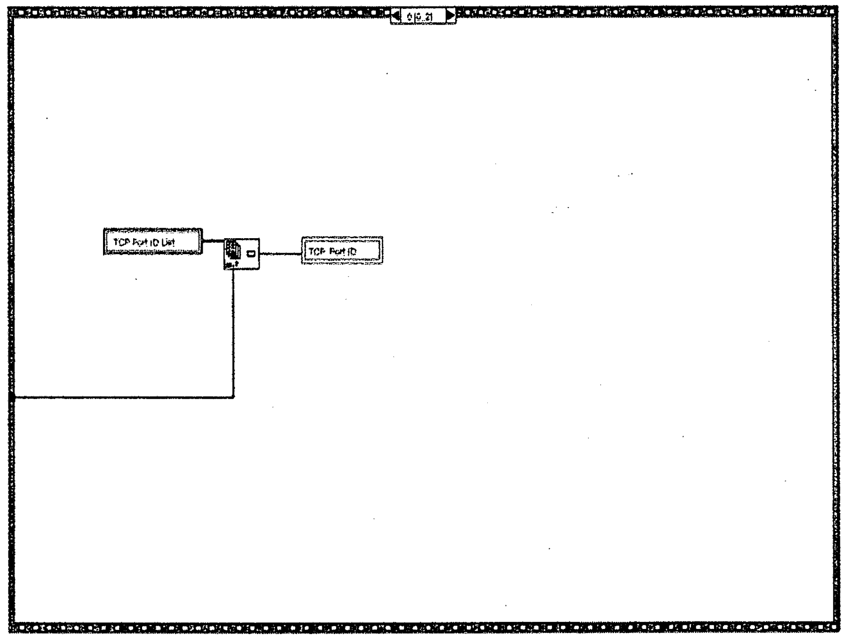


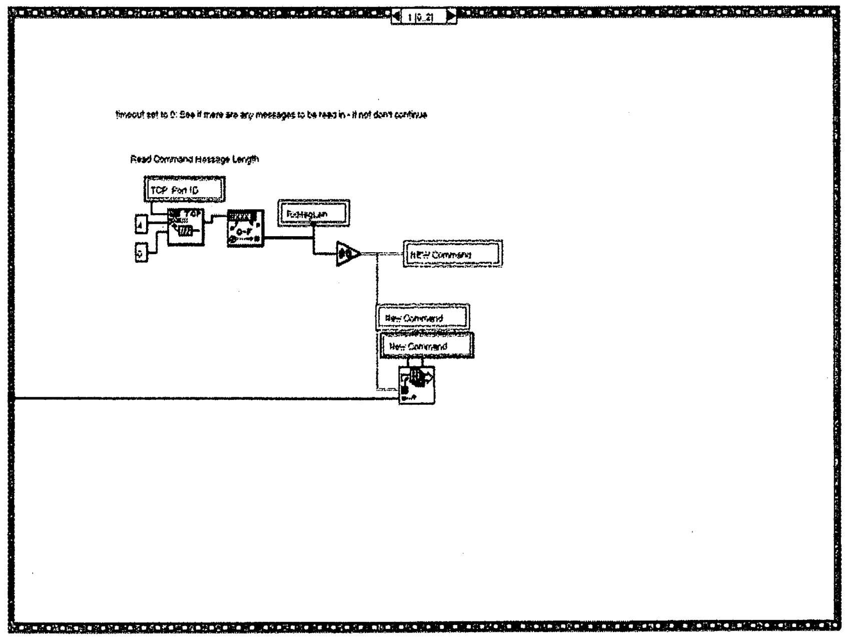


DATA SERVER

R ultiple Connect Server1.vi
\GARCON\Labview\Vir_serv.llb\VIR Multiple Connect Server1.vi
ast modified on 6/27/96 at 11:34 AM
in d on 5/3/99 at 3:01 PM






1 [0..2]
timeout set to 0: See if there are any messages to be read in - if not don't continue
Read Command Message Length
TCP Port ID
4
0
RdMsgLen
NEW Command
New Command
New Command

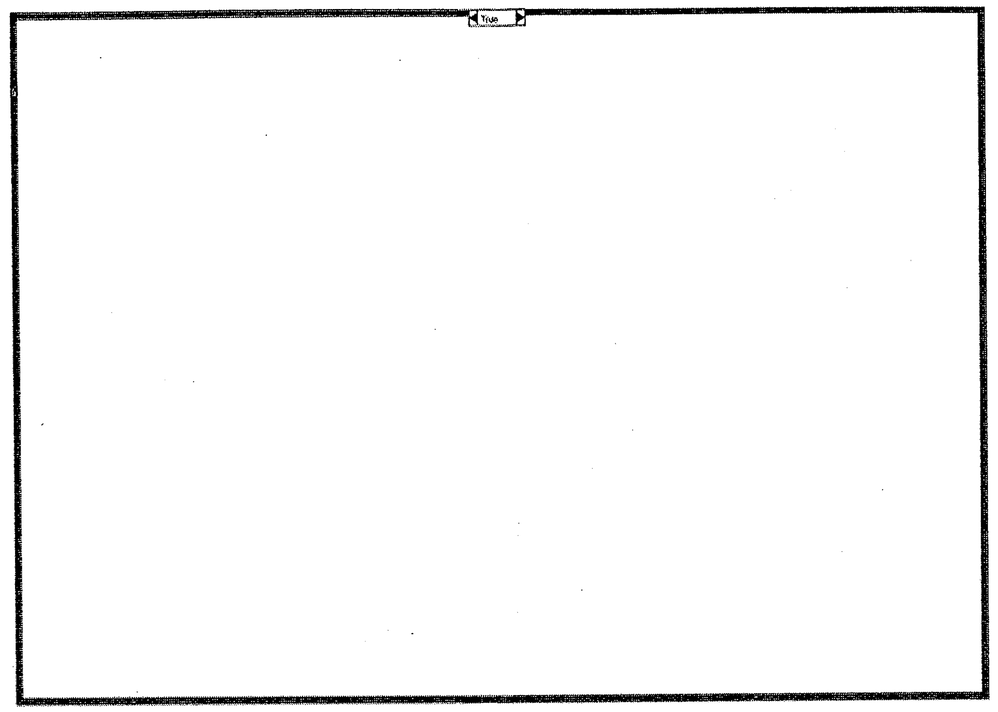
DATA
SERVER

lultiple Connect Server1.vi

RCON\Labview\Vir_serv.llb\VIR Multiple Connect Server1.vi

st modified on 6/27/96 at 11:34 AM

d on 5/3/99 at 3:01 PM



2.6 SERVER'S MONITOR DATA RETRIEVAL

This section describes the operation of the software and hardware components that SERVER uses to obtain CLIENT monitor data. Figure 2 (next page) is a block diagram showing these components and the major linkages between them. They consist of a Dynamic Linking Library (DLL), a Virtual Device Driver (VXD), a Monitor Data Interface (MDI) and a multidimensional Monitor Data Array in computer RAM. SERVER's Frame 1[0..2] Code Interface Node (CIN) is the address-data linkage to the DLL. See Section 2.5 for a description of SERVER's operations.

Note that the block diagram shows that the CIN uses LabVIEW DOWNSHIFT.VI to pass the addresses and data; it's operations are transparent to the CIN and other components. Since DOWNSHIFT is an internal LabVIEW function and was not particularized for the SERVER application, it's operations are not discussed here.

To SERVER's CIN, the Dynamic Linking Library literally looks like a VLA monitor data library. The CIN obtains the CLIENT's monitor data, one 24-bit data word at a time, by passing four address arguments to the DLL; these arguments designate the requested data. The DLL obtains this data (plus an 8-bit WCC count) from the data array in RAM via the VXD and returns it to the CIN. The four CIN address components are: Antenna Address (also called the DCS Address), Data Set Address, Multiplex Address and the MW1/2 designator.

As shown on the block diagram, the VXD performs two hardware driving functions: 1) In response to an IRQ2F by the DLL, the VXD obtains the requested 24-bit data word from the computer's RAM and returns it to the DLL. 2) At the end of each 19.2 Hz VLA cycle, in response to an IRQ5 generated by the MDI, the VXD reads the previous cycle's set of 384 monitor data messages from the MDI and stores them in the RAM's monitor data array. These messages were polled from the Central Buffers by the SLC during the previous 19.2 Hz VLA machine cycle. (See Appendix A for M&C System details.) These two VXD operations are independent and asynchronous but there are no time conflicts because both functions are interrupt-driven.

Figure 3 (following Figure 2) depicts the data retrieval process through the DLL-VXD software: The process starts with the SERVER-CIN's function call to the DLL with the data address; the DLL activates INT2F which calls the assembly language part of VXD to service the interrupt; this in turn calls the "C" language part of VXD which obtains the requested data from the data array in RAM. The VXD then returns the data to the DLL which completes the function call by a data return to the SERVER-CIN. As shown in the figure, registers convey the address and data through the DLL-VXD forward and return paths.

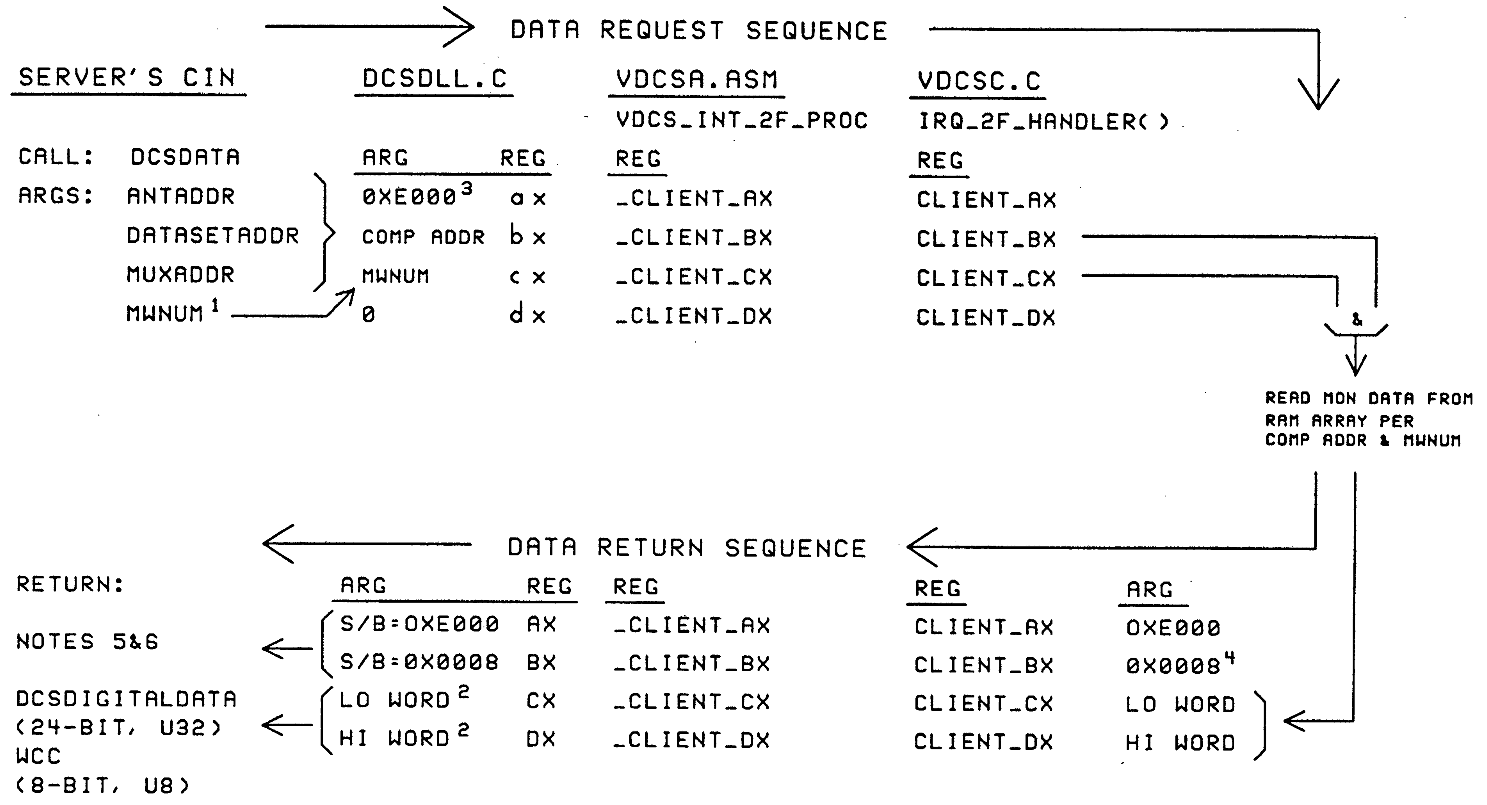
Dynamic Linking Library (DLL) Description

The DLL was written and compiled in Borland C, version 4.02. The current DLL version is DCSDLLA.C, 3/7/95, P. Dooley. During this DLL description, refer to the DCSDLLA.C listing which follows this section's text.

The #include <windows.h) and #include <dos.h> directives search for these files in the special directory for included files. Data from these files may be referenced by the program. The #include "dcsdef.h" directive extends the search to involve the current directory before searching the include directory. The dcsdef.h listing is included with the DLL and VXD listings at the end of this section's text.

The DLL starts with a call to function **dcsdata**. The first parameter of the call is the return type DWORD, which specifies that the returned data value is a double word (unsigned 32-bit value). The next parameter is **FAR PASCAL** which specifies that the function uses the FAR PASCAL method of passing

FIGURE 3, DATA TRANSFERS THROUGH DLL-VXD



NOTES:

1) MWNUM SELECTS EITHER MW1 (WNUM=0) OR MW1 (MWNUM=1) FOR RETURNED VALUE
2) HI WORD IS 16 MSB OF DATA VALUE & LO WORD IS 8 LSB OF DATA VALUE + 8 WCC BITS
3) 0XE000 LOADED IN ax BY DCSDLL.C IS MUX_HANDLE_BASE & IS RETURNED BY IRQ_2F_HANDLER( )
4) IRQ_2F_HANDLER( ) LOADS REG CLIENT_BX WITH 0X0008; THIS IS INT_2F_RESPOND
5) IN DCSDLL.C IF UPON RETURN CONTENTS OF AX & BX ARE BOTH CORRECT, 24-BIT DATA VALUE (U32) AND 8-BIT WCC (U8) ARE RETURNED TO SERVER'S CIN
6) IN DCSDLL.C IF UPON RETURN, CONDITION OF 5) IS NOT MET, RETURNED VALUE IS 0XBAD2000L

that specified for the program, the **return** statement is executed. This returns a 32-bit fault code, 0XBAD10000L to the CIN.

To verify that the three linked sections of the monitor data retrieval code (DLL, Vdcs_Int_2F_Proc in VDCSA.ASM and Irq2F_Handler in VDCSC.C) are executing properly, the AX and BX registers are loaded with test constants, and then tested after passage through the three pieces of code. In DCSDLL.C, RegAX is loaded with MUX_HANDLE_BASE (0xE000 from DCSDEF.H); this value is then passed through Vdcs_Int_2F_Proc to Irq2F_Handler and then returned to DCSDLL.C (via the same pieces of code) where it is tested in the **dcsdata** function's return (described below). This is a round-trip test. The second test value is the INTERRUPT_2F_RESPOND (0X0008 from DCSDEF.H) that is loaded into RegBX by Irq2F_Handler; this enables verification of RegBX's Irq2F_Handler to DCSDLL.C return path. RegBX's contents are tested in the **dcsdata** function's return statement, described below.

The next group of instructions are the **dcsdata** function's normal (error-free) data return. The logic operations in the **return**'s double parenthesis test the states of RegAX and RegBX to verify proper operation in the sections of linked code; this was mentioned above. In the first parenthesis, the contents of RegAX is Bitwise ANDed (by the **&** operator) with MUX_HANDLE_BASE; if there is a mismatch, something is wrong. The result is a Boolean true or false. The second parenthesis is a similar test of the contents of BX, which was loaded with INTERRUPT_2F_RESPOND in Irq2F_Handler(); again the result is a Boolean true or false. The two Boolean products are compared by a Logical AND (the **&&** operator); if the result is true (identical values), the dcsdata function's **return** exit is executed which carries the function's data to the CIN. Just before the return, RegDX (Hi Word in VDCSC.C) is loaded into LongTemp and left-shifted 16 bits, 0s are shifted into the LSB. The **return** instruction loads LongTemp with the Bitwise OR of CX, (Lo Word in VDCSC.C); this places both 12-bit data values and the 8-bit WaveGuideTick (WCC count) in the 32-bit data value returned to the CIN.

If the result of the Logical AND (described above) is false, the VXD did not respond to the int2fh call and the **return** following the **else** statement is executed. In this case the return carries the fault code 0XBAD20000L. The 0XBAD20000L and 0BAD10000L fault codes indicate a DLL execution fault to SERVER.

There are two additional function calls (LibMain and WEP) in the DLL that are not directly related to the CIN data retrieval function. They perform functions required by Windows in DLL applications. Since they are standardized and unrelated to the CIN function, they are not described here.

Monitor Data Array in RAM

Since the VXD stores and retrieves monitor data from the multidimensional array in computer RAM, it may be useful to describe the array structure. Note the array declaration on page 1 of the VDCSC.C listing. All array elements are WORD (unsigned 16-bit) values). The "fixed-array" parameters are enclosed by brackets that define the number of parameter elements. Three of these parameters correspond to addresses: DCS (Antenna), Data Set and Multiplex. Although only 28 of the 32 possible DCS addresses are used and there are unimplemented multiplex addresses, the array is sized to accommodate the ultimate address capacity. This is: 32 x 6 x 256, or 49,152 addresses (actually, only 192 addresses are used for monitor data; the remaining 64 are used for commands). When the Irq5_Handler() stores data in the array, it is stored in the location corresponding to the composite address components.

Note that below the array's multiplex address parameter, there are two additional two-element parameters; the first is labelled "2 Monitor Words" by the comment and the second is labelled "2 Data Words." The "2 Monitor Words" parameter is a 32-bit **Monitor Word 1** value; the second parameter is a 32-bit **Monitor Word 2** value. Thus for each DSA-Data Set-Multiplex Address state (32 x 6 x 256), both types of monitor data words are stored in the array. The paragraph below describes the two types.

A brief digression here: Monitor Word 1's multiplex address sequence is determined by a table in a Data Set EPROM; Monitor Word 2's multiplex address is either a sequential scan through all addresses or it can be command-set to any address so that it samples the same data channel every VLA cycle. The fixed address mode is useful for intensive sampling purposes. The multiplex address uniquely designates a single analog or digital data channel; the only difference between Monitor Word 1 and Monitor Word 2 data are the address sampling schedules and sampling times; Monitor Word 2 is sampled 700 μs after Monitor Word 1. Why the emphasis here upon the differences since both types refer to the same signal? The Irq5_Handler() described below stores both types in different elements of the same array address; some application programs designate either (but not both) type for processing. A user may elect to command a data channel to the intensive sampling mode for higher time resolution in an application program.

Below the table statement described above is a second table statement with a different order; this table is not used and is "commented out" by the \\ prefix.

The keyword **extern** defines four WORD register pseudovariables, Client_AX,..Client_Dx that enables arguments to be exchanged between VDCSC.C and the assembly program VDCSA.ASM. In VDCSA.ASM the _Client_AX, etc., are declared to be global variables by a PUBLIC declaration.

The **Byte The_Shell_message_Proc[12]** and **extern Shell_Message_Proc** are artifacts of a call to debugging code at the end of the listing. This procedure was used during program development to check argument transfers in the Client_AX, etc., registers and is no longer used. Portions of this procedure have been removed from VDCSC.C.

The **BYTE** variable **ParityByte** is not used and it is an artifact of a function that was not implemented.

Irq5_Handler()

This function is called by VDCS_HW_INT in VDCSA.ASM in response to interrupt IRQ5 from the MDI which signals that the most recent VLA cycle's monitor data is available in the MDI's FIFO memory. This function reads the 384 monitor data messages (five bytes) stored in the FIFOs and stores the message's 24-bit data component in the array's address-designated Monitor Word 1 or Monitor Word 2 locations. When the storage operation is completed, program control returns to Vdcs_Hw_Int. The **void** return type in the Irq5_Handler() function call specifies that the function does not return a value; it also means that the function does not have to use a **return** statement when the function has completed its operations. Since this is a called function, there is not a **main** statement.

The **WORD i,j,InWord0,InWord1,** etc., statement defines 16-bit operands; InWord0 through InWord3 are four values read from the FIFO memory. **InWord0** is the composite message address (DCS, DSA and Mux); **Inword1** is the upper two message data bytes; **InWord2** is the lowest data byte, read out in the word's upper byte; the lower byte is null. **InWord3** is the MDI FIFO Status Register; an encoded value indicates completion of the FIFO read operation. The Antenna (DCS), MuxAddr and Dsa operands are the familiar address parameters. The **MwCtr** state is an index to the **MwTable** described above; **Mw** is the value read from the table (0 or 1) and determines whether the message's data is stored in the array's MW1 or MW2 elements.

Ignoring for the moment the WaveGuideTick statements, the data storage while loop starts with the **while** ((InWord3&FIFO_EFN) == FIFO_EFN) statement. The parenthetic Boolean condition controls the while loop execution and is described below. There are two loop count variables: **i** and **j** which are initialized to 0. i has a range of 0 to 383; the loop repeats 384 times to read the 384 monitor data messages from the FIFO memory and store them in the data array. j has a range of 0 to 11 and causes

2.6 SERVER'S MONITOR DATA RETRIEVAL

This section describes the operation of the software and hardware components that SERVER uses to obtain CLIENT monitor data. Figure 2 (next page) is a block diagram showing these components and the major linkages between them. They consist of a Dynamic Linking Library (DLL), a Virtual Device Driver (VXD), a Monitor Data Interface (MDI) and a multidimensional Monitor Data Array in computer RAM. SERVER's Frame 1[0..2] Code Interface Node (CIN) is the address-data linkage to the DLL. See Section 2.5 for a description of SERVER's operations.

Note that the block diagram shows that the CIN uses LabVIEW DOWNSHIFT.VI to pass the addresses and data; it's operations are transparent to the CIN and other components. Since DOWNSHIFT is an internal LabVIEW function and was not particularized for the SERVER application, it's operations are not discussed here.

To SERVER's CIN, the Dynamic Linking Library literally looks like a VLA monitor data library. The CIN obtains the CLIENT's monitor data, one 24-bit data word at a time, by passing four address arguments to the DLL; these arguments designate the requested data. The DLL obtains this data (plus an 8-bit WCC count) from the data array in RAM via the VXD and returns it to the CIN. The four CIN address components are: Antenna Address (also called the DCS Address), Data Set Address, Multiplex Address and the MW1/2 designator.

As shown on the block diagram, the VXD performs two hardware driving functions: 1) In response to an IRQ2F by the DLL, the VXD obtains the requested 24-bit data word from the computer's RAM and returns it to the DLL. 2) At the end of each 19.2 Hz VLA cycle, in response to an IRQ5 generated by the MDI, the VXD reads the previous cycle's set of 384 monitor data messages from the MDI and stores them in the RAM's monitor data array. These messages were polled from the Central Buffers by the SLC during the previous 19.2 Hz VLA machine cycle. (See Appendix A for M&C System details.) These two VXD operations are independent and asynchronous but there are no time conflicts because both functions are interrupt-driven.

Figure 3 (following Figure 2) depicts the data retrieval process through the DLL-VXD software: The process starts with the SERVER-CIN's function call to the DLL with the data address; the DLL activates INT2F which calls the assembly language part of VXD to service the interrupt; this in turn calls the "C" language part of VXD which obtains the requested data from the data array in RAM. The VXD then returns the data to the DLL which completes the function call by a data return to the SERVER-CIN. As shown in the figure, registers convey the address and data through the DLL-VXD forward and return paths.

Dynamic Linking Library (DLL) Description

The DLL was written and compiled in Borland C, version 4.02. The current DLL version is DCSDLLA.C, 3/7/95, P. Dooley. During this DLL description, refer to the DCSDLLA.C listing which follows this section's text.

The #include <windows.h) and #include <dos.h> directives search for these files in the special directory for included files. Data from these files may be referenced by the program. The #include "dcsdef.h" directive extends the search to involve the current directory before searching the include directory. The dcsdef.h listing is included with the DLL and VXD listings at the end of this section's text.

The DLL starts with a call to function **dcsdata**. The first parameter of the call is the return type DWORD, which specifies that the returned data value is a double word (unsigned 32-bit value). The next parameter is **FAR PASCAL** which specifies that the function uses the FAR PASCAL method of passing

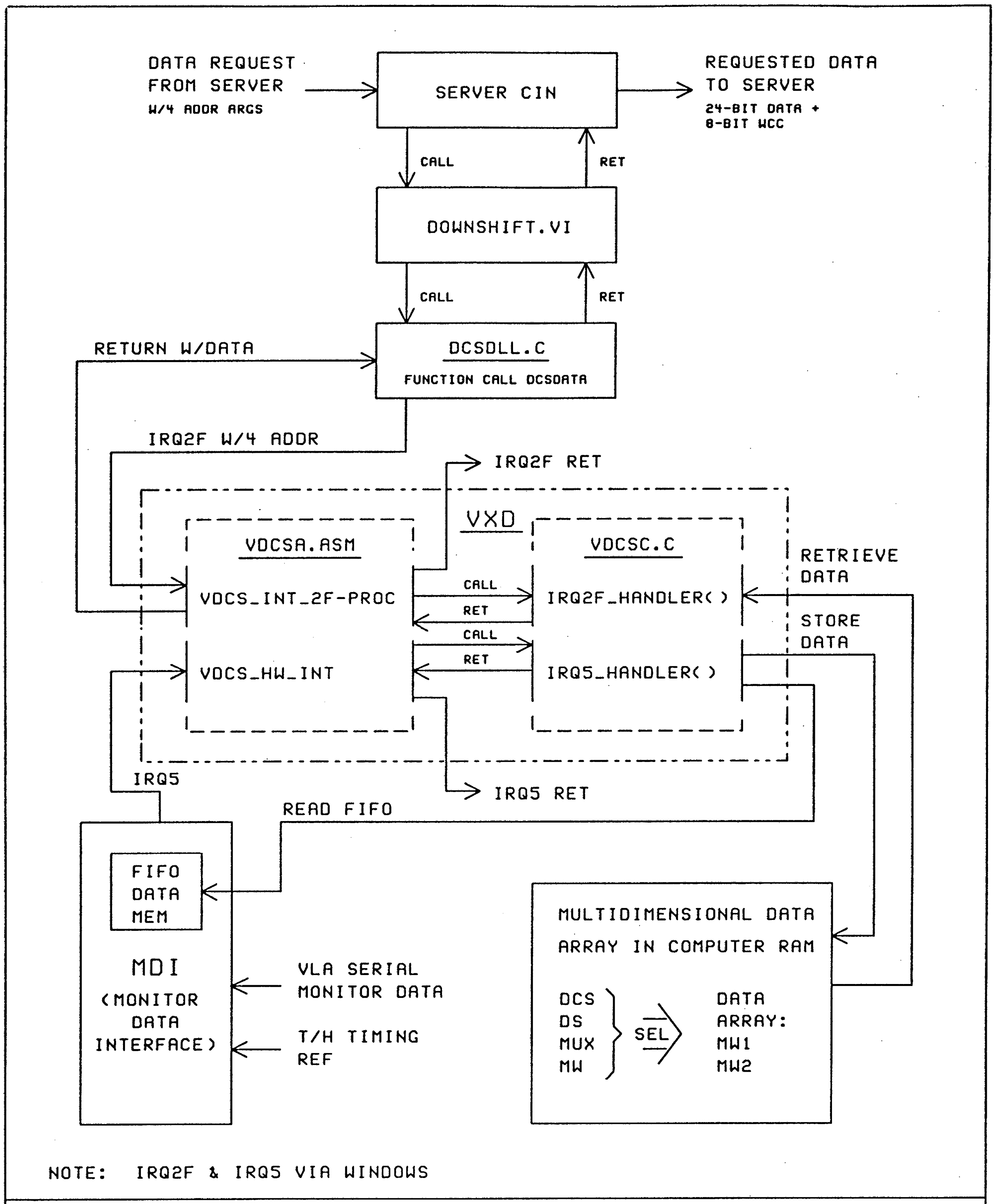
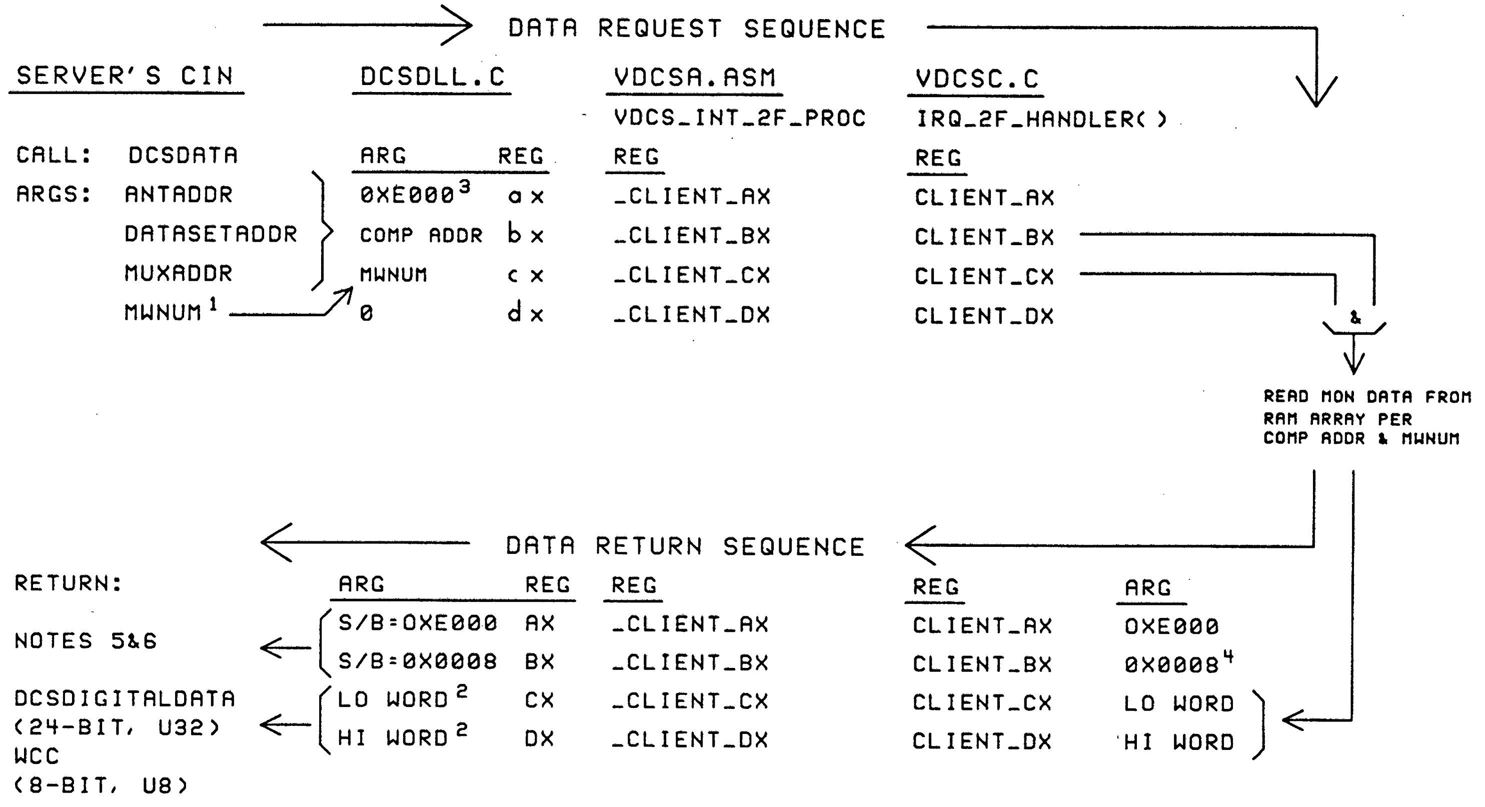


FIGURE 2, SERVER'S MONITOR DATA RETRIEVAL STRUCTURE

FIGURE 3, DATA TRANSFERS THROUGH DLL-VXD



NOTES:

1) MWNUM SELECTS EITHER MW1 (WNUM=0) OR MW1 (MWNUM=1) FOR RETURNED VALUE
2) HI WORD IS 16 MSB OF DATA VALUE & LO WORD IS 8 LSB OF DATA VALUE + 8 WCC BITS
3) 0XE000 LOADED IN ax BY DCSDLL.C IS MUX_HANDLE_BASE & IS RETURNED BY IRQ_2F_HANDLER( )
4) IRQ_2F_HANDLER( ) LOADS REG CLIENT_BX WITH 0X0008; THIS IS INT_2F_RESPOND
5) IN DCSDLL.C IF UPON RETURN CONTENTS OF AX & BX ARE BOTH CORRECT, 24-BIT DATA VALUE (U32) AND 8-BIT WCC (U8) ARE RETURNED TO SERVER'S CIN
6) IN DCSDLL.C IF UPON RETURN, CONDITION OF 5) IS NOT MET, RETURNED VALUE IS 0XBAD2000L

parameters. The **_export** parameter specifies that the returned data value is to be transferred to an external program. The function name dcsdata is followed by the list of four typed parameters in parenthesese; commas delimit the parameters. The parameter types are all WORD, unsigned 16-bit values: AntAddr, DataSetAddr, MuxAddr and MwNum.

A **return** statement is executed upon completion of the dcsdata function; data returned by the function immediately follows the return keyword. Note that there are three returns from the dcsdatafunction, two of these carry fault codes. The **return** (LongTemp | RegCX); statement (second return) carries the requested data back to the CIN. The fault code **returns** are described below.

The lines WORD wselector; and DWORD LinearAddress, LinearArrayAddress; are vestiges of an earlier version of the DLL and have no effect upon the current version.

The next three lines start with Macro declarations DWORD and WORD that define the three statement's variables types. Variables LongTemp and TempInt1 are initialized to 0, and TempInt1 is declared. RegAX,.. RegDX are 16-bit register pseudovariables that access CPU registers. This is done because calculations are faster when frequently used operands are put into registers rather than memory (see Figure 3). RegAX is loaded with MUX_HANDLE_BASE, obtained from the #include "DCSDEF.H" file and has the value 0xE000. (The usage of MUX_HANDLE_BASE is described below.) In the next statement, RegBX is loaded with a composite data address, identical to the first two bytes of a monitor data message: AntAddr, DataSetAddr and MuxAddr. All three addresses are brought into the DLL as unsigned 16-bit values, right-adjusted. In the statement's first parenthesis, AntAddr is left-shifted 11 bits; 0's are shifted into the variable's LSB. In the second parenthesis, DataSetAddr is left-shifted 8 bits. Two Bitwise OR operators merge the three operands into a single word with the five AntAddr bits in the most significant position, the three DataSetAddr bits next, and the eight MuxAddr bits in the least significant position. RegCX is loaded with the MwNum designator and RegDX is set to 0. Registers RegCX and RegDX will contain the dscdata function's return data value after returning from the int2fh call described below.

Turbo C executes inline assembly language instructions within the C source code. The assembly code portion is started by the keyword **asm** and delimited by curly brackets. The pusha instruction pushes the processor's eight general purpose 16-bit registers onto the stack. The following four mov instructions load the ax,..dx registers with the RegAX,..RegDX register contents. As noted above, these respectively contain the MUX_HANDLE_BASE, the composite monitor data address, MwNum and the dcsdata function's return value. The int2fh instruction generates the software interrupt IRQ2F. Note from the block diagram that the 2Fh interrupt service routine, Vdcc_Int_2F_Proc is in the VDCSA.ASM assembly language program in the VXD (described in the VXD paragraphs below). Vdcs_Int_2F_Proc calls the C language Irq2F_Handler() routine in the other VXD program (VDCSC.C) which retrieves the address-designated data from the array in RAM and returns it to the DLL's dcsdata function call. Upon return from the int2fh interrupt routine, four arguments in the ax,.. dx registers that were loaded by VDCS_INT_2F_PROC are loaded into RegAX,..RegDX registers by the mov instructions. The final assembly instruction, popa pushes the eight general purpose registers off the stack into their respective registers and the C program operations continue.

The next group of instructions is a comparison of Window's operating mode with the mode specified in the #include <windows.h> file. The **if** statement tests the boolean result of a bitwise AND of the logic inverse of the Windows Flag file and WF_MODE read from the #include <windows.h> file. The **!** character is the logical not function which specifies the logical inverse of the flags read from GetWinFlag() file. If these Windows Flag states are identical to the WF_MODE flag states, the logical product of their complement and the WF_MODE flag states is boolean false and control proceeds on the **else** path. If the bitwise AND is true, indicating a mismatch between the Windows operating mode and that specified for the program, the **return** statement is executed. This returns a 32-bit fault code, 0XBAD10000L to the CIN.

To verify that the three linked sections of the monitor data retrieval code (DLL, Vdcs_Int_2F_Proc in VDCSA.ASM and Irq2F_Handler in VDCSC.C) are executing properly, the AX and BX registers are loaded with test constants, and then tested after passage through the three pieces of code. In DCSDLL.C, RegAX is loaded with MUX_HANDLE_BASE (0xE000 from DCSDEF.H); this value is then passed through Vdcs_Int_2F_Proc to Irq2F_Handler and then returned to DCSDLL.C (via the same pieces of code) where it is tested in the **dcsdata** function's return (described below). This is a round-trip test. The second test value is the INTERRUPT_2F_RESPOND (0X0008 from DCSDEF.H) that is loaded into RegBX by Irq2F_Handler; this enables verification of RegBX's Irq2F_Handler to DCSDLL.C return path. RegBX's contents are tested in the **dcsdata** function's return statement, described below.

The next group of instructions are the **dcsdata** function's normal (error-free) data return. The logic operations in the **return**'s double parenthesis test the states of RegAX and RegBX to verify proper operation in the sections of linked code; this was mentioned above. In the first parenthesis, the contents of RegAX is Bitwise ANDed (by the **&** operator) with MUX_HANDLE_BASE; if there is a mismatch, something is wrong. The result is a Boolean true or false. The second parenthesis is a similar test of the contents of BX, which was loaded with INTERRUPT_2F_RESPOND in Irq2F_Handler(); again the result is a Boolean true or false. The two Boolean products are compared by a Logical AND (the **&&** operator); if the result is true (identical values), the dcsdata function's **return** exit is executed which carries the function's data to the CIN. Just before the return, RegDX (Hi Word in VDCSC.C) is loaded into LongTemp and left-shifted 16 bits, 0s are shifted into the LSB. The **return** instruction loads LongTemp with the Bitwise OR of CX, (Lo Word in VDCSC.C); this places both 12-bit data values and the 8-bit WaveGuideTick (WCC count) in the 32-bit data value returned to the CIN.

If the result of the Logical AND (described above) is false, the VXD did not respond to the int2fh call and the **return** following the **else** statement is executed. In this case the return carries the fault code 0XBAD20000L. The 0XBAD20000L and 0BAD10000L fault codes indicate a DLL execution fault to SERVER.

There are two additional function calls (LibMain and WEP) in the DLL that are not directly related to the CIN data retrieval function. They perform functions required by Windows in DLL applications. Since they are standardized and unrelated to the CIN function, they are not described here.

Monitor Data Array in RAM

Since the VXD stores and retrieves monitor data from the multidimensional array in computer RAM, it may be useful to describe the array structure. Note the array declaration on page 1 of the VDCSC.C listing. All array elements are WORD (unsigned 16-bit) values). The "fixed-array" parameters are enclosed by brackets that define the number of parameter elements. Three of these parameters correspond to addresses: DCS (Antenna), Data Set and Multiplex. Although only 28 of the 32 possible DCS addresses are used and there are unimplemented multiplex addresses, the array is sized to accommodate the ultimate address capacity. This is: 32 x 6 x 256, or 49,152 addresses (actually, only 192 addresses are used for monitor data; the remaining 64 are used for commands). When the Irq5_Handler() stores data in the array, it is stored in the location corresponding to the composite address components.

Note that below the array's multiplex address parameter, there are two additional two-element parameters; the first is labelled "2 Monitor Words" by the comment and the second is labelled "2 Data Words." The "2 Monitor Words" parameter is a 32-bit **Monitor Word 1** value; the second parameter is a 32-bit **Monitor Word 2** value. Thus for each DSA-Data Set-Multiplex Address state (32 x 6 x 256), both types of monitor data words are stored in the array. The paragraph below describes the two types.

The array size is 32 x 6 x 256 x 2 x 2 which is 196,608 16-bit words or 393,216 bytes. _MemoryArrayAddress is an artifact of an earlier VXD version.

Virtual Device Driver (VXD) Description

The Virtual Device Driver was written in two parts. The first part, VDCSA.ASM contains the shell of the VXD and all of the Operating System calls. It is written in assembly language and compiled by MASM. The second part, VDCSC.C contains all of the called functions of the VXD and is written in "C" language and compiled by MSVC. Both object files were linked by a MS linker to form the VXD.

The current versions of the two VXD components are: VDCSA.ASM P. Dooley 1/9/95 and VDCSA.C P. Dooley 1/9/95. During this VXD description, refer to the VDCSA.ASM and VDCSC.C listings at the end of this section's text.

The VXD is "flat"; all operands and addresses are treated as 32 bit integers which obviates the use of segment registers.

As noted above, two interrupt-triggered, hardware-driving functions are performed by the VXD:
1) Retrieving the DLL-requested data from the RAM array in response to software interrupt int2fh.
2) Reading the most recent VLA cycle's set of 384 monitor data messages from the Monitor and Control Interface and storing them in the RAM array in response to int5fh. Note from the block diagram that these two operations are independent in both VDCSA.ASM and VDCSC.C, and each operation involves linked assembly language and "C" language code.

VDCSC.C Description

VDCSC.C is the "C" language part of the VXD.

Like the DLL, VDCSC.C references the #includes files, "dcsdef.h", "windows.h", and "dos.h", the latter two being general operating system files. The "dcsdef.h file was mentioned in the DLL description and is included with the other listings at the end of this section's text. Additional #include files it references are: "conio.h" which declares functions used in calling the DOS console I/O routines, "stdlib.h" which declares several commonly used conversion, search/sort and other miscelaneous routines, "hungaria.h" which permits abbreviated notation, and "vxd.h" which are pointer conversion macros for converting among types of 4-bit pointers. The last, "vxd.h" does not seem to have obvious utility to VDCSC.C but it is in the #include list.

There are six #define variables, none of which are used in VDCSC.C.

Note the **BYTE MwTable[12] = {0,...0,1,...1,0,1}** statement which defines a 12-byte table of 0s and 1s. This table designates the monitor word type time sequence order for each DCS's six Data Sets. The table's first five values are type 0 (for Monitor Word 1, Data Sets 0,..4), the second five are type 1 (for Monitor Word 2, Data Sets 0,..4) and the last two, types 0, and 1, for Data Set 5. The table's Monitor Word order is an artifact of the Antenna and Central Buffer's Data Set polling sequences. The SLC sequentially polls the 32 Central Buffers, DCS0 through DCS31, twelve times, each time evoking a Data Set monitor message. During the first pass, Data Set 0's Monitor Word 1 is read out of the 32 buffers; during the second pass Data Set 1's Monitor Word 1 is read out of the 32 buffers; the Monitor Word 1 readout sequence continues through Data Set 4. During the sixth pass, Data Set 0's Monitor Word 2 is read out of the 32 buffers and the Monitor Word 2 readout sequence continues through Data Set 4. During the 11th pass, Data Set 5's Monitor Word 1 is read out, and during the 12th pass, Data Set 5's Monitor Word 2 is read out. Thus the table designates a DCS's six Data Set's MW1-MW2 time-sequence order; this order is the only way to distinguish between MW1 and MW2.

A brief digression here: Monitor Word 1's multiplex address sequence is determined by a table in a Data Set EPROM; Monitor Word 2's multiplex address is either a sequential scan through all addresses or it can be command-set to any address so that it samples the same data channel every VLA cycle. The fixed address mode is useful for intensive sampling purposes. The multiplex address uniquely designates a single analog or digital data channel; the only difference between Monitor Word 1 and Monitor Word 2 data are the address sampling schedules and sampling times; Monitor Word 2 is sampled 700 μs after Monitor Word 1. Why the emphasis here upon the differences since both types refer to the same signal? The Irq5_Handler() described below stores both types in different elements of the same array address; some application programs designate either (but not both) type for processing. A user may elect to command a data channel to the intensive sampling mode for higher time resolution in an application program.

Below the table statement described above is a second table statement with a different order; this table is not used and is "commented out" by the \\ prefix.

The keyword **extern** defines four WORD register pseudovariables, Client_AX,..Client_Dx that enables arguments to be exchanged between VDCSC.C and the assembly program VDCSA.ASM. In VDCSA.ASM the _Client_AX, etc., are declared to be global variables by a PUBLIC declaration.

The **Byte The_Shell_message_Proc[12]** and **extern Shell_Message_Proc** are artifacts of a call to debugging code at the end of the listing. This procedure was used during program development to check argument transfers in the Client_AX, etc., registers and is no longer used. Portions of this procedure have been removed from VDCSC.C.

The **BYTE** variable **ParityByte** is not used and it is an artifact of a function that was not implemented.

Irq5_Handler()

This function is called by VDCS_HW_INT in VDCSA.ASM in response to interrupt IRQ5 from the MDI which signals that the most recent VLA cycle's monitor data is available in the MDI's FIFO memory. This function reads the 384 monitor data messages (five bytes) stored in the FIFOs and stores the message's 24-bit data component in the array's address-designated Monitor Word 1 or Monitor Word 2 locations. When the storage operation is completed, program control returns to Vdcs_Hw_Int. The **void** return type in the Irq5_Handler() function call specifies that the function does not return a value; it also means that the function does not have to use a **return** statement when the function has completed its operations. Since this is a called function, there is not a **main** statement.

The **WORD i,j,InWord0,InWord1,** etc., statement defines 16-bit operands; InWord0 through InWord3 are four values read from the FIFO memory. **InWord0** is the composite message address (DCS, DSA and Mux); **Inword1** is the upper two message data bytes; **InWord2** is the lowest data byte, read out in the word's upper byte; the lower byte is null. **InWord3** is the MDI FIFO Status Register; an encoded value indicates completion of the FIFO read operation. The Antenna (DCS), MuxAddr and Dsa operands are the familiar address parameters. The **MwCtr** state is an index to the **MwTable** described above; **Mw** is the value read from the table (0 or 1) and determines whether the message's data is stored in the array's MW1 or MW2 elements.

Ignoring for the moment the WaveGuideTick statements, the data storage while loop starts with the **while** ((InWord3&FIFO_EFN) == FIFO_EFN) statement. The parenthetic Boolean condition controls the while loop execution and is described below. There are two loop count variables: i and j which are initialized to 0. i has a range of 0 to 383; the loop repeats 384 times to read the 384 monitor data messages from the FIFO memory and store them in the data array. j has a range of 0 to 11 and causes the MwCtr to be incremented; the count state is an index to the MwTable. MwCtr is also initialized to 0.

The **WaveGuideTick** is the WCC count mentioned above. This parameter is incremented after it contributes its value to the next expression, the **if(WaveGuideTick == 192)** statement. This expression resets the count to zero if it reaches 192. The WaveGuideTick is incremented each time the Irq5_Handler() is called; therefore it counts the 19.2 Hz VLA machine cycles and the count period is ten seconds.

The four InWord statements, **InWord0 = inpw(DCS_BASE)**, InWord2 = etc., each read 16 bits from the FIFO; **inpw** is a word (two-byte) read. InWord0, ..,Inword3 contents were described above. The FIFO read pointer is decremented each time the InWord statement is executed; this selects the next 16 bits for readout. (The FIFO's logic increments the address during a write and decrements it during a read). DCS_BASE is the MDI's select address, 0x300 (Hex, from file DCSDEF.H) and DCS_BASE+6 is the FIFO's status register address.

The next series of statements check to see if the array addressing is in bounds; this check is described below.

Monitor data is stored in the computer RAM's multidimensional array by the two consecutive **MonData[AntAddr][Dsa][MuxAddr][Mw][0] or [1] = X,.....** statements with the familiar address operands. Note the **[Mw]** and **[0]** or **[1]** parameters: Mw's state determines whether the data is stored in the MW1 or MW2 array elements; the way that Mw's state is set is described below. The [0] or [1] parameter determines whether the 16-bit data is stored in "LoWord" or "HiWord" of the MW1 or MW2 locations in the array. Remember that the stored data value consists of 24 data bits (three bytes) to which is appended a WaveGuideTick byte; thus each data value requires two 16-bit words in RAM. The first statement stores InWord2 in "Lo Word" in either MW1 or MW2, and consists of the lowest data byte and the WaveGuideTick. The second statement stores InWord1 in "Hi Word" in either MW1 or MW2 array locations and consists of the upper two data bytes. Note that the composite address read from the FIFOs is not stored.

In the first MonData storage statement, the operation performed on the right side of the = (assignment) operator is a mask-merge. Inside the parenthesis, InWord2 is Bitwise ANDed with hex 0xFF00; this clears the lower byte of InWord2. The resultant product is then Bitwise ORRed with the WaveGuideTick byte; the final state is stored in the "LoWord" array locations determined by the address and Mw and parameters.

We now consider the Mw parameter. Note the **Mw = MwTable[MwCtr]** statement that precedes the data storage statements. This sets the Mw variable to the state in the MwTable pointed to by the MwCtr state. (The MwTable was described above.) The MwCtr is post-incremented at the bottom of the while loop and is initialized to 0 before the while loop is entered. Note that the Mw = 0 and ParityByte = 0 statements have been "commented out" by the \\ characters.

Note that indices i and j are post-incremented near the bottom of the while loop. After j is incremented, it is compared with 28 by the == (equal) operator. If true, j is set to 0; this eliminates the unnecessary data readout from the FIFO's DCS 28, DCS 29, DCS 30 and DCS 31 data storage locations since this data is null and No Response-flagged by the SLC. If one of these DCS addresses is activated, this section of VDCSC.C should be adjusted to the new conditions.

Now, referring to the **while** loop Boolean expression **((InWord3&FIFO_EFN) == FIFO_EFN)**. When this condition becomes false, the while loop execution is stopped. InWord3 is initially set to FIFO_EFN, 0x0010 from the "dcsdef.h" file. InWord3 is the fourth FIFO read instruction in the four word sequence and reads the FIFO's status register, at the DCS_BASE+6 select address. If the number of monitor data word readouts is less than 384, the status register readout is 0x0010: when the count reaches 384, the status register value becomes 0 which indicates that the last data byte has been read from the FIFO memory. The different InWord3 value changes the Bitwise And of InWord3 and FIFO_EFN; this makes the == (equal to) relational operator false which terminates the while loop execution.

Inside the while loop there are some address tests to insure that array addressing stays within bounds. The first is the **if** test of the i index, if it is greater than 383, InWord3 is set to 0 which makes the while loop Boolean expression (described above) false so that the loop stops executing.

The next series of tests isolates the three address components in InWord0 (the composite address). The AntAddr (i.e. DCS) variable is derived by Bitwise right-shifting Inword0 and Bitwise ANDing it with 0x1F; this places the five AntAddr bits in the least significant location in the 16 bit word; higher bits are all zeros. The Dsa and MuxAddr are derived by the same technique. An **if** statement tests the AND of a concatenated set of relational expressions to verify that AntAddr does not exceed 27, that Dsa is greater than -1 and less than or equal to 5 and that MuxAddr is greater than -1 and less than or equal to 255.

Irq2F_Handler()

This function is called by Vdcs_Int_2F_Proc in VDCSA.ASM in response to interrupt IRQ2F from the DLL. It reads address-designated data from the mulitdimensional monitor data array and returns it to the DLL.

The voids in the function's first statement, **void** Irq2F_Handler(**void**) have the same usage as the voids in the initialization function described in the previous paragraph.

The **WORD DcsMwNum,DcsAntAddr,DcsDsa,DcsMuxAddr** statement defines these familiar address components as 16-bit register pseudovariables; the syntax is altered to make them compatible with their usage in VDCSA.ASM and distinct from their usage in the DLL. These variables are brought into the handler from the DLL (the Rx'd Parameters) in registers Client_BX (the composite address, described in the DLL) and Client_CX, the MW1/2 designator. Again, the Client prefix makes them compatible with their usage in VDCSA.ASM and distinct from the DLL.

The requested 32-bit data value is returned to the DLL in variables Client_CX and Client_DX. Variable Client_BX is not altered by the handler and is returned to the DLL by Vdcs_Int_2F_Proc in VDCSA.ASM; as described in the DLL paragraphs, the DLL tests the contents to verify that the VXD functions were performed.

To read data from the array, tt is necessary to extricate the three address components from Client_BX. The five-bit DcsAntAddr is extracted from Client_BX by a Bitwise right shift of 11 bits and the result is Bitwise ANDed with 0x001F; the result is a 16-bit value with the five DcsAntAddr bits in the least-significant position. The two other address components are extracted by similar operations. The MW1/2 designator is derived by a Bitwise AND of Client_CX and 0x1.

Thwo variables and two test constants are returned to the DLL via Vdcs_Int_2F_Proc in VDCSA.ASM. We consider the first test constant, 0x0008. The contents of Client_BX is loaded with INTERRUPT_2F_RESPOND (from DCSDEF.H). INTERRUPT_2F_RESPOND is tested by the DLL to verify the integrity of the program-to-program round-trip sequence. This test verifies that Irq_2F-Handler() was actually called by the DLL since the probability that another non-VIR system program loaded this value into Client_Bx and set IR2F is vanishingly small. The second test constant is the MUX_HANDLE_BASE (0xE000) loaded into RegAX by the DLL and passed through Vdcs_Int_2F_Proc (in VDCSA.ASM) to Irq_2F_Handler(). Irq_2F_Handler() does not change Client_AX, so this test constant value is returned intact to the DLL where it is tested. If this test value is returned to the DLL, it is a second verification that only the VIR system set IRQ2F.

The two variables are Hi Word and Lo Word, each 16 bits, read from the multidimensional monitor data array in RAM. The DLL combines the 32 bits into a single 24-bit value and an 8 bit WCC tick value. The data value can be either MW1 or MW2, designated by the **DcsMwNum** variable brought into Irq_2F_Handler() by variable Client_CX. The Hi Word component is read by the **Client_CX = MonData[DcsAntAddr][DcsDsa][DcsMuxAddr][DcsMwNum][0]** statement and the Low Word component is read by the **Client_DX = MonData[,....][1]** statement. This completes the operations performed by Irq2F_Handler and control is returned to the DLL via Vdcs_Int_2F_Proc in VDCSA.ASM.

VDCSC.C Initialization

On page 2 of VDCSC.C is an initialization function called from VDCSA.ASM that initializes the array's Monitor Word 1 and Monitor Word 2 elements. The **void** before the function name, **Initialization** means that the data type it points to is not specified. The **(void)** after the function name means that the function does not return a value. This procedure consists of three nested loops which sequence through the DCS, Data Set and Mux addresses. Monitor Word 1's LoWord elements are set to 0x0100 and the HiWord elements are set to 0xdead. Monitor Word 2's LoWord elements are set to 0x0200 and the HiWord elements are set to 0xdead. This initialization procedure is not a vital function of VDCSC.C, and is unnecessary in normal operation and is an artifact of the VDCSC.C program development.

VDCSA.ASM Description

VDCSA.ASM is the assembly language portion of the VXD.

VDCSA.ASM operates under Windows 95 and performs the interrupt-triggered data handling functions that cannot be performed in LabVIEW or by the "C" language (VDCSC.C) part of the VXD. See Figure 2, SERVER's Monitor Data Retrieval Block Diagram (above, second page of this section). Note the relationships between VDCSA.ASM, VDCSC.C, DCSDLL.C, the Monitor Data Interface (MDI) and the Monitor Data Array in RAM. Two VDCSA.ASM functions, **Vdcs_Int_2F_Proc** and **Vdcs_Hw_Int** are, respectively, the linkages between Window's IRQ2F and IRQ5 interrupt handlers and the VXD's "C" language **Irq2F_Handler()** and **Irq5_Handler()**.

Also see Figure 3 (following Figure 2) which depicts the address and data transfers through the SERVER-CIN, DLL and VXD software.

When SERVER requests a data value via its CIN, the DLL sets an IRQ2F interrupt. Window's IRQ2F handler then calls **Vdcs_2F_Proc,** which in turn calls **Irq2F_Handler()** to obtain the SERVER-requested data value from the Monitor Data array in RAM. The data value is returned to the DLL via register variables as depicted in Figure 3.

When the Monitor Data Interface (MDI) has completed storage of the current cycle monitor data, it sets IRQ5. Windows' IRQ5 interrupt handler calls **Vdcs_Hw_Int**, which in turn calls **Irq5_Handler()** to transfer the monitor data from the MDI's FIFO memory to the Monitor Data Array in RAM. Like the IRQ2F operations, the data values are conveyed via register variables.

On program startup, VDCSA.ASM's initialization procedure establishes the required linkages with Windows's Virtual Machine Manager (VMM), Virtual Programmable Interrupt Control Driver (VPICD) and SHELL so that VDCSA.ASM executes as a standard Windows-compatible program.

The term "virtualizing" and labels derived from the word are frequently used in VDCSA.ASM; in fact, "virtual" properties are a requisite for Windows-compatible programs. Windows is a multi-tasking operating system in which one or more programs may be operating concurrently. Windows provides the functional equivalent of multiple independent "virtual machines," one for each executing program, using what appear to the programs to be the full resources of the machine. Each virtualized machine has its dedicated CPU, memory, registers, interrupts, I/O, etc. One of these VMs is the System Virtual Machine or **System VM** which controls the execution of the other virtual machines. Windows allocates the computer's resources and resolves contentions between the programs so that there are no unwanted interactions between them. Although these programs operate independently, data can be passed between them. Managing the concurrent execution of several programs is obviously very complicated. The following description is a brief, broad-brush descripton of VDCSA.ASM and its Windows linkages; a detailed description of Windows' operation is way beyond the scope of this manual.

Readers interested in a more extensive treatment of driver-Windows linkages are referred to "Writing Windows Device Drivers" by Norton (1992) or "Systems Programming for Windows 95" by Oney (1996). The VMM, VPICD and SHELL macro calls used in VDCSA.ASM are listed in the three "Include" files, VMM.INC, VPICD.INC and SHELL.INC.

LabVIEW SERVER and the Borland "C" DCSDLL.C programs have Windows-compatible interfaces. However, the MDI is a custom logic design and there are no commercially available drivers for it; consequently VDCSA.ASM incorporates a Windows-compatible interface which enables it to operate as a "virtualized program" in one of the "virtual machines." The VM that is executing VDCSA.ASM is identified by its **"VM Handle"**, which is a flat pointer to its VM Control Block. During program initialization and execution there are many macro calls to Windows' VMM and VPICD files. VMM.INC is the Virtual Machine Manager operating in the System VM. Among other functions, the VMM provides an I/O port handler for the MDI. In the PC hardware, interrupts are handled by a programmable interrupt controller (PIC); in Windows the virtual PIC driver (VPICD) manages interrupts for Virtual Device Drivers (VxDs). In executing VDCSA.ASM, the VPICD provides interrupt handling services for IRQ5 and IRQ2F. The Shell services provide the user a way to interact with the VxD via messages and responses.

In the 386 there are four CPU privilege rings that regulate execution priority; Windows uses two of these rings. The Windows 95 Kernel and VDCSA.ASM operate in Ring 0; the System VM, and application programs (word processors, etc.) and other VMs operates in Ring 3. (The .386 directive mentioned below allows the assembler to generate i80386 instructions).

The term **VxD** is a Windows VMM formalism for "Virtual Device Driver" and the term VxD is used in macro calls and procedure labels throughout VDCSA.ASM. The VxD "name" for VDCSA.ASM is **Vdcs**; this label is used in the following discussion.

To a limited extent, Vdcs is modeled after VxD MonDel because MonDel demonstrates the structure and calling conventions required by a Windows VxD. MonDel is a Virtual Device Driver that watches system-wide for file deletes and notifies the user via a "C" language program; control is returned to the VMM. A MonDel listing is not included in this manual.

At the machine level, interrupts involve the INT and IRET instructions; INT pushes the flags and registers CS and IP onto the stack, clears the trap and interrupt flags before jumping into the interrupt handler. The jump is to the first instruction in the interrupt handler whose segment:offset address is contained in the interrupt vector that corresponds to the interrupt number. The code execution sequence broken by the execution of an interrupt handler should be properly resumed after the handler's functions have been performed. This requires that the interrupted program's critical information be saved before control is passed to the interrupt handler. This information must then be restored before execution of the interrupted program can resume.

Windows uses Interrupt 2F in video applications and in a virtual display driver (VDD). To avoid potential confusion over which program set IRQ2F, the Vdcs_Int_2F_Proc (described below) has provisions to identify when it has been set by Vdcs. In DOS, IRQ5 is the "print screen" hardware interrupt. Since Windows does not use the print screen feature, Vdcs can use this interrupt without having to test for non-Vdcs usage as is the case with IRQ2F.

Referring to the VDCSA.ASM listing, note the installation instruction (add "device=\path\VDCS.386" in [386Enh] in SYSTEM.INI).

The program structure consists of five Windows-required blocks: 1) Directives (.386, INCLUDE, and EXTERN references); 2) EQUATES; 3) VIRTUAL DEVICE DECLARATION; 4) LOCKED DATA SEGMENT; and 5) LOCKED CODE SEGMENT. The first three are briefly described below; the DATA SEGMENT and LOCKED CODE SEGMENT descriptions follow.

DIRECTIVES

The .386 directive allows the assembler to generate i80386 instructions.

The INCLUDEs (VMM.INC, VPICD.INC and SHELL.INC) are Windows files mentioned above; the LOCKED CODE SEGMENT has many macro calls to functions in these files.

The EXTERNs Irq5_Handler_:NEAR, Irq2F_Handler_:NEAR and Initialization_:NEAR reference functions in the "C" language portion of VDCSC.C; the first two were mentioned above and are shown on Figure 2, SERVER's Monitor Data Retrieval Structure Block Diagram. Initialization is also in VDCSC.C and initializes the contents of the Monitor Data Array in RAM.

The EXTERNs Debug_Routine_:NEAR is a Windows program debugging feature which is not discussed here. _The_Shell_Message:BYTE is used with the Shell_Message_Proc described below.

EQUATES

EQUATES are data values used by the assembler; the two Ver... values identify the VxD version; the PORT_ADDR EQU 300h defines the first (base) address of the Monitor Data Interface (MDI) which has four addresses: 300, 302, 304 and 306. IRQ5 and IRQ2F identify the interrupts used by Vdcs. Vdcs_Init_Order EQU 0C0000000h defines Vdcs's Device Descriptor Block (DDB) address in the VIRTUAL DEVICE DECLARATION bloc below. MY_MUX_ADDR EQU 0000E000h defines a value that is tested in DCSDLL.C to verify that the DLL to DCSC.C and back to DLL linkages were properly executed.

VIRTUAL DEVICE DECLARATION

This section is the Device Descriptor Block (DDB) which provides the linkage between Vdcs and the Windows Kernel. The linkage is a data structure in the device's permanant data segment. The DDB is defined by invoking the **Declare_Virtual_Device** macro which can take up to nine parameters.

The first parameter is the device name, **Vdcs.** The macro creates a public symbol using this name with _DDB appended. The second and third parameters define the major and minor Vdcs version numbers, in this case version 1.0 (from the EQUATES block). The fourth parameter specifies the address of the VxD "control" entry point; it is **Vdcs_VxD_Control** in the VXD_LOCKED_CODE_SEG. The fifth parameter specifies the virtual device ID number; in the Vdcs case, an ID number is unnecessary so the **Undefined_Device_ID** symbol is used. The sixth parameter specifies the order in which Vdcs should be initialized relative to the initialization order of other VxDs. The initialization order of other VxDs is defined in the VMM.INC file; in the case of Vdcs, its order is not defined by VMM.INC because Vdcs's initialization order is not dependent on that of other VxDs. Note that all these parameters are prefixed by a comma and postfixed by a \ (backslash).

DATA SEGMENT

This section is delimited by the **VxD_LOCKED_DATA_SEG** and **VxD_LOCKED_DATA_ENDS** macros. This segment and the following LOCKED CODE SEGMENT are always resident in memory or "locked" into place by the VMM.

The IRQ Descriptor Structure Elements (listed as comments) are functions that may be invoked by the VPICD in servicing the interrupts. These functions are listed for informational purposes and are not directly called by Vdcs itself.

A VxD can modify the normal VPICD default procedures by virtualizing the IRQ itself. this is done for IRQ5 and IRQ2F and avoids complications that might result from potential contentions for IRQ5 and IRQ2F services by other VxDs (if other VxDs are installed in GARCON that might try to use IRQ5 and IRQ2F). Virtualizing is done in the initialization code (page 3) by calling **VPICD_Virtualize_IRQ** service for IRQ5; IRQ2F is virtualized by a **VMMcall Hook_V86_Int_Chain.** In the call to this service, the VxD passes a structure that contains option indicators and callback routines. In response, the VPICD returns an IRQ handle; after a VxD has received the virtual IRQ handle, it can assert an interrupt by calling **VPICD_Set_Int_Request.**

The VPICD's process of virtualizing the two interrupts is too complex to describe here; for the purposes of this Vdcs description, it is sufficient to describe the structure table and note the virtualizing macro calls to VPICD.

The table headed by the comment ;IRQ descriptor to "virtualize Irq 5" is the IRQ5 structure passed to VPICD to perform this function. The two macros, **Vdcs_IRQ_Descriptor** and **VPICD_IRQ_Descriptor** order the VPICD to perform the virtualizing function for IRQ5 for use by Vdcs. The table format conforms to VPICD_IRQ_Descriptor STRUC specifications (not described here). Note that the table is delimited by <\ and > and each line starts with a comma and ends with a \. The lines and spaces shown are required table features. VPICD reads the **IRQ5** interrupt identifier and the address of the associated **Vdcs_Hw_Int** procedure (page 3) which is called by the VPICD when the MDI sets IRQ5. Note that this address is preceded by an **OFFSET32**. After the VPICD processes this call, it provides an **Irq5_Handle**, listed below as a dd define double word variable.

Immediately following the above is the similar table for IRQ2F. The **Vdcs_IRQ_2F_Descriptor** macro and the **VPICD_IRQ_Descriptor** macro order the VPICD to virtualize IRQ2F. The table cites **IRQ2F** but does not contain what might be the expected **OFFSET32** and **Vdcs_Int_2F_Proc,** lines. The offset and linkage to Vdcs_Int_2F_Proc is provided in the initialization code labelled: ;set up to respond to Int 2f on page 3.

The next set of variables are Define Word (dw) and Doubleword (dd) reference values; note that the first three are assigned a value of 0 and the next three are ?, an undefined (at this point) value. As noted above, the **VM_Handle** label identifies the virtual machine that is executing Vdcs. VM_Handle is used with ebx in the Vdcs_VxD_Control, Vdcs_Hw_Int and Shell_Message_Proc procedures below.

Following are a list of five Global Variables used by VDCSC.C, the "C" language portion of the VXD. These psuedo-register variables are all declared PUBLIC and (AX through DX) respectively, and carry the MUX_Handle_Base, Composite Address, Lo Word and Hi Word variables. _MemArrayAddress is an artifact of an earlier version of Vdcs.

The next six lines are an unused Shell message that is "commented out" by ~ (tilde) delimiters. Following this are data for a Shell message, accessed by the Shell_Message_Proc on page 4. The procedure provides a way to return a Caption Message and a "Vslc Debug Point #1" test value.

LOCKED CODE SEGMENT

This block (page 2) contains the "working" part of Vdcs and is delimited by the **VxD_LOCKED_CODE_SEG** and **VxD_LOCKED_CODE_ENDS** macros. The operations performed in this section are Initialization, IRQ5 service, IRQ2F service and Shell message service.

The locked code segment has many VMMcall and VxDcall macro service calls to the VMM and VPICD, respectively.

Vdcs's linkages to VMM, and VPICD are established in the Initialization Code that installs the MDI I/O port, and IRQ5, and IRQ2F interrupt handlers. Two procedures perform this initialization function; the first is **Begin Proc Vdcs_VxD_Control** and this procedure ends with the **EndProc Vdcs_VxD_Control** macro. All operations required to link a VxD to Windows must be included in this procedure. Note that the control entry point, **Vdcs_VxD_Control** was cited in the Virtual Device Declaration described above. Within this procedure the **Control_Dispatch** macro builds a jump table and dispatch code for the control entry point. Upon entry to the first procedure, Windows creates the Vdcs VM and loads the address of the control block (**VM_Handle**) into the EBX register. In this first procedure the **Device_Init** macro calls the **InitVdcs** procedure which installs the I/O and interrupt handlers mentioned above.

After executing the **InitVdcs** procedure, control is returned to the **Vdcs_VxD_Control** procedure (above) and register ebx is moved into [VM_Handle], the dd variable in the data segment mentioned above. The brackets denote that VM_Handle is an operand used in register indirect addressing in the IRQ5 (Vdcs_Hw_Int Proc) and SHELL message functions. In the LOCKED DATA segment, VM_Handle is initialized to 0 which designates that the MDI VxD is the focus of the VMM for the Vdcs VxD and that edx will contain the device ID code. No other VMM application can change this status. After clearing the carry flag (an important requirement), the ret statement ends the **Vdcs_VxD_Control** procedure. This tells Windows that the VxD installation has been completed.

At the start of the **InitVdcs** procedure, the extended Count, Source Index and Destination Index registers are pushed onto the stack to preserve their states for return to Windows after initialization has been completed.

The MDI I/O handler is next installed by the **Install_IO_Handler** macro call in the four-pass VPD_Itrap_Loop. The edx register is loaded with the MDI port base address (300h from the EQUATES); esi is loaded with the address of the **Port_Trap_Addr** procedure (page 4) with an OffSET of 32. This procedure simply consists of a ret instruction. During each pass, edx is increased by 2 which defines 300, 302, 304 and 306 as MDI I/O port addresses. The number of loop passes, 4, is determined by NUM_PORTS from the EQUATES block. During each pass of the loop, the **VMMcall Install_IO_Handler** macro tells the VMM that edx contains an MDI I/O port address. When the four loop passes have been executed, the MDI I/O handler is ready for use after the edx, esi and ecx register contents have been popped off the stack.

The IRQ5 handler is virtualized next. First the extended Destination Index Register (edi) is pushed onto the stack. The address of the **Vdcs_IRQ_Descriptor** structure table (Page 1) is loaded into edi with an OFFSET of 32. The **VxDcall VPICD_Virtualize_IRQ** macro calls this VPICD function to make IRQ5 service a "virtual" process and upon return, eax contains the **Irq5_Handle**. This was defined as a dd variable in the Locked Data Segment above. The Irq5_Handle is used in the IRQ5 interrupt handler below. The next function is to physically unmask Irq5 so that it can be used. The Extended Accumulator (eax) is pushed onto the stack and the [Irq5_Handle] is loaded into eax. The **VxDcall VPICD_Physically_Unmask** macro performs this function using the contents of eax.

IRQ2F is virtualized next. 2Fh is loaded into eax, this identifies the interrupt and Vdcs_Int_2F_Proc, the flat address of the IRQ2F handler in Vdcs is loaded into esi with an OFFSET of 32. A VMMcall (not a VPICDcall) to **Hook_V86_Int_Chain** virtualizes IRQ2F. eax is popped off the stack which completes the IRQ2F virtualization process.

Next the Initialization procedure in VDCSC.C is called. This function initializes the contents of the Monitor Data Array in RAM, a debugging convenience and is not important to Vdcs. The sti, set interrupt flag instruction sets the Interrupt Flag which enables the maskable interrupts. The procedure ends with a ret which ends the **InitVdcs** procedure. This process also yields a virtual **Irq2F_Handle**, defined as a dd variable in the Locked Data Segment above. This Irq2F_Handle is not directly invoked in the Locked Code Segment but may be used in VPICD and VMM calls. Control then returns to the **Vdcs_VxD_Control** procedure described above. The terminal operations of the Vdcs_VxD_Control procedure was described above.

At this point, the MDI port I/O handler has been installed and IRQ5 and IRQ2F have been virtualized and Windows resumes its multiprocessing operations.

Vdcs_Hw_Int

The IRQ5 handling function is performed by the **Vdcs_Hw_Int** procedure which has the **High_Freq** modifier. The High_Freq macro aligns the Vdcs_Hw_Proc entry point to a 32-bit boundary to speed up execution upon entry. This procedure is called by VPICD in response to an IRQ5 from the MDI and the first instruction is a call to **Irq5_Handler_** in VDCSC.C, the "C" language part of the VXD that transfers the 384 monitor data values from the MDI FIFO to the Multidimensional Monitor Data Array in RAM. This function was described above. See Figure 2 for a graphical overview of this function.

After pushing eax and ebx onto the stack, eax is loaded with the register indirect address of [Irq5_Handle] and ebx is loaded with the register indirect address [VM_Handle]. The **VxDcall VPICD_Phys_EOI** macro tells VPICD that the IRQ5 interrupt-handling function (Irq5_Handler() in VDCSC.C has been completed and the reference addresses in eax and ebx contain return addresses. The pre-IRQ5 contents of eax and ebx are restored, the interrupt flag (sti) has been set, ret ends the procedure returning control to Windows.

Vdcs_Int_2F_Proc

The IRQ2F handling function is more involved because there is a possibility that the 2F interrupt could have been set by some other non-VIR system program. To protect the system from this possibility, two loop tests are performed; one of them is in Vdcs_Int_2F_Proc, the other is in the DLL. Figure 3 depicts the AX,...DX register contents through the DLL-VXD-DLL sequence. The first test passes a constant value through the sequence. Mux_Handle_Base (0xE000) is loaded in ax by the DLL before it calls Vdcs_Int_2F_Proc. It is tested by the DLL after the data has been retrieved by Irq2f_Handler(). This test verifies that the DLL to Vdcs_Int_2F_Proc to Irq2f_Handler() and back to the DLL path has been properly executed. This value is also tested in Vdcs_Int2F_Proc but is called MY_MUX_ADDR. If ax contains this value, the 2F interrupt is considered to have been set by the DLL since the probability is very low that another non-Vdcs program could have first loaded this test value into ax and then set interrupt 2F. The contents of ax are tested by the and ax, MY_MUX_ADDR (bitwise logical AND) and cmp ax, MY_MUX_ADDR (Compare Two Operand) instructions. If the the values are unequal, interrupt 2F was not set by the DLL; some other non-VIR system program set it. In this event, eax is popped off the stack and a jne (Jump If Not Equal) to the Int_2F_Exit instruction at the end of the procedure.

The second test (a slight digression here) is performed by the DLL and uses the INTERRRUPT_2F_RESPOND value (0x008 from DCSDEF.H) loaded by Irq_2F_Handler (in VDCSC.C) into Client_BX and returned to DLL. If both the Mux_Handle_Base and INTERRUPT_2F_RESPOND have passed through the sequence unaltered, the DLL returns the Irq2F-requested data to SERVER via the CIN.

If the IRQ2F call from the DLL is valid, eax (containing Client_AX) is pushed onto the stack and the three other registers Client_BX, Client_CX and Client_DX are sequentially loaded into _Client_BX, etc., via the ax register. The instruction notation [ebp.Client_Bx], etc., indicates a data structure with ebp as the highest element and Client_BX as the subordinate element. Note that upon entry BX has the composite address (DCS, DSA and Mux), DX has the MW1/2 designator and DX has 0. The _ (underscore) prefix is added to the four register psuedo-variables. This is done to make the four register psuedo-variable arguments (Client_AX,..Clint_DX) brought into Vdcs_Int_2F_Pros appear "distinct" from the register variables that are returned to the DLL. The sequence of four register indirect instructions use the ebp.Client_BX (where ebp is the Extended Base Pointer) segment register as an offset in the register indirect instructions.

After the register transfers have been done, the **Irq2F_Handler()** in VDCSC.C is called; this obtains the address-designated monitor data value from the multidimensional data array in RAM. (The data array was described above.) Upon return, the four variables are loaded into _Client_AX,..._Client_DX by four transfers similar to those mentioned above; this makes them available to the DLL for its return call. Following this, eax is popped to its state prior to the IRQ2F interrupt. The procedure ends by setting the Interrupt Flag (stc) and a return. Note that there is not a VxDcall VPICD_Phys_EOI return as is the case in the IRQ5 interrupt return above.

Shell_Message_Proc

This procedure was used in debugging Vdcs and is not an operational function. The BeginMsg macro at the end of the VxD_Locked_Data_Segment defines the linkage parameters to this procedure.

VXD Batch Files

Several Batch files establish the compilation and operating environment for two components of SERVER's data retrieval code: VDCSC.C, and VDCSA.ASM. DCSDLL.C is not influenced by batch files; this is also the case with LabVIEW code. Remember that DCSDLL.C (compiled by the Watcom C compiler) is the linking library called by SERVER's CIN; VDCSA.ASM (assembled by MASM5) is the assembly language linkage between the DLL and VDCSC.C; VDCSC.C (compiled by the Watcom C compiler) is the MDI and RAM data array I/O driver. Figures 2 and 3 show the linkages and address-data transfers. Since Windows is the operating system, the three programs conform to Window's VXD conventions.

Operative system programs are **wcc386,** the Watcom C compiler for 32-bit address code and **masm5,** the Microsoft assembler version 5. **flatobj** makes all memory references 32-bit (in place of the segment register + offset address convention). **link386** is a linking loader. **addhdr** does a conversion to make a vxd file executable. **wdisasm** is a disassembler for the assembly code generated by wcc386.

There are two important files used in the batch files which deserve mention: **.lrf** and **vdcs.def.**

.lrf is a text file invoked in Buildall.Bat and combines the vdcsc and vdcsa object files into a single file, vdcs.386 for the linker, link386. It links library functions from vclib3.lib for vdcs386, and links vdcs.def (see below). **.lrf** also has user-selected convenience switch functions (shown in caps) that affect the screen display or output file.

.lrf is: vdcsc+vdcsa,vdcs386,,vclib3.lib,vdcs.def/INFORMATION/CODEVIEW/NOEXTDICTIONARY

LINENUMBERS assigns line numbers and MAP shows the vxd's virtual memory map; the other features are of a similar character.

The VDCSA.ASM listing has the note: To install this vxd, add "device=\path\VDCS.386" [in 386 Enh] in SYSTEM .INI.

vdcs.def (invoked in Buildall.Bat) defines the structure of the vxd files, vdcsc and vdcsa after the files have been linked. This file defines properties of the vxd macro calls: _ITEXT, _LDATA, _ITEXT, _TEXT, and _DATA, and is required by the Windows DDK. **vdcs.def** is not listed here because a description of these vxd calls is beyond the scope of this manual. The interested reader is referred to the two Windows references cited in the seventh paragraph of the **VDCSA.ASM Description** above.

Comments delimited by [] brackets in the file listings below briefly indicate the line's directive action. Note that there is some redundancy in the batch files. Directives are alternatively prefixed by a hyphen (-), which is equivalent to a slash (/). Operative system program directives are generally listed by a /h (help) command.

The directives preceded by / select features displayed on the monitor screen.

MASM5-related directives:

-d	generates a pass 1 listing
-p	checks for pure code
-s	orders segments sequentially
-Zi	generates symbolic information for CodeView
-Zd	generates line number information
-w2	set warning level 2
-Mx	preserve case of labels

"Batch" files used with VDCSA.ASM and VDCSC.C are:

A.Bat:

```
masm5 -p-w2-Zi-Zd-Mx vdcsa;                        [operates with MASM during assembly]
pause
```

C.Bat:

```
wcc386 vdcsc -dl -w4 -mf -s -z1 -nt=LTEXT -nd=W386   [operates with the Watcom C compiler]
pause
```

Buildall.Bat:

```
wcc386 -od -w4 -mf -s -zl -nt=LTEXT -nd=W#*^ vdcsc   [operates with the Watcom C compiler]
masm5 -p-w2-Zi-Zd-Mx vdcsa;                          [performs MASM functions listed above]
flatobj vdcsc                                        [converts vdcsc to 32-bit addressing mode]
link386 @vdcs.lrf                                    [links .lrf]
addhdr vdcs.386                                      [see "device=\path\VDCS.386" note above]

pause
```

D.Bat:

```
wdisasm vdcsc -1=vdcsc.txt -s=vdcsc.c -a -e -p -b
            edit vdcsc.txt
```

All.Bat:

```
a
c
l
m
pause
```

```
// DCSDLLA.C  3.7.95 P. Dooley
#include <windows.h>
#include <dos.h>
#include "dcsdef.h"
//#define INT2F_UPDATE 0X00000080
// A .dll that responds to a call from a DOWNSHIFT .vi
// DCS data is returned to the VI via function dcsdat.
// 7.28.95 Removed references to channels - VI will handle channels
//----------------------------------------------------------------------
DWORD FAR PASCAL _export dcsdata (WORD AntAddr,
                                  WORD DataSetAddr,
                                  WORD MuxAddr,
                                  WORD MwNum)
{
  WORD wSelector;
  DWORD LinearAddress,LinearArrayAddress;
  DWORD LongTemp=0;
  WORD TempInt=0,TempInt1;
  WORD RegAX,RegBX,RegCX,RegDX;

  RegAX = MUX_HANDLE_BASE;
  RegBX = (AntAddr << 11) | (DataSetAddr << 8) | MuxAddr;
  RegCX = MwNum;
  RegDX = 0;
  // Note: If intr 0X2F VxD or TSR cannot be found it will not'fail
  // It only fails if the interrupt routine which responds
  // to 2F has a different MUX HANDLE
  asm
  {
    pusha
    mov ax,RegAX
    mov bx,RegBX
    mov cx,RegCX
    mov dx,RegDX
    int 2fh
    mov RegAX,ax
    mov RegBX,bx
    mov RegCX,cx
    mov RegDX,dx
    popa
  }

  if(!(GetWinFlags() & WF_PMODE)) // Check to see if we are in the
    return 0XBAD10000L;           // right windows mode
  else
  {
    if(((RegAX & MUX_HANDLE_BASE) == MUX_HANDLE_BASE) &&
        ((RegBX & INTERRPUPT_2F_RESPOND) == INTERRPUPT_2F_RESPOND))
    {
      LongTemp = RegDX;
      LongTemp = LongTemp << 16;
      return (LongTemp | RegCX);
    }
    else
      return 0XBAD20000L; // No response from the Int 2F call to VxD
  }
}
//----------------------------------------------------------------------
int FAR PASCAL _export LibMain(HINSTANCE hInst,
                               WORD wDataSeg,
                               WORD cbHeapSize,
                               LPSTR lpszCmdLine)
{
  if (cbHeapSize)
    UnlockData(0);
  return (TRUE);
}
//-----------------------------------------------------------------------------
int FAR PASCAL WEP(int bSystemExit)
{
  return WEP_FREE_DLL;
}
//*****************************************************************************
```

```
// VDCSC.C
// P. Dooley 1/9/95
// Functions for VDCSA.ASM
// Note: Since this is being compiled as a 32 bit flat model - all integers
// are treated 32 bits by the WATCOM compiler
#include "dcsdef.h"
#include "conio.h"
#include "stdlib.h"
#include "dos.h"
#include "windows.h"
#include "hungaria.h"
#include "vxd.h"

#define DATA_BIT 0x01
#define TRACK_HOLD 0x02
#define MONITOR_TIME  0x04
#define INT70 0x70
#define STROBE_BIT 0x01
#define ACK 0x40

WORD WaveGuideTick=0;
// Big Array containing current Monitor Data from the VLA

WORD MonData[32]  // 32 Ultimate DCS Antenna Adresses
            [6]   // 6 Data Set Addresses
            [256] // 256 Monitor Word Addresses
            [2]   // 2 Monitor Words
            [2];  // 2 Data Words;

BYTE MwTable[12] = {0,0,0,0,0,1,1,1,1,1,0,1};
//BYTE MwTable[12] = {0,0,0,0,0,0,1,1,1,1,1,1};
extern WORD Client_AX,Client_BX,Client_CX,Client_DX;
BYTE The_Shell_Message[12];
extern  Shell_Message_Proc();
BYTE  ParityByte;
//------------------------------------------------------------
// This function responds to IRQ 5 "DATA AVAIL"
// Read in 12, 4 word messages.
// Compare Ant. addresses of messages - if all equal store away.
// Do this 32 times for 32 antennas.
void Irq5_Handler()
{
  WORD i,j,InWord0,InWord1,InWord2,InWord3,AntAddr,MuxAddr,Dsa,Mw,MwCtr;

  i = j = MwCtr = 0;
  WaveGuideTick++;
  if(WaveGuideTick == 192)
    WaveGuideTick = 0;
  InWord3 = FIFO_EFN;

  while((InWord3&FIFO_EFN) == FIFO_EFN) // Loops up to 316 times
  {
    InWord0 = inpw(DCS_BASE); // Automatically increments read ptr in FIFO
    InWord1 = inpw(DCS_BASE); //      "              "        "   "   "   "
    InWord2 = inpw(DCS_BASE); //      "              "        "   "   "   "
    InWord3 = inpw(DCS_BASE+6);

    if(i > 383)
      InWord3 = 0;

    AntAddr = (InWord0 >> 11) & 0x1F;
    Dsa = (InWord0 >> 8) & 0x0007;
    MuxAddr = InWord0 & 0x00FF;

    if((AntAddr > -1) &&  // Check to see if Array addressing is in bounds
        (AntAddr <= 27) &&
        (Dsa > -1) &&
        (Dsa <= 5) &&
        (MuxAddr > -1) &&
        (MuxAddr <= 255))
      {

       Mw = MwTable[MwCtr];
//       Mw = 0;
//       ParityByte = InWord2 & 0xFF;  // For future parity checking
       // Hi word contains 16 MSBits of analog data
       // Lo word contains 8 LSBits of analog data + WaveGuideTick cnt in the 8 LSBits of
d
       MonData[AntAddr][Dsa][MuxAddr][Mw][0] = (InWord2 & 0xFF00) | WaveGuideTick; // Lo Wc
       MonData[AntAddr][Dsa][MuxAddr][Mw][1] = InWord1;  // Hi Word
//       MonData[0][0][0][0][0] = 0;
//       MonData[0][0][0][0][1] = InWord0;
      }
     i++;
     j++;
     if(j == 28)
     {
       j = 0;
       MwCtr++;
     }
   }
}
//---------------------------------------------------------------------------
// This function responds to interrupt 2F from a Windows DLL
void Irq2F_Handler(void)
{
  WORD DcsMwNum,DcsAntAddr,DcsDsa,DcsMuxAddr;
//  Rx'd Parameters
  DcsAntAddr = (Client_BX >> 11) & 0x001F;
  DcsDsa = (Client_BX >> 8) & 0x0007;
  DcsMuxAddr = Client_BX & 0x00FF;
  DcsMwNum = Client_CX & 0x1;
// Returned Data
  Client_BX = INTERRPUPT_2F_RESPOND;
  Client_CX = MonData[DcsAntAddr][DcsDsa][DcsMuxAddr][DcsMwNum][0]; // Lo Word
  Client_DX = MonData[DcsAntAddr][DcsDsa][DcsMuxAddr][DcsMwNum][1]; // Hi Word
}
//---------------------------------------------------------------------------
void Initialization(void)
{
  WORD i,j,k;

  for(i=0;i<32;i++) // Antenna addresses
  {
    for(j=0;j<6;j++)  // Data Set addresses
    {
      for(k=0;k<256;k++)  // Mux addresses
      {
        // Monitor Word 1
        MonData[i][j][k][0][0] = 0x0100;
        MonData[i][j][k][0][1] = 0xdead;
        // Monitor Word 2
        MonData[i][j][k][1][0] = 0x0200;
        MonData[i][j][k][1][1] = 0xdead;
      }
    }
  }
}
//---------------------------------------------------------------------------
void Debug_Routine(void)
```

```
{
  if((Client_AX & MUX_HANDLE_BASE) == MUX_HANDLE_BASE)
  {
    itoa(Client_AX,The_Shell_Message,16);
    Shell_Message_Proc();
  }
}
//-----------------------------------------------------------------
```

```
comment~
VDSCA.ASM P. Dooley 1/9/95
VDCS - A VxD to handle interrupts from a Custom interface board @ 19.2 Hz
        (VLA Waveguide Polling Frequency - Modeled after VxD MonDel
To install this VxD, Add "device=\path\VDCS.386" in [386Enh] in SYSTEM.INI
~
.386p
INCLUDE VMM.INC
INCLUDE VPICD.INC
INCLUDE SHELL.INC
EXTRN   Irq5_Handler_:NEAR  ; C routine that services interrupts
EXTRN   Irq2F_Handler_:NEAR ; C routine that communicates with IRQ 2F
EXTRN   Initialization_:NEAR ; C routine to initialize other program elements
EXTRN   Debug_Routine_:NEAR
EXTRN   _The_Shell_Message:BYTE
;***************************************************************************
;                    E Q U A T E S
;***************************************************************************
verMaj          EQU 1
verMin          EQU 0
ver             EQU ((verMaj SHL 8) OR verMin)
PORT_ADDR       EQU 300h
IRQ5            EQU 5
IRQ2F           EQU 2Fh
NUM_PORTS       EQU 4
Vdcs_Init_Order EQU 0C0000000h  ; init after the shell
MY_MUX_ADDR     EQU 0000E000h
;***************************************************************************
;       V I R T U A L   D E V I C E   D E C L A R A T I O N
;***************************************************************************
Declare_Virtual_Device Vdcs     \
            , verMaj            \
            , verMin            \
            ,Vdcs_VxD_Control   \
            ,Undefined_Device_ID\
            ,Vdcs_Init_Order     ;Device not defined by Microsoft(vmm.inc)
;***************************************************************************
;                   D A T A   S E G M E N T
;***************************************************************************
VxD_LOCKED_DATA_SEG

; IRQ descriptor Sructure elements
    ; VID_IRQ_Number
    ; VID_Options
    ; VID_Hw_Int_Proc
    ; VID_Virt_Int_Proc
    ; VID_EOI_Proc
    ; VID_Mask_Change_Proc
    ; VID_IRET_Proc
    ; VID_IRET_Time_Out

; IRQ descriptor to virtualize Irq 5
Vdcs_IRQ_Descriptor VPICD_IRQ_Descriptor\
    <\
      IRQ5,                        \
      ,                            \
      OFFSET32 Vdcs_Hw_Int,        \
      ,                            \
      ,                            \
      ,                            \
      ,                            \
                                   \
    >

; IRQ descriptor to virtualize Irq 2F
```

```
Vdcs_IRQ_2F_Desc VPICD_IRQ_Descriptor\
    <\
      IRQ2F,                          \
      ,                               \
      ,                               \
      ,                               \
      ,                               \
      ,                               \
      ,                               \
                                      \
    >

    Irq5_Handle          dd     0
    Irq2F_Handle         dd     0
    VM_Handle            dd     0
    Vdcs_Next_I2F_CS     dd     ?
    Vdcs_Next_I2F_EIP    dd     ?
    MemBlockHandle       dd     ?

    ; Global Variables used by "c" program
    PUBLIC  _Client_AX
    PUBLIC  _Client_BX
    PUBLIC  _Client_CX
    PUBLIC  _Client_DX
    PUBLIC  _MemArrayAddress

    _Client_AX         dw    ?
    _Client_BX         dw    ?
    _Client_CX         dw    ?
    _Client_DX         dw    ?
    _MemArrayAddress   dd    ?

comment~
BeginMsg
PUBLIC The_Shell_Message
The_Shell_Message label byte
  db  " Debug Message #1",0
EndMsg
~

BeginMsg
PUBLIC  The_Caption_Message
The_Caption_Message label byte
  db  " Vslc Debug Point #1",0
EndMsg
PUBLIC  Shell_Message_Proc_
VxD_LOCKED_DATA_ENDS
;*****************************************************************************
;          L O C K E D   C O D E   S E G M E N T
;*****************************************************************************
VxD_LOCKED_CODE_SEG

BeginProc Vdcs_VxD_Control  ; Vxd Initialization

    Control_Dispatch Device_Init,InitVdcs ; returns VM_Handle in EBX
    mov [VM_Handle],ebx
    clc
    ret

EndProc Vdcs_VxD_Control
;----------------------------------------------------------------------------
BeginProc InitVdcs

    ; Hook I/O ports starting with port # in EDX.
    push  ecx       ; Save loop counter
    push  esi       ; and data structure pointer
    push  edx

    mov edx,PORT_ADDR
    mov esi,OFFSET32 Port_Trap_Addr ; ESI -> Procedure to call
    mov ecx, NUM_PORTS              ; ECX = # of ports to hook
VPD_ITrap_Loop:
    VMMcall Install_IO_Handler      ; Install port hook
    inc edx                         ; EDX = Next device port to hook
    loopd VPD_ITrap_Loop            ; Loop until all hooked

    pop edx
    pop esi            ; Restore
    pop ecx

    ; Get the Irq5 handle
    push  edi
    mov   edi, OFFSET32 Vdcs_IRQ_Descriptor
    VxDcall VPICD_Virtualize_IRQ
    mov   [Irq5_Handle], eax          ; handle is returned in eax
    pop   edi

    ; Unmask in the Virtual Programmable Interrupt Controler "Irq5"
    push  eax
    mov   eax, [Irq5_Handle]
    VxDcall VPICD_Physically_Unmask ; This takes care of the interrupt ctlr

    ; Set up to respond to Int 2F
    mov   eax, 2Fh
    mov   esi, OFFSET32 Vdcs_Int_2F_Proc
    VMMcall Hook_V86_Int_Chain
    pop   eax

    call  Initialization_
    sti                               ; Global interrupt enable
    ret

EndProc InitVdcs
;---------------------------------------------------------------------------
BeginProc Vdcs_Hw_Int, High_Freq

    call  Irq5_Handler_   ; Jump to "C" routine to do real stuff
    push  eax
    push  ebx
    mov   eax, [Irq5_Handle]
    mov   ebx, [VM_Handle]
    VxDcall   VPICD_Phys_EOI  ; Do an EOI
    pop   ebx
    pop   eax
    sti
    ret

EndProc Vdcs_Hw_Int
;--------------------------------------------------------------------------
BeginProc Vdcs_Int_2F_Proc, High_Freq

    ; Find out if this multiplex address is for me
    push  eax
    mov   ax, [ebp.Client_AX]
    mov   [_Client_AX], ax
    and   ax, MY_MUX_ADDR
    cmp   ax, MY_MUX_ADDR
    pop   eax
    jne   SHORT Int_2F_Exit   ; !=0, This Multiplex address is not for me
    push  eax
    mov   ax, [ebp.Client_BX] ; Get the sender's regs
    mov   [_Client_BX], ax
    mov   ax, [ebp.Client_CX]
    mov   [_Client_CX], ax
    mov   ax, [ebp.Client_DX]
    mov   [_Client_DX], ax
    pop   eax

    call  Irq2F_Handler_      ; Do the necessary work in "C"

    push  eax
    mov   ax, [_Client_AX]    ; Return Regs to sender
    mov   [ebp.Client_AX], ax

    mov   ax, [_Client_BX]
    mov   [ebp.Client_BX], ax

    mov   ax, [_Client_CX]
    mov   [ebp.Client_CX], ax

    mov   ax, [_Client_DX]
    mov   [ebp.Client_DX], ax
    pop   eax

Int_2F_Exit:
    stc
    ret

EndProc Vdcs_Int_2F_proc
;-----------------------------------------------------------------------------
BeginProc Port_Trap_Addr
    ret
EndProc Port_Trap_Addr
;-----------------------------------------------------------------------------
BeginProc Shell_Message_Proc_

    pushad
    mov   eax, 0
    mov   ebx, [Vm_Handle]
    mov   ecx, OFFSET32 _The_Shell_Message
    mov   edi, OFFSET32 The_Caption_Message
    mov   esi, 0
    VxDcall SHELL_Message
    popad
    ret

EndProc Shell_Message_Proc_
;-----------------------------------------------------------------------------
VxD_LOCKED_CODE_ENDS
    END
;*****************************************************************************
```

```
// DCSDEF.H
// 11/4/93
// P. Dooley
// Header file for SLCCTL.C & RXSLC.C
#define TIMER_TICK 0x1C
#define DCS_BASE 0x300
#define DCS_IRQ_5 0x0D
#define DSA_SIZE 6
#define MUX_ADDR_SIZE 256
#define MON_WORD_SIZE 2
#define DCS_DATA_SIZE 3
#define DCS_MSG_MAX 192
#define MUX_HANDLE_BASE 0xE000
#define _CARRY_FLAG 0x0001
#define MW1 0
#define MW2 1
#define INTERRPUPT_2F_RESPOND 0x0008
#define FIFO_EFN 0x0010
```

Monitor Data Interface

These paragraphs describe the VIR system's Monitor Data Interface (MDI) which detects and temporarily stores 384 monitor data messages during one VLA cycle; at the end of the storage period, the MDI generates an IRQ5 and the messages are read by the PC's SERVER-DLL-VXD software.

The MDI is functionally similar to the VLA Serial Line Controller (SLC); see VLA Technical Report No. 63 for a description of the VLA SLC's operation. The MDI is an adaptation of the Single DCS SLC that operates under control of an AT-class, X86 PC. This adaptation consists of the addition of a FIFO data storage memory and IRQ5 interrupt logic. The MDI does not use some of the Single DCS SLC features; therefore they are not described in this manual. The Single DCS SLC logic schematic is C13710L02; it is not included in this manual.

The MDI logic schematic is C81002L01 and follows this section's text. The MDI's assembly drawing is D81002P001. The MDI uses custom-built DIP headers that are installed on the MDI board; these headers are described by assembly drawing A81002P002. The assembly drawings are not included in this manual.

The MDI is constructed on a JDR Microdevices JDR-PR10 prototype board that is installed as a system board in the PC case. The board has AT Bus 62 pin and 36 pin connectors; monitor data message and timing signal lines are connected via a 37-pin DB37 connector. The JDR PR10 is a general-purpose PC AT bus interface board that is particularized by the user to implement special-purpose functions that are not commercially available. Some of the MDI's logic elements are Programmable Array Logic (PAL) chips. The ADDRESS BUS and ADDRESS DECODE (U2 through U7, Sheet 5) PALs are provided by JDR; the balance (U16, U17, U23 and U24) are Single DCS SLC PALs. The PAL equations are not included in this text but are described in "The Single-DCS Serial Line Controller, VLA Techincal Report No. X". The PR10 is equipped with I/O bus buffers and I/O control PALs to simplify interface logic design; these are U1,..U10 on Sheet 5. The interface logic uses the 20 bit address bus, SA00,...SA19 on the ADD header, the 16-bit data bus, SD00,...SD15 on the I/O1 and I/O2 headers, and the U7 and U8 PAL logic outputs on headers SEL1 and SEL2. The latched address bits on the LA header are not used. The JDR-provided PALs and I/O line buffers connect to the custom-design interface circuitry via wire wrap pins soldered to JDR-provided ADD, SEL1, SEL2, MMEM1 and MEM2 header pads. The SLC interface circuitry is wire-wrapped on WW IC sockets.

The VLA M&C System uses time-serial command and data messages; see Appendix A for additional details on this system. The SLC outputs computer-generated command messages to Central Buffers and uses "Q" characters to poll monitor data messages from the Buffers for input to the computers. The MDI is tapped onto the stream of serial monitor data messages evoked by the SLC.

Command and monitor data message formats are identical with the exception of the multiplex address ranges. Addresses 0 to 191_D are reserved for monitor data and addresses 192 to 255_D are reserved for commands. The messages are prefixed by a 10-bit S (synchronizing) character and consist of five address and data bytes; each byte is followed by a parity bit that forms odd parity over the preceding byte. The first byte consists of five DCS and three Data Set address bits; the second byte consists of eight Multiplex address bits; the third, fourth and fifth bytes can be either a 24 bit argument or in the case of converted analog data, two 12-bit arguments. The MDI data bit rate is 1 MHZ and the S and Q characters bit rate is 2 MHz. All data is NRZ.

The MDI logic includes a COMMAND STORAGE register and a Q CHARACTER generator which outputs 12 Qs; these are Single DCS SLC functions not used by the MDI, therefore they are not described.

In the description below, circuit functions are designated by the upper-case labels used on the schematic diagram.

TimeBase and Timing Discretes

In the VLA, time governs all machine operations; consequently this is the case in the M&C system which is time-referenced to the VLA's 19.2 Hz cycle by the T/H signal from the Control Room L8. The MDI uses the T/H reference.

The VLA 19.2 Hz machine cycle period is 52,083.333,.. μs. The MDI 19.2 TIMEBASE counter (Sheet 2) can be reset in two ways: 1) a reset by the External T/H from the Master LO Rack's L8 - the actual usage; 2) an internally-generated QQ from U16 in the event that an External T/H signal is not available. This feature is not used.

The External (L8) T/H signal goes low at -106 μs and returns high at +1500 μs. U49A is triggered by the leading (falling) edge of External T/H and, with minor trim-adjustment of R2, provides a 106 μs delay. The falling edge of U49A's Q output is the time base zero or reset time and triggers U49B which produces about a 250 ns QQ pulse on its Q output. This QQ derived from the External T/H is the normal time base reset. U49B's Q output is connected to one input of a 74128 NOR driver (U51B); the driver's low-true output QQ* clears five of the six stages (U12,..U15) of the time base.

U49A's low-true QN output, MRFN resets the FIFO's address registers.

When the T/H is not available, the MDI's internal TIMEBASE is reset by a slightly slower (52,088 μs period) signal decoded from the TIMEBASE counters. The other 74128 (U51B, mentioned above) input QQ is generated by the QQ equations in U16 (DCS_PAL1). U16-22's QQ period is 52,088 μs, 5 μs longer than the standard 52,0833.3 μs period.

The MDI's TIMEBASE consists of a 10MHz crystal clock, U18 and five stages of 74LS161 binary counters, U11,..U15. Note that U11's QD output (1 MHZ) clocks the other four stages. The time base resolution is 0.1 μs.

Three control discretes are generated in U16 (DCS_PAL1) from TIMEBASE counter terms: A) MON_TIME (-104 to 22,272 μs); B) Q_PERIOD (1024 to 13,312 μS); C) CMD_TIM (28,952 to 49,456 μs). The TIMEBASE circuitry and PAL equations are described in the Single-DCS SLC manual mentioned above. The interested reader is referred to this manual for additional details on the operation of the time base logic.

The only time-base derived discrete signal that the MDI uses is MON_TIME. Its trailing edge clock-sets the DATA READY INTERRUPT TO PC flip flop U54B (74LS74); this activates the PC's low-true IRQ5 line on J1. Timing term 8 (for 8μs) from the TIMEBASE counter clears U54B.

The MDI does not use the CMD_TIME, Q_PERIOD and QQ signals; they are mentioned because they are intrinsic functions of the timing logic. The internally generated T/H signal, the QQ output, 5 MHz and CMD_Q_GOSSIP signals are not used.

Internal Data Bus

An internal two-byte, tri-state, parallel DATA BUS (D0,..D15, Sheets 3 and 5) enables transfers between the PC's SD00,...SD15 I/O lines, the STATUS FLAGS part (1 byte) of the MONITOR DATA STORAGE AND FLAGS register, and the COMMAND STORAGE register. Direction control and enable terms on bus drivers U5 and U6 control the data flow direction (into or out of the PC) and assert the driver's outputs on the bus.

The two FIFO's outputs are asserted on the PC's SD00,..SD15 I/O lines by U6 and U10. Each time the PC executes an InputWord instruction, the low-true RD1N signal reads two bytes from the FIFO's "Q" outputs and asserts them on the SD00,...SD15 lines.

Note from the drawings that the DATA REGISTER can only assert data on the D00,..D07 portion of the DATA BUS. This is not a problem because the register's data is input to the PC via the FIFOs.

The DATA BUS drivers are 74LS245 octal tri-state, bi-directional bus drivers. A low on the G inputs enables the outputs; a high forces the outputs to the high impedance state. A high on the DIR input causes A to B data flow and a low causes the converse. The MDI's design is inherently capable of both 8 and 16-bit transfers but has been jumpered (JMP tie to Gnd) to operate in the 16-bit mode. An 8-bit capability requires separate enables on the G inputs of U1 and U5; the LO_ENABLE on U5 enables lower byte transfers and HI_ENABLE on U1 enables high byte transfers. Both enables are active in 16-bit transfers.

Monitor data messages are serially loaded into the MONITOR DATA STORAGE AND FLAGS register, U32,..U39 by the MONITOR DATA SYNC AND TIMING logic described below. The data register DM8546 chips have tri-state outputs that are enabled by a low-true enable on pin 9. The message's 45-bit data value (5 data bytes and 5 parity bits) is input to the two FIFOs (U57 and U58) "D" inputs in three sequential, two-byte transfers by three low-true output enables (RD1N, RD2N and RD3N) on U32-U37, U33-U38 and U34-U31, respectively. These read enables are described in the FIFO Read and Write paragraphs below.

When reading the STATUS FLAGS from the MONITOR DATA AND STATUS FLAGS register, the DIR inputs of both U5 and U1 are driven low by !BIOR (buffered -IOR) from U7, pin 18; data flow is from the registers to the PC's SD00,.. SD15 lines. The only STATUS FLAG term of interest to the VDCSC.C software is the FIFO Empty Flag (EFN) discrete.

FIFO Read and Write

The FIFOs are First In/First Out 9-bit memories with an internal address register. The IDT 7203 has a 2K X 9 capacity; this paragraph cites only the properties that are important in the MDI usage. A low strobe on the Write (WRN) or Read (RDN) pins writes data into or reads data out of the FIFO, respectively. The internal address register is incremented after a read and decremented after a write and is reset to zero by a low strobe on the Reset (MRFN) pin. An Empty flag discrete (EFN) goes low to indicate that all data has been read out. The FIFO's ninth data bit D8-Q8 are not used.

The FIFO's address register is reset to address 0 by the low-true MRFN signal from U49A in the TIMEBASE logic at the end (at -106 μs) of the VLA cycle. This initializes the address register in preparation for the software start-up; after the start-up, the normal FIFO write-read cycle will return the address register to 0.

When the MONITOR DATA STORAGE register has been loaded with a new monitor data message, the MON DONE term from the MONITOR DATA SYNC AND TIMING logic (described below) clocks U52B true. The rising edge clocks a 0 into U54A, the input to a sequence generator (U55) that generates the low-true RD1N, RD2N, RD3N and RD4N terms. U54A's Q output is connected to U55's Data input. When U54A's Q goes low, a 0 is shifted into U55's first stage (QA) by the rising edge of the 2 MHz clock. This term is RD1N (a data register read strobe) that is also fed back to U54A's direct set input, quickly making it's Q output a 1. The next rising edge of the 2 MHz clock shifts U54's 1 into U55's QA; RD1N goes high and U55's QB is set to 0 (low). The 0 state is thus shifted through U55 by the clock; after eight shift clocks, U55's Q outputs are all 1s (high). U55's RD1N,..RD4N low-true outputs are separated by one shift register stage; as a result, each term is low-true for 500 ns and the four terms are spaced at 1 μS time intervals. These four terms do two things: first, they each sequentially enable two bytes of the MONITOR DATA STORAGE register onto the FIFO inputs; second, they generate four FIFO WRN strobe terms via the four input, low-true OR gate U56A (74LS21). RD1N enables U32 and U37 Q outputs onto the bus, RD2N enables U33 and U38, etc. U32 contains the message's MSB so RD3N reads the data's LSB and parity bits. RD4N does not enable any data bits onto the DATA BUS but does strobe the FIFO's WRN line (via UU56A) so there are four FIFO data writes, the last being the disconnect state of the bus which could be either 0's or 1's. The state is not important because, when the FIFO contents are read out, the fourth readout (via READ4) are the STATUS FLAGS in U30, not the fourth (bus disconnected state) value in the FIFO.

Note that the FIFO read strobes (pin 15) and the U6 and U10 (74LS245) bus driver output enables are READ1 from U24, (DCS_PAL4). This output is low-true when the PC's I/O bus select address is 300h. Referring to the FIFO data read code in VDCSC.C: in the sequence of four FIFO read instructions, InWord0,...InWord3, the first three InWordn instructions use the select address DCS_BASE (300h); this reads the three sets of 16-bit data values from the FIFO's. The InWord3 instruction uses BCS_BASE+6 as the select address (306h). Select address 306h makes READ4 from U24 low true; this reads out the STATUS FLAGS register U30. The FIFO Empty Flag (EFN) on pin 21 is connected to pin 6 on U30; if the FIFO is empty (all contents read out), U30's readout will be XXX1XXXX, where X denotes "don't Care". VDCSC.C's present version does not test the Empty Flag or message parity bits in the STATUS FLAGS register.

Note that the DATA BUS is only used to read the eight STATUS FLAGS bits (U30); this transfer uses only U5.

Monitor Data Sync and Timing

In response to a Q character from the SLC, the selected DCS emits a monitor data message. The MONITOR DATA SYNC AND TIMING logic detects the message and loads it into the MONITOR DATA STORAGE AND STATUS FLAGS circuitry (Sheet 3) described below.

This process involves the following: detection of the data bit edges, development of a shift clock synchronized to the message bit edges, sampling the data bits in the middle and loading them into a shift register to detect the message's S character. When the S character is detected, load clocking logic generates shift clocks to load the message's 40 data bits and 5 parity bits into the data register.

The input data stream is ground-isolated by an optical isolator (U50) to eliminate common-mode noise effects. The isolator's gate input is connected to INHIBIT*, described below. The isolator's output, GOSSIP DATA is input to the JK* input of U42 (74LS195). U42 and U43 (74LS160) are clocked at 10Mhz.

A 2 MHz clock synchronized to the data bit edges is developed from U42's QA and QB outputs, DS1 and DS2 by U43 (74LS160) and the SYNC_CLK equation in U23 (DCS_PAL3). U43 operates as a divide by 5 counter clocked at 10 MHz. The 10 MHz clock and U23's SYNC_CLK* signal on U43's load input set a count of 5 in U43; it then sequences through states 5, 6, 7, 8, 9, 5, 6,,.. etc. SYNC_CLK* is formed in U23 by OR-ing U43's RCO (Ripple Carry Out) with the XOR of DS1 and DS2. Remember that the S character bits are 500 ns wide, so U43's state sequence is synchronized to the data bit edges. U43's QD output rises at state 8 and is U44's 2MHz clock.

The next step in monitor data message detection and loading is to detect the 10-bit S character that prefixes each message. This is done by the S_DET equation in U17 that detects the character's bit pattern in shift register U44, a 74LS164 (serial input, 8-stage parallel out). DS2 is connected to U44's A and B data inputs and the 2MHz shift clock samples DS2 300 ns after each S bit edge. As the S bits shift through U44, the S_DET equation outputs the low-true S DET* signal. For simplicity in implementation, only eight of the ten S bits are detected; the first two bits are assumed to be correct.

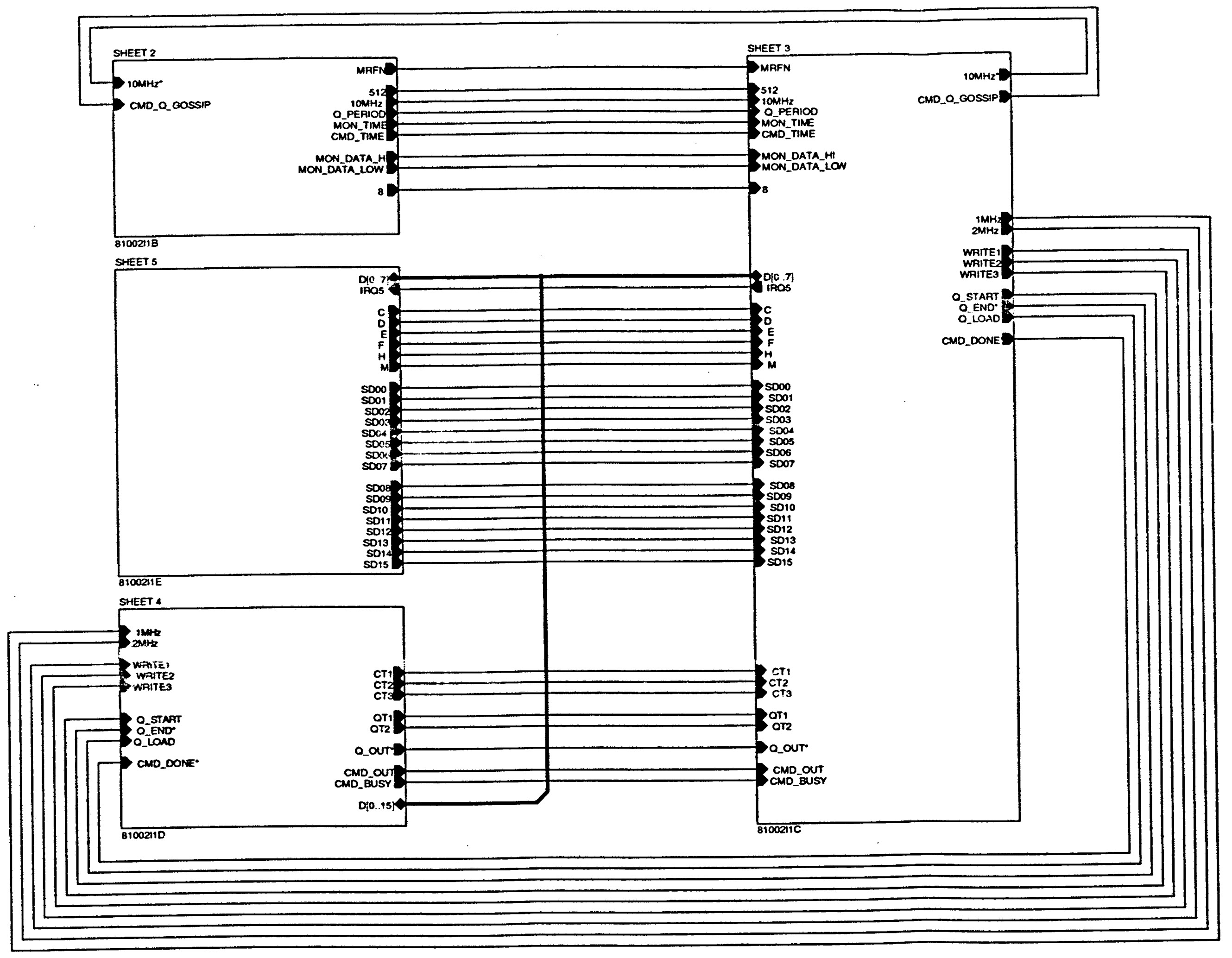
When the S is detected, 45 shift clocks must be generated to load the message data into the MONITOR DATA AND STATUS FLAGS register, U30,...U39. this is done by binary counters U47, U48, D flip-flop U52A (7474) and the MON_END equation in U17 (DCS_PAL2). S DET* clears binary counters U47 and U48 (74LS161s) that generate four timing terms, MT1,..MT4 which are input to U17 (DCS_PAL2). The message data bit rate is 1 MHz; toggling D flip-flop U52A divides the 2MHz shift clock from QD in U43 to provide the U47 and U48 clock.

The MON_END* output of U17-19 (DCS_PAL2) provides a low-true version of the MON_END equation which is an AND of the four timing terms, MT1,...MT4 mentioned above. This line is high when the equation is not true (high) and enables the counter's ENT and ENP inputs. When the four terms are true (high) at 45 counts, the counters are disabled and remain at the 45 count state.

Toggling flip-flop U52B is initialized to the reset state by the SYNC_LOW_CLK* output from U17-21. This is generated by the equation SLC = NOT S_DET OR NOT T1MHZ. If S has not been detected or the 1MHz clock (U52A-5) is low, this output is low, holding U52A reset. When the S is detected, this term goes high, enabling U52B to toggle to produce a properly-phased 1 MHz clock.

The MONITOR DATA AND STATUS FLAGS register is clocked by the 1 MHz LOAD CLKS term generated by the GLC (gated load clocks) equation in U17; this provides the 45 clocks required to shift the data and parity bits into the register. This NOT GLC term is generated by the AND of the four MT1,..MT4 terms of U47 and U48 OR the 1MHz output of U52B mentioned above. Remember that U52B is held reset until the S is detected.

REV.	DATE	DRAWN BY	APPRV'D BY	DESCRIPTION



ORCAD : 81002L1A

D81002P02	HEADERS
D81002P01	ASSEMBLY
NEXT ASSEMBLY	DWG TYPE

UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES: ANGLES ±
3 PLACE DECIMALS(.XXX): ±
2 PLACE DECIMALS (.XX): ±
1 PLACE DECIMALS (.X): ±

MATERIAL:

FINISH:

VLA PROJECT: VIRTUAL INSTRUMENT RECORDER

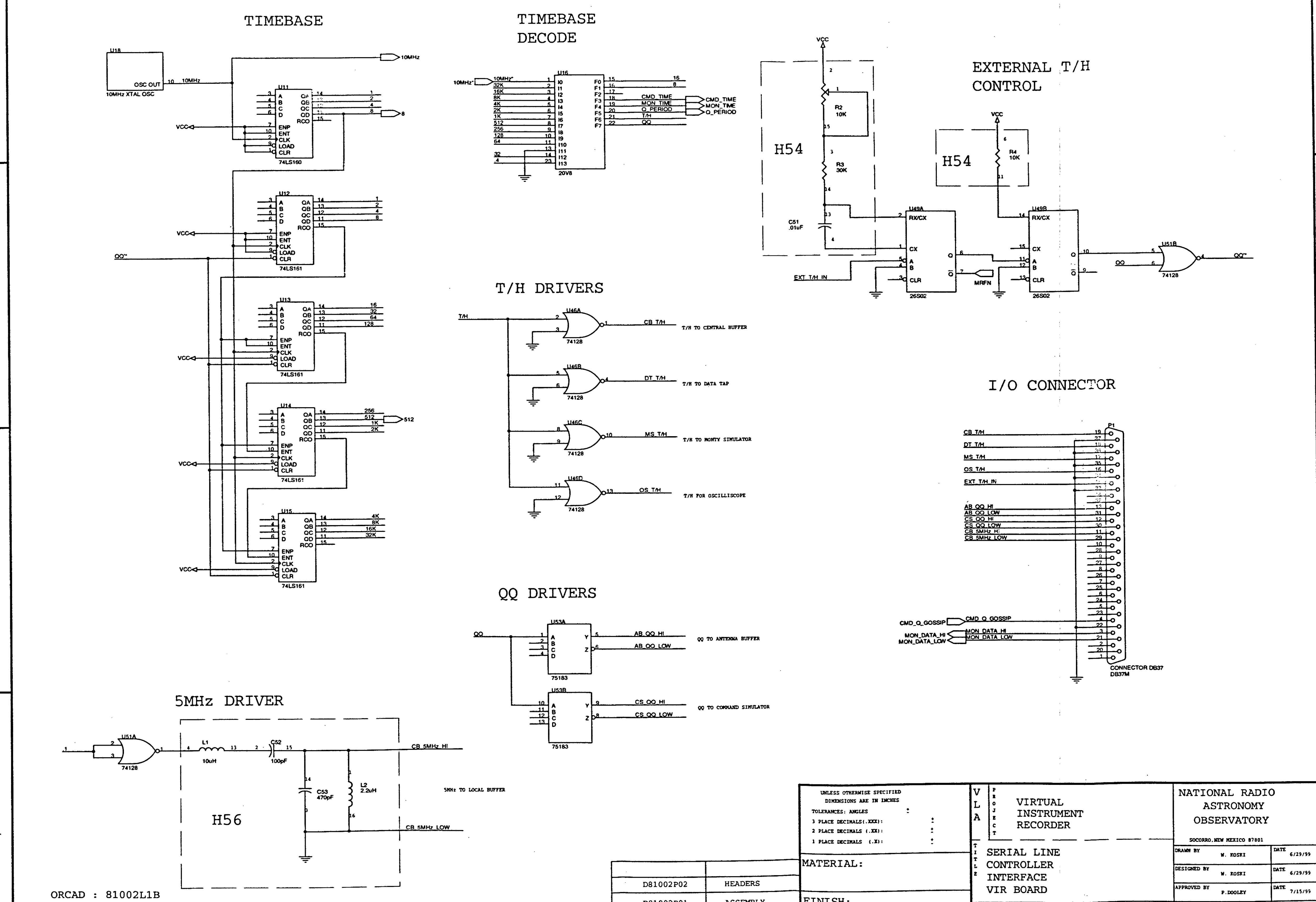
TITLE: SERIAL LINE CONTROLLER INTERFACE VIR BOARD

NATIONAL RADIO ASTRONOMY OBSERVATORY
SOCORRO,NEW MEXICO 87801

DRAWN BY	W. KOSKI	DATE	6/29/99
DESIGNED BY	W. KOSKI	DATE	6/29/99
APPROVED BY	P.DOOLEY	DATE	7/15/99

SHEET NUMBER 1 OF 5 | DRAWING NUMBER C81002L01 | REV. - | SCALE NONE

PROPERTY OF NRAO



REV.	DATE	DRAWN BY	APPRV'D BY	DESCRIPTION

UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES: ANGLES ±
3 PLACE DECIMALS(.XXX): ±
2 PLACE DECIMALS (.XX): ±
1 PLACE DECIMALS (.X): ±

MATERIAL:

FINISH:

D81002P02	HEADERS
D81002P01	ASSEMBLY
NEXT ASSEMBLY	DWG TYPE

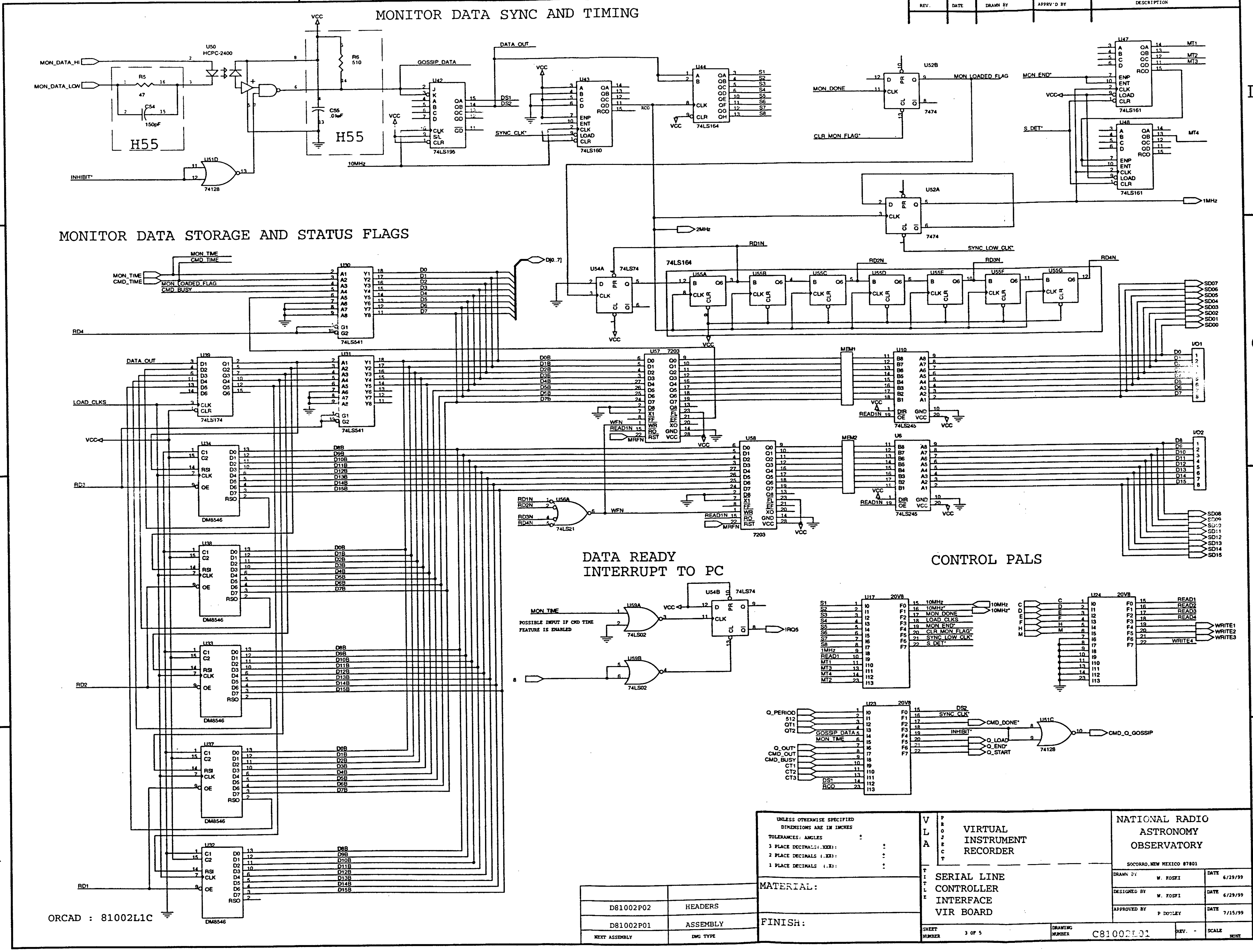
VLA PROJECT: VIRTUAL INSTRUMENT RECORDER

TITLE: SERIAL LINE CONTROLLER INTERFACE VIR BOARD

NATIONAL RADIO ASTRONOMY OBSERVATORY
SOCORRO, NEW MEXICO 87801

DRAWN BY	W. KOSKI	DATE	6/29/99
DESIGNED BY	W. KOSKI	DATE	6/29/99
APPROVED BY	P.DOOLEY	DATE	7/15/99

SHEET NUMBER 2 OF 5 — DRAWING NUMBER C81002L01 — REV. - — SCALE NONE



4 3 2 1

D C B A

REV.	DATE	DRAWN BY	APPRV'D BY	DESCRIPTION

ORCAD : 81002L1C

UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES: ANGLES ±
3 PLACE DECIMALS (.XXX): ±
2 PLACE DECIMALS (.XX): ±
1 PLACE DECIMALS (.X): ±

VLA PROJECT: VIRTUAL INSTRUMENT RECORDER

TITLE: SERIAL LINE CONTROLLER INTERFACE VIR BOARD

NATIONAL RADIO ASTRONOMY OBSERVATORY
SOCORRO, NEW MEXICO 87801

DRAWN BY	W. KOSKI	DATE	6/29/99
DESIGNED BY	W. KOSKI	DATE	6/29/99
APPROVED BY	P DOOLEY	DATE	7/15/99

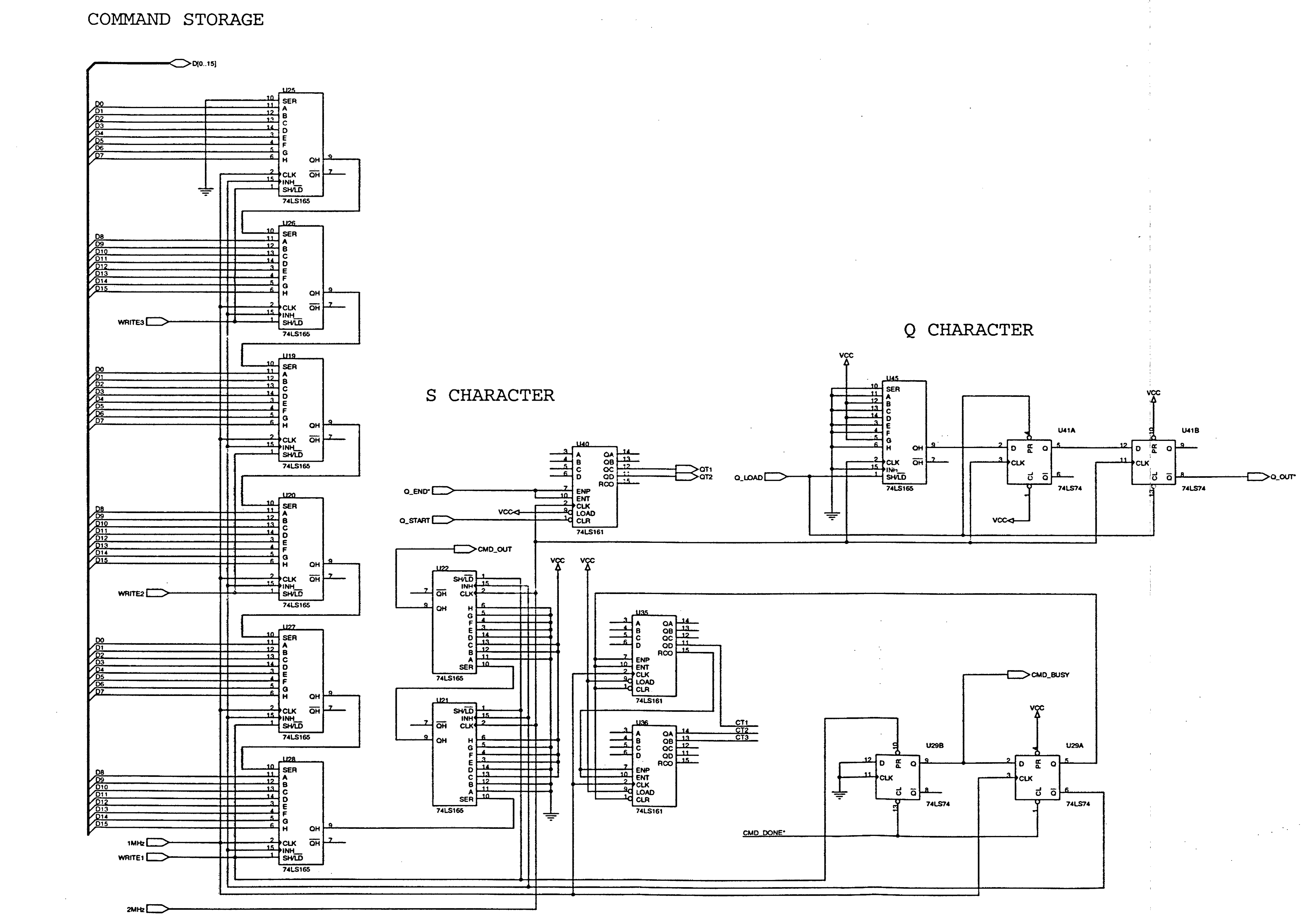
MATERIAL:

FINISH:

D81002P02	HEADERS
D81002P01	ASSEMBLY
NEXT ASSEMBLY	DWG TYPE

SHEET NUMBER 3 OF 5 DRAWING NUMBER C81002L01 REV. - SCALE NONE

PROPERTY OF NRAO



ORCAD : 81002L1D

PROPERTY OF NRAO

D81002P02	HEADERS
D81002P01	ASSEMBLY
NEXT ASSEMBLY	DWG TYPE

UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN INCHES
TOLERANCES: ANGLES ±
3 PLACE DECIMALS(.XXX): ±
2 PLACE DECIMALS (.XX): ±
1 PLACE DECIMALS (.X): ±

MATERIAL:

FINISH:

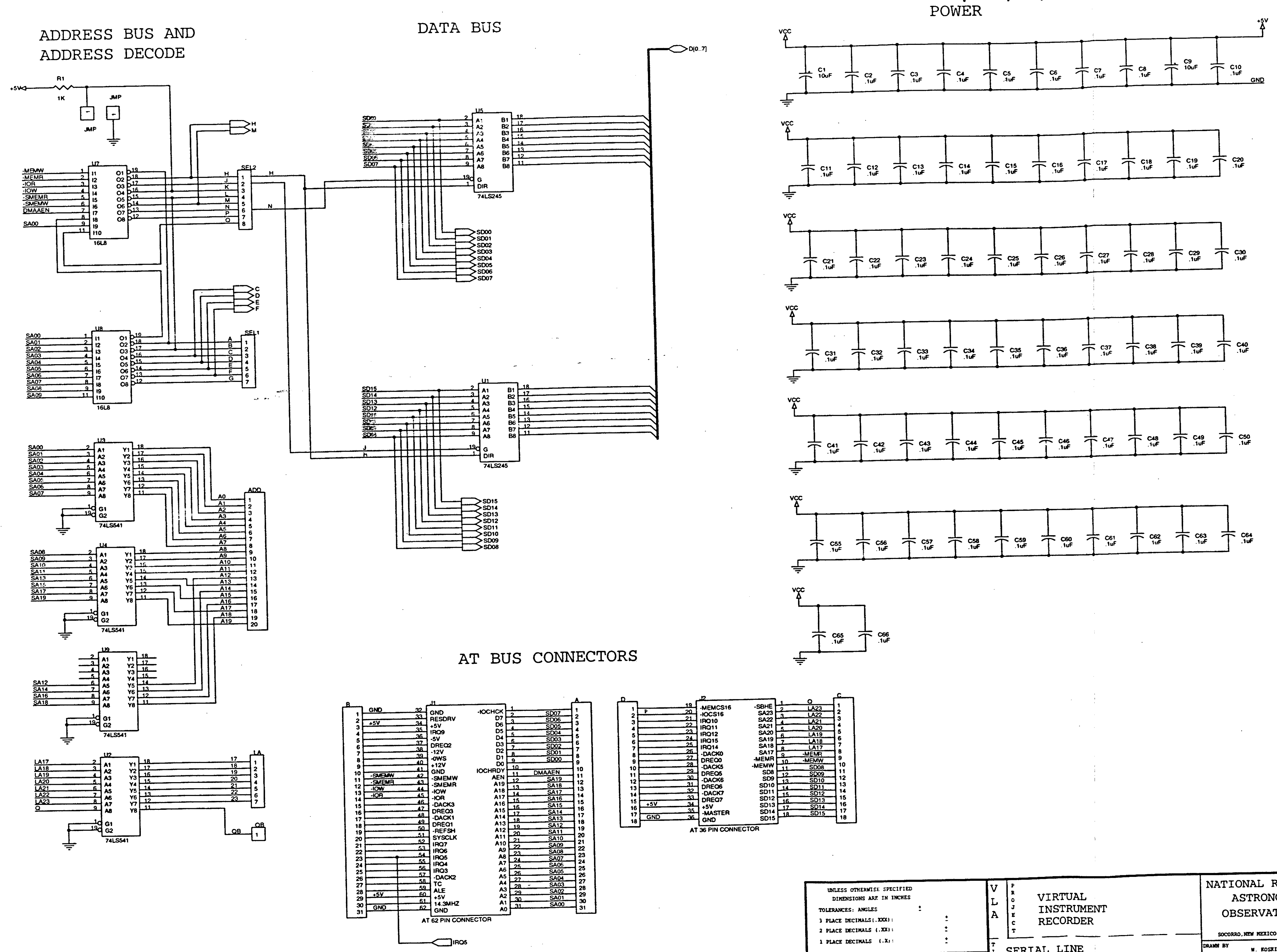
VLA PROJECT: VIRTUAL INSTRUMENT RECORDER

TITLE: SERIAL LINE CONTROLLER INTERFACE VIR BOARD

NATIONAL RADIO ASTRONOMY OBSERVATORY
SOCORRO, NEW MEXICO 87801

DRAWN BY	W. KOSKI	DATE	6/29/99
DESIGNED BY	W. KOSKI	DATE	6/29/99
APPROVED BY	P.DOOLEY	DATE	7/15/99

SHEET NUMBER 4 OF 5 DRAWING NUMBER C81002L01 REV - SCALE NONE



REV.	DATE	DRAWN BY	APPRV'D BY	DESCRIPTION

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES

TOLERANCES: ANGLES :

3 PLACE DECIMALS (.XXX) :

2 PLACE DECIMALS (.XX) :

1 PLACE DECIMALS (.X) :

MATERIAL:

FINISH:

D81002P02	HEADERS
D81002P01	ASSEMBLY
NEXT ASSEMBLY	DWG TYPE

VLA PROJECT: VIRTUAL INSTRUMENT RECORDER

TITLE: SERIAL LINE CONTROLLER INTERFACE VIR BOARD

NATIONAL RADIO ASTRONOMY OBSERVATORY

SOCORRO, NEW MEXICO 87801

DRAWN BY	M. KOSKI	DATE	6/29/99
DESIGNED BY	M. KOSKI	DATE	6/29/99
APPROVED BY	P.DOOLEY	DATE	7/15/99

SHEET NUMBER 5 OF 5 — DRAWING NUMBER C81002L01 — REV. - — SCALE NONE

ORCAD : 81002L1E

PROPERTY OF NRAO

2.7 VIR_GRAF APPLICATION

Introduction

VIR_GRAF is the most general VIR system application program because the plotting operations are not specialized to any module or system. It emulates an analog strip chart recorder that can plot any eight VLA M&C system **analog** data channels (addresses 0 to 127_{10}) versus time. See the VIR_GRAF Control Panel (next page) which depicts the chart traces, operating controls, and indicators. VIR_GRAF does not plot digital data channels (addresses 128_{10} to 192_{10}).

Briefly, the program can be considered to consist of three operations: 1) The program is initialized and then the user's control switch settings are read; these are channel addresss, timing, chart length and channel arithmetic functions. 2) The program establishes the plotting parameters, loads the addresses into the CLIENT's Channel Entry List and transmits it to SERVER. 3) The data provided by SERVER is charted in accordance with the parameters established in 2) above.

Control Panel

The CONTROL PANEL and BLOCK DIAGRAM follows this text. A glance at the CONTROL PANEL shows that nine chart traces span most of the panel. All nine traces are plotted at the same rate, however; the top trace plots a marker tick function and the lower eight plot address-selected monitor data.

Note that the operating controls are the functional groups: CHART CONTROL, TIME BASE, Channel Select, channel address (MW-DCS-DSA-MUX), and trace arithmetic operations (Gain-Offset-Avg). These functions are described in more detail below.

With the exception of the CHART CONTROL switches, all switches are digital numeric controls with an alphanumeric display of the switch's setting. The setting may be changed by left-side increment (up) and decrement (down) selector arrows.

The PLOT (psuedo-LED) indicator flashes each time the program plots a new set of data values. The SERVER psuedo-LED flashes when SERVER provides a new set of data to the VIR_GRAF CLIENT.

The CONNECTED TO VLA-VIR Server message in the Connect Status indicator confirms the VIR-GRAF's CLIENT-SERVER connection. The day-date and time indicator shows when a chart is started.

Each chart trace's vertical axis is annotated to show the trace's analog signal span, determined by the trace's Gain and Offset switch settings. Each channel trace has an Editable Channel Label to describe the channel's identity and character.

Chart time is recorded in two forms: an annotated time base with time ticks runs along the bottom and the top (ninth) trace is a time marker (tick) trace. The two time functions are independent. The Marker Sel (tick) choices are: 1/sec, 1/10 sec, 1/min, 1/10 min, 1/hr, 1/4hr, 1/day and 1/wk.

The Chart Length choices are: 1 minute, 1 hour, 1 day, 1 week and 1 year.

Plot Time (reciprocal of the data sample rate) choices are: 0.1, 1.0, 1.0/10 (seconds/sample), 1.0, 1.0/10 (min/sample), 1.0, 1.0/4 (hours/sample), and 1 day/sample.

The **Channel Number** displayed in the Channel Select switch is associated with the adjacent channel address component switches: MW, DCS, DSA and MUX. Channel 1's address components are thetop set, Channel 2's are the next down, etc. **Chart Traces** are trace physical locations, Trace 1 is the top data trace, Trace 2 is the next down, etc. If the top Channel Select switch is set to Channel 1, Trace 1 will plot Channel 1 data. Setting the second Channel Select switch to Channel 2 will plot channel 2 data, etc. However, a chart trace can also be a plot of the algebraic **sum** or **difference** of two channel's values. Channel sum/difference selections are made by using the Channel Select switch's Increment or Decrement functions (Up or Down arrows). For example, Trace 1 can be the **sum** of Channel 1 (defined by the top set of address components) and another channel, say 8 (the bottom set of address components), by using the Channel Select control's **Increment** function. This causes the Channel Select switch's displayed value to increment to higher channels, 2, 3,..8. When 8 is displayed, Trace 1 plots the sum of channel's 1 and 8 data values. Similarly, using the **Decrement** function causes Chart Trace 1 to be the difference between Channel 1 and higher channels, for example, 8. When 8 is displayed, the Trace 1 plot is the Channel 1 value **minus** the Channel 8 value. Assigning a Chart trace to either the sum or difference of two channels does not preclude using the other traces in the direct manner described above. For example, if Trace 1 is assigned to plot the Channel 1 to Channel 8 difference, Traces 2 through 8 may be used to plot channels 2 through 8 in the direct (not sum or difference) mode. When using the sum or difference charting modes, the trace Gain-Offset-Avg arithmetic operations are applied to the sum or difference values, not to the individual channel values.

Mode control is very simple: setting the CLEAR switch initializes the program variables; the TIME BASE, Channel Select, MW-DCS-DSA-MUX, and trace arithmetic function switches are read; the values determine the channel scaling, sample rate, chart period, etc. Setting the RUN/STOP switch to RUN starts the plotting operation. Conversely setting the switch to STOP stops the plotting operation without altering the TIME BASE or other switch settings; the chart traces remain unchanged. Setting the switch to RUN again, resumes plotting with the same control switch settings but the traces are cleared and a new chart is started. This prevents a discontinuity in the chart traces resulting from the STOP-RUN transition, which could be confusing and perhaps cause an erroneous interpretation of the channel's behaviour. Any control switch setting can be changed while in the RUN mode without causing a charting error.

The strip charting operation is not limited to the specified chart period (Chart Length). When a chart reaches the selected period, charting continues and the time annotations, chart traces and marker trace scroll in a manner similar to conventional strip chart recorder scrolling.

Each trace's analog span can be altered by selectable arithmetic functions: Gain, Offset and Avg (averaging filter). The program's initial settings are: Gain = 1.00, Offset = 0.00 and Avg = Off. With these settings, the channel trace spans are labelled -10.5 to +10.5 (for volts). The Data Set's analog signal span is -10.240 to +10.235 volts and its quantization is 5 mV/count. The settable gain-offset features enable the user to expand a fraction of the channel signal to make it cover the full trace span. For example, if a user wanted to examine a channel's behavior in the - 2.5 to -3.0 volt range, he would set the gain to 40 and offset to +3.0. The chart trace would then plot this signal range and the trace annotations would show the -3.0 to -2.5 volt trace span. Since this trace span is 0.5 volts, or 100 counts, a one-count change in the analog signal would be a change of 1/100-th of the trace span; this is small but visible. Each trace's signal level is displayed in Digital (hex counts) and Analog (volts) format numeric indicators. If the trace is plotting a single channel, the displays show the channel values; if the trace is the sum or difference of two channels, the display shows the sum or difference value.

The Averaging function is a simple recursive filter that operates at the channel update rate, Plot Time. The filter's equation is: $Y_N = 0.9Y_{N-1} + 0.1X_N$ where N is the sample time number (N is the current sample time, N-1 is the previous, etc.) Y_{N-1} is the previous filter output value and X_N is the new data value. The filter emulates a simple RC low-pass filter and one "time-constant" is about 10N. Thus a step change in a data level reachs its full value in about 5 time-constants.

Initialization

The Initialization code is on Block Diagram Page 3. The Control Panel has many variables; these must be initialized before program execution is started. Since there are many control parameters and data variables, they are introduced below in their functional categories rather than piecemeal throughout the code description.

Program Mode-Status variables: (Boolean): RUN, CLEAR, PLOT, SERVER; (String) Connect Status; (U32, Unsigned 32-bit Integer) Date/Time.

Channel Address variables: (U8, Unsigned 8-bit integer) MW, DCS, DSA, MUX; (Boolean) MUX+1. Note that although MUX is a U8 variable, MUX+1 is Boolean. (Unsigned 16-bit Integer) Ch 1 Select,..Ch Select.

A note about the MW address variable; MW is not an operative parameter in VIR-GRAF's data charting operations; it specifies to SERVER which time-ordered data value (MW1 or MW2) should be provided to VIR_GRAF from the data array in RAM. See Appendix A for a description of the usage of MW1 and MW2. Also see the **Monitor Data Array in RAM** paragraphs in Section 2.6 for a description of this array.

Channel Data variables: (DBL, Double-precision floating point) Ch 1 Ln,..Ch 8 Ln (current trace data value); Ch 1 Ln Prev,..CH 8 Ln Prev (previous trace data value); Ch1 Gain,...Ch8 Gain; Ch1 Offset,..Ch8 Offset; (String) Ch 1 Digital,..Ch 8 Digital. Not shown on Page 3 are the Boolean channel average select functions, Ch1 Avg,...Ch8 Avg.

Time variables: (String) Date/Time; (U8, Unsigned 8-bit integer) Day ofWk, Day ofWeek Prev, Hours, Hours Previous, Minutes, Minutes Previous, Seconds, Seconds Previous; Time Changes (Boolean) 1/day Chg, 1/Hr Chg, 1/Min Chg, 1/Sec Chg; (U32, Unsigned 32-bit integer) Plot Time Value (time annotation below trace 8); (U8 Unsigned 8-bit) Plot Time (time to plot data and marker trace values).

Marker trace tick variables: (Boolean): Marker, 1 Week Marker, 1 Day Marker, 4 Hour Marker, 30 Min Marker, 10 Min Marker, 1 Min Marker, 10 Sec Marker; (U8, Unsigned 8-bit integer) Marker Sel; (U32, Unsigned 32 bit integer) Marker Time Value; (Boolean) InvalidMarker. (DBL Double precision, floating point) Marker Value. The marker trace has two data levels: 0.25 for the space level and 0.75 for the tick level.

Chart control variables: (U8, Unsigned 8-bit integer) Chart Length, Chart Length Prev; (DBL, Double-precision floating point) Xmax, Xincr; (Boolean) Chart Length Chg, Invalid Length.

Trace labels: (String) Marker Label, Ch 1 Label,..Ch 8 Label.

Graf Data (String) is the input data stream from SERVER via VIR_graf Global VI.

The GRAF CLIENT's Ch Entry List array is created and initialized by an Initialize Array. The array has two indices: 8 for the eight Channel Entry List address parameter rows and 4, for the number of address components in each row (MW, DCS, DSA and MUX). The element input is the string constant 1; this sets all element's to the value of 1, an arbitrary value which is over-written later in the program when the Control Panel's address switches are read. The reader is urged to take a quick glance at the CLIENT Channel Entry List description in Section 2.3.

The BUNDLE-CLUSTER icons define a waveform chart function, set to the Strip Chart update mode. Waveform Charts retain previous data (up to a maximum value) and new data is appended to the old data, enabling the user to see the current value in the context of previous data. The Strip Chart mode is a scrolling display similar to a strip chart recorder; the trace starts from the chart's left margin and plots data to the right margin. As each new data value is received, it is plotted at the right margin and old values shift to the left - a format similar to a strip chart. The associated time base annotations scroll with the data traces. In this initialization code, the data trace inputs to the chart BUNDLE function are initialized to 0.0 and the Marker trace is initialized to 0.25.

Setting the CLEAR switch initializes the variables and starts a one-pass While loop to set up time variables. In this loop, a GET DATE/TIME IN SECONDS function obtains time in seconds and a SECONDS TO DATE/TIME function converts this value to a cluster of nine date and time parameters, all 32-bit integers. An UNBUNDLE function disassembles the cluster into the components shown. The program uses Seconds, Seconds Previous, Minutes, Minutes Previous, Hours, Hours Previous, Day of Week and Day of Week Previous.

Main While Loop

The Main While loop execution rate is 275 ms, determined by the WAIT UNTIL NEXT ms MULTIPLE (Metranome icon). If there is a data update from SERVER via VIR Graf global.vi, the SERVER LED is flashed.

While Loop execution consists of the eight-frame, N[0..7] sequence, Block Diagram Page 3.

Frame 0[0..7] is on Block Diagram Page 13. In this frame the presence of data from VIR Graf global.vi causes the "CONNECTED to VLA-VIR Server" message to be displayed. There are two Boolean case structures in this frame. The first is entered if CLEAR is true; this causes entry to the N[0..3] (Sequence B) sequence which clears the chart and sets the chart's X (horizontal) scale. Within 0[0..3] is a case structure of five case of the Chart Length numeric variable (a Control Panel switch setting). Cases 0, 1, 2 and 3 are on Page 13, case 4 is on Page 14. Each case assigns time values to the Xmax and Xincr variables.

If CLEAR is not true, the second case structure is entered; this case is described below.

Frame 1[0..3] (Page 14) has nine INITIALIZE ARRAY functions (eight data traces and the marker trace). The Element inputs (the value to be stored) are all 0.0 and the Index is 0. The array outputs are connected to an INDEX AND BUNDLE function whose output variable is Strip Chart. Appended to the Strip Chart variable is a Chart Attribute node. The History Data attribute clears the charting function.

Frame 2[0..3] (Page 14) is another Chart Attribute node; this one sets the X Increment attribute equal to the variable Xincr, and the X Maximum attribute equal to the variable Xmax.

Frame 3[0..3] (Page 14) sets the CLEAR state false; this completes sequence N[0..3].

Now, referring back to the second case structure in 0[0..7] (BD Page 13), mentioned three paragraphs back. The second case is entered if Chart Length is different from Chart Length Prev(ious). If true, it causes entry to sequence N[0..1] (Page 13). Frame 0[0..1] has a five case structure in which Chart Length is the Selector. The five cases of Chart Length range from 1 minute to 1 year; these were noted above. In each case, the Xmax and Xincr variables are set to values appropriate for the charts. Frame 1[0..1] contains a Chart Attribute Node which sets four attribute values. X Maximun is set to the variable Xmax, X Increment is set to the variable Xincr, X Minimum is set to 0. The X Scale Fit attribute controls how the X scale is fitted to the data is set to 0; this means never. The X Scale is not dependent upon the data values. At this point, Frame 0[0..7] operations are complete and control passes to Frame 1[0..7]. It should be noted that the two cases described above appear to be almost redundant, but there is an important difference. The case depending upon CLEAR is entered from program start-up; the second case is entered if the user changes the Chart Length after start-up. This second case accommodates situations in which the user first selects a short chart period to view data with a high-sampling rate and then shifts to a lower sample rate, longer period chart.

A final note on 0[0..7], if neither (of the two cases described above) is true, control passes to frame 1[0..7].

Frame 1[0..7] (BD P15) reads the time and separates the components into Seconds, Minutes, Hours and Day of Week variables. The functions that do this are the same ones described in the Initialization paragraphs above (i.e., GET DATE/TIME IN SECONDS, SECONDS TO DATE/TIME, and UNBUNDLE.) Although it's not drawn on the diagram, the Seconds output of the DATE/TIME TO SECONDS is connected to a GET DATE/TIME STRING function. The function's Want Second? input is set true. The Date string (upper output) and Time string (lower output) are combined with a string constant (value nul) in a CONCATENATE STRINGS function which concatenates the three strings into a single string.

Frame 2[0..7] (BD P16) detects two types of timing parameter changes: time changes that are related to data sampling and marker (tick) time changes. Three (nearly) identical sets of functions operate on Seconds-Seconds Previous, Minutes-Minutes Previous and Hours-Hours Previous, respectively. Consider the top set, the Seconds operation. Seconds and Seconds Previous are compared in a NOT EQUAL function, if true, the 1/Sec Chg variable is true. Seconds are also connected to a QUOTIENT & REMAINDER function with 10.0 connected to the divisor input. The Qoutient (lower) output is not used. If the Remainder output is 0, it means that the time input is an integer multiple of 10.0 seconds. The Remainder output is input to an EQUAL TO 0? function; if the output is true, the 10 Second Marker variable is set true. The EQUAL TO 0? function is also connected to an EQUAL function with a Boolean True connected to the other input; the output of the EQUAL function is connected to one input of an AND function. The other input is connected to the output of an EQUAL function that compares 10 Sec Marker Previous with Boolean False. If the 10 Second Marker Previous is also false, the EQUAL output is True so the AND function output, 1/10 Sec Change variable is true.

This comparison structure is repeated for the Minutes-Minutes Previous and Hours-Hours Previous variables. A similar but somewhat simpler comparison is performed on the Day of Week-Day of Week Previous variable. It should be noted that only one time change and only one marker change variable can ever be true; secondly, because the program's execution rate is 275 ms, during most of the passes through the N[0..7] sequence, there is a high probability that neither change will be true.

Frame 3[0..7] tests the state of the time change and marker change parameters detected in 2[0..7]. If a time change corresponds to the setting of the Control Panel Plot Time switch, channel data and the marker trace are plotted. This is also the case for the marker tick function; if a marker tick time change corresponding to the Control Panel's Marker Sel switch setting is true, the trace's tick level (0.75) will be plotted. If it is not true, the trace's space level is 0.25.

Frame 4[0..7] (Page 3) contains sequence N[0..1]. Frame 0[0..1] of this sequence is on BD Page 3. Within 0[0..1] is an eight-pass FOR loop; within each pass there is a case structure that uses the FOR loop index. Inside each of these case structures is an N[0..3] sequence structure. This looks rather complex but it performs the simple operation of setting the eight sets of address parameters (MW, DCS, DSA and MUX) into the eight rows of VIR_Graf Client's Channel Entry List (CEL). In the first FOR loop pass, the four CEL row 1 components are set to the first set of switch values, in the second pass the next four are set into the CEL row 2 components, etc, through row 8. The N[0..3] sequence converts these switch numeric components to decimal or octal string formats before they are entered in the CEL.

Frame 0[0..3] converts <u>MW</u> to decimal string, 1[0..3] converts <u>DCS</u> to decimal string, 2[0..3] converts <u>DSA</u> to decimal string and 3[0..3] converts <u>MUX</u> to octal string. The operations are all nearly identical and are performed on 32 sequence structures, BD pages 3 through 12. In each structure, a TO DECIMAL (MW, DCS, DSA) or TO OCTAL (MUX) converter function converts the switch's numeric value to a string of decimal or octal digits. These are input to the Element input of a REPLACE ARRAY ELEMENT function. The arrays have two indices: the first is the FOR loop index and the second corresponds to each parameter in a CEL row; 0 for MW, 1 for DCS, 2 for DSA and 3 for MUX. The array variable is Ch Entry List and each frame of N[0..3] modifies the CEL row corresponding to the FOR loop index.

After the 0[0..1] frame's FOR loop operations have been completed, the CLIENT's CEL has been set to the string equivalents of the eight sets of address switches. Frame 1[0..1] transmits the CEL to SERVER. The reader will remember that this CEL is VIR Graf Global VI which enables it to be modified and transmitted to SERVER as an integral data package. This transmission completes frame 4[0..7] operations. SERVER will provide the requested data which is input to VIR_GRAF via its CLIENT in frame 5[007], below.

In frame 5[0..7] (BD P18) eight sets of 12-bit data values are extracted from the eight sets of 24-bit values provided by SERVER. The selection logic is based upon MUX address and chart trace index and consists of eight pairs of Boolean case structures, where each pair provides a 12-bit MUX address-designated data value for a trace (Ch N Ln) and (Ch N Digital) for the associated Digital (hex) numeric indicator. Boolean case selection is based upon the MUX address (an 8-bit integer) or by MUX+1, a Boolean value. If the MUX address is an even value, the MUX case structure selects the lower 12 bits of the 24-bit data argument; conversely, if the MUX address is an odd value, the MUX+1 case structure selects the upper 12 bits of the 24-bit argument. Remember that each 12-bit value in string format is three, four bit characters.

The case structure code elements are very similar but rather compactly packed in this page, therefore the connections are described in some detail. Both cases consist of two INDEX ARRAYs, a string STRING SUBSET function, a string FROM EXPONENTIAL/FRACT/ENG string to number conversion function, and a CONCATENATE STRINGS function. The trace data values (Ch N Ln) comes from the left arrays and the CH N Digital values displayed by the CP's Digital indicators comes from the right arrays.

Common features of both cases are: 1) SERVER's VIR Graf Global Data (string format) is input to each of the two INDEX ARRAYs. 2) The trace index (1,..8) is connected to both array's Index 0 inputs. 3) In both cases, the left array's output is connected to the FROM EXPONENTIAL/FRACT/ENG string to number conversion function which provides the numeric trace value, Ch N Ln. 4) In both cases the right array's output is connected to the STRING SUBSET function, the output of which is connected to the CONCATENATE STRINGS function. In this function, a string constant is concatenated with the data value. The CONCATENATE STRINGS function output drives the Ch N Digital indicators on the Control Panel. 5) In both cases the right array's Index 1 input is connected to 0. 6) In both cases 3 is connected to the Length input of the STRING SUBSET function.

Features that are different are: 1) In the MUX+0 case, a 1 is connected to the left array's Index 1 input; this selects the lower 12 bits for the Ch N Ln outputs. In the MUX +1 case a 2 is connected to the left array's Index 1 input; this selects the upper 12 bits for the Ch N Ln outputs. In the MUX+0 case, a 3 is connected to the Offset Input of the STRING SUBSET function; this selects the lower 12 bits for the CH N Digital outputs. In the MUX+1 case, a 0 is connected to this Offset input; this selects the upper 12 bits for the Ch N Digital outputs.

One final point, how do the Index arrays know that the data values are 12-bit rather than say a byte value? This is determined by the code context; the 3 on the Length input of the STRING SUBSET function defines the values as 12-bit.

Frame 6[0..7] (BD Page 18) contains a Boolean case structure. If PLOT is true (from 3[0..7] which detects (plot) time changes), a two-frame sequence, N[0..1] (Sequence F) is entered. In frame 0[0..1], data averaging functions are performed on the Ch N Ln data if the channel's ChN Avg switch is true (On). The filter equation was described above. For each trace case, if true, a N[0..3] sequence is entered. In the first frame the Ch N Ln Prev(ious) value is modified by multiplying it by 0.9. In the next frame, the Ch N Ln value (the new data value) is multiplied by 0.1. In the third frame the two sets of modified values are summed to make the filtered Ch N Ln value. In the fourth frame the resultant Ch N Ln value is made the Ch N Ln Prev value. If the trace's Avg switch is Off, data filtering is not done.

In frame 1[0..1] (BD Page 20) the eight Ch N Ln values from 0[0..1] are input to a BUNDLE function which produces a CLUSTER. This CLUSTER output is connected to eight sets of 8-case Numeric Case Structures in which the Channel Sum-Difference functions are performed; there are fifteen cases which are all combinations of one channel summed/differenced with another. The Numeric Case selector inputs are Ch N Select from the CP's Chan Select switches. Looking at case 0 (inside the 1[0..1] frame figure) which is the direct (not a sum or difference case), the output data is Ch N Ln. Cases 1 through 14 are shown below; each of these cases either sums or differences the Ch N Ln value with another Ch N Ln value.

The eight Ch N Ln outputs of the direct or sum-difference operations are multiplied by the trace's Gain multiplier and the product is summed with the trace's Offset value. The eight resultant Ch N Ln values and the Marker Value are input to a BUNDLE function; the CLUSTER output is connected to the strip chart function which plots another point on the Marker and Channel traces.

Frame 7[0..7] (BD Page 22) sets all current time parameter values to the previous time values. In addition, if Plot is false (this pass was not a plot cycle), Chart Length from the CP switch is set into the Chart Length Prev variable. After completion of these operations, control reverts to frame 0[0..7].

VIR

VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:09 PM

Front Panel

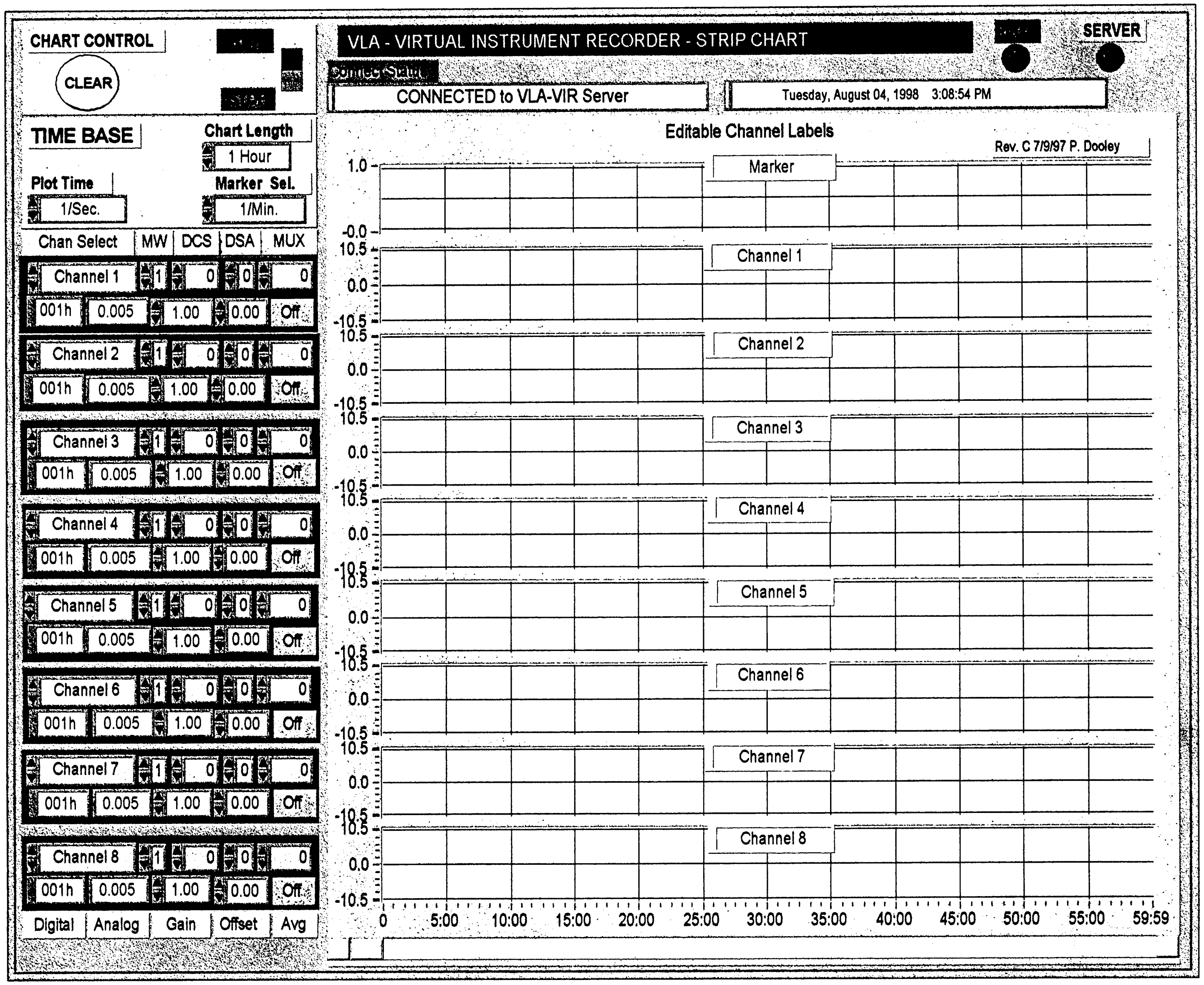


CHART CONTROL

CLEAR

VLA - VIRTUAL INSTRUMENT RECORDER - STRIP CHART

SERVER

CONNECTED to VLA-VIR Server

Tuesday, August 04, 1998 3:08:54 PM

TIME BASE

Chart Length: 1 Hour

Plot Time: 1/Sec.

Marker Sel.: 1/Min.

Chan Select	MW	DCS	DSA	MUX
Channel 1	1	0	0	0
Channel 2	1	0	0	0
Channel 3	1	0	0	0
Channel 4	1	0	0	0
Channel 5	1	0	0	0
Channel 6	1	0	0	0
Channel 7	1	0	0	0
Channel 8	1	0	0	0

Digital	Analog	Gain	Offset	Avg
001h	0.005	1.00	0.00	Off
001h	0.005	1.00	0.00	Off
001h	0.005	1.00	0.00	Off
001h	0.005	1.00	0.00	Off
001h	0.005	1.00	0.00	Off
001h	0.005	1.00	0.00	Off
001h	0.005	1.00	0.00	Off
001h	0.005	1.00	0.00	Off

Editable Channel Labels

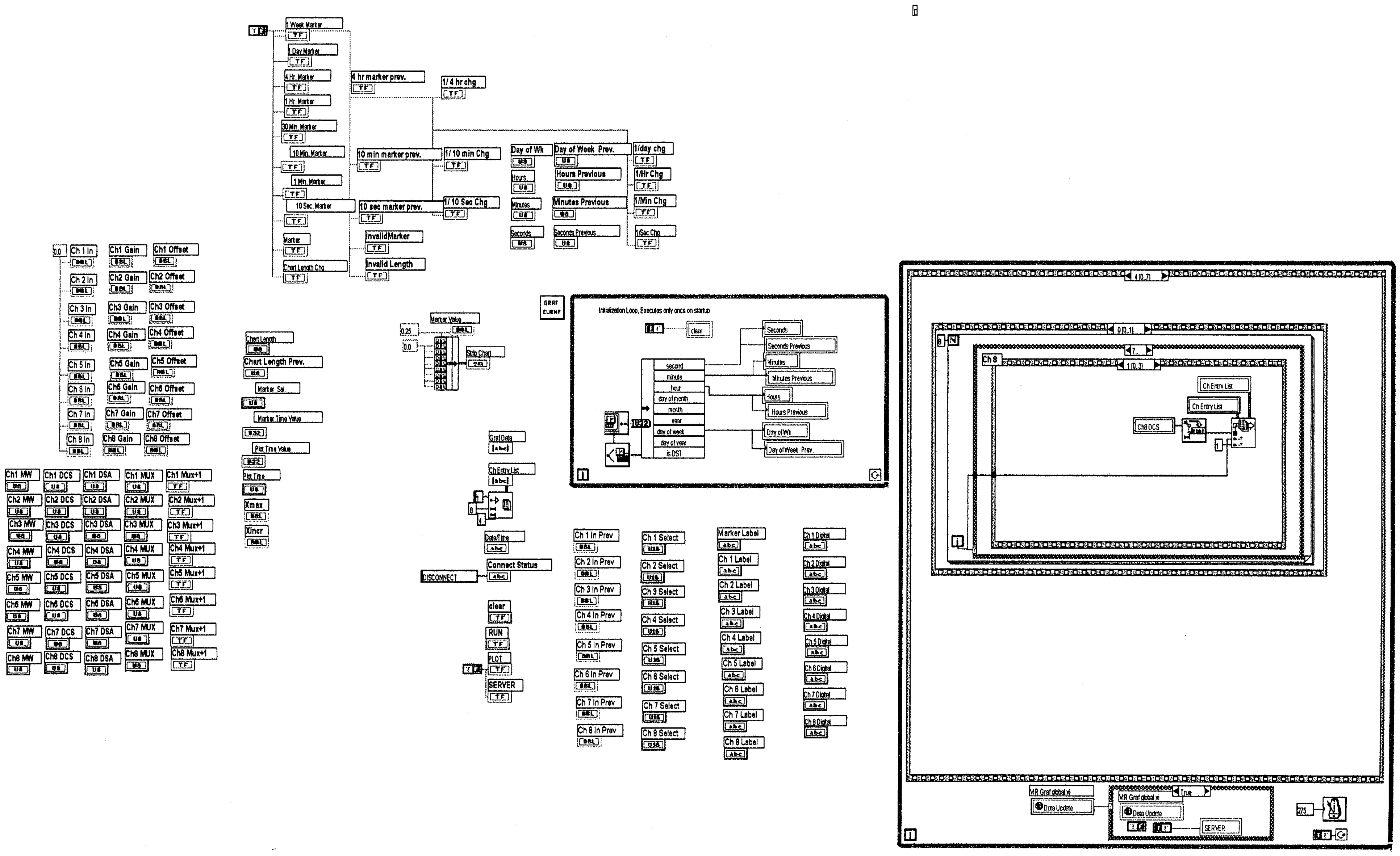
Rev. C 7/9/97 P. Dooley

VIR_GRAF.VI

VIR

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:10 PM

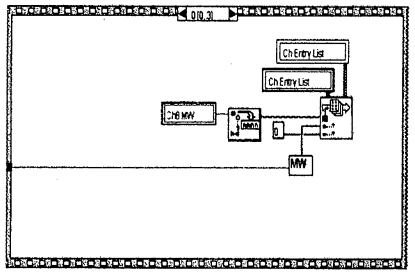
Block Diagram

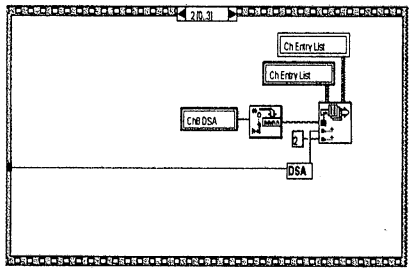


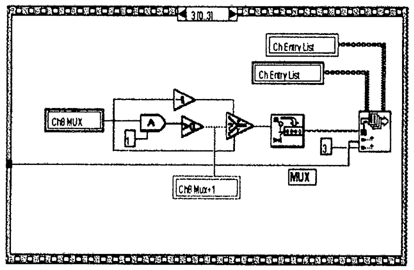
VIR_GRAF.VI

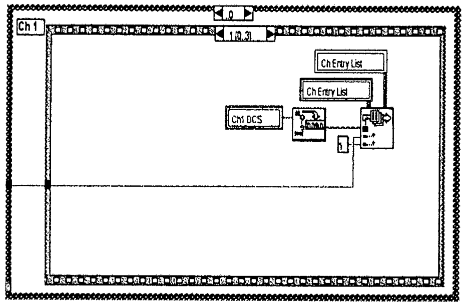
VIR

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:10 PM





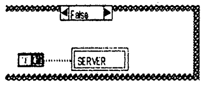




VIR

VIR_GRAF.VI

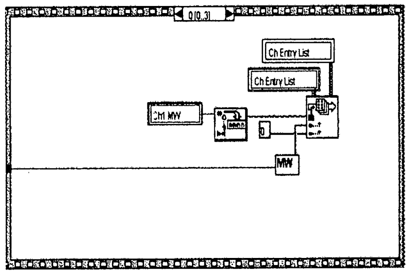
Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:10 PM

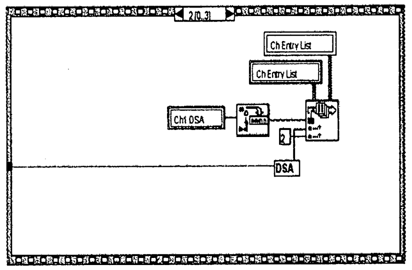


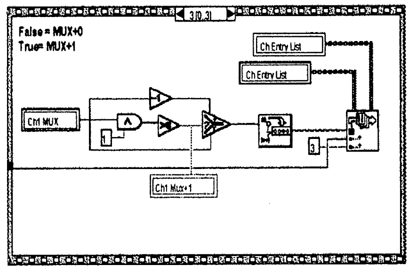
VIR

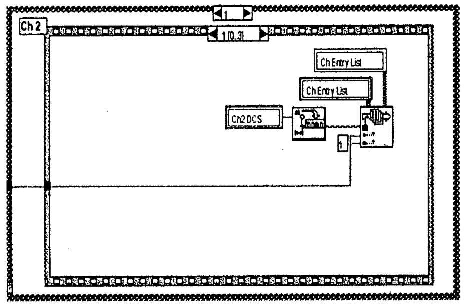
VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:10 PM





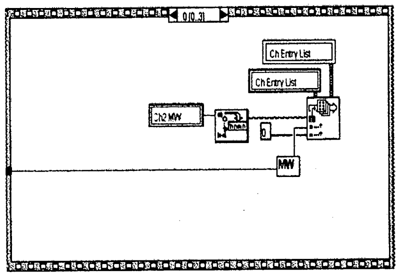


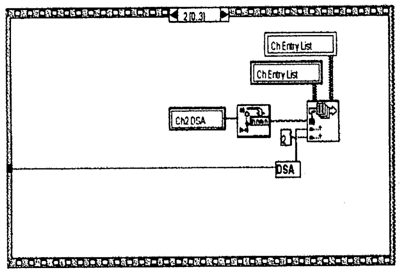


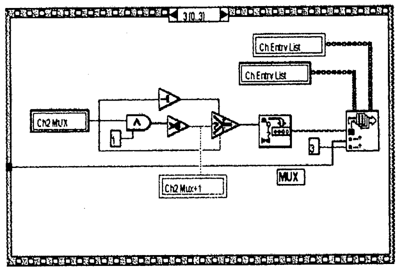
VIR

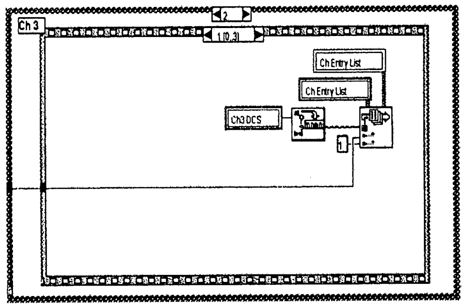
VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:10 PM





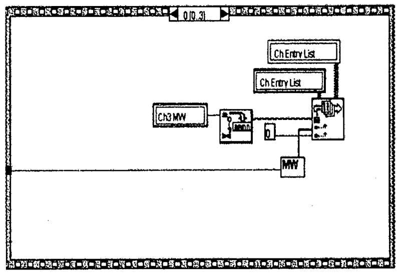


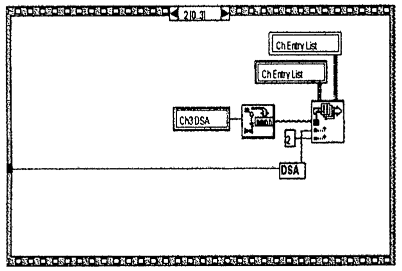


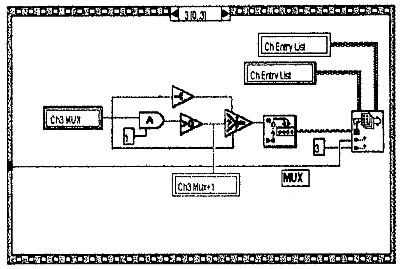
VIR

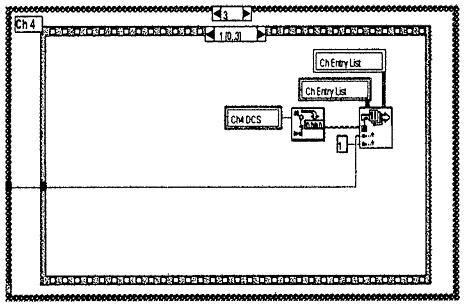
VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:10 PM



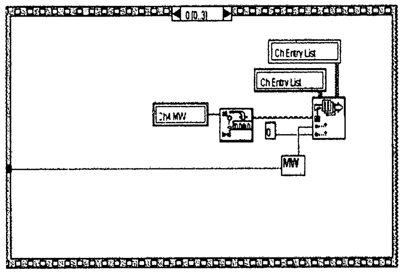


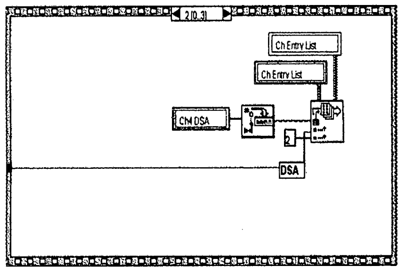


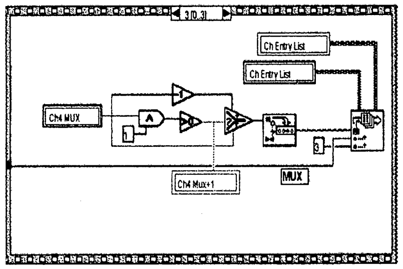


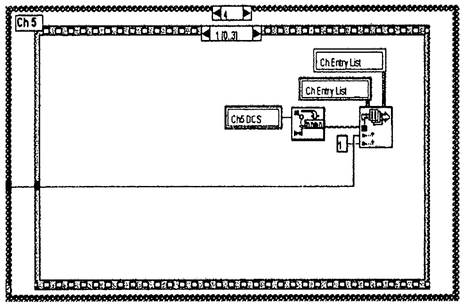
VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:11 PM





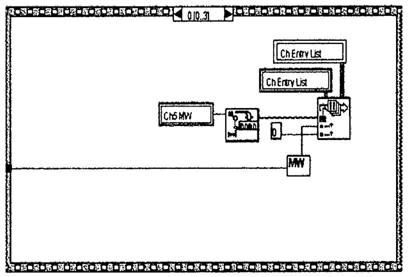


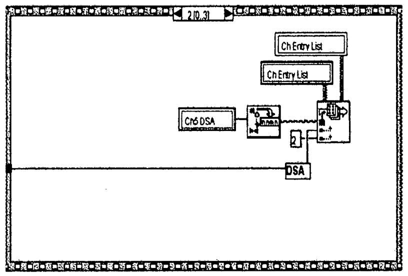


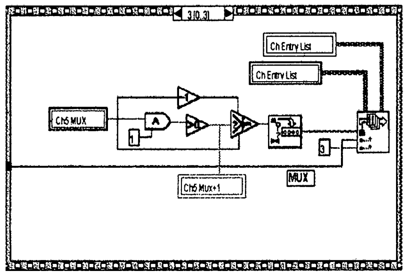
VIR_GRAF.VI

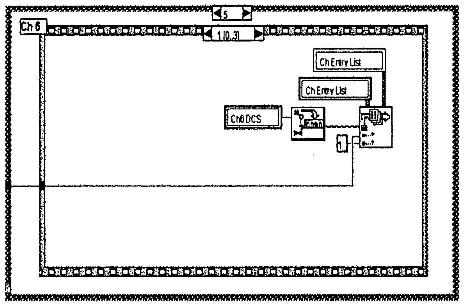
VIR

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:11 PM



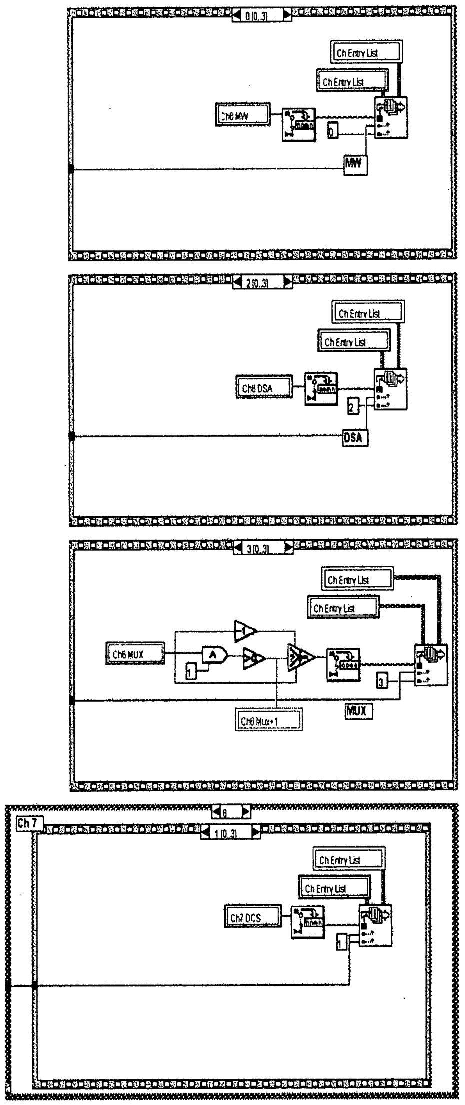






VIR_GRAF.VI

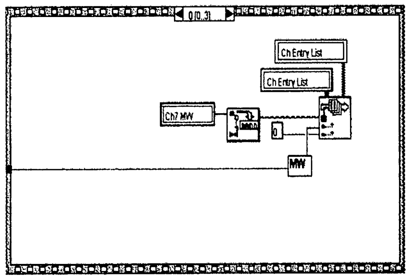
Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:11 PM

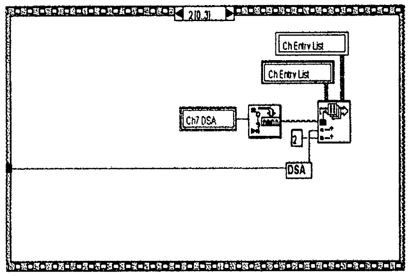


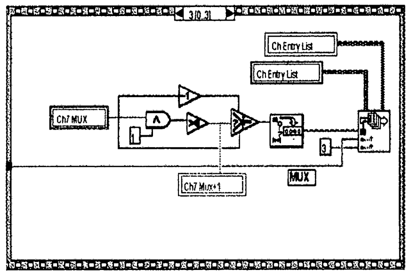
VIR

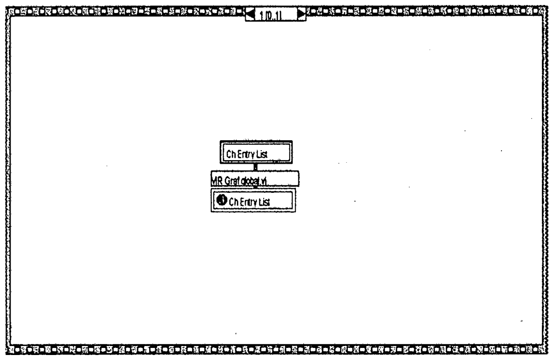
VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:11 PM



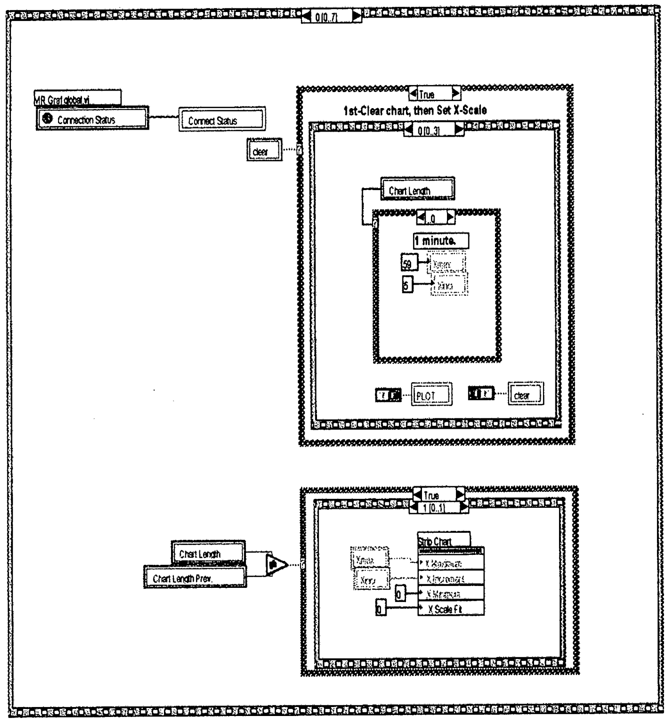


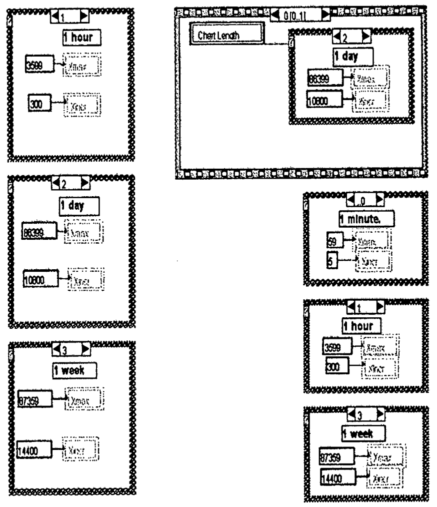




VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:11 PM

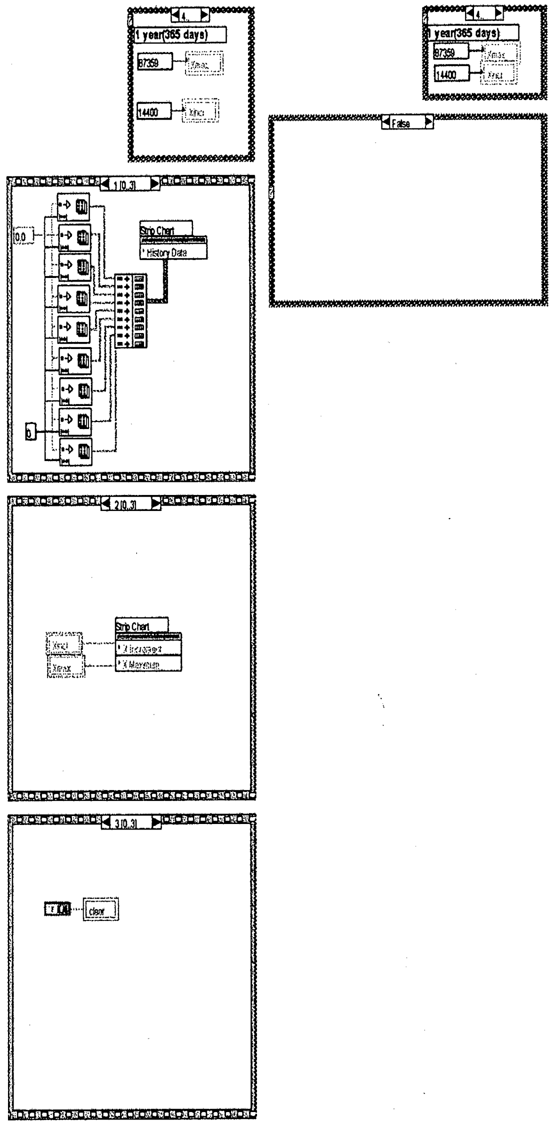




VIR_GRAF.VI

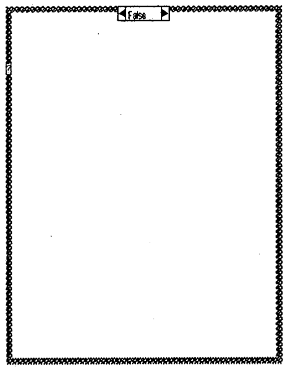
VIR

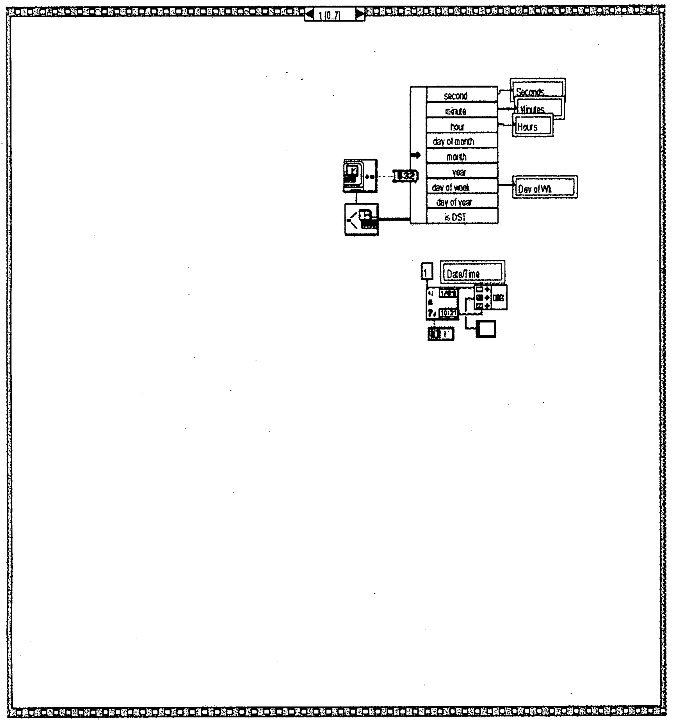
Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:11 PM



VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:12 PM

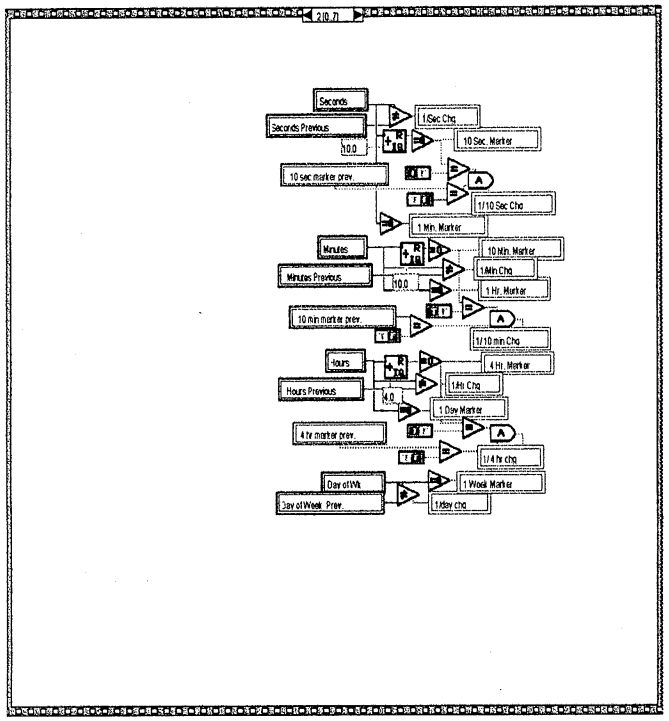




VIR_GRAF.VI

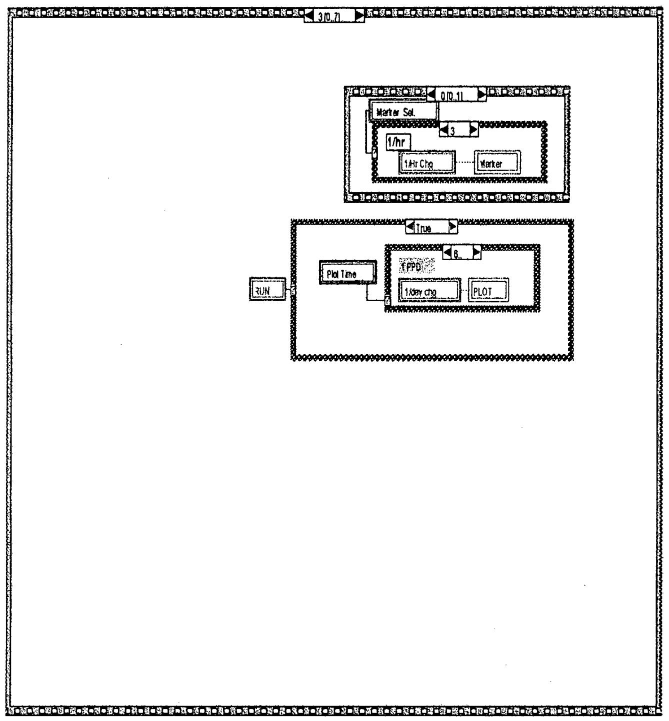
VIR

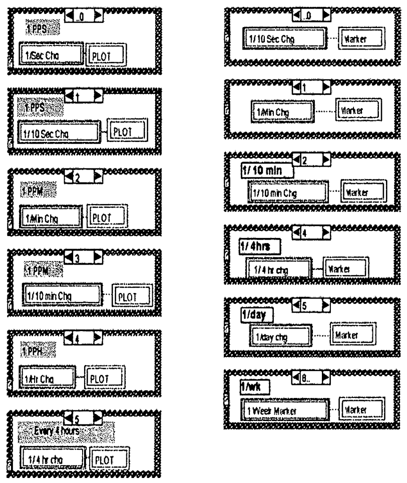
Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:12 PM



VIR_GRAF.VI

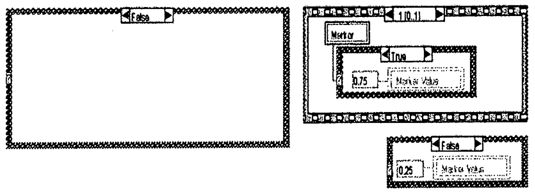
Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:12 PM

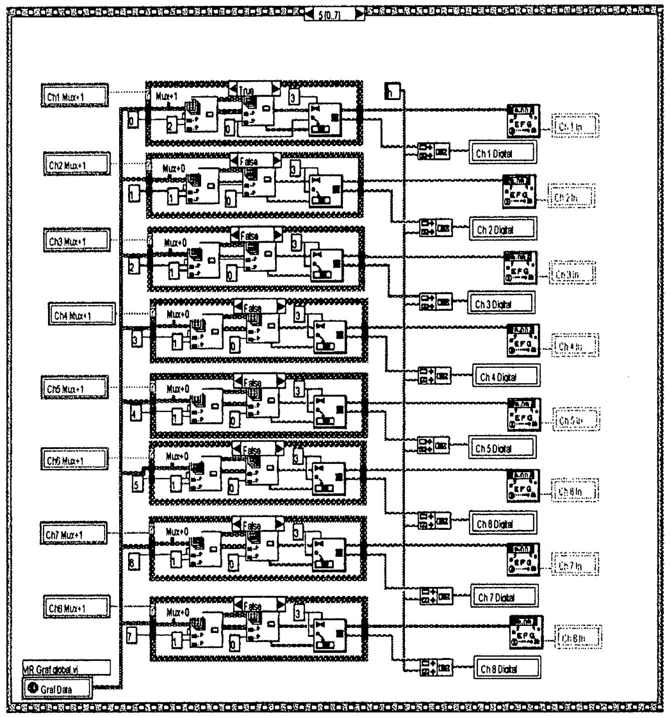


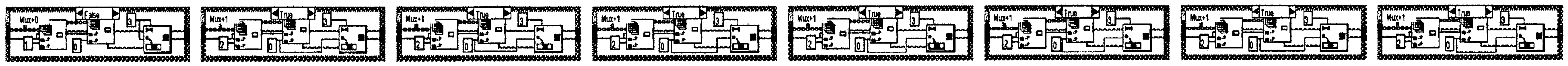


VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:12 PM

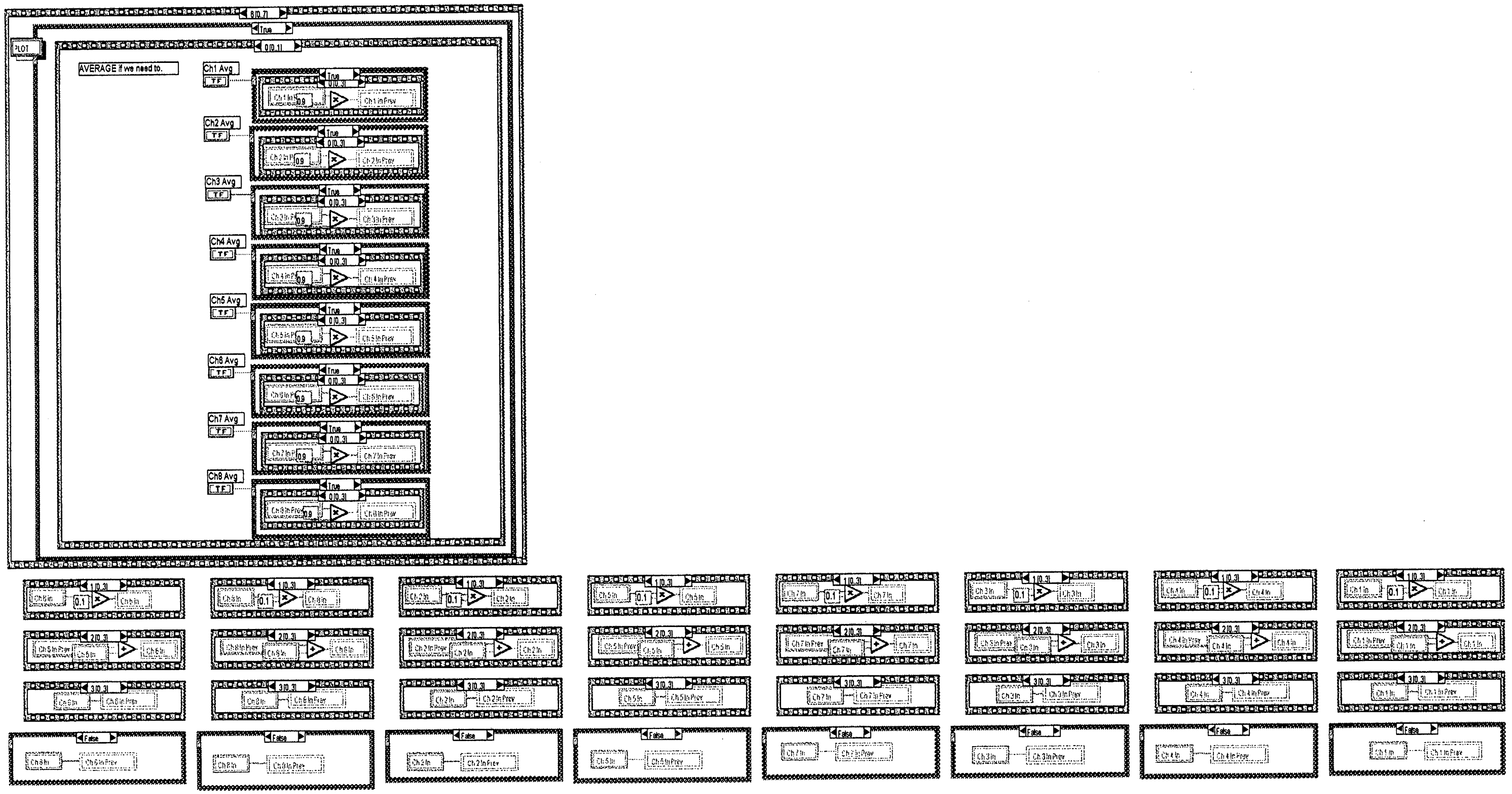






VIR_GRAF.VI

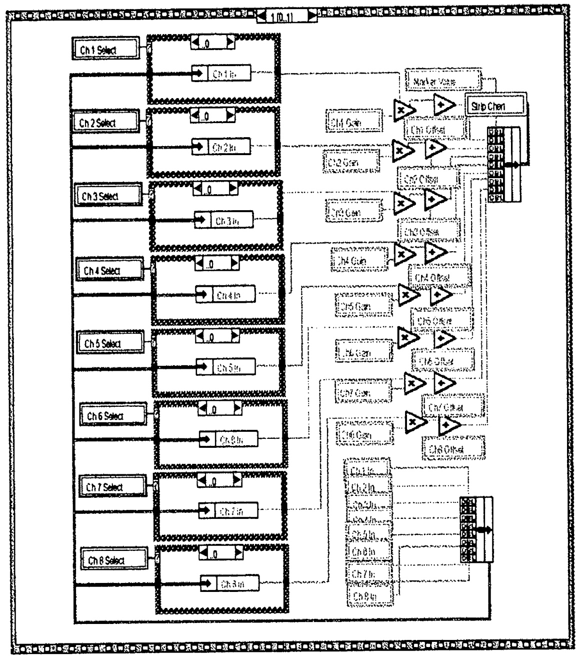
Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:12 PM

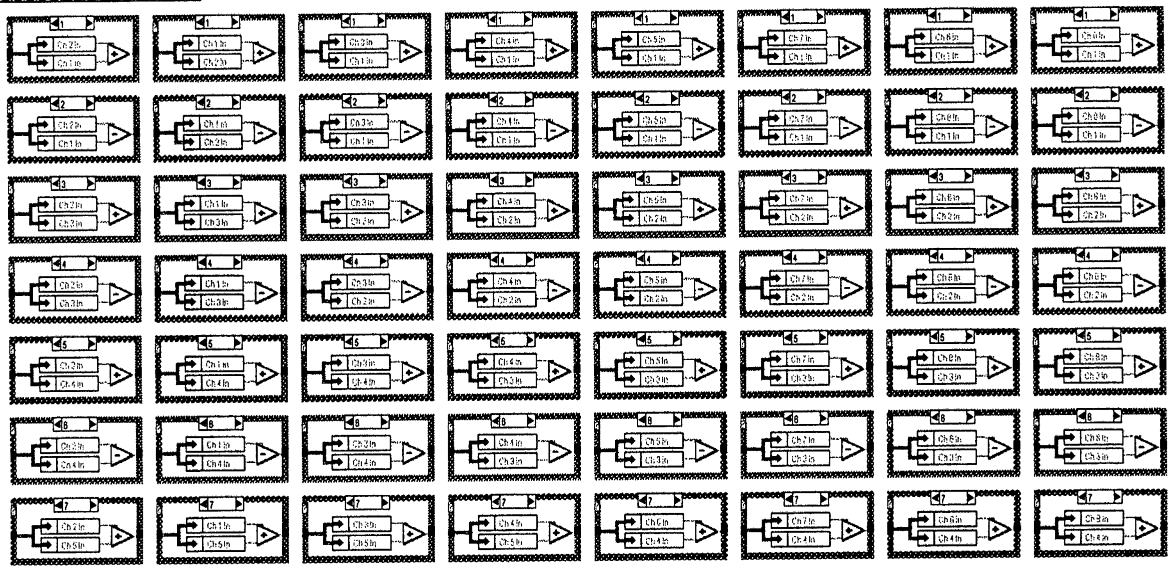


VIR

VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:13 PM

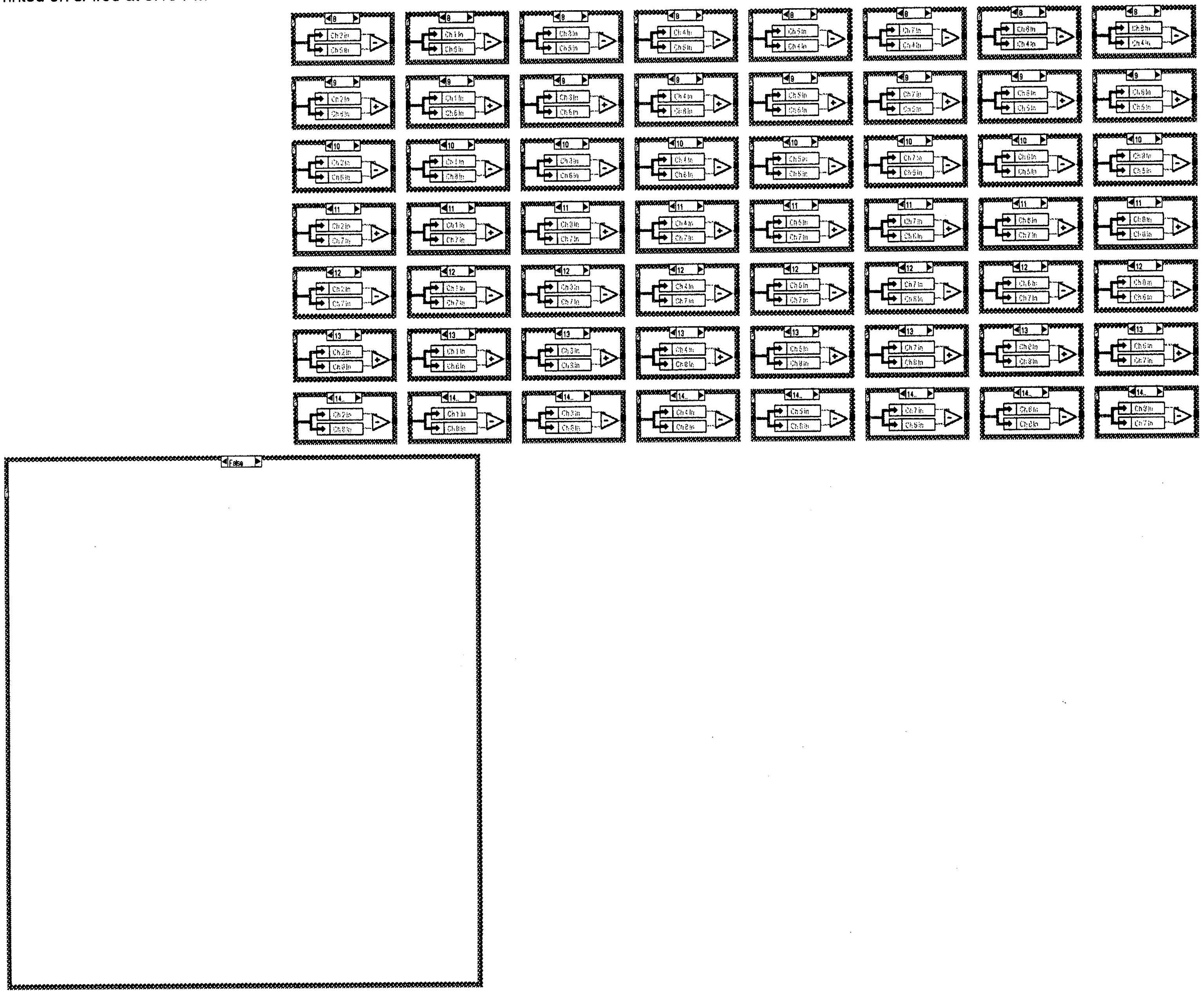




VIR

VIR_GRAF.VI

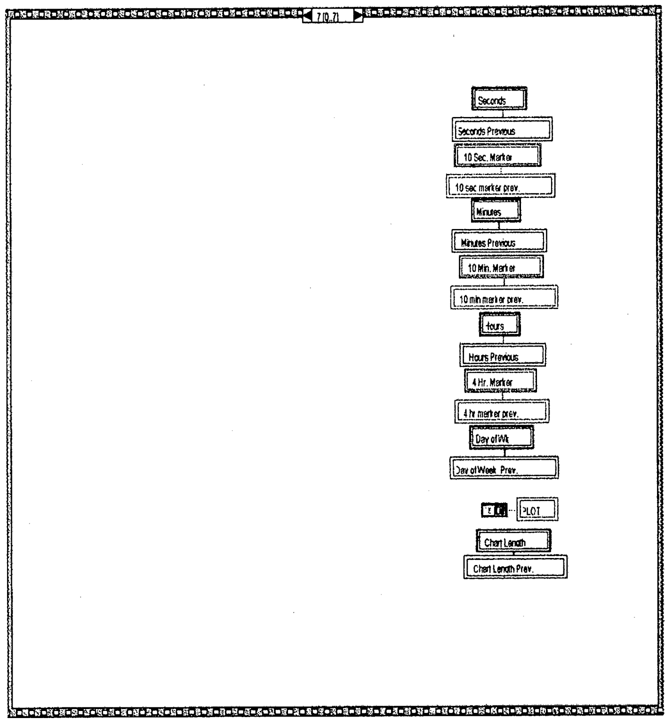
Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:13 PM



VIR

VIR_GRAF.VI

Last modified on 7/9/97 at 9:58 AM
Printed on 8/4/98 at 3:14 PM



Position in Hierarchy

2.8 WEATHER VI APPLICATION

The Weather VI displays and plots VLA weather station data from Data Set 0, DCS0. The CONTROL PANEL and BLOCK DIAGRRAM follow this text.

The weather parameters, their addresses and scaling are:

Parameter & Units	MUX $Addr_8$	Abbrv	CEL Index (Note 1)	Scale
Windspeed, M/Sec	2	WS	0	56.0 mV = 0 M/Sec 5.0 V = 25 M/Sec 10.0 V = 50 M/Sec
Wind Direction, deg	3	WD	0	0 to 10 V = 0 to 360 deg
Temperature, °C	4	T	1	3.770 V = 0 deg °C 9.490 V = 45 °C
Barometric Pressure, mB	5	BP	1	0.00 V = 650 mB 10.0 V = 950 mB
Tracked Dew Point, °F	6	TDP	2	scaling ?
Held Dew Point, °F	7	HDP	2	scaling ?
Black Solar Ball Temp, °C	10	BB	3	100 mV/°C
White Solar Ball Temp, °C	11	WB	3	100 mV/°C
Black Ball-White Ball diff, °C	12	BB-WB	4	100 mV/°C
" " "	13		4	100 mV/°C
Rain Gauge (Note 2)	220		7	Note 2

Notes:
1 CLIENT Channel Entry Table Row Index, designates parameters stored in Index Arrays.
2 Weather parameter is input to this application but is not used.

Two numeric indicators at the top of the CONTROL PANEL show the date and time of day. The first indicator shows day name (e.g. Tuesday), month name (e.g. August), day of the month and year. The second indicator shows time of day and AM or PM. Analog meters show Wind Speed (WS) and Barometic Pressure levels; their numeric values are shown on a numeric indicator below the meter. Wind Speed (WS) is often quite variable and is an important antenna and personnel safety concern; it is easier to judge average, maximum and minimum Wind Speed from a meter display than from a numeric display. Temperatures are shown as thermometers and the numeric value is shown on three digit numeric indicators below the thermometers. Wind Direction (WD) is also quite variable; a 16-element circular array of LED indicators shows wind direction with a resolution of ± 11.25 degrees; the numeric value is shown on a three digit numeric indicator above the array. Three 24-hour charts plot one or two parameters versus elapsed time from the start of the Weather VI. The time axis runs from 0 to 2400, plotted in 1/100 th of an hour increments with 15 minute ticks. The Temperature/Dewpoint chart (black) shows temperature (°F, yellow trace) and Tracked Dew Point (°F, green trace) versus time. The Wind Speed/Direction chart uses a blue trace for wind speed and a red trace for direction. This chart's vertical axis is labeled 0 to 90, indicating a 0 to 90 MPH wind scale and a 0 to 90 degree wind direction scale; the actual direction is 4 times the trace value. The circular LED display is a more granular version of this parameter. The VLA site Barometric Pressure is displayed on a meter and numeric indicator. The equivalent Sea Level Barometric Pressure is calculated from the VLA site barometric pressure; this is displayed on the bottom meter-numeric indicator and is also plotted on the bottom chart.

Note that the Block Diagram has two regions: a WHILE LOOP in the center of the page and an initialization region on the left side of the page (it's not labelled as such). The chart plotting functions performed in the weather station application necessitate data storage for up to 24 hour periods; this requires LabVIEW data storage arrays that are accessed for current and earlier values that are to be plotted. The arrays are initialized when the Weather VI is started. The Global Weather input array receives all data from the WEATHER CLIENT (see the WEATHER CLIENT description below). This array is an INITIALIZE ARRAY with three inputs: the element input which has a value of 0 sets each element's value to 0; dimension size 0 which has a value of 8 declares that there are 8 array elements (i.e. weather parameters), and dimension size 1, which has a value of 3 declares that each element consists of 3 bytes. Note that the Global Weather Input array outputs are defined as string variables (designated by the [abc] symbol); this is an important aspect inside the While Loop. Below the Global Weather array are five INITIALIZE ARRAYS. These arrays each initialize **one weather variable** (BP, T, HDP, WS and WD) of the Index Arrays that operate inside the While Loop. The array's element input are initialized to a value of -40, a value never seen in in any of the weather parameters and the dimension inputs are initialized to 2400 elements; this provides 24-hour plot periods in increments of hundredths of an hour. Note that the outputs of these five arrays are defined as double-precision variables designated by the [DBL] symbol. Since the Black Ball and White Ball and Black minus White Ball temperatures are not plotted, their arrays are not initialized.

In operation, the WEATHER CLIENT stores each 3-byte parameter in its Weather Global INDEX ARRAY in the element location designated by the CLIENT CEL Row Index. When this happens, the element's previous contents are over-written. This array drives five INDEX ARRAYS inside the WHILE LOOP. Two Control Panel indicators (lower left corner of the Block Diagram) are hidden: 1) the UPDATE indicator for the weather global.vi and 2) the Connection Status, just below the UPDATE indicator.

Inside the WHILE LOOP, the operations on the five data paths (CEL Indexes 0,1,.. ,4) are very similar: storage in an INDEX ARRAY, conversion of the hex string value to a hex numeric value, and separation of the two packed 12-bit values into individual hex 12-bit values. The 12-bit values are then summed with an offset and multiplied to produce CONTROL PANEL indicator values. Because they are all very similar, for brevity only two types of process operations are described: 1) the BB/WB temperature operations that drive CP thermometers (only), and 2) the WS/WD operations that drive CP meters, charts and an WD LED array.

Inside the WHILE LOOP there are five INDEX ARRAYS, all having a common Chart Index (1 to 2400, described below) and a CEL Row Index (listed in the table above) that designates a specific parameter in the stream of Global Weather data. Since Wind Speed (WS) and Wind Direction (WD), Temperature (T), Tracked Dew Point Temperature (TDP), and Barometric Pressure (BP) parameters are plotted in the three CONTROL PANEL Chart displays, they must be stored in INDEX ARRAYS connected to the WHILE LOOP's Global Weather Data input. Although the BB and WB temperatures are not plotted on charts, they are also input to INDEX ARRAYS. The CEL Indexes (0,1,..,5 noted in the table above) designate data paths into WHILE LOOP INDEX ARRAYS.

Each weather parameter is a string variable and consists of two independent 12-bit weather variables; for example, CEL Row Index 0 designates both WS and WD. The outputs of the five INDEX ARRAYS are input to the String input of a HEX STRING-TO-HEX NUMBER function that interprets string characters 0 through 9 and A through F (or a through f) as Hex integers which are output on the icon's number terminal. At this point, the two sets of 12-bit values on the number output can be separated for independent processing.

The simplest BLOCK DIAGRAM processing operations are those performed on the Black Ball, White Ball and BB minus WB temperatures, and with the exception of detailed differences, exemplify the first parts of all the weather application operations.

CEL Row Index 3 designates the BB and WB temperatures and Index 4 designates the BB minus WB temperature difference. All three CP thermometers show double precision floating point ([DBL]) temperatures in °C. The lower 12 bits of the HEX STRING-TO-HEX NUMBER output are selected by the FFF mask on the AND function. Minus 429 is added to the resultant value which is then multiplied by

0.10523616 and displayed on the CONTROL PANEL BB Thermometer. To obtain the WB parameter, the upper 12-bit portion of the number output is right-shifted 12 bits in the LOGICAL SHIFT function. The LOGICAL SHIFT icon's upper terminal specifies the number of bits and the minus sign specifies a right shift. 0's are shifted into the top 12 bits. The balance of WB operations are identical to that described for the BB temperature thermometer and numeric display.

The BB minus WB temperature operations are very similar to the above. CEL Row Index 4 causes this parameter to be stored in its INDEX ARRAY. The lower 12 bits of the array output are masked, summed, multiplied and displayed on the CP by operations identical to the BB and WB process above. The upper 12 bit value is not processed. Referring to the BLOCK DIAGRAM, note that the BB/WB operations do not use REPLACE ARRAY ELEMENT arrays; they are not required since these temperatures are not plotted on CP charts.

The BLOCK DIAGRAM's Wind Speed (WS) and Wind Direction (WD) operations are more complex than the BB and WB operations described above because these parameters are plotted on CP charts. In addition, WD is compared to reference values by 16 Comparison functions to develop Boolean discretes that activate a 16-element CP Wind Direction LED array. WD is scaled by a factor of 0.25 for the CP WD chart.

Although the first functions in the process are virtually identical to the BB and WB operations described above (i.e. storage in an INDEX ARRAY, conversion to Hex values and mask selection of the upper and lower 12 bit values), operations peculiar to the CONTROL PANEL displays are required because wind data is displayed in four formats: numeric displays, meter displays, a WS & WD chart and a wind direction LED display. Wind Speed and Direction in the Global Data stream are stored in an Index Array by 0, the CEL Row Index.

After separation of the two 12-bit hex values by masking-shifting, WS is multiplied by 0.025 and displayed on the CONTROL PANEL Wind Speed M/S meter. The M/S value is scaled to MPH by multiplying by 2.237 to convert it to WS MPH which is displayed on the Wind Speed MPH meter. Both values are also displayed on numeric indicators below the meters. The 16 bit integer values are designated by the bracketed [I16] symbol. The weather station's WS analog channel scaling is 5 M/s per volt: thus at 50 M/s the analog channel voltage is 10 volts.

Wind Direction is multiplied by 0.18 and displayed in a CONTROL PANEL numeric indicator above the 16-element circular Wind Direction LED display which indicates the wind's current azimuth bearing with a resolution of ± 11.25 degrees. This data is also in 16-bit integer format. The WD value is input to 17 limit comparators; if the WD value is within the comparator's limits, the associated LED is illuminated. The two limit values are + and - 22.5 degrees from the LED's angular position. The Control Panel LED display shows the SSE LED (@ 168.75 degrees) illuminated; the comparators upper and lower limits are 146.25 and 168.75 degrees, + and - 22.5 degrees from the 168.75 SSE bearing. The comparison for the N LED is a bit more complicated because two comparators are required: the 22.5 degree segment around zero degrees is double-valued; limit values are 348.75 and 11.25 degrees, respectively. One comparator has limits of 348.75 and 0; the other has limits of 0 and 11.25. If either comparator output is true, an OR function illuminates the N LED. The LED array is outside the While loop; connections from the comparators exit the loop through tunnels. All comparator outputs are Boolean, indicated by the boxed [TF] disignator.

Both WS and WD are plotted on the middle CP chart. WS is plotted as MPH and WD is plotted as WD ÷ 4 and the chart's ordinate is labelled 0 to 90. LabVIEW's data charting operation require an Index Array followed by a REPLACE ARRAY ELEMENT, both driven by the Chart Index, ranging from 1 to 2400. This function replaces with new element (the center terminal) the element in the array at the index. This causes all chart values up to the current index value to be plotted, thus simulating a running strip chart format. WS and WD REPLACE ARRAY ELEMENT array outputs are coupled by a BUNDLE FUNCTION which causes them to be plotted on a single chart. The array outputs are double precision values, denoted by the boxed [DBL] symbol.

The WHILE LOOP's control operations are the [TF] Boolean symbol in the loop's lower right corner, while T (true) the loop continues to execute. The loop index i (in the lower left corner) is driven by the WAIT UNTIL NEXT ms MULTIPLE function (Metronome icon). This function is a millisecond timer that generates an output when the elapsed time (in ms) equals the input delay value; after this output, the timer register is reset to zero and another timing cycle begins. The 1000 count input causes a 1000 ms delay; thus the loop executes at a 1/second rate.

The Chart Index (1 to 2400) is a time-related operation mentioned above; it provides a sampling signal for the charting function. The Chart Index drives the Index 1 input on the INDEX ARRAY functions and the Index input on the REPLACE ARRAY ELEMENT arrays. The Chart Index is time-dependent; the time clock is the GET DATE/TIME IN SECONDS clock immediately to the right of the metronome icon. This function returns a time-zone-independent number that contains the number of seconds that have elapsed since 12:a.m., Friday, January 1, 1904, Universal Time. This elapsed time is input to a SECONDS TO DATE/TIME function to a cluster of nine, signed 32-bit integers that specify second, minute, hour, day of the month, number of month, year, day of week, day of the year, and Standard or Daylight Savings time in the VLA's time zone. The Standard or Daylight Savings time parameter is set in accordance with the User's computer operating system setting. This cluster thus describes time in named components. The cluster is input to an UNBUNDLE BY NAME function which disassembles the time cluster into its individual named components.

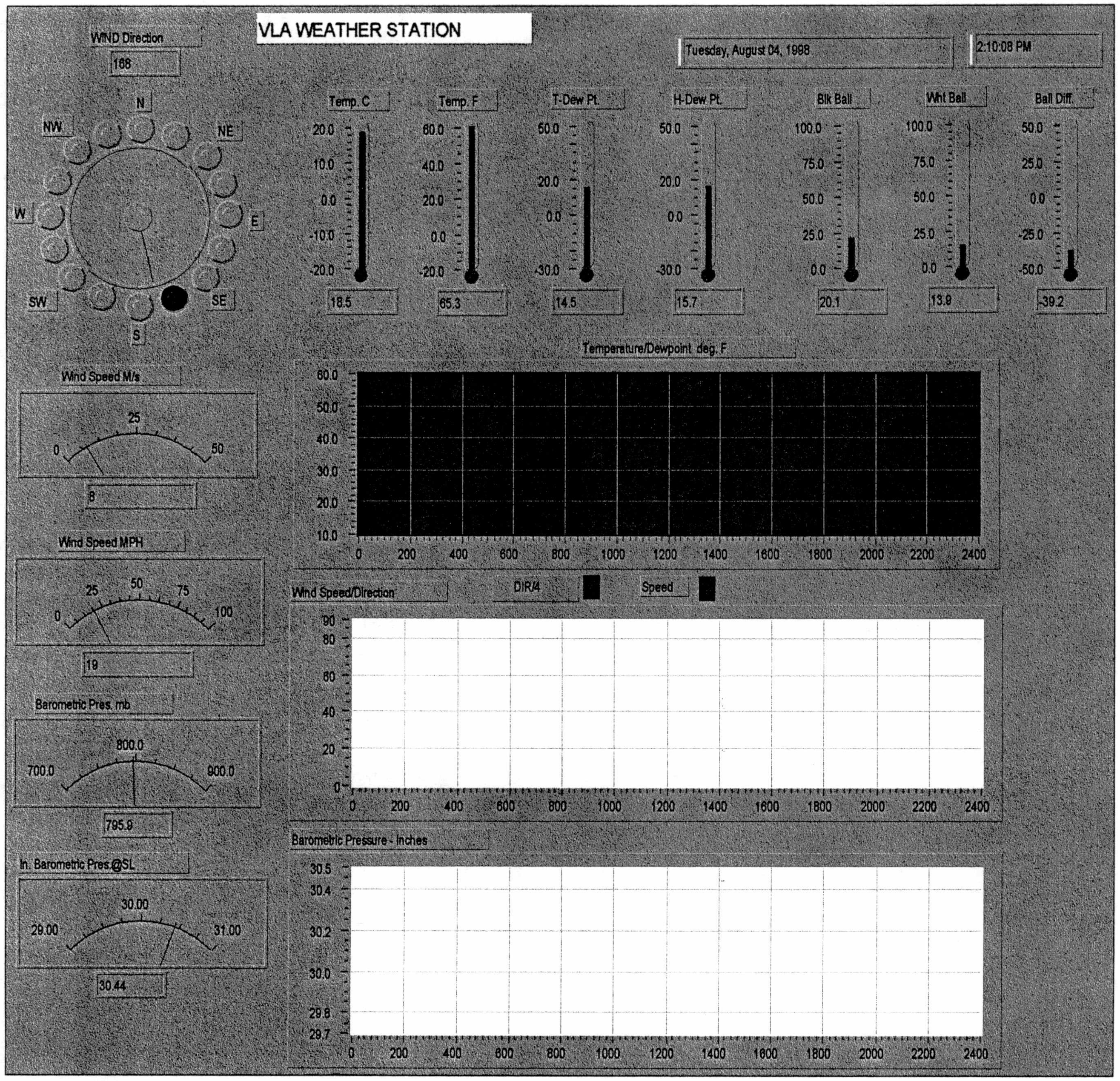
The unbundle function's minute and hour terms are connected to a scaling and sum function to produce the 1 to 2400 Chart Index. The unbundle function's minute output is divided by 60 to make a fraction of an hour value. An example Chart Index value is 2365, developed by taking minute 39, dividing it by 60 to produce a fractional hour value, (39/60 ths of an hour or 0.65). This 0.65 is added to the unbundle function's hour output (e.g. 23) to produce a 23.65 hour value which is then multiplied by 100 to produce 2365. The Chart Index plotting resolution is thus one-hundredth of an hour. The Chart Index is the index to the five INDEX ARRAYS and five REPLACE ARRAY ELEMENT arrays mentioned above.

The CONTROL PANEL's <u>date and time of day indicators</u> are developed by the GET DATE/TIME STRING function on the lower left corner, just above the Global Weather Update indicator. This function converts a time-zone-independent number assumed to be the number of seconds that have elapsed since 12:00 a.m., Friday, January 1, 1904, Universal Time, to a string in the configured time zone of the User's computer. Note that although this function appears similar to the operation performed by the GET DATE/TIME IN SECONDS and DATE/TIME TO SECONDS functions described immediately above, this function's output is a string value ([abc] symbol), not integer values of time components. This function has two string outputs: the upper output is the date string and the lower output is the time string. The 1 on the icon's upper input specifies that the date be fully spelled out on the CP's day; month, day of month and year indicator (top right). The boxed [TF] input on the lower input specifies that seconds are to be displayed on the CP's time of day indicator to the right of the date indicator.

VLA
WS

'eather.vi
:\LabviewNT\vla_wsa.llb\weather.vi
ast modified on 5/13/96 at 7:25 AM
rinted on 8/4/98 at 2:10 PM

ront Panel


VLA WEATHER STATION

Tuesday, August 04, 1998

2:10:08 PM

WIND Direction: 168

Temp. C: 18.5

Temp. F: 65.3

T-Dew Pt.: 14.5

H-Dew Pt.: 15.7

Blk Ball: 20.1

Wht Ball: 13.9

Ball Diff.: -39.2

Wind Speed M/s: 8

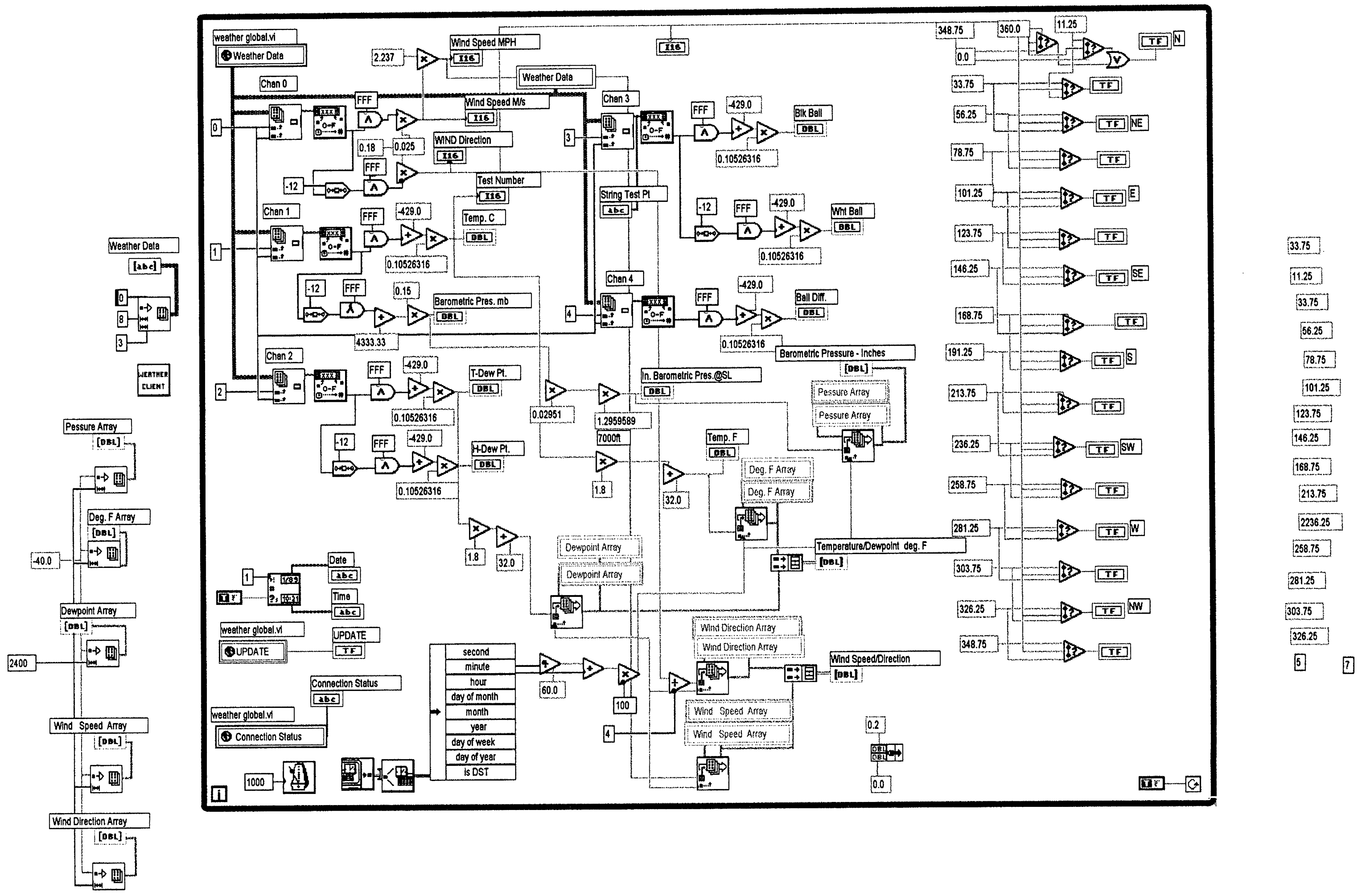
Wind Speed MPH: 19

Barometric Pres. mb: 795.9

In. Barometric Pres.@SL: 30.44

ather.vi
\LabviewNT\vla_wsa.llb\weather.vi
st modified on 5/13/96 at 7:25 AM
inted on 8/4/98 at 2:10 PM

ock Diagram



2.9 ROUND TRIP PHASE DISPLAY APPLICATION

The Round Trip Phase application is a specialized program that charts a selected antenna's 600 MHz Round Trip Phase and Azimuth and Elevation positions against time on Control Panel charts. See the RT PHASE CONTROL PANEL at the end of this text. Time is plotted along the X axis with time tick marks and time and date annotations. The chart's Y axes cover the full physical span of Azimuth (80 to 640 degrees), Elevation (0 to 130 degrees), and 600 MHz RT Phase (360 degrees).

The RT Phase Display BLOCK DIAGRAM follows the CONTROL PANEL at the end of this text.

The Control Panel has four user input parameters: <u>DCS_8</u> address numeric (octal) switch, <u>Chart Length - Minutes</u> (decimal) numeric switch, and <u>Clear Chart</u> and <u>STOP</u> push-button switches. (In this text Control Panel switches and indicators are underlined to make them distinct from program variables.)

The Control Panel displays <u>ANT_{10}</u> (Antenna ID) on a decimal numeric indicator, <u>Connection Update</u> (flashing LED) and the three function Vs. time charts. The <u>Connection Update</u> LED flashes when the RT Phase CLIENT provides a new set of data values for charting.

In using this application, an antenna is selected via the Control Panel's <u>DCS_8</u> address selector switch, the usual M&C System antenna reference. The program obtains the selected antenna's ID number from the monitor data and displays it on the Control Panel's <u>ANT_{10}</u> numeric indicator. After the <u>DCS_8</u> switch has been set to the desired address, the user actuates <u>Clear Chart</u>. This removes any existing chart traces, reads the DCS_8 switch, initializes the program in accordance with the DCS_8 and Chart Length settings, and starts the charting sequence.

The user may exit a charting sequence at any time by setting the Stop switch; this causes the program to enter the Pause state which halts data charting but keeps the charts intact for the user's review. There is not a resume function after STOP has been actuated because resumption of charting would cause a potentially confusing discontinuity in the charts.

After setting <u>Clear Chart</u> to remove the existing traces, charting can be restarted with either the current or another DCS address or a different Chart Length setting.

After completing or exiting a charting sequence, the user may print the charts (i.e., the Control Panel) via Windows facilities.

Refer to Section 2.3 which shows the Weather CLIENT's Control Panel; the RT Phase CLIENT is virtually identical but the Channel Entry List addresses differ. Note that the list's entries are MW Select, and **decimal** DCS, **octal** DSA and Mux addresses for each of the eight channels (0,..7). The RT Phase CLIENT's Channel Entry List addresses are preset as follows: Channel 0: 1, N, 0, 210 (Azimuth position); Channel 1: 1, N, 0, 212 (Elevation position); Channel 2: 1, N, 5, 220 (RT Phase); Channel 3: 1, N, 2, 220 (Antenna ID); Channels 4,..7 are not used. N is the initial DCS address which is modified by the program to the address set on the DCS_8 switch. The program's operations assume this channel order. It must not be changed because monitor data is accessed as a function of the list's row indices: 0, 1, 2 and 3. The RT Phase CLIENT Channel Entry List is actually another VI which is sent by the RT Phase CLIENT to the SERVER after the initial DCS address has been replaced by the value set on the DCS_8 selector switch.

The RT Phase application is capable of producing plots with a high data bandwidth. The WHILE LOOP execution frequency, is 6.66 Hz, roughly the sampling rate of the antenna position data.

The Channel Entry List's MW select entries are all preset to 1; this yields antenna position sampling rates of 9.6 Hz (by DS0) and a RT Phase sampling rate (by DS5) of 0.8 Hz. (DS5's MW1 samples 600 MHz Round-Trip phase every 24 WG cycles; since there are 192 WG cycles in 10 seconds, the sampling rate is 0.8 Hz, 192/24 = 8.) Antenna ID is sampled every ten seconds by DS2. It is potentially possible to increase the plot data bandwidth by changing some parameters. The RT Phase's MW select entry could be changed to MW2, and DS5 could be commanded to sample Mux 220_8 in the intensive sample mode. This would make the RT Phase sampling rate 19.2 Hz. Since the While Loop execution rate is 150 ms, the loop period should be reduced to avoid data aliasing. Although using RT Phase's MW2 entails fiddling with the M&C System and temporarily changing the RT PHASE CLIENT, doing so provides the highest possible chart phase-time resolution. Attempting to increase position readout sampling rates by selecting MW 2 (for DS0) is not a valid option because DS0's MW2 functional usage differs from MW2 usage in the other Data Sets (DS1,..DS5).

The RT Phase and position data readout resolutions are: 600 MHz RT Phase, 1 LSB count = $1/2^{18}$ turns, Azimuth and Elevation, 1 LSB count = 1.2 arc-seconds.

This program can be considered to consist of three operations: 1) initializing the program by changing the DCS address value in the CLIENT's Channel Entry List to the value set on the $\underline{\text{DCS}}_8$ switch and sending it to SERVER; 2) scaling the chart's X-axis update rate in accordance with the value set in the <u>Chart Length</u> switch; 3) charting the three parameters vs. time.

Initialization

The left edge of Page 3 of the BLOCK DIAGRAM, outside the MAIN WHILE LOOP, contains the initialization functions. Notable among these are the BUNDLE AND CHART Function (labelled 600 MHz Round Trip Phase vs Antenna Position) which defines the three variables as double precision floating point and sets the initial values to 0.0. The CHART FUNCTION icon is the small two-line rectangle enclosing tiny squares and rectangles. (This is a CP Indicator; a CP Control would be denoted by a three-line rectangle.) The GET DATE AND TIME function (clock icon) returns a time zone independent number (unsigned 32-bit integer) that contains the number of seconds that have elapsed since 12:00 A.M., Friday, 1/1/1904. The RT Phase CLIENT.vi is also defined. Within the CLIENT is the Channel Entry List (string variables) which will be modified in the first part of the program. Other variables that are initialized are Seconds Now and Seconds Previous (unsigned 32-bit integers), Chart Length - Min (16-bit unsigned integer), and Boolean variables, Pause - Pause Previous, Clear, and Init (Initialize).

Main While Loop

As noted above, the MAIN WHILE LOOP (BD Page 3) executes at a 6.66 Hz rate, determined by the WAIT UNTIL NEXT ms MULTIPLE function (Metranome icon, lower left corner). The program executes when the millisecond timer becomes a multiple of the specified period – in this case, 150 ms.

The logic inverse of the CP <u>STOP</u> switch state drives the WHILE LOOP enable function (lower right corner). After program execution has been started, actuating the <u>STOP</u> switch causes the program to enter the Pause state which halts data charting but keeps the chart intact for the user's review. As noted above, there is not a resume function after <u>STOP</u> has been actuated because resumption of charting would cause a potentially confusing discontinuity in the charts.

After establishing the loop execution rate and run/stop condition, the program enters frame 0 of N[0..2]. This MFS is the program's main operation sequence; frame 0[0..2] initializes the charting sequence and frames 1[0..2] and 2[0..2] perform the charting function.

0[0..2] does two things: 1) It reads current time via the GET DATE/TIME IN SECONDS function (clock icon); this reading is the Seconds Now variable. Time is the driving parameter (X_0) in the charting function; the value read by the GET DATE/TIME function is the chart start time. 2) The DCS_8 address entries in the first four lines of the RT Phase CLIENT Channel Entry List are changed to the value set by the user on the Control Panel $\underline{DCS_8}$ switch. This second function is performed by the N[0..4] sequence. The switch value is converted to decimal by a TO DECIMAL function which converts the octal numeric value to a decimal string that is input (via a tunnel) to the first four frames of the N[0..4] MFS (Sequence B).

Modifying the DCS Address in the CLIENT's CEL

Multi-Frame Sequence N[0..4] within 0[0..2] (BD P1) modifies the DCS address value in the first four lines of the RT Phase CLIENT's Channel Entry List array. Frames 0,..3 each replace the initial DCS value with the decimal equivalent (converted by the TO DECIMAL function) of the value set on the CP's DCS_8 switch. Frame 5 sends the modified CEL to SERVER.

In each of the first four frames, the (decimal) converted DCS address value is input to the New Element input of the Channel Entry List array, a REPLACE ARRAY ELEMENT array. Note the array indices. The upper index is the channel (or row) index of the Channel Entry List; in frame 0 it is 0, denoting the Azimuth line, in frame 1, it is 1 denoting the Elevation line, in frame 2 it is 2 denoting the RT Phase line and in frame 3 it is 3, denoting the Antenna ID line. The lower index of all four arrays is 1 denoting X_0, the current chart time value, described later. Frame 5, 4[0..4] invokes the RT Phase CLIENT which sends the modified Channel Entry List array to SERVER; this CLIENT operation is described in Section 2.3. These operations are depicted in Block Diagram Pages 3, 4 and 5 and in Sheets 1 and 2 of the Sequence Diagram.

Scaling the Chart X Axis

Frame 1 of N[0..2] (BD P5) uses a NOT EQUAL? function to compare the Seconds Now variable with the Seconds Previous variable. The comparison result is input to a Boolean Case Structure; if the comparison is true (i.e. variables not equal), another set of chart values should be calculated and plotted. If the comparison is false, the program loops until the variables differ.

Inside this Seconds comparison BCS is frame 0 of N[0..1] which contains two independent Boolean Case Structures. Entry to one is a logic function of the Clear/Init state. Entry to the other is a logic function of the Pause/Pause Previous state. The two sets of variables are unrelated and have no precedence relationship. The Clear/Init BCS operations are considered first; the Pause/Pause Previous operations are considered below.

Clear is the state of the Control Panel <u>CLEAR CHART</u> switch. Init was set true during program initialization, prior to entry of the MAIN WHILE LOOP. Clear and Init can both be false since Clear results from the actuation of the <u>CLEAR CHART</u> switch and Init is false after program execution has started. In either case, the chart time base X_0 (Seconds Now) should be reset to 0 because the user wants to start a new charting sequence. If either Clear or Init are true, the OR function causes causes entry to the BCS and execution of MFS N[0..1].

Frame 0 of N[0..1] is on BD P6. This frame's operations are a bit obscure but it resets the chart time base, X_0. A value of 0 is entered to a one-dimensional BUILD ARRAY function which then drives an ARRAY TO CLUSTER function that converts the array to a cluster of elements (1) of the same type as the array element (1). The cluster output is then re-converted to a one-element array by another BUILD ARRAY function. The array output is the one-element RT Phase vs ANT Position array, X_0 that is operated on in Frame 1 of N[0..1], BD P5.

In frame 1 of N[0..1] X Maximum is calculated from the X_0 (Seconds Now) variable and the Control Panel's <u>Chart Length-Min</u> control. X Maximum = (60 times CL-Min - 1) + X_0. It should be noted that this array is distinct from the three element array, 600 MHz Round Trip Phase vs Antenna Position which is the array that is used for plotting in 1[0..1], Sequence C. This frame completes the operations based upon the Clear OR Init logic function. Also in this frame, Clear and Init states are set false so that the Clear/Init OR logic does not cause this sequence to be entered again.

We now consider the other BCS of frame 0 of N[0..1], the BCS dependent upon the (Boolean) Pause/Pause Previous logic comparison, mentioned four paragraphs above. The Pause state is compared with Pause Previous state; if they are unequal, the output is true. This BCS is entered if there is more than one actuation of the CP <u>STOP</u> switch. X Maximum is calculated as described in the previous paragraph. The Pause state is set to the Previous state.

Charting the Data Values vs. Time

Frame 1 of N[0..1] is where the data charting function is performed. If the Pause state is false, the new Azimuth, Elevation and RT Phase data values are plotted and Antenna ID is displayed on the ANT_{10} CP indicator. SERVER provides the data via RT Phase Global.vi which is connected to the Element input of four INDEX ARRAYS, one for each variable. Note the array indices; the upper indices are the RT Phase CLIENT CEL row indices. Remember from the CEL description above that 0 designates Az position data, 1 designates El Position data, 2 designates RT Phase data and 3 designates the Antenna ID data. The lower array indices are all driven by X_0, the chart time variable. The four Index Arrays returns the array element designated by the two indices and each value is converted to a hexadecimal string by TO HEXADECIMAL functions. The upper bits in the 24-bit data values are masked by 21-bit Azimuth and 20-bit Elevation position mask, 1FFFF and FFFF, respectively. The resultant data is multiplied by a conversion factor, 0.0003433 and the double precision, floating point values are input to the BUNDLE-CHARTING function. The 24-bit fixed-point RT Phase data (a fraction of a turn) is converted to double precision, floating point format, divided (i.e. scaled) by FFFFFF, multiplied by 360 and input to the Bundle-Charting function. The string Antenna ID value is masked by 1F and the unsigned 8-bit output drives the ANT_{10} CP indicator.

Frame 2 of N[0..2] (BD P8) sets the Seconds Previous variable to Seconds Now for the next comparison in 1[0..2]. The connection update from RT Phase Global.vi is connected to the selector of two BCSs; if there is a new packet of data from SERVER, the CP's Connection Update LED indicator flashes. After completing frame 2[0..2] operations, control reverts to the start of 1[0..2] for another plotting cycle.

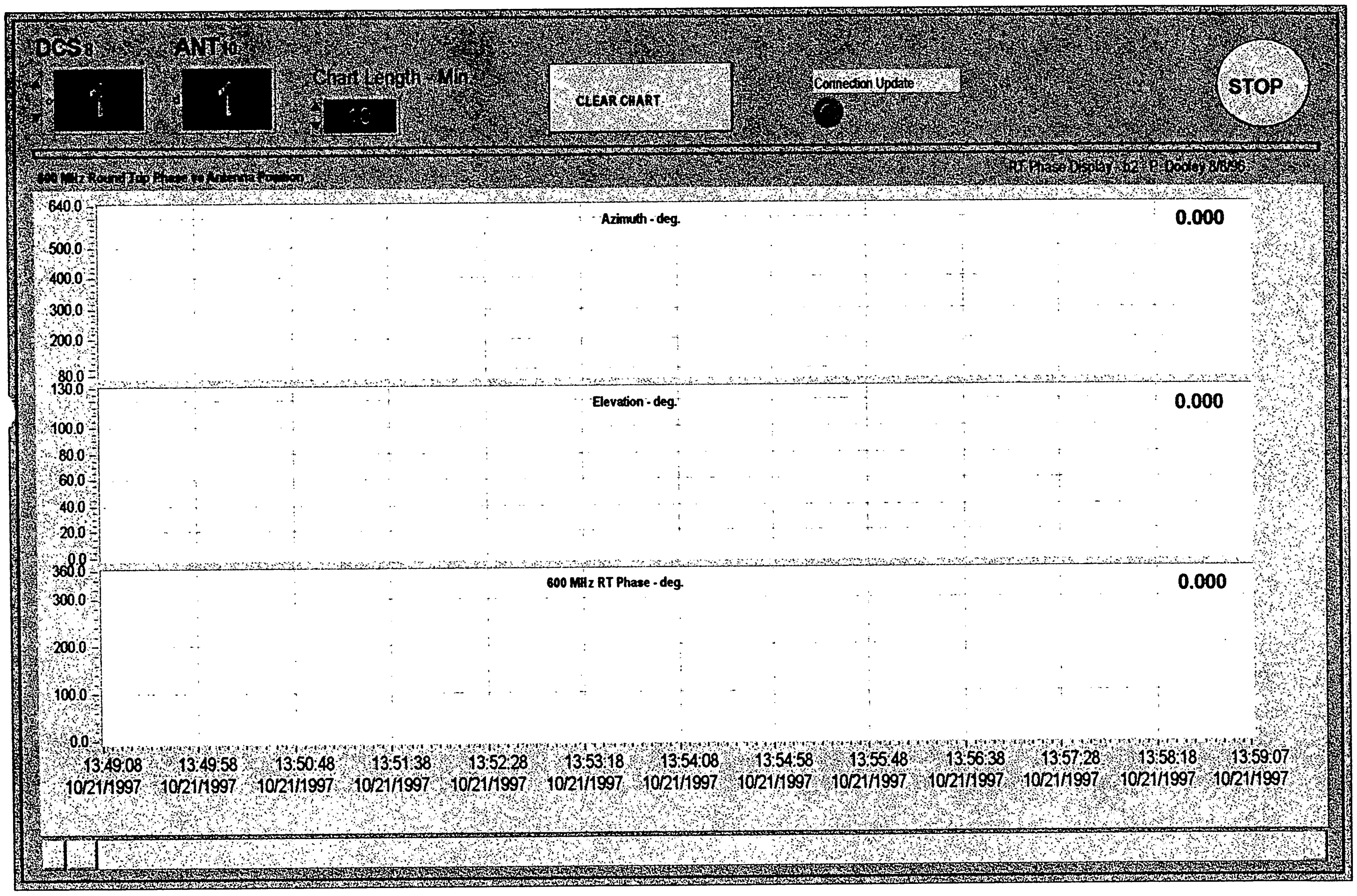
RT D:\LabviewNT\RT Phase App.vi

ast modified on 10/21/97 at 1:50 PM
Printed on 7/7/98 at 11:58 AM

Connector Pane

RT Phase App.vi

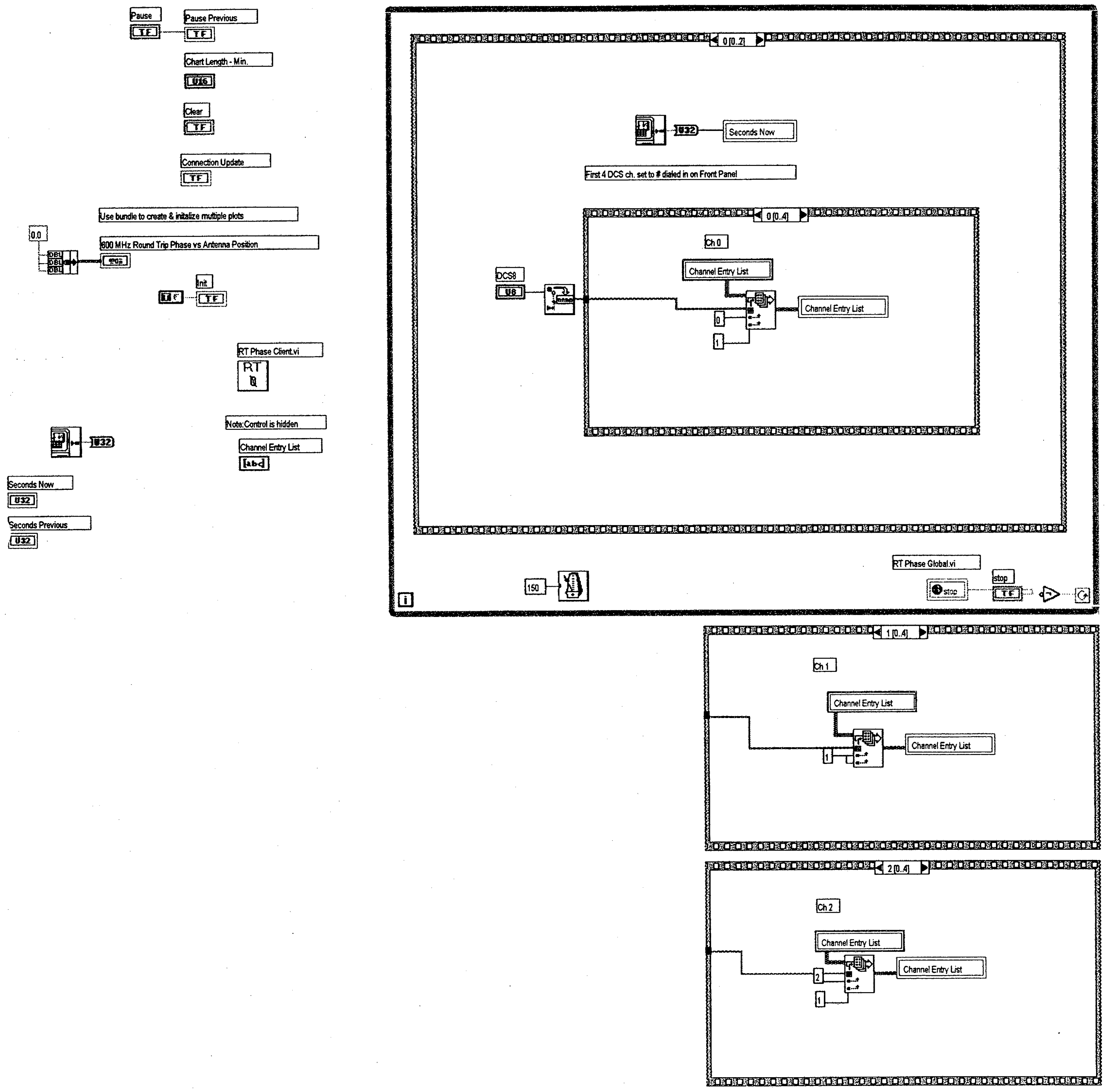
Front Panel



RT |D:\LabviewNT\RT Phase App.vi

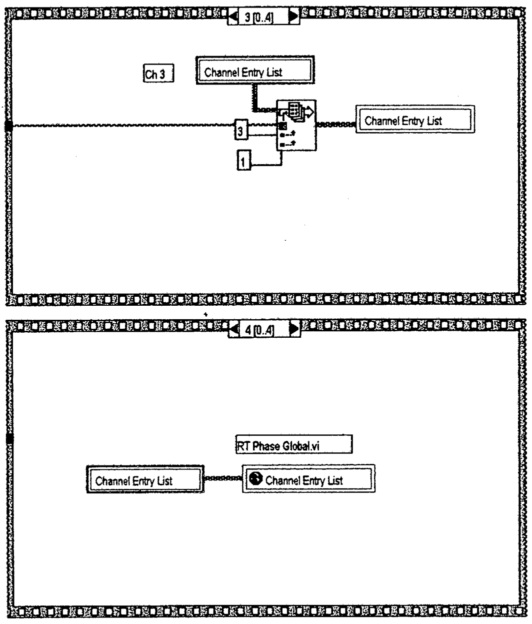
'.ast modified on 10/21/97 at 1:50 PM
Printed on 7/7/98 at 11:58 AM

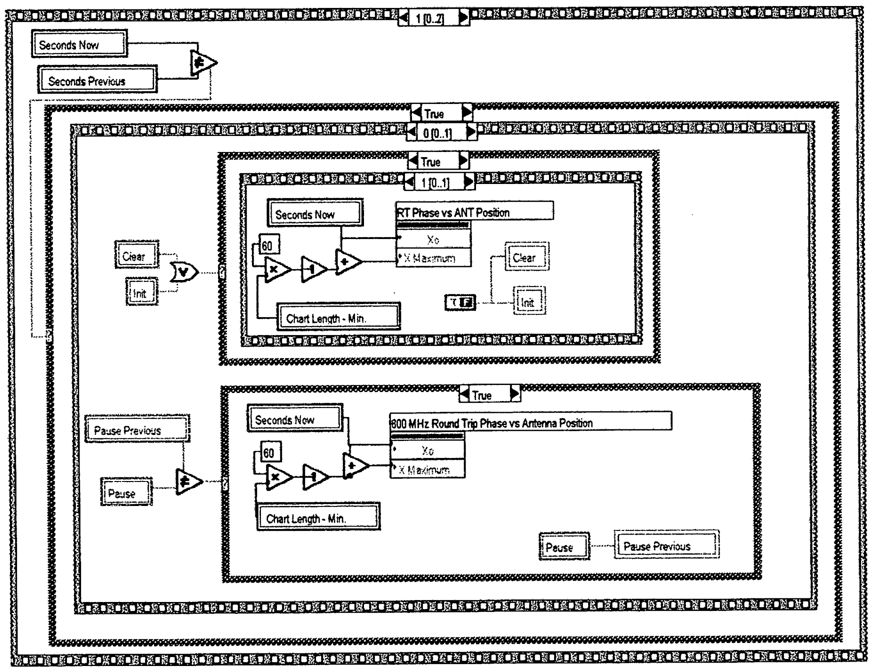
Block Diagram

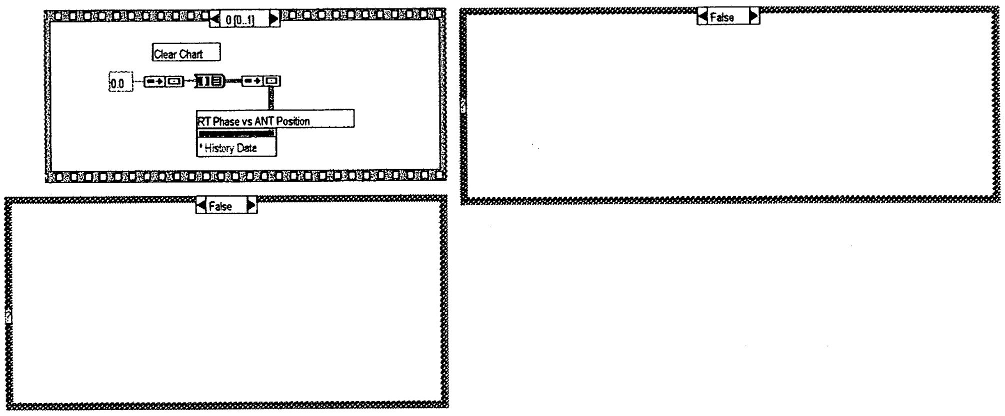


RT PD:\LabviewNT\RT Phase App.vi

Last modified on 10/21/97 at 1:50 PM
Printed on 7/7/98 at 11:59 AM

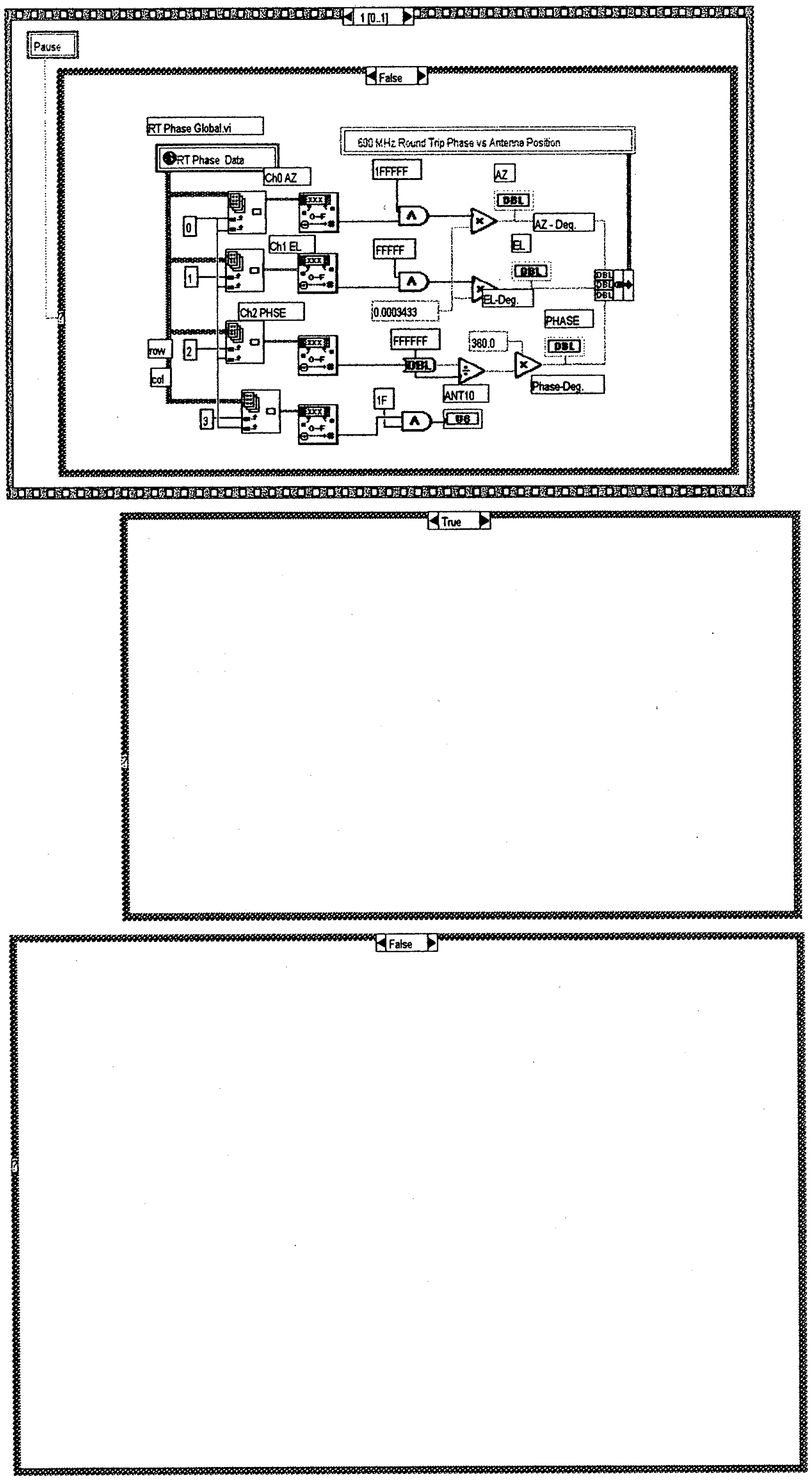






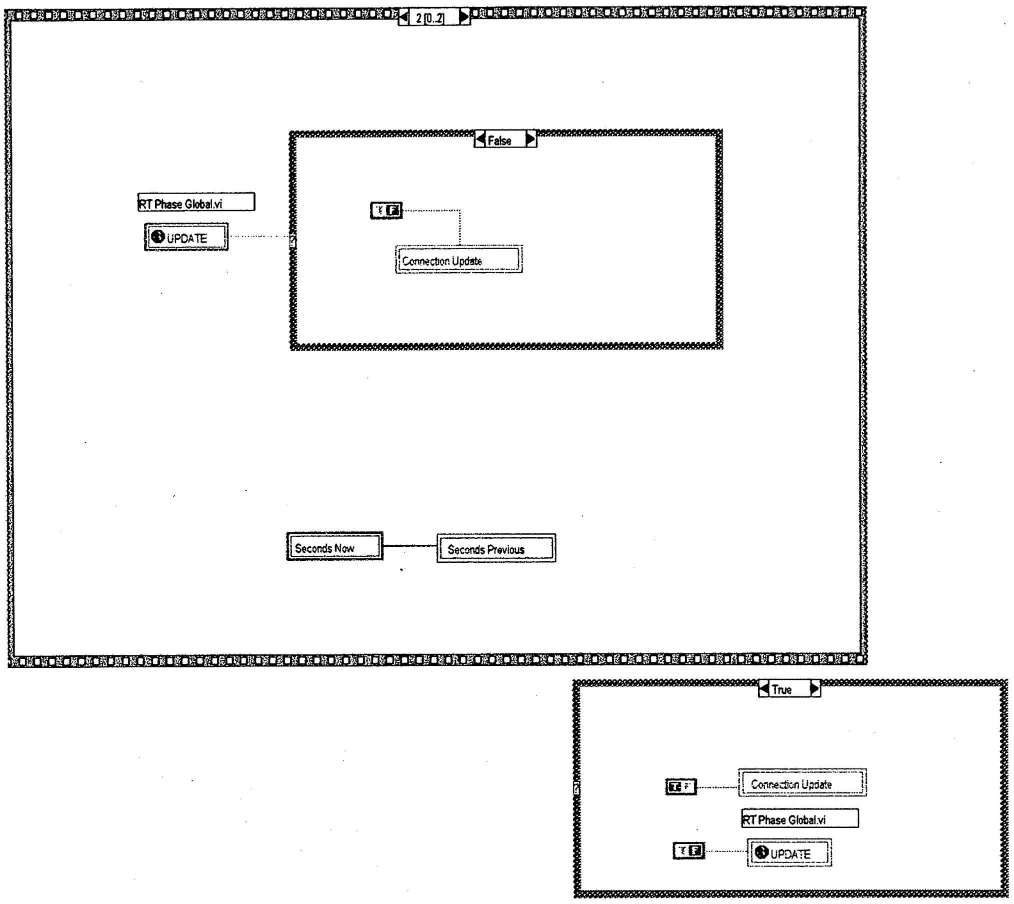
RT [D:\LabviewNT\RT Phase App.vi

'Last modified on 10/21/97 at 1:50 PM
Printed on 7/7/98 at 11:59 AM

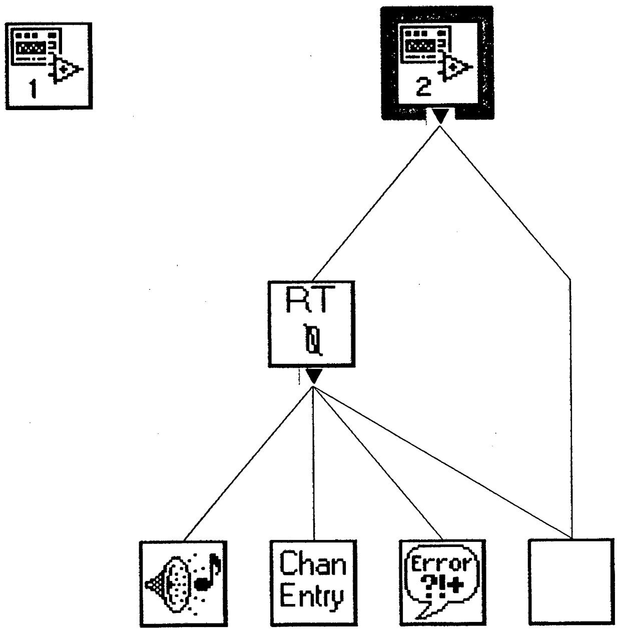


RT P D:\LabviewNT\RT Phase App.vi

Last modified on 10/21/97 at 1:50 PM
Printed on 7/7/98 at 11:59 AM



Position in Hierarchy



APPENDIX Monitor and Control System Overview

This Appendix is a brief overview of the monitor data gathering aspects of the VLA Monitor and Control System (M&CS). The VLA M&CS is the control-monitor interface between the VLA control computers and the VLA antennas and electronics. The M&CS uses time-serial digital messages to distribute computer commands to address-designated devices, and evokes monitor data messages from these devices which are input to the computers for VLA system performance analysis. For additional details on monitor data functions in the M&CS, see VLA Technical Reports No. 30 (the Data Set) and 63 (the Serial Line Controller).

Monitor and Control System Structure

The M&CS consists of four types of units: Data Sets which are the electronics rack local data-control terminals, Antenna and Central Buffers which poll Data Sets for monitor data messages, and the Serial Line Controller which polls the Central Buffers for monitor data messages which it inputs to the VLA control computers.

The M&CS is closely linked to the Local Oscillator and Waveguide transmission systems (LO & WG) because these systems are the transmission medium for antenna command and monitor data messages. These messages are modulated on 1800 MHz reference signals in the waveguide modem channels and message transmissions are synchronized to the modem's transmit-receive cycle. The LO & WG systems operate half-duplex at a 19.2 Hz rate; during the first phase (1000 μS), the Control Building Electronics Room modems transmit command and reference signals to the antenna modems which operate in the receive mode. During the second phase (about 51,000 μS), the modems operate in the opposite modes; the antenna modems transmit and the Electronics Room modems receive. During this second phase, the Antenna Buffer polls monitor data messages from five antenna Data Sets. These messages and I. F. signals are transmitted to the Electronics Room modems where the antenna monitor data messages are detected and stored in Central Buffers. The Central Buffers also poll monitor data messages from the Data Sets in the Electronics Room "D" racks. After these messages have been stored, the Serial Line Controller (SLC) sequentially polls the Central Buffers for the Data Set's monitor data messages which are input to the control computers. During the polling process, the SLC "repeats" the serial monitor data messages on its command ports. The VIR system is tapped onto this stream of command and monitor data messages; it ignores command messages but stores monitor data messages for subsequent use.

Antenna-Related Data Sets

Data Sets are installed in each electronics rack to perform the rack's control and monitor data gathering functions. In the antenna, Data Set 0 is the Antenna Control Unit (ACU); Data Set 1 services the "A" rack front ends, Data Set 2 services the "B" rack LO and modem system; Data Set 3 services the Subreflector Focus and Rotation system, and Data Set 4 services the X, L and Q band front ends. In the Electronics Room, Data Set 5 services the "D" rack LO and modem.

Electronics Room Functions Data Sets

DCS0 (see the DCS address usage described below) is not associated with an antenna but services systems in the Control Building Electronics Room. Data Set 0 services the VLA weather station; Data Set 1 services the Active Master L.O.; Data Set 2 services the Spare Master L.O.; Data Set 3 services the Fluke Synthesizers; Data Set 4 services the Analog Sums and Data Set 5 services the Hydrogen Maser.

Monitor and Control Message Format

A fundamental parameter in the VIR system is the M&CS 5-byte time-serial message format. The VIR system decodes the DCS, DS and Mux address components in gathering VLA monitor data. The message format is depicted below.

```
DCS Address  DS Addr  Mux Addr         Data Byte 1      Data Byte 2      Data Byte
5 bits       3 bits   8 bits           8 bits           8 bits           8 bits
D - - - D    D - D    M - - - - - - M  D - - - - - - D  D - - - - - - D  D - - - - - - D
C - - - C    S - S    U - - - - - - U  B - - - - - - B  B - - - - - - B  B - - - - - - B
S - - - S    2 1 0    X - - - - - - X  2 - - - - - - 1  1 - - - - - - 0  0 - - - - - - 0
4 3 2 1 0             7 6 5 4 3 2 1 0  3 - - - - - - 6  5 - - - - - - 8  7 - - - - - - 0
```

The message's MSB is DCS4; this is the first bit transmitted in the time-serial sequence. The DCS and DS addresses are packed into one byte.

The DCS Address capacity is 32. Addresses 1 through 27 designate 27 "D" racks in the Electronics Room. These "D" racks communicate with the 27 antennas via the modems and waveguide as described above. DCS address 0 is assigned to Electronics Room functions. At present, there are four unused DCS addresses. It should be noted that the DCS Address is not the antenna serial number; any antenna can be assigned to any modem channel.

The DS address designates Data Sets 0 through 5. Data Sets 0 through 4 are in the antenna; Data Set 5 is in the the Electronics Room "D" rack. Although the DS address could designate eight Data Sets, the Antenna and Central buffers are only equipped with ports for six Data Sets.

For simplicity, the format shown above does not include the serial parity bit that follows each byte. The SLC tests monitor data message parity as it is polled from the Central Buffers; if an error is detected, a flag bit is set to indicate a parity error. If the M&CS-VIR interface detects a parity error, the error-tainted data is thrown away.

There are three types of Monitor and Control messages, each of which is assigned unique multiplex address ranges: 1) Analog monitor data messages, addresses 0 to 127 (0 to 177_8); 2) Binary monitor data messages, addresses 128 to 191 (200_8 to 277_8); and 3) Command messages, addresses 192 to 256 (300_8 to 377_8). Thus the Data Set's data-control capacity is 128 analog monitor channels, 64 binary monitor channels and 64 command channels (256 channels total).

Binary monitor data messages use the format depicted above, but Analog monitor data messages pack two 12-bit values into the three data bytes. These are the digital values of two different analog signals, sequentially converted by the Data Set's ADC. The Analog monitor data format is depicted below.

```
DCS Address  DS Addr  Mux Addr         Mux + 1 Analog Data        Mux Analog Data
5 bits       3 bits   8 bits           12 bits                    12 bits
D - - - D    D - D    M - - - - - - M  D - - - - - - - - - - - D  D - - - - - - - - - - - D
C - - - C    S - S    U - - - - - - U  B - - - - - - - - - - - B  B - - - - - - - - - - - B
S - - - S    2 1 0    X - - - - - - X  1 - - - - - - - - - - - 0  1 - - - - - - - - - - - 0
4 3 2 1 0             7 6 5 4 3 2 1 0  1 - - - - - - - - - - - 0  1 - - - - - - - - - - - 0
```

Note that there is only one multiplex address in the message format; the Mux + 1 address is implied and is one count higher than the Mux Address. The Data Set ADC first converts the analog signal selected by the Mux address; after conversion, the Mux address counter is incremented and the analog signal selected by the incremented address is converted.

During each VLA machine cycle, the Antenna and Central buffers poll the six Data Sets for six MW1 and six MW2 monitor data messages. The Antenna Buffer polls five MW1 and five MW2 messages from Data Sets 0 through 4 and the Central Buffer polls Data Set 5 for a MW1 and a MW2.

The central control computer's Data Checker program analyzes MW1 data to evaluate VLA system performance. Since Data Sets are installed in racks that perform different system functions, the composition of monitor data channels and MW1 sampling sequence are peculiar to each Data Set. MW1 multiplex addresses are read from an EPROM programmed to sample data at rates appropriate for the device being monitored. The EPROM channel sequence does not include unimplemented addresses. The highest possible EPROM program channel sampling rate is 19.2 Hz (which in practice is never used), and the lowest possible rate is one sample per 10 seconds.

MW2 addresses are the state of a MW2 register-counter. MW2 addressing can be set to either of two modes: intensive sampling of a single channel at a 19.2 Hz rate, or a sequential scan mode in which the 128 analog data channels are sampled once in 5 seconds and the 64 binary monitor data channels are sampled once in 10 seconds. MW2's mode and the intensive sampling multiplex address can be controlled by the MW SCREEN overlay program; this can be done without perturbing the telescope operation. MW2 is intended to be a diagnostic tool to enable monitor data to be sampled at the fastest possible M&CS rate. A VIR user may want use this feature to increase a channel's sampling rate. If intensive sampling is not required, the sequential scan mode increases the MW1 + MW2 data sampling rate of the 192 data channels (19.2 Hz x 10 seconds) in a rather complicated way.

Data Set 0, the ACU, is the exception to the Data Set monitor data MW1-MW2 address usage described above. Antenna position and mode commands are executed by the ACU which also reads out drive system monitor data. The ACU command and monitor data message formats are compatible with the M&CS but the ACU differs from the Data Set in that it does not have an EPROM programmed with an ACU MW1 address sequence. Secondly, the MW1, MW2 address sequences are fixed by hardwired logic and ACU monitor data is conveyed by both MW1 and MW2. Consequently, the Data Checker program (mentioned above) uses both MW1 and MW2 monitor data messages in evaluating the antenna drive system performance.

M&CS Monitor Data Address Space

The M&CS's monitor data address space is 36,864 channels (194 analog and binary data addresses per Data Set X 6 Data sets per DCS address X 32 DCS addresses). Since two analog monitor data channels are packed into one monitor data message, the system's monitor word message capacity is 24,576 (128 monitor data messages per Data Set X 6 Data Sets X 32 DCS Addresses). Currently, about 9,900 monitor data channels are used.

SLC Monitor Data Message Polling Rate

During one 19.2 Hz cycle the SLC polls 384 monitor data messages from the Central Buffers (32 DCS addresses x 6 Data Sets x 2 Monitor words per Data Set). Therefore the VIR system monitor data message input rate is 1228.2 messages per second (384 X 19.2).

Monitor Word Message Rate

It may be useful to cite a few examples of MW1 monitor data message rates. As described above, MW1 message address schedules are programmed into the Data Set's 192-state address EPROMs (the EPROM address register is incremented each 19.2 Hz cycle); therefore the resultant channel data sampling rates are a function of this address schedule. A listing of all six Data Set's MW1 address sampling rates is beyond the scope of this appendix; however the following are some frequently-sampled functions: The ACU's antenna azimuth and elevation position sampling rate is 9.6 Hz. The subreflector's rotation and focus positions are sampled at a 4.8 Hz rate. The following Front End functions are sampled every 16 machine cycles, which is a 1.2 Hz rate: Gated Total Power, ALC and Sync Det. Many B-rack LO functions are sampled every 24 machine cycles, which is a 0.8 Hz rate. The sampling rate of some lightly-loaded Data Sets is quite high, not because it is necessary, but just to fill the EPROM's address space. The VLA Weather Station in Data Set 0, DCS 0 is an example of such. In gereral, one can assume that most addresses are sampled two to five times in ten seconds; only a few channels are sampled once per ten seconds.