

2030
Analog Programming

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.1 AUXILIARY RELAY AND FET BOARDS (AUXRLY)

BASIC VERSION

AUXRLY's have both sides of a Form A contact connected to the patchboard. They are provided to the user for auxiliary switching. There are 32 switches / circuit board.

The AUXRLY call provides on/off control for both the AUXRLY and AUXFET boards. The AUXRLYs are reed relays used for utility and digital switching. The AUXFET are opto-isolated, bipolar, VMOS, FETS used for power switching.

NOTE: AUXFETs have 1500pf open circuit capacitance, and are not useful for digital switching. AUXRLYs can be damaged by the high in-rush currents of capacitive loads.

run AUXRLY (Chan1, State1, C2, S2, . . . Cn, Sn)

where:	chan	=	0	to 31, (1 Auxrly or Auxfet board)
		=	0	to 63, (2 boards), etc.
		=	<0	Implies a list.
	State	=	0	Turn relay off.
		=	1	Turn relay on.

EXAMPLES:

run AUXRLY (9, 1)

run AUXRLY (9, 1, 4, 0, 5, 1)

run AUXRLY (2, 0, -5)

run AUXRLY (0, 1, -5)

Turn AUXRLY 2 through 5 off

Turn AUXRLY 0 through 5 on

Turn AUXRLY 9 on
Turn AUXRLY 9 on, 4 off, and 5 on

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.1 AUXILIARY RELAY AND FET BOARDS (AUXRLY)

SPECIFICATIONS:

Number of boards per testhead		10 maximum
AUXRLY Software execution time		6 milliseconds
AUXRLY Contact configuration	1 FORM A	
AUXRLY Contact resistance, initial		0.2 ohms
AUXRLY Contact material		Ruthenium
AUXRLY Life expectancy		200x10 ⁶ @ 10V and 10milliamps
AUXRLY Operate time, w/bounce		1 milliseconds maximum
AUXRLY Release time	1 milliseconds maximum	
AUXRLY Maximum voltage		200 VDC or AC peak
AUXRLY Maximum switched power		10VA
AUXRLY Maximum switching current		0.5 amps
AUXRLY Maximum carry current		1.0 amps
AUXRLY Insulation resistance	10 ¹⁰ ohms	
AUXRLY Off-state capacitance	5 picofarads typical	
Number of boards per testhead		10 maximum
AUXFET Software execution time		6 milliseconds
AUXFET Contact configuration	1 FORM A.(opto-isolated)	
AUXFET Contact resistance		0.6 ohms maximum
AUXFET Contact bounce		None
AUXFET Maximum voltage		+/- 200 VDC or AC peak
AUXFET Maximum switched current		5 amps
Off-state leakage		1 microamp at 100 VDC
AUXFET Leakage to tester ground		10 nanoamps at 100 VDC
AUXFET Off-state capacitance	1500 picofarads typical	

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.2 DIGITAL INPUT OUTPUT (DIO)

BASIC VERSION

The DVREN call specifies which of the 8-Bit drivers are going to be enabled.

NOTE: All driver outputs are tristated on power-up and reset.

run DVREN (Byte1, Data1, Byte2, Data2, . . . ByteN, DataN)

where:	ByteN	=	0	Byte #0, First DIO board
		=	1	Byte #1, First DIO board
		=	2	Byte #2, First DIO board
		=	3	Byte #3, First DIO board
		=	4	Byte #0, Second DIO board
				Etc. to 39.
	DataN	=	\$00	All bits disabled (Tristate)
		=	\$01	Bit #0 enabled only.
		=	\$02	Bit #1 enabled only.
		=	\$03	Bit #0 and Bit #1 enabled.
				Etc. to \$FF.
		=	\$FF	All bits enabled. (Default)

EXAMPLES:

run DVREN

run DVREN (0)

run DVREN (0, \$FF, 5, \$00)

Disables all bytes on all boards

Enables all bits of byte #0 on board #1

Enables all bits of byte #0 on board #1 and
disables all bits of byte #1 on board #2

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.2 DIGITAL INPUT OUTPUT (DIO)

BASIC VERSION

The DVRDAT call is used to set/reset the logic-state of the driver bits after they have been enabled.

NOTE: DIO outputs do not change until the call DVRCLK is run.

run DVRDAT (Byte1, Data1, Byte2, Data2, . . . ByteN, DataN)

where:	ByteN	=	0	Byte #0, First DIO board
		=	1	Byte #1, First DIO board
		=	2	Byte #2, First DIO board
		=	3	Byte #3, First DIO board
		=	4	Byte #0, Second DIO board
				Etc. to 39.
	DataN	=	\$00	All bits set to logic-0. (Default)
		=	\$01	Bit #0 set to logic-1 only.
		=	\$02	Bit #1 set to logic-1 only
		=	\$03	Bit #0 and Bit #1 set to logic-1.
				Etc. to \$FF.
		=	\$FF	All bits set to logic-1.

EXAMPLES:

run DVRDAT

Reset all bits on all boards to logic-0

run DVRDAT (0) Reset all bits of byte #0, board #1 to logic-0

run DVRDAT (0, \$FF, 5, \$0F)

Set all bits of byte #0, board #1 to logic-1 and bits 0 through 3 of byte #1 on board #2 to logic-1

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.2 DIGITAL INPUT OUTPUT (DIO)

BASIC VERSION

The DVRCLK call updates all of the output latches on all of the DIO boards in the system.

run DVRCLK (Mode, Ext Rise/Fall)

where:	Mode	=	0	Drivers clocked by CPU (Default).
		=	1	Drivers clocked by External Driver Clock.
	Ext Rise/Fall	=	1	Drivers are clocked by a rising edge of the External Driver Clock (Default).
		=	-1	Drivers are clocked by a falling edge of the External Driver Clock.

EXAMPLES:

run DVRCLK	CPU clocks (updates) all drivers
run DVRCLK (0)	CPU clocks (updates) all drivers
run DVRCLK (1)	Drivers are clocked by a rising edge of the External Clock
run DVRCLK (1, -1)	Drivers are clocked by a falling edge of the External Clock

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.2 DIGITAL INPUT OUTPUT (DIO)

BASIC VERSION

The RSTROBE call strobes all of the receivers on all DIO boards in the system.

run RSTROBE (Mode, Delay, Ext Rise/Fall)

where:	Mode	=	0	Receivers strobed by CPU (Default).
		=	1	Receivers strobed by External Clock.
		=	2	Receivers strobed by a Driver Clock when the Mode parameter for the DVRCLK call is 0.
		=	3	Receivers strobed automatically during the RCOMP call.
	Delay	=	0.00	to 2.00 microseconds in 0.01 microsecond increments (Default = 0).
	Ext Rise/Fall	=	1	Receivers are clocked by a rising edge of the External Receiver Clock (Default).
		=	-1	Receivers are clocked by a falling edge of the External Receiver Clock.

EXAMPLES:

run RSTROBE	CPU clocks all receivers (no delay)
run RSTROBE (0)	CPU clocks all receivers (no delay)
run RSTROBE (1)	Receive-Data is latched by a rising edge of the External Receiver Clock
run RSTROBE (1, 1, -1)	Receive-Data is latched 1 microsecond after a falling edge of the External Receiver Clock.
run RSTROBE (2, 2)	Receive-Data is latched 2 microseconds after the drivers have been clocked by the call: run DVRCLK(0).
run RSTROBE (3)	Receive-Data will be automatically latched by the call RCOMP

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.2 DIGITAL INPUT OUTPUT (DIO)

SPECIFICATIONS:

Number of DIO boards per system	10
MASTER Software execution time	5 milliseconds
DVREN Software execution time	5 milliseconds
DVRDAT Software execution time	5 milliseconds
DVRCLK Software execution time	5 milliseconds
RSTROBE Software execution time	7 milliseconds
RCOMP Software execution time	5 milliseconds
Driver high level output voltage	2.4 volts @ 250 microamps
Driver low level output voltage	0.34 volts @ 20 milliamps
Driver tristate output current	20 microamps
Driver tristate output resistance	approximately 250 Kohms

DRIVER OUTPUT NETWORKS

Series resistors socketed DIP pack (51 ohm standard)	
Pull-up resistors	socketed SIP pack (4.7K ohm standard)
Pull-down resistors	socketed SIP pack (4.7K ohm standard)
Driver voltage protection	20 volts

RECEIVER INPUT NETWORKS

Series resistors socketed DIP Pack (510 Ohm Standard)	
Receiver protection voltage, with standard resistor and diode clamps	50 volts
External driver clock input	standard TTL levels
External driver clock to output delay	30 +/-10 nanoseconds
Receiver input voltage range	standard TTL levels
Receiver hi level input current (5V)	100 microamps
Receiver low level input current (.4V)	-200 microamps
Receiver input capacitance	30 picofarads
External receiver strobe input	standard TTL levels
Programmable receiver strobe delay	0 to 2000 Nanoseconds in 10 Nanosecond steps
Programmable delay accuracy	10 +/-

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.3 DIGITAL TO ANALOG CONVERTORS (DA)

BASIC VERSION

The DA convertors provide DC voltages for the Test System. DA pairs 4 & 5, 6 & 7, 8 & 9, and 10 & 11 have external reference inputs tied to each pair and brought to the patchboard in the 2030.

Each channel is auto-calibrated via TMUX and against TDAC in the SelfTest Patchboard.

Since DA convertors provide +/- 16 volts @ 100milliamps and update simultaneously per board, they can be used as tracking power supplies in low current applications.

run DA (Chan0, Volts0, C1, V1, . . . Cn, Vn)

where: **Chan** = 0 to 11 (1st Analog Source Board)
 = 12 to 23 (2nd Analog Source Board)
 = 24 to 35 (3rd Analog Source Board)
 = 36 to 47 (4th Analog Source Board)

If multiple channels are programmed in one call, they update simultaneously. The DA call can set channels on only one board. Each board requires a separate call. The last channel in the parameter list is multiplexed to TMUX (See TMUX call). Negative numbers for Channels 4 thru 11 activate External References.

Volts = +16 to -16 in approximately 2 millivolt steps

EXAMPLES:

run DA
run DA (1, 1.234, 4, 2.468)
run DA (-4, 1.234)

Reset all DAs and mux DA Internal Reference to TMUX.
Set DA1 to 1.234 volts and set DA4 to 2.468 volts.
Set DA4 and 5 to external reference and
DA4 to [1.234(external reference)/10]volts

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.3 DIGITAL TO ANALOG CONVERTORS (DA)

SPECIFICATIONS:

Analog Source Boards per testhead		4
DA Software execution time		7 milliseconds
DA Resolution		14 bits, 2 millivolts
DA Output range		+/-16 volts @ 100 milliamps
DA Slew rate		0.5 volts / microseconds
DA Accuracy		V prog. +/- 10 millivolts
DA Output protection		+/- 30 volts
DA External reference	0 to +10 volts	
DA Output settling time		100 microseconds maximum (16 to -16V)

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.4 ARBITRARY WAVEFORM GENERATORS (ARB)

BASIC VERSION

The ARBfreq call sets the clock generator frequency for the selected ARB.

run ARBfreq (Chan, Freq, Delay)

where:	Chan	=	0	1st ARB, first board. (Default)
		=	1	2nd ARB, first board.
		=	2	1st ARB, second board.
		=	3	2nd ARB, second board. Etc. to four boards/system.
	Freq	=	0	External clock. (Default)
		=	10	to 10,000,000Hz, sets clock frequency to 3 significant digits. Output Freq=Clock freq / # of steps.
				<i>See ARBsin & ARBprog for step programming.</i>
	Delay	=	-1	Automatic delay, typically 300 milliseconds.(Default)
		=	0	to 65535 milliseconds. Typically set less than 300 millisecond to save test time if ARB waveform can settle before it's used.

EXAMPLES:

run ARBfreq Set ARB 0 to External Clock
run ARBfreq (1, 99999, 50) Set ARB 1 Frequency to 99.9 Kilohertz
and proceed after 50 millisecond delay.(2 digits of precision are dropped)

Generate a 1 KiloHertz sinewave:

run ARBfreq (0, 1000000) Set ARB 0 Clock to 1 Megahertz
run ARBsin (0, 5, 0, 1000) Set 10 volts peak to peak, zero offset, 1000 step sinewave on ARB 0
run ARB Turn ARB 0 on

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.4 ARBITRARY WAVEFORM GENERATORS (ARB)

BASIC VERSION

Each Arbitrary Waveform Generator (ARB) has two TTL - type logic (pulse) outputs, programmable in steps synchronous with the voltage steps of the analog ARB output. Pulse channels 0 and 1 are part of ARB0; pulse channels 2 and 3 are part of ARB1, etc.

Each time ARBpulse is called, the associated ARB analog and pulse outputs will stop and hold at the existing state until restarted by the ARB Functional Call or by an external start pulse, if so programmed .

Step-level pairs need only be programmed when the state changes. ARBpulse will fill in the intermediate steps.

To generate the initial waveform, at least 3 step-level pairs must be used. A recycle bit is at the last step-voltage pair.

To append, the first step-level pair must begin with the recycle bit.

To modify, one or more step-level pairs may be used but must end before or at the recycle bit.

To modify and append, 3 or more step-level pairs must be used to start before and transcend the recycle bit.

NOTE: There is only one recycle bit per ARB; therefore, its placement should be identical for related ARBprog, ARBsin, ARBpulse calls, i.e. calls are same length.

run ARBpulse (Chan, Step0, Level0, S1, L1, . . . Sn, Ln)

where:	Chan	=	0	1st ARB, first pulse, first board. (Default)
		=	1	1st ARB, second pulse, first board.
		=	2	2nd ARB, first pulse, first board.
		=	3	2nd ARB, second pulse, first board.
		=	4	1st ARB, first pulse, second board.
		=	5	1st ARB, second pulse, second board.
				Etc. to four boards per system
	Step	=	0	to 4095. (Default = 2048)
	Level	=	0	Set output to logic "0".
		=	1	Set output to logic "1".

EXAMPLES:

run ARBpulse (0, 0, 1, 1, 1, 2, 1)

Program TTL "one" level

run ARBpulse (0, 0, 1, 49, 1, 50, 0, 99, 0)

Program TTL squarewave of 100 steps

run ARBpulse (0, 74, 1) Modify previous call and place a one step "one" level at 75%

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.4 ARBITRARY WAVEFORM GENERATORS (ARB)

BASIC VERSION

ARBget copies the contents of the specified ARB memory into a Basic array variable for convenient disk storage. ARBput copies an array variable back into the ARB memory. This is a much faster way to program the ARBs than using ARBsin or ARBprog statements. The data copied includes the analog output sequence and the two associated digital ARBpulse outputs.

NOTE: Data from one ARB should not be put into another ARB, since voltages would not be calibrated.

run ARBget (Chan, Array)

run ARBput (Chan, Array)

where:	Chan	=	0	1st ARB, first board. (Default)
		=	1	2nd ARB, first board.
		=	2	1st ARB, second board.
		=	3	2nd ARB, second board
				Etc. to four boards per system.
	Array	=		Name of array. Must be declared in a DIM statement. Can be any data type. Array must allocate at least 2 bytes / ARB step, up to 8192. For speed, data transfer stops at the first recycle bit.

EXAMPLE PROGRAM:

```
DIM SINE (200):BYTE
run ARBsin (0, 5, 2, 100)
run ARBpulse (0, 0, 1, 49, 1, 50, 0, 99, 0)
run ARBget (0, SINE)
```

```
Dimension 200 byte array
Program 100 step (200 byte) sinewave.
Program 100 step (200 byte) square wave
Put ARB 0 memory contents in SINE array
```

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.4 ARBITRARY WAVEFORM GENERATORS (ARB)

SPECIFICATIONS:

Analog Source Boards per testhead	4
ARB Software execution time	5 milliseconds
ARBfreq Software execution time	6 milliseconds + delay
ARBprog Software execution time	13 milliseconds + 30 microseconds/step
ARBSin Software execution time	8 milliseconds + 225 microseconds/step
ARBpulse Software execution time	5 milliseconds + 13 microseconds/step
ARBget Software execution time	5 milliseconds
ARBput Software execution time	5 milliseconds
ARB Frequency synthesizer range	1-10.0 megahertz
ARB Frequency dividers	6 decades
ARB Frequency accuracy	programmed value +/- 0.1%
ARB Frequency settling time	300 milliseconds maximum
ARB Resolution 12 bits, 7.8 millivolts	
ARB Output range	+/-16 volts @ 50 milliamps
ARB Slew rate	40 volts / microseconds
ARB Accuracy	V programmed +/- 16 millivolts
ARB Output protection +/- 30V + short circuit	
ARB Memory length	4096 digital words
ARB Program clock range	0.01Hz to 10.0 megahertz
ARB Program burst control	1 to 255 patterns
ARB External reference	0 to +10 volts
ARB External start/stop Inputs TTL or equivalent	
ARB External start/stop delay	500 nanoseconds
ARB External start/stop width	100 nanosecond minimum
ARB Pulse outputs	TTL levels 3.4 volts @ 3 milliamp source 0.4 volts @ 12 milliamp sink 0.5 volts @ 24 milliamp sink
ARB Clock outputs	TTL levels 3.4 volts @ 3 milliamp source 0.4 volts @ 4 milliamp sink

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.5 SELFTEST MULTIPLEXER (TMUX)

BASIC VERSION

The Selftest Multiplexer is on the Instrumentation Amplifier board. It provides direct readback of system signals via Sig3, which is returned to the AMS via the Analog Motherboard. It is used in calibrate the DA's, ARB's and the AMS using TDAC as a Reference. TDAC is calibrated to a secondary standard during the DLI Certification Procedure.

TMUX is available to the USER and may be used to readback Instrumentation Amplifier outputs.

run TMUX (Chan)

where:	Chan	=	0	TDAC input via patchboard.
		=	1	Instrumentation amplifier #0.
		=	2	Instrumentation amplifier #1.
		=	3	Instrumentation amplifier #2.
		=	4	Instrumentation amplifier #3.
		=	5	Obsolete.
		=	6	Obsolete.
		=	7	Obsolete.
		=	8	Obsolete.
		=	9	Obsolete.
		=	10	Obsolete.
		=	11	1st Analog Source Board
		=	12	2nd Analog Source Board
		=	13	+19.5 TBUS power supply
		=	14	-19.5 TBUS power supply
		=	15	+15 TBUS power supply
		=	16	+5 TBUS power supply
		=	17	-15 TBUS power supply
		=	18	-5.2 TBUS power supply.
		=	19	Obsolete.
		=	20	Obsolete.
		=	21	+15PB Patchboard power supply
		=	22	+5PB Patchboard power supply
		=	23	-15PB Patchboard power supply
		=	24	Analog ground
		=	25	Obsolete.
		=	26	Obsolete.
		=	27	1st Analog Source Board.
		=	28	2nd Analog Source Board.
		=	29	3rd Analog Source Board.
		=	30	4th Analog Source Board.
		=	31	TDAC via internal P/S mux.

EXAMPLES:

run TMUX (1)

Multiplexes Instrumentation Amp #0 to Sig3 on the AMS

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.5 SELFTEST MULTIPLEXER (TMUX)

SPECIFICATIONS:

Number of boards per testhead

1

TMUX Software execution time

5 milliseconds

TMUX Gain bandwidth product

200 kil

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.6 PROGRAMMABLE POWER SUPPLIES

FEATURES: Voltage programming.
Current programming.
Voltage measurement.
Current measurement.
Dedicated micro-controller.
Serial data communication to host computer via an opto-isolated ring network (9600 baud).

Programmable relay disconnect from unit under test. Capable of continuous monitoring of output voltage for limits.
Capable of continuous monitoring of output current for limits.
Internal, power-up SelfTest.
Internal "fault" monitoring.
 Micro-controller failure.
 A/D failure.
 Internal power supply failure.
 Configuration card removal.
 EEPROM failure.

External fault monitoring
 Communication failure from host.
 Voltage and/or current out of programmed limits.
 Able to receive "fault" status from other units.
 Able to detect cable disconnection.

Automatic shutdown and disconnect upon fault detection. Allows "sense" terminals of power supply to be used.
Software calibrated.
Calibration constants stored in units EEPROM.
Programming through low level commands or functional calls.

The programmable power supply system is composed of two major components, the power supply and the controller. The controller can be made to work with any programmable power supply as long as the programming cable and configuration card are available for that supply. Another way of looking at it is that the controller is universal while the power supply, configuration card and programming cable are a matched set.

Commands are given to the controller via an opto-isolated current loop.

The output of the power supply goes to the controller, where it is switched, by a mercury wetted relay, to the test-head. The cable that is between the controller and the testhead, besides providing a path for the output, has a fault loop, which if broken, will cause the controller to shut down the power supply and report an error to the computer (the next time the computer tries to give it a command). The fault loop is broken by the controller itself any time the controller detects an error. This signals to the other controllers in the loop that they too should shut down. Programming of the power supplies is accomplished with functional calls POWER and PCHEK.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.6 PROGRAMMABLE POWER SUPPLIES

BASIC VERSION

Most of the parameters are optional. The exceptions are if the over voltage limit is given then the under voltage limit must be given; and if the over current limit is given then the under current limit must be given.

The U.U.T. power supply will connect (if disconnected), to the patchboard on any programming statement with two or more parameters. This assumes that the parameters are in legal range.

run POWER (Unit, Voltage, Amperage, Timeout, UVL, LVL, UCL, LCL)

Where: **Unit** = -1 to 63

The Unit parameter is used to select which power supply is to be programmed. Note -1 corresponds to the patchboard power supplies. (See examples).

Voltage = The value of the Voltage parameter is what the output voltage of the U.U.T. power supply will be set to assuming that:

- 1.) The power supply is not in current limit.
- 2.) The value does not exceed the monitor limits set.
- 3.) The power supply is allowed enough time to settle within the monitor limits set.

In the case of the patchboard supplies the Voltage parameter is used to connect and disconnect the supply rather than program it. The two values that do this are 0 for disconnecting and 1 for connecting.

Amperage = The value of the amperage parameter is what the power supply current limit will be set to.

The patchboard power supplies are not voltage or current programmable. Values placed in the Amperage parameter will be ignored in the case of the Unit parameter equal to -1.

Timeout = 0 to 65,535

The value of the Timeout parameter is used to determine how long the functional call will wait before returning to the calling program. Each count of the value in the parameter will cause the program to wait 1 millisecond before returning. (0 - 65,535 ms)

UVL, LVL = The value of the UVL/LVL parameters are used to set the sides of a window that the U.U.T. power supply controller will monitor.

UCL, LCL = The value of the UCL/LCL parameters are used to set the sides of a window that the U.U.T. power supply controller will monitor.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.6 PROGRAMMABLE POWER SUPPLIES

EXAMPLES:

run POWER (0,10,1)

run POWER

run POWER (Unit)

run POWER (-1,0)

run POWER (-1,1)

Programs U.U.T. power supply #0 for 10 volt output, one amp current limit

This statement disconnects all U.U.T. power supplies in the tester

The supplies are programmed for 0 volts, 0 amps before disconnection

This statement disconnects only that U.U.T. power supply with
the logical unit address specified by "Unit"

Disconnects patchboard supply

Connects patchboard supply

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.6 PROGRAMMABLE POWER SUPPLIES

BASIC VERSION

run Pchek (Unit, Voltage, Amperage)

Where: **Unit** = 0 to 63

The Unit parameter is used to select which power supply is to be measured.

Voltage =

Voltage is a variable in which the value of the output voltage of the unit selected is returned.

Amperage =

Amperage is a variable in which the value of the output current of the unit selected is returned.

EXAMPLES:

run Pchek (0, v, i)

Voltage measured will return in variable v,
current measured will return in variable i.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.6 PROGRAMMABLE POWER SUPPLIES

SPECIFICATIONS:

Number of controllers per testhead	5
POWER Software execution time	216 milliseconds+programmed delay
Pchek Software execution time	195 milliseconds
POWER Current output accuracy	0.3% of full scale
Pchek Voltage measurement accuracy	0.3% of full scale
Pchek Current measurement accuracy	0.3% of full scale
Time to disconnect after fault	100 milliseconds
Time to programmed disconnect	100 milliseconds programmed delay

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.7 INSTRUMENTATION AMPLIFIER (INST)

BASIC VERSION

There are 4 Differential Instrumentation Amplifiers in the system. The differential inputs are brought to the patchboard. The single ended outputs, with an associated ground, are also brought to the patchboard. Each amplifier has programmable gain and programmable filters and can be readback with the TMUX call.

These amplifiers have +/-200 volt Common Mode Voltage Range, and as such, are useful to read voltages above ground.

run INST (Chan, Gain, Filter)

where:	Chan	=	0	to 3
	Gain	=	0	1 (Default)
		=	1	10
		=	2	100
	Filter	=	0	No filter (Default)
		=	1	16000 Hertz, -3db (-20db/decade)
		=	2	1600 Hertz, -3db (-20db/decade)
		=	3	160 Hertz, -3db (-20db/decade)

EXAMPLES:

run INST
run INST (2, 1, 3)

All channels set to Gain = 1 no filter.
Amplifier 2 set to Gain = 10 with 160 Hz filter.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.7 INSTRUMENTATION AMPLIFIER (INST)

SPECIFICATIONS:

Number of boards per testhead		1
INST Software execution time	5 milliseconds	
INST Common mode rejection ratio		86db minimum
INST Common mode voltage range		+/- 200 volts
INST Gain inaccuracy		0.05 %
INST Offset drift vs temperature, gain = 1		30 microvolts / degrees C
INST Offset drift vs temperature, gain = 10		92 microvolts / degrees C
INST Offset drift vs temperature, gain = 100		850 microvolts / degrees C
INST Full power bandwidth		30 kilohertz
INST Output voltage range		+/- 10 volts
INST Output current range @ +/- 10 volts		5 milliamps
Input impedance		400 K ohms per leg

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.8 RELAY MUX AND AMPLITUDE MEASUREMENT SYSTEM (AMS)

BASIC VERSION

The AMS takes a voltage reading on Sig1 or Sig2 provided by the MUX call or Sig3 by the TMUX call.

run AMS (Vret, Sig, Mode, Timeout, Autozero, InterruptMode, Readings)

where: **Vret** = Returned reading in volts.

An array of REAL numbers when using the interrupt modes or a single reading for non-interrupt driven modes.

Sig	=	0	System measurement ground.
	=	1	A multiplexed Patchboard signal (MUX Call) for voltage measurement and is displayed on upper scope trace. (Default)
	=	2	A multiplexed Patchboard signal (Mux Call) for voltage measurement and is displayed on lower scope trace.
	=	3	An internal multiplexer signal Sig3 used for SelfTest and Calibration of the system. See TMUX call for signal selection.
	=	4	Differential (Sig1-Sig2).
Mode	=	0	Straight to A/D, no signal processing.
	=	1	Low-pass 1 kilohertz filter. (Default)
	=	2	RMS, AC-coupled, 500 millisecond integration.
	=	3	High speed sampling mode for voltage measurement on waveforms.
	=	4	RMS, DC-coupled, 500 millisecond integration.
	=	5	TrigA DAC (SelfTest only)
	=	6	Trig1 DAC (SelfTest only)
	=	7	Trig2 DAC (SelfTest only)
Timeout	=	-1	Return previously triggered reading. (Past)

If in the RMS modes, the negative value selected here sets the integration time in milliseconds. (0 to 60000 maximum)

	=	0	Take instantaneous reading. (Default)
	=	>0	Wait for trigger for timeout in milliseconds. (Future)
Autozero	=	-4	Return reading in raw A/D counts. (SelfTest)
	=	-1	Raw readings, no gain or offset correction.
	=	0	Normal gain and offset corrections (Default).
	=	1	Autozero reading, using MUX reference channel.

MUX channel 0 is the autozero channel for channels 1 thru 15. A corrected reading is taken on channel 0 and subtracted from the selected channel of 1 thru 15. (Channel 0 normally tied to product ground) Channel 16 is autozero Channel for 17 - 31. Channel 32 is autozero Channel for 33 - 47. Etc.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.8 RELAY MUX AND AMPLITUDE MEASUREMENT SYSTEM (AMS)

BASIC VERSION - CONTINUED

InterruptMode	=	-2	Arm and pass through for interrupt mode 2
	=	-1	Arm and pass through for interrupt mode 1
	=	0	Non-triggered burst of samples.
	=	1	Wait for trigger (from MARK), then burst samples.
	=	2	Every reading is triggered (by MARK).
Readings	=		Number of readings to take in the interrupt mode. (Default = size of Vret [] array).

NOTES:

- 1). When using the RMS conversion modes, the integration time may be adjusted from 1ms to 60s. When the time parameter is zero or greater, the integration period is 500ms. When the time parameter is negative, it sets the integration time.
- 2). For all other signal processing options, a negative parameter means return the previously triggered reading(s) from the A/D.
- 3). If the time parameter is positive and the interrupt mode has been selected, the value of the time parameter is then taken as CPU "ticks" worth 10 milliseconds each (a value of zero means wait **FOREVER!**).

Example: Timeout = 10 in the interrupt mode means wait 100 milliseconds. See the OS-9 Operating System Manual for more information on computer ticks and sleep time.

- 4). To retrieve arm & pass through readings taken in the interrupt driven mode, this routine must be run again with a negative time parameter. The previous value for the timeout is used when retrieving values from the interrupt handler.

You must also either: (a) OPEN a path to /AMSint in your program, OR (b) \$iniz /AMSint.

See OS-9 Operating System Manual for more information on INIZ or the Language Manuals for more information on OPEN.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.8 RELAY MUX AMPLITUDE MEASUREMENT SYSTEM (AMS)

BASIC VERSION - CONTINUED

EXAMPLES:

run AMS (V, 1, 1)

Read Sig1 with filter and return as V
(Slowest but Most accurate DC reading)

run AMS (V, 2, 3, 12)

Read Sig2 with High Speed Sample/Hold
Triggered from MDE MARK. Return unconditionally in 12 milliseconds
Software must handle the Not Triggered Error

DIM Rdgs(10):REAL

run AMS(Rdgs,1,3,100,1,10)

Wait for a trigger from MARK (for the first reading) and then
take 9 additional non-triggered readings as fast
as the hardware and software is capable.

The next example combines both the interrupt mode readings and the Arm and Pass through feature:

DIM Rdgs(100):REAL

OPEN #pth,"/AMSint":READ

run AMS(Rdgs,1,3,1.0,0,-1)

open a path & initialize "/AMSint"
arm the AMS and interrupt handler
user section of program (causes trigger)

run AMS(Rdgs,1,3,-1.0,0,1)

CLOSE #pth

retrieve 100 readings from interrupt handler
close path to interrupt handler

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.8 RELAY MUX AND AMPLITUDE MEASUREMENT SYSTEM (AMS)

SPECIFICATIONS:

Relay Mux boards per testhead	4
AMS boards per testhead	1
MUX Software execution time	14 milliseconds
AMS Software execution time	9 milliseconds. (Mode 0 - straight)
AMS Software execution time	19 milliseconds. (Mode 1 - filter)
AMS Software execution time	9 milliseconds. (Mode 3 - HSS/H)
AMS Software execution time	1000 milliseconds. (Modes 2 & 4 - RMS)
AMS Software execution time	Add 2 milliseconds for Autozero
Input voltage ranges	200, 20, 2, 0.2 volts
Input resistance	10 megohms.(200 & 20 volt ranges)
Input resistance	
Input capacitance	5 picofarads maximum
Crosstalk	-60 db at 10 megahertz
Frequency response (SE)	DC to -3db at 2 megahertz.(0.2 V range)
Frequency response (SE)	DC to -3db at 10 megahertz.(other ranges)
Frequency response (Diff)	DC to -3db at 5 kilohertz
	NOTE !! Any differential is limited to 5 kilohertz
DC accuracy (SE & Diff)	0.1 % of full scale.(0.2 V range ,Mode 1)
DC accuracy (SE & Diff)	.05 % of full scale.(2.0 V range, Mode 1)
DC accuracy (SE & Diff)	.02 % of full scale.(20 V range, Mode 1)
DC accuracy (SE & Diff)	.02 % of full scale.(200 V range, Mode 1)
DC accuracy (Mode 0)	add +/- 3 millivolts to all ranges
RMS accuracy (SE only)	0.1 % of full scale.(DC to 25 Kilohertz)
	1.0 % of full scale.(to 800 kilohertz)
	5 % of full scale.(to 2 megahertz)
Mode 3 accuracy (SE only)	0.1 % of full scale to 100 kilohertz
	1 % of full scale.(to 2 megahertz)
	5 % of full scale.(to 5 megahertz)
	10 % of full scale.(to 8megahertz)
Definitions:	SE = Single ended
	Diff = Differential

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.9 TIME AND FREQUENCY MEASUREMENT SYSTEM (TMS)

BASIC VERSION

The Time and Frequency Measurement System receives SigA, Sig1 and Sig2 signals from the RMUX and AMS boards by way of the MUX call. These signals are squared up with level and slope programmable Crossover Detectors (TRIG CALL) to form TrigA, Trig1, and Trig2 and are gated with MDE signals to form measurement controlled Gate signals for the TIME Call. TrigA, Trig1, and Trig2 are used directly as inputs for the FREQ and TCOUNT call.

FEATURES:

- 100 Megahertz base count frequency.
- 10×10^{-7} long term stability.
- Time, frequency, ratio, and count capability.
- Phase, rise/fall time, pulse width, duty cycle etc.
- 3 Crossover Detectors (triggers) with programmable:
 - Level (10 bits).
 - Slope (rising or falling).
 - Filters - (4)
- External Event counting.
- External Clock operation.
- Certification output provided.

NOTE: The RMUX, AMS, TMS, and MDE form a highly integrated Measurement System which provide speed and measurement capability far beyond GPIB Rack-and-Stack equipment. This integration requires understanding of the Measurement System, as a whole, to capture its full advantages.

If continuous counting during the test is required, the MUX call selecting the Signal must not be tampered with.

run TCOUNT (Cret, Signal)

where:	Cret	=	Return variable
	Signal=	0	TrigA (MUX & TRIG calls required)(Default)
		= 1	Trig1 (MUX & TRIG calls required)
		= 2	Trig2 (MUX & TRIG calls required)
		= 3	External Event Input (CX5 on Digital Motherboard)

EXAMPLES:

run TCOUNT (C, 1)

run TCOUNTMux to TrigA, reset, arm and leave
Read Mux to Sig1, reset, arm and leave

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.9 TIME AND FREQUENCY MEASUREMENT SYSTEM (TMS)

BASIC VERSION

The TIME Call generally requires setup of the MDE calls. The MDE calls allow a visual representation of the measurement on the MDE scope, with suitable event and period marking for measurement aid.

run TIME (Tret, Gate, Average, Timeout, Mode)

where:	Tret	=		Return variable in seconds. (10 nanoseconds resolution)
	Gate	=	0	TrigA to Trig1 (Default).
		=	1	TrigA to Trig A - Bypass MDE. (For Period >> 0.5 Seconds)
		=	2	Start of trace2 to Trig2.
		=	3	TrigA to MARK on trace1.
		=	4	Trig1 to MARK on trace2.
		=	5	TrigA to end of delay1.
		=	6	Trig1 to end of delay2.
		=	7	Sweep time 1.(SelfTest and Calibration)
		=	8	Sweep time 2.(SelfTest and Calibration)
		=	9	Mark width 1.(SelfTest)
		=	10	Mark width 2.(SelfTest)
		=	11	DMERST pulse width.(SelfTest)
		=	12	Crossover A high pulse width.(SelfTest)
		=	13	Crossover 1 high pulse width.(SelfTest)
		=	14	Crossover 2 high pulse width.(SelfTest)
		=	15	TrigA to Trig1 - Bypass MDE. (For Time 0.5 Seconds)
	Average	=	0	Take one reading. (Default)
		=	2	through 65536 = number of readings to average. (Average not used in Modes 0 and 1)
	Timeout	=	0	Wait forever.(5 seconds in 2025)(Default)
		=		Time limit in seconds.(Up to 60 seconds)
	Mode	=	0	Wait and read a previously armed TIME call.
		=	1	Clear and arm counter and return. This implies a later call to read the result. Use for 1 shot TIME measurements.
		=	2	Clear and arm counter, wait for reading or timeout, and return. (Default).

NOTE: If the Gate does not begin within the time limit, a zero will be returned. If the Gate begins but does not end within the time limit, the time limit will be returned.

EXAMPLES:

run TIME (Tret, 0, 0, 1)	Take immediate time reading on Gate 0, with 1 second timeout
run TIME (Tret, 2, 0, 0, 1)	Arm TIME Call to Gate 2 and pass through, no is reading
run TIME (Tret, 2, 0, 1, 0)	Retrieve a reading from a previously armed TIME Call

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.9 TIME AND FREQUENCY MEASUREMENT SYSTEM (TMS)

SPECIFICATIONS:

TMS Boards per testhead		1
TCOUNT Software execution time		5 milliseconds
TIME Software execution time	10 milliseconds + gate time	
FREQ Software execution time	6 milliseconds + timebase	
RATIO Software execution time		6 milliseconds + gate time
TCOUNT Register range		16 bits.(65536 counts)
TCOUNT Counting frequency		10 megahertz
TCOUNT Edge sensitivity		rising edge
TCOUNT External event frequency		10 megahertz maximum
TMS TCXO Internal frequency		100 megahertz
TMS TCXO Stability		5×10^{-7} (15-45 degrees C)
TMS Timebase resolution		10 nanoseconds
TMS Timebase range		10 microseconds to 100 seconds
TMS time measurement accuracy		+/- 10 nanoseconds
TMS External clock frequency		100 megahertz maximum (Requires board jumper)

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.10 MEASUREMENT DISPLAY ELECTRONICS SYSTEM (MDE)

BASIC VERSION

The Measurement Display Electronics (MDE) is integrated into the measurement system to provide waveform measurement capability. The MDE provides a "picture" of the waveforms to be measured, and allows the Test Engineer to position measurement marks, and delays to his satisfaction.

The MDE is an alternate trace oscilloscope which is triggered by TrigA and displays Sig1 on trace1 and Sig2 on trace2. The "Z" axis is modulated with intensified Trig1 and Trig2 marks (TRIG Call), and a voltage measurement mark (MARK Call). The "Z" axis is also modulated from the start of each trace with trigger inhibiting, intensified analog delay bands: Delay1 on trace1 inhibits Trig1 and delay2 on trace2 inhibits Trig2. Both are positioned by the DELAY Call. The sweep on each trace, is set by the SWEEP Call in seconds for total sweep time. The vertical amplitude is set, on each trace, with the VERT Call. Finally, the T2DEL Call sets the trigger mode for trace2.

run TRIG (Signal, Level, Slope, Filter)

where: **Signal** = 0SigA (Creating TrigA)(Default).
 = 1 Sig1 (Creating Trig1, displayed on trace1).
 = 2 Sig2 (Creating Trig2, displayed on trace2).

Level = 2.000 to -2.000 in steps of .001. The actual trigger voltage is related to the MUX Call Range parameter. (Default = 0.000) i.e.

LEVEL	MUX RANGE	ACTUAL TRIGGER VOLTAGE
1.000	200	100 volts
1.000	20	10 volts
2.000	2	2 volts

NOTE: 2001 through 2025 Systems have a high Range of 100, and Level of 1 to -1.

Slope = 1 Trigger on rising edge of Signal.(Default)
 -1 Trigger on falling edge of Signal.

Filter = 0 5 Megahertz, single break. (Default)
 1 3 Megahertz, single break.
 2 370 Kilohertz, single break.
 3 38 Kilohertz, single break.

EXAMPLES:

run TRIG

MUX Call Range = 20:

Trigger on rising edge of SigA at 0 volts with 5 Megahertz filter.

run TRIG (2, 1.5, -1, 3)

MUX Call Range = 2:

Trigger on falling edge of Sig2 at 1.5 volts with 38 Kilohertz filter.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.10 MEASUREMENT DISPLAY ELECTRONICS SYSTEM (MDE)

BASIC VERSION

The SWEEP CALL sets the Sweep Time on trace1 and trace2.

run SWEEP (Signal, Time)

where:	Signal	=	1	For trace1 (upper) (Default)
		=	2	For trace2 (lower)
	Time	=	.5	Seconds full sweep
		=	.2	Seconds full sweep.
		=	.1	Seconds full sweep.
		=	.05	Seconds full sweep.
		=	.02	Seconds full sweep.
		=	.01	Seconds full sweep.
		=	.005	Seconds full sweep. (Default)
		=	.002	Seconds full sweep.
		=	.001	Seconds full sweep.
		=	.0005	Seconds full sweep.
		=	.0002	Seconds full sweep.
		=	.0001	Seconds full sweep.
		=	.00005	Seconds full sweep.
		=	.00002	Seconds full sweep.
		=	.00001	Seconds full sweep.
		=	.000005	Seconds full sweep.
		=	.000002	Seconds full sweep.
		=	.000001	Seconds full sweep.
		=	.0000005	Seconds full sweep. (500 nanoseconds)

EXAMPLES:

run SWEEP

run SWEEP (2, .000005)

Set trace1 sweep to 5 milliseconds
Set trace2 sweep to 5 microseconds

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.10 MEASUREMENT DISPLAY ELECTRONICS SYSTEM (MDE)

BASIC VERSION

The T2DEL Call programs the start of sweep2 on trace2. Mode 2 allows trace2 to start after a digital count of events on the selected Source. This count delay is generally more stable for long delays than the Mode 1 analog delay.

run T2DEL (Mode, Count, Source)

where:	Mode	=	0	Start sweep2 at end of delay on trace1.
		=	1	Start sweep2 on trig1.(Default)
		=	2	Start sweep2 after countdown of Source. (Count = 65535 maximum.)
		=	3	Disable sweep2. (single-trace oscilloscope)
	Count	=		Number of counts for Mode 2. (Default = 1000)
	Source	=	0	TrigA. (TRIG Call Required)(Default)
		=	1	Trig1. (TRIG Call Required)
		=	2	Trig2. (TRIG Call Required)

Source used in Mode 2 only. Only counts occurring during sweep1 time will be honored; if the count programmed is too great to meet this requirement, trace2 will not occur. TCOUNT cannot be used if Mode 2 is used. The Event counter is used in both calls.

EXAMPLES:

run T2DEL
run T2DEL (0)
run T2DEL (3)
run T2DEL (2, 10, 1)

Start trace2 at Trig1
Start trace2 at end of Delay1
Disable trace2
Start trace2 at the tenth Trig1 presented on trace1

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.10 MEASUREMENT DISPLAY ELECTRONICS SYSTEM (MDE)

BASIC VERSION

The VERT Call sets the size of Sig1, displayed on trace1, and Sig2, displayed on trace2.

run VERT(Signal, Range, AC)

where: **Signal** = 1 Set Range of Sig1 on trace1.
 = 2 Set Range of Sig2 on trace2.

		- Relay Multiplexer Voltage Range -					
		200	20	2.0	0.2	volts	
Range	= 1	10	1	0.1	0.01	volts/division	
	= 2	20	2	0.2	0.02	volts/division	
	= 5	50	5	0.5	0.05	volts/division	
	= 10	100	10	1.0	0.10	volts/division (Default)	

The Range parameter is related to the MUX Range parameter as shown above. Proper oscilloscope setup is essential, see Oscilloscope Setup Section.

AC = 0 DC coupled amplifier. (Default)
 = 1 AC coupled amplifier.

EXAMPLES:

run VERT

run VERT (1)

run VERT (2, 2, 1)

run VERT (2, 2, 1)

Both traces set to 10 Volts/division, DC
(If MUX Call is on 20 volt range)
Trace1 set to 10 Volts/division, DC
(If MUX Call is on 20 volt range)
Trace2 set to 2 Volts/division, AC
(If MUX Call is on 20 volt range)
Trace2 set to 20 Volts/division, AC
(If MUX Call is on 200 volt range)

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.10 MEASUREMENT DISPLAY ELECTRONICS SYSTEM (MDE)

SPECIFICATIONS:

MDE boards per system		1
TRIG Software execution time	7 milliseconds	
SWEEP Software execution time		28 milliseconds
DELAY Software execution time		7 milliseconds
MARK Software execution time		7 milliseconds
T2DEL Software execution time		5 milliseconds
VERT Software execution time	5 milliseconds	
TRIG Level accuracy		programmed value +/-5%
TRIG Filter		5 megahertz, single break
		3 megahertz, single break
		370 kilohertz, single break
		38 kilohertz, single break
SWEEP accuracy		programmed value +/-5%
DELAY Position accuracy		programmed value +/-5%
MARK Position accuracy		programmed value +/-5%
T2DEL Count accuracy		programmed value +/-1 count
VERT Amplitude accuracy		programmed value +/-5%

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.12 PATCHBOARD IDENTIFICATION (PBID)

BASIC VERSION

It is possible to assign a patchboard identification code to any patchboard assembly. The code is hard-wired into each patchboard assembly. Identification codes fall in the range of 0-255 (0 - FF hex) and are coded by grounding pins that correspond to the bits set in the desired identification code. An ID GROUND pin is provided adjacent to the identification pins for this purpose. The PBID Function Call is used to read a patchboard's identification code when installed on the Testhead.

A patchboard's identification code is read through the 2030's Instrumentation Amplifier Board (ISA). Patchboard ID pins therefore are found in slot # 1 where the ISA board is installed. The following are the ID pin assignments on the patchboard interface. These locations are shown graphically on a Patchboard Map (see manual section on Patchboard Software for information on printing out Patchboard Maps).

Patchboard ID bits are inverted. Internal pullup resistors produce zeroes and must be wired to ground on the patchboard to produce ones.

PIN NAME	COORDINATE	INDEX #	COMMENT
ID01	C-26	119	Bit 0
ID02	D-26	120	Bit 1
ID04	C-27	121	Bit 2
ID08	D-27	122	Bit 3
ID10	C-28	123	Bit 4
ID20	D-28	124	Bit 5
ID40	C-29	125	Bit 6
ID80	D-29	126	Bit 7
IDGD	C-30	127	GROUND

run PBID (IDcode)

where: **IDcode=** Return variable (0-255, 0-FF hex).

EXAMPLES:

run PBID (IDcode)

Returns in the variable IDcode the identification code of the patchboard installed on the Testhead.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.12 SOFTWARE DELAY (IDLE)

BASIC VERSION

The IDLE Function Call is used whenever a software delay is required within a test program.

run IDLE (Time)

where: **Time** = Time in milliseconds to delay. Minimum delay is 2 milliseconds. Maximum delay is 65535 milliseconds. Tolerance is +/- 2 milliseconds.

NOTE: The delay caused by IDLE is not predictable when multiple processes are running (i.e. CPU is multitasking). Analog test programs should not be executed with multiple processes running.

EXAMPLES:

run IDLE (0)
run IDLE (50)

Software delay of 2 milliseconds (2 - 4 milliseconds)
Software delay of 50 milliseconds (48 - 52 milliseconds)

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.12 PRINT ANALOG TEST FAILURE (PTA)

BASIC VERSION

The functional call PTA is used to obtain standardized test failure output messages for analog type tests and to easily control output redirection. Its main use is with Digalog calibration and SelfTest programs and their operation under the Test Executive, but it can be applied by users in their application programs as well. See also Print Digital Test Failure (PTD).

run PTA (Output, Path, Errtn, "DESCR", Dnum, Tnum, Rdg, "Units", Hi, Lo)

where:	Output =	0Fail data to terminal
	=	1 All data to terminal
	=	2 Fail data to path
	=	3 All data to path
	=	4 No data output (Errtn still valid) BYTE, INTEGER, REAL
	Path	= n Where n is the path number to send the output information. The path may be to any I/O device (i.e. /p or a disk file) but must be OPENed or CREATEd prior to running PTA. Path is ignored if Output parameter is 0 or 1. BYTE, INTEGER
	Errtn	= Return parameter. PTA will return 0 in Errtn if the Rdg parameter falls within the limits Hi and Lo . PTA will return 1 in Errtn if the Rdg exceeds the limits. INTEGER ONLY
	"DESCR"	= A string expression/constant that is a description of the test or what is being tested. A text comment field. STRING
	Dnum	= Device number. The number of the device being tested. BYTE, INTEGER, REAL
	Tnum	= Test number. The number of the test being made. BYTE, INTEGER, REAL
	Rdg	= Reading. The measured/calculated value. The number to be compared to limits. REAL ONLY
	"Units" =	A string expression/constant with the units to be output with Rdg . STRING
	Hi	= The high limit for comparison with Rdg . If Rdg >>Hi, Errtn will be 1. REAL ONLY
	Lo	= The low limit for comparison with Rdg . If Rdg <<Lo, Errtn will be 1. REAL ONLY

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.12 PRINT ANALOG TEST FAILURE (PTA)

BASIC VERSION - CONTINUED

EXAMPLES:

run PTA (1, 0, rtn, "ChAfrq", 5, 101, 1003., "Hz", 1005., 995.) causes output to the terminal of...

ChAfrq 5 Test 101 RDG=1.003E3 Hz HI=1.005E3 LO=0.995E3 T> PASS
(Errtn equal to 0)

run PTA (0, 0, rtn, "ChAfrq", 5, 101, 1003., "Hz", 1005., 995.) causes no output
(output was 0 (FAIL data only) and the test passed)

To output data to the printer or disk file from a BASIC application program, you must open a path to the desired device before calling PTA. Example:

```
PROCEDURE PTA_test
DIM p_path:BYTE
DIM rtn,Test_Errs:INTEGER
Test_Errs=0
OPEN #p_path,"/p":WRITE
RUN PTA(3,p_path,rtn,"ChAfrq",5,101,1111.,"Hz",1005.,995.)
IF rtn>>0 THEN Test_Errs=Test_Errs+1 \ ENDIF
CLOSE #p_path
END
```

This program will cause output to the printer of

ChAfrq 5 Test 101 RDG=1.111E3 Hz HI=1.005E3 LO=0.995E3 FAIL

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.12 PRINT DIGITAL TEST FAILURE (PTD)

BASIC VERSION

The functional call PTD is used to obtain standardized test failure output messages for digital type tests and to easily control output redirection. Its main use is with Digalog calibration and selftest programs and their operation under the Test Executive, but it can be applied by users in their application programs as well. See also Print Analog Test Failure (PTA).

run PTD (Output, Path, Errtn, "DESCR", Dnum, Tnum, Addr, Expect, Actual)

where:	Output =	0	Fail data to terminal
		= 1	All data to terminal
		= 2	Fail data to path
		= 3	All data to path
		= 4	No data output (Errtn still valid) BYTE, INTEGER, REAL
	Path	= n	Where n is the path number to send the output information. The path may be to any I/O device (i.e. /p or a disk file) but must be OPENed or CREATEd prior to running PTD. Path is ignored if Output parameter is 0 or 1. BYTE, INTEGER
	Errtn	=	Return parameter. PTD will return 0 in Errtn if expect equals actual . PTD will return 1 in Errtn if expect does not equal actual . INTEGER ONLY
	"DESCR"	=	A string expression/constant that is a description of the test or what is being tested. A text comment field. STRING
	Dnum	=	Device number. The number of the device being tested. BYTE, INTEGER, REAL
	Tnum	=	Test number. The number of the test being made. BYTE, INTEGER, REAL
	Addr	=	Address. The address being tested. BYTE, INTEGER
	Expect =		Expected data. This is the expected data value that actual will be compared to. If the two are unequal, Errtn is 1. If expect is equal to actual , Errtn is 0. BYTE, INTEGER
	Actual	=	Actual data. This is the value that the test obtained to be compared to expect . If the two are unequal, Errtn is 1. If expect is equal to actual , Errtn is 0. BYTE, INTEGER.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.12 PRINT DIGITAL TEST FAILURE (PTD)

BASIC VERSION - CONTINUED

EXAMPLES:

run PTD (1, 0, rtn, "MEMa", 1, 100, \$fbe900, \$ff, \$ff)

causes output to the terminal of

```
MEMa1      Test 100      ADDR=$00FBE900      EXP=$0000FF      ACT=$0000FF      PASS
(Errtn equal to 0)
```

run PTD (0, 0, rtn, "MEMa", 1, 100, \$fbe900, \$ff, \$ff)

(causes no output since Output parameter was 0 (FAIL data only) and the test passed.)

To output data to the printer or disk file from a BASIC application program, you must open a path to the desired device before calling PTD. Example:

```
PROCEDURE PTD_test
DIM p_path:BYTE
DIM rtn,Test_Errs:INTEGER
Test_Errs=0
OPEN #p_path,"/p":WRITE
RUN PTD(3,p_path,rtn,"MEMa",1,100,$fbe900,$ff,$ff)
IF rtn>>0 THEN Test_Errs=Test_Errs+1 \ ENDIF
CLOSE #p_path
END
```

This program will cause output to the printer of

```
MEMa 1      Test 100      ADDR=$00FBE900      EXP=$0000FF      ACT=$0000FF      PASS
```

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.12 SOFTWARE TESTHEAD RESET (TCLEAR)

BASIC VERSION

The TCLEAR Functional Call is used to reset the Testhead to its power-up state. Including:

- 1) Faults the power supply system. This causes all UUT product power supplies to shut down (i.e. their voltage and current outputs are programmed to 0 and the relays open disconnecting them from the patchboard interface). Also, the patchboard power supplies will turn off and disconnect.
- 2) Selects the lowest Relay MUX channel in each group mux and sets all to the 200 volt range.
- 3) Resets all D/A's and ARB's to zero volts.
- 4) Tristates DIO drivers.
- 5) Opens all AuxRLY/AuxFET channels.
- 6) Resets the MDE, TMS and AMS.

run TCLEAR

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.12 STORE DATA IN MODULE (PUTDATMOD)

BASIC VERSION

The PutDatMod Functional Call is provided to allow the easy creation of and/or writing to OS9 data memory modules. The call is run specifying a module name and a data structure that is to be stored. If the module does not already exist it is created and the data is written to it. If the module already exists, the data is stored in it. For more information see the OS9 User's Manual on:

- 1) OS9 memory modules
- 2) OS9 utilities LOAD, SAVE, FIXMOD
- 3) BASIC "Shell" statement

Information that is stored in OS9 memory modules can be read using the Functional Call GetDatMod. SEE ALSO GetDatMod.

run PutDatMod (ModName, Data, Size)

where:	Modname	=	A string expression for the module name to be created/written to.
	Data	=	A structure containing the data you want to store in the memory module. Default is 256 byte array of 0's.
	Size	=	Size of the data structure to be stored in bytes. This parameter defaults to the size of the data structure being passed to PutDatMod so is generally not needed.

EXAMPLE PROGRAM:

```
PROCEDURE PutMod_Test
DIM measurements (100):REAL \ REM Storage Space for 100 measurements
RUN Test (measurements) \ REM Run a test that takes the measurements
RUN PutDatMod ("TMOD",measurements) \ REM Store array in "TMOD"
END
```

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.12 READ DATA FROM MODULE (GETDATMOD)

BASIC VERSION

The GetDatMod Functional Call is provided to allow the easy retrieval of information from OS9 memory modules previously stored using PutDatMod (see PutDatMod). The call is made specifying a module name and a data structure into which the data previously stored will be placed.

run GetDatMod (ModName, Data, Size)

where:	Modname	=	A string expression for the module name to be read from.
	Data	=	A structure into which read data will be placed. This should be a structure of the same type as was used with PutDatMod to create/write the module.
	Size	=	The size of the data structure to be read in bytes. This parameter defaults to the size of the data structure passed to GetDatMod so is generally not needed.

EXAMPLE PROGRAM:

```
PROCEDURE GetMod_Test
DIM measurements(100):REAL\ REM Same data structure as used before
REM Now read the measurements stored before by PutDatMod
RUN GetDatMod("TMOD",measurements)
FOR Print_Out=1 to 100
```

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

```
PRINT "Value # ";Print_Out;" = ";measurements (Print_Out)
NEXT Print_Out
```

SECTION - 2.13 GPIB PROGRAMMING

The GPIB port is called /G. Its operation is similar to the other system I/O ports such as the printer and terminal ports. The main Basic statements used with an I/O port are OPEN, CLOSE, READ, PRINT, and INKEY. /G may also be used in Shell commands like Copy, Echo, etc. For more information on the Microware I/O system, refer to section 2 of the OS9 technical manual. I/O redirection from Shell is covered in chapter 5 of the OS9 users manual. I/O from Basic is treated in chapter 11 of the Basic manual.

The following sections discuss features peculiar to the GPIB. It must be kept in mind that all GPIB instruments behave somewhat differently, so only a general outline may be given here.

IFC/REN

When the GPIB port is initialized by opening a path to /G, the Interface Clear (IFC) line is pulsed and the Remote Enable (REN) line is asserted. When the path is closed, REN is dropped.

Pulsing IFC resets all instruments on the bus to their power-on states. Asserting REN disables the front panel controls. When REN is dropped, instruments revert to manual control.

To avoid resetting the bus every time a path is opened, it is advisable to open a dummy path that never gets closed. Then, additional paths may be opened and closed at any time without affecting IFC and REN.

A good way to do this is with the OS9 utility command Iniz. You may put the statement Iniz /G in your Startup file so that the GPIB port is automatically initialized once and for all. You may also use an OPEN statement in your main Basic program, but then be sure to close the path before quitting the program. Otherwise, each time the program runs another path will be opened and the system will eventually run out of available path numbers.

When using /G in a subprogram, you may either pass an open path number from the main program, or let the subroutine open and close its own path. The latter method is preferable since it does not require parameter passing and it keeps all I/O operations local to the subprogram. A dummy path must still be open at all times to avoid resetting the bus.

HEXADECIMAL CONVERSION

Many GPIB commands require the use of non-ASCII character codes. To simplify programming, the \$ character automatically performs ASCII-to-binary conversion on the following two characters. For example, the three-character sequence \$1B would be sent as an escape character.

If the \$ character itself is to be sent, it may be written twice as \$\$, similar to the way quotation marks must be written in print statements. It may also be written as \$24.

GPIB COMMANDS

GPIB commands are introduced by the ` character (this is NOT the single quote character. The ASCII code for the ` character is \$60). Most of these commands are non-ASCII codes and are written using hexadecimal conversion:

`\$01 GTL Go To Local	(enables manual controls)
`\$04 SDC Selected Device Clear	(resets listeners)
`\$05 PPC Parallel Poll Configure	(not used)
`\$08 GET Group Execute Trigger	(triggers listeners)
`\$09 TCT Take Control	(see slave operation, below)

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.13 GPIB PROGRAMMING

`\$11 LLO Local Lockout	(disables manual controls)
`\$14 DCL Device Clear	(resets all devices)
`\$15 PPU Parallel Poll Unconfigure	(not used)
`\$18 SPE Serial Poll Enable	(see Serial Poll, below)
`\$19 SPD Serial Poll Disable	(see Serial Poll, below)
`\$20 through `\$3E LAG Listen Address Group	(select listeners 0 through 31)
`\$3F UNL Unlisten	(deselects all listeners)
`\$40 through `\$5E MTA My Talk Address	(select talkers 0 through 31)
`\$5F UNT Untalk	(deselects all talkers)
`\$60 through `\$7F SCG Secondary Command Group	(not used)

Once again, if the ` character itself is to be sent, it may be written as either `` or \$60.

GPIB ADDRESS

The tester has a GPIB address, just like any other instrument. This address is normally 0. The address is stored in the device descriptor module G, and may be changed with the Shell command Xmode /G GPIB=xx. Legal addresses are 00 through 1E (decimal 0 through 30).

CONTROLLER PROTOCOL

Before the controller can send data (other than GPIB commands) it must select itself as talker by sending `\$40 (talk address 0). Failure to do so will result in Error #000:203 Bad I/O Mode.

Before the controller can receive data, it must select itself as listener by sending `\$20 (listen address 0). Failure to do so will result in Error #000:203 Bad I/O Mode.

Before the controller can send data, there must be a listener on the bus. The listener is selected with one of the Listen Address Group commands `\$21 through `\$3E (address 1 through 30). Failure to have a listener will result in Error #000:245 Write Error. This error will also occur on GPIB commands if there are no devices attached whatsoever (or all instruments are powered off). All instruments listen to GPIB commands whether they are selected or not.

When sending GPIB commands, a PRINT statement should end with a semicolon to suppress the carriage return. Here is an example of a typical exchange with a voltmeter:

DIM #g;integer	REM path number must be byte or integer
OPEN #g,"/g"	REM initialize port, clear GPIB
PRINT #g,"`\$21`\$40DCV"	REM meter listen, tester talk, DC volts mode
PRINT #g,"`\$20`\$41";	REM tester listen, meter talk, no terminator
READ #g,voltage	REM get voltage reading
CLOSE #g	REM return meter to manual

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.13 GPIB PROGRAMMING

TIMEOUT

A timeout feature may be enabled with the Shell command `Xmode /G TIME=nnn` (nnn ranges from 0 to 255 in tenths of a second). If the timeout is set to zero, or if the system clock is not running, then the timeout feature is disabled.

Timeout on input or output will result in Error 000:246 Device not ready.

The Basic function `INKEY(#path)` may be used to determine if data is ready to be received, so as to avoid getting hung up waiting for I/O. Since the GPIB is unbuffered, `INKEY` will only indicate 1 or 0 for ready or not ready, respectively.

TERMINATOR/EOI

`PRINT` statements automatically end with carriage return. If the `Xmode LF` parameter is on, this becomes carriage return + linefeed. If a semicolon is used at the end of the print statement, there is no automatic terminator, but you may explicitly code a terminator. For example, `PRINT "hello$0A";` would send a linefeed only.

The bus signal `EOI` (End or Identify) is asserted whenever a carriage return character is sent. The standard descriptor module `G` is set for `Xmode NOLF`, so the terminator consists of carriage return with `EOI`.

When receiving messages, a carriage return is required and the `EOI` signal is ignored. Warning: if an instrument sends carriage return + linefeed, the linefeed will not be read at the end of the message, but will probably appear at the beginning of the following message. If the instrument cannot be programmed to omit the linefeed, it may be necessary to strip it off in your program.

XMODE

The Shell command `Xmode` is used to set the GPIB address, timeout, and terminator parameters. `Xmode` works by modifying the device descriptor module `G`. This must be done before initializing the GPIB port. If the parameters are to be changed on the fly, all paths to the GPIB port must be closed. The new parameters take effect when the next path is opened.

To make the changes permanent, you may save the modified descriptor. However, the module `CRC` will be wrong and must be corrected before the module can be loaded. The following example command sequence will modify, save, and correct the descriptor:

```
Xmode /G time=10 ; Save-rx G ; Fixmod-ux G
```

Note that Xmode refers to device name /G while Save and Fixmod use the module name G.

SLAVE OPERATION

When the GPIB port is first initialized, it assumes that it is the Controller-In-Charge. However, up to 15 testers may be networked on a GPIB, and only one of them can be controller. In this case, each slave unit must have a unique address (`Xmode /G GPIB=xx`), and the slaves must relinquish control. This can be done two ways.

First, the tester automatically relinquishes control when it sees bus activity indicating the presence of another controller. When a unit sees its own talk or listen address command occur on the bus, it assumes another controller is present and relinquishes control.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.13 GPIB PROGRAMMING

Second, a unit may be programmed to relinquish control by giving the Take Control command (`\$09). This passes control to any unit currently addressed as talker. If each slave unit starts by giving the command sequence `\$40`\$09 (Talk Address 0, Take Control), then unit #0 will be the controller.

lave units cannot send any GPIB commands. Attempts to print the ` character will produce Error #000:203, Bad I/O Mode. The same error will occur on any PRINT or READ statement, unless the slave unit has been selected to talk or listen by the controller.

When a slave unit wishes to send or receive data, it must wait its turn. This means using the ON ERROR GOTO statement to handle I/O Error #000:203. The program should continue trying to send or receive until it succeeds, that is, until error 203 does not occur.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.14 RELAY MATRIX BOARD (MRLY)

INTRODUCTION

The Digalog Systems Relay Matrix board contains 256 relays organized in a 64 x 4 matrix (see Figure 1).

Up to 4 Relay Matrix boards may be placed into the testhead at a time. The Relay Matrix board may be operated in any of the five possible modes set by the ModMRLY functional. Mode 0 is the most flexible mode of the Relay Matrix board which allows any channel to any bus connection. A break before make feature is included and can be enabled or disabled by software control (mode 1). The duration of the break is programmable. Mode 2 emulates the Digalog Relay Multiplexer assembly. Mode 3 allows a bus to be connected to one channel only and mode 4 allows a channel to be connected to one bus only.

The relays used on this board are rhodium plated reed relays. They can be individually switched on or off using the MRLY functional call. The power on and reset state of the relays is open, with the break before make feature disabled.

This board, as with most of Digalog's products, contains built in self test. The functional call software checks the condition of the board every time it is run.

THEORY OF OPERATION

One slot of the Digalog analog testhead has 68 connections to the patch board receiver. Of the 68, 64 of these connections are used for one side of the matrix (see Figure 1). The remaining four receiver connections are used for the other side of the matrix. This allows the user to connect any number of 64 points to any of four "buses". This also means that any of the 64 points can be connected to any other of the remaining 63.

Circuitry on the relay matrix board will insure break before make operation of the relays. This feature is software controlled and may be enabled / disabled with the ModMRLY functional call.

APPLICATIONS

A typical application of this board is to replace the auxiliary relay board when switching to a common bus is desired. The relay matrix board allows the user a higher relay density per analog test head slot used. Such an application might be in pulling up (or down) high voltage UUT inputs (or outputs).

Another application this board might be used for is a multiplexor for external equipment used in highly specialized applications.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.14 MATRIX RELAY BOARD

BASIC VERSION

ModMRLY functional call changes the Mode of operation of the specified Matrix Relay board. BOARD is to select which board in the system to set the mode on. MODE is used to select the various ways in which the MRLY assembly can operate. TIME is the amount of time, in milliseconds, that the relays will remain off when the state of the relays is changed. The default time is 10 milliseconds.

run ModMrly (Board,Mode,Time)

Where:	Board	=		Selects the board with channels 0-63
		=	1	Selects the board with channels 64-127, etc.
	Mode	=	0	The mode is returned to normal. Any relay can be turned on or off at will and the break before make feature is disabled. All relays on the selcted board are turned off during this mode set call.
		=	1	The break before make feature is enabled. The duration of the break is set by the third parameter of this call.
		=	2	This mode makes the Matrix Relay assembly emulate the switching of the relays on the Relay Multiplexer assembly. The emulation divides the Matrix Relay board into four groups of sixteen channels each (ex. Channels 0-15, 16-31, 32-47, 48-63). Channels 48-63 are not accessible. When one relay in a group is turned on all other relays in the group are turned off. After this call is run the selected board will have Channel #0 connected to Bus #1, Channel #16 connected to Bus #2, and Channel #32 connected to Bus #3.
		=	3	In this mode, a bus can be connected to ONLY one channel. Connection to any other channel is removed prior to a new connection to the bus in question. All relays on the selected board are turned off during this mode set call.
		=	4	In this mode, a channel can be connected to ONLY one bus. Connection to any other bus is removed prior to a new connection to the channel in question. All relays on the selected board are turned off during this mode set call.
	Time	=	0-10	The relays will break for 10 milliseconds.
		=	10	The relays will break for the requested amount of time, up to 500 milliseconds.

EXAMPLES:

RUN ModMrly(0,1)

Sets the break before make feature to 10 milliseconds on board number 0 (channels 0-63).

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

RUN ModMrly(1,1,70)

Sets the break before make feature to 70 milliseconds on board number 1 (channels 64-127).

SECTION - 2.14 MATRIX RELAY BOARD (MRLY)

BASIC VERSION

The MRLY functional call is used to open(close), i.e., change State, of a Channel from(to) a Bus. CHANNEL is one of the 64 inputs from the testhead patch board receiver. BUS is one of 4 buses on a board. The board is determined by the channel number. STATE describes the desired condition of the relay selected by the channel and bus parameters. If STATE is:

run Mrly (Channel1,Bus1,State1.....,ChannelN,BusN,StateN)

Where:	Channel	=	0-255	The channel is selected.
		=	-1	All relays are opened.
	Bus	=	0	All 4 buses will be connected (disconnected) to the channel number selected by the previous parameter.
		=	1-4	A relay will connect (disconnect) the channel selected by the previous parameter to the selected bus number.
	State	=	0	The relay is opened.
		=	1	The relay is closed.

NOTE: Functional call ModMrly must be used to set the Mode of a MRLY board before functional call Mrly can be used. This is because there is no longer a power up default mode for a board.

RUN Mrly(-1) will reset the mode of MRLY board(s) to NO MODE as well.

EXAMPLES:

RUN Mrly(-1)

Opens all relays.

RUN Mrly(0,0,1)

Connects channel 0 to all 4 buses.

RUN Mrly(2,3,1)

Connects channel 2 to bus 3.

RUN Mrly(2,3,0)

Disconnects channel 2 from bus 3.

RUN Mrly(2,3,1,4,1,0)

Connects channel 2 to bus 3 and disconnects channel 4 from bus 1.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.14 MATRIX RELAY BOARD (MRLY)

SETTING THE BOARD NUMBER

The board number (channel number range) is set using a group of switches located on the Matrix Relay assembly. This group of switches is labeled SW1 and the individual switch numbers range from 1 to 4. Only switches SW1-1 and SW1-2 are used to set the board number. SW1-3 and SW1-4 should always be in the "ON" position.

	SW1-1	SW1-2
Board number 0, channels 0-63	ON	ON
Board number 1, channels 64-127	OFF	ON
Board number 2, channels 128-191	ON	OFF
Board number 3, channels 192-255	OFF	OFF

SPECIFICATIONS:

MRLY boards per testhead		4
MRLY software execution time	6 milliseconds	
ModMRLY software execution time		15 milliseconds
Contact resistance		0.200 2 typ.
Contact turn on time		0.5 milliseconds typ.
Contact turn off time		0.5 milliseconds typ.
Maximum switching current	0.5 Amps	
Maximum carry current		1.0 Amps
Maximum input voltage		100 Volts
Channel to channel crosstalk		-30 dB @ 1 kHz

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.15 CHECKING THE EXECUTION TIME OF PROGRAMS

BASIC VERSION

The functional call "NOW" is used to read the value of the real time clock. The value returned, by itself is of little interest. However, when BASIC program statements are placed between two successive NOW calls, the time that the program statements took to run can be calculated. NOW has a resolution of +/-10 milliseconds.

NOW can also be used in functional testing when long time periods need to be measured. Example 2 shows NOW being used with a repeat-until loop and the AMS functional call.

run Now(time)

Where: **time** = The value of time in seconds that the real time clock had in its registers during the NOW call is returned in this variable.

EXAMPLE PROGRAM 1:

PROCEDURE Test

REM This procedure will check the timing of a for-next loop.

RUN Now(time1)

FOR i=1 to 10000

a=b

NEXT i

RUN Now(time2)

Print "The time of the loop was";time2-time1;"seconds."

END

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.15 CHECKING THE EXECUTION TIMES

BASIC VERSION - CONTINUED

EXAMPLE PROGRAM 2:

REM We assume that the rest of the program has set up the
REM relay multiplexer and anything else needed. The 1 second
REM maximum times are used for examples and could be anything.

REM Wait for the signal to go high.
REM 1 second maximum

```
RUN Now(time1)
REPEAT
RUN ams(v,1,1)
RUN Now(time2)
UNTIL v >> 5.0 OR time2-time1 >> 1.0
```

```
IF time2-time1 >> 1.0 THEN 100
```

```
REM Wait for the signal to go low.
REM 1 second maximum
REPEAT
RUN ams(v,1,1)
RUN Now(time3)
UNTIL v << 0.5 OR time3-time2 >> 1.0
t=time3-time2
GOTO 200
```

```
100 t=0
200 PRINT "The time of the pulse was ";t;" seconds."
END
```

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.16 HIGH LEVEL I/O BOARD (HLIO)

INTRODUCTION:

The High Level Input Output Board (HLIO) is designed to drive and receive "digital" signals with high voltage levels. This design is an addition to the Series 2030 Analog Test System and can be inserted in any testhead slot. A 2030 Testhead can have a maximum of eight HLIO boards at a time.

Each slot of the Digalog Series 2030 Analog Test System has 68 connections to the patch board receiver (on the testhead). On HLIO boards, sixty four of these connections are used for Driver/Receiver bits, 2 are used for External Rail Voltage inputs, and the remaining 2 are connected to Digital Ground on the Test System.

The output drivers are fuse protected open-drain type capable of handling up to 50 Volts and sinking up to 0.5 Amp. Each Driver has its own internal pull-up resistor which simplifies patchboard wiring. The pull-up resistor can be disconnected, or can be connected to either an external rail voltage pin or the internal +5 Volts on the board. Each output pin is wrapped back to a receiver through a current limiting diode to voltage clamps.

FEATURES:

- 64 Input/Output channels.
- Open-drain type output drivers.
- Receivers tied directly to the output pins.
- Outputs current protected to 0.5 Amp.(fused).
- Inputs protected with current limiting diode to voltage clamps.
- 2 External Rail input pins.
- Software selectable internal/external rail voltages.
- All 64 channels updated simultaneously.
- Internally socketed rail pull-up resistor packs.
- No SelfTest Adaptor upgrades required.

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.16 HIGH LEVEL I/O BOARD (HLIO)

BASIC VERSION

The HLEn functional call is used to enable/disable the internal pull-up resistors for a selected byte of Drivers.

run HLEn (Byte0, Data0, ... ByteN, DataN)

where:	Byte	=	-1	All internal pullups on all boards will be disabled.
		=	0	Driver bits 0 through 7 on the first HLIO board.
		=	1	Driver bits 8 through 15 on the first HLIO board.
		=	8	Driver bits 0 through 7 on the second HLIO board.
	Data	=	\$00	All internal pull-ups disabled (disconnected). (Default)
		=	\$FF	All internal pull-ups enabled (connected to Rail).

EXAMPLES:

run HLEn

run HLEn(2,\$f0)

run HLEn(3)

Displays the HLEn helpfile
run HLEn(-1)All internal pullups will be disabled
Four high bit pullups in byte 2 will be enabled
Byte 3 pullups will be disabled

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.16 HIGH LEVEL I/O BOARD (HLIO)

BASIC VERSION

The HLRail functional call is used to select internal or external rail voltage for a specified bank of 32 bits on a HLIO board.

run HLRail (Bank0, Rail0, ... BankN, RailN)

where:	Bank	=	-1	All rails on all boards will be reset to internal.
		=	0	Driver bits 0 through 31 on the first HLIO board.
		=	1	Driver bits 32 through 63 on the first HLIO board.
		=	2	Driver bits 0 through 31 on the second HLIO board.
	Rail	=	0	Internal 5 Volt rail selected (default).
		=	1	External rail selected.

EXAMPLES:

run HLRail	Displays the HLRail helpfile
run HLRail(-1)	All rails will be reset to internal
run HLRail(0)	Bank 0 will be connected to internal 5V rail
run HLRail(0,1)	Bank 0 will be connected to external rail

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.16 HIGH LEVEL I/O BOARD (HLIO)

BASIC VERSION

The HLData functional call is used to setup and latch the output drive data for the selected byte of drivers. All drivers are updated simultaneously. Board containing the first byte specified in a HLData call acts as the master.

run HLData (Byte0, Data0, ... ByteN, DataN)

where:	Byte	=	-1	All data bits on all boards are reset to 0 and the Master is disabled.
		=	0	Driver bits 0 through 7 on the first HLIO board.
		=	1	Driver bits 8 through 15 on the first HLIO board.
		=	8	Driver bits 0 through 7 on the second HLIO board.
	Data	=	\$00	All bits set to logic-0 (default).
		=	\$FF	All bits set to logic-1.

EXAMPLES:

run HLData	Displays the HLData helpfile
run HLData(-1)	Resets all data bytes on all boards
run HLData(0)	Data byte 0 will be reset to 0(low)
run HLData(9,255)	All bits of byte 9 will be 1(high) and board 2 will be Master

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.16 HIGH LEVEL I/O BOARD (HLIO)

BASIC VERSION

The HLRead functional call is used to latch and read back the input data on the selected byte of drivers. All bytes are latched simultaneously. Board containing the first byte specified in a HLRead call acts as the master.

run HLRead (Byte0, Data0, ... ByteN, DataN)

where:

Byte	=	0	Receiver bits 0 through 7 on the first HLIO board.
	=	1	Receiver bits 8 through 15 on the first HLIO board.
	=	8	Receiver bits 0 through 7 on the second HLIO board.
Data	=		Returned data variable for ByteN - dimensioned as an INTEGER or BYTE.

EXAMPLES:

run HLRead

Displays the HLRead helpfile

DIM RtnData1,RtnData2:BYTE(* dimension RtnData1 and RtnData2 *)

run HLRead(5,RtnData1)

Board 0 will be the Master and byte 5 status will be returned in RtnData1

run HLRead(9,RtnData1,6,RtnData2)

Board 1 will be the Master, byte 9 will be returned in RtnData1
and byte 6 status will be returned in RtnData2

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.16 HIGH LEVEL I/O BOARD (HLIO)

SPECIFICATIONS:

HLIO boards per testhead	8
HLEn software execution time	3 milliseconds
HLRail software execution time	3 milliseconds
HLData software execution time	7 milliseconds
HLRead software execution time	9 milliseconds
Driver pull-up resistor	510 ohm socketed DIP
Maximum Driver current	0.5 A fuse protected
Maximum external rail voltage	50 Volts
Maximum Receiver current	240 milliamps

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.17 SET CALIBRATION DA VOLTAGE (TDAC)

BASIC VERSION

The TDAC Functional Call is used to program the output voltage of the precision digital to analog converter contained in the SelfTest Assembly. The output range of the TDAC is - 10/+9.99976 VDC. Patchboard power MUST be on in order for the TDAC to function. This is done by running POWER(-1,1). If patchboard power is not supplied, Error 099:170 will result .

The output of the TDAC can be monitored from the front of the SelfTest Assembly at the BNC connector marked TDAC. This is the upper-left most connector.

Accuracy of the TDAC is established during system certification. See Certification for more information.

run TDAC (Volts, Actual)

where:	Volts	=	The desired output voltage for the TDAC in the SelfTest Assembly. The value ranges from -10 VDC to +9.99976 VDC.
	Actual	=	Return of the actual value programmed after conversion from real to binary and roundoff.

EXAMPLES:

[run POWER (-1,1)]
run TDAC (9.375, actual)
run TDAC (0, actual)

Patchboard power must be on
Set TDAC output to 9.375 VDC
Set TDAC output to 0.000 VDC

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.17 SELFTEST ASSEMBLY RELAYS (TSTRLY)

BASIC VERSION

There are 64 test relays in the SelfTest Assembly that are used by various SelfTest programs. Only one test relay may be on at any one time, all others are automatically turned off. Running TSTRLY with no argument turns them all off.

run TSTRLY (chan)

where: **Chan** = The number of the test relay to be turned ON. Only one can be on at a time. If chan is omitted, all test relays are turned OFF.

EXAMPLES:

run TSTRLY
run TSTRLY(1)

Turn OFF all test relays in the Selftest Assembly
Turn ON test relay channel 1

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.17 READING TESTHEAD EEPROMS (GETEEP)

BASIC VERSION

The GetEEP Functional Call is used to read the contents of the EEPROMs in the Testhead boards. The contents are written using the Functional Call PutEEP. GetEEP and PutEEP are used by CONSTAT software to transfer data to and from the EEPROMs. They may be called from BASIC, though the information stored in the EEPROMs is not in a human readable form. For reference, however, the storage format is shown here:

BYTES	CONTENTS	TYPE
0-1	Serial Number	UNSIGNED SHORT INT
2-3	Part Number	UNSIGNED SHORT INT
4-5	Ship Date	SHORT INT
6-15	ECO NUMBER	UNSIGNED CHAR[10]
16-31	Customer Name	CHAR[16]
32-47	Description	CHAR[16]
48-67	ECO Install Dates	SHORT INT[10]
68-77	RMA list	UNSIGNED SHORT INT[5]
78-79	Checksum	UNSIGNED SHORT INT
80-123	UNUSED	
124	Test Byte (Currently Unimplemented)	
125-127	"DLI" in most boards written by CONSTAT	

run GetEEP (Array, FromSlot)

where: **Array** =This parameter is the data structure into which the data will be placed by GetEEP. It makes best sense to DIM as an array of BYTEs. An array to hold all Testhead board EEPROMs could be DIM a(2048):BYTE. Each EEPROM has 128 bytes of data.

FromSlot =The starting slot number to begin reading from (-1 to 15). The numbers 0 to 15 correspond to the slots in the Testhead, 0 being the PS Controller, 14 being the MDE, and 15 being the SelfTest Assembly. The slot -1 is used to access a second EEPROM on the SelfTest Assembly not used by CONSTAT. Reading begins from the "From Slot" and continues as long as there is space in Array to place the data.

EXAMPLE PROGRAM:

```
PROCEDURE ReadBack
DIM info(2048):BYTE
run GETEEP(info, 0)
/REM The contents of all the Testhead board EEPROMs are now in the
/REM array "info" for reference.
```

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.17 READING TESTHEAD EEPROMS (PUTEEP)

BASIC VERSION

The PutEEP Functional Call is used to write the contents of the Testhead board EEPROMs. The contents can later be read with the GetEEP Functional Call. PutEEP and GetEEP are used by CONSTAT software to transfer data to and from the EEPROMs. They may be called from BASIC, though the information stored in the EEPROMs is not in a human readable form. **Using PutEEP can alter/destroy information stored in the EEPROMs. Its use outside of CONSTAT software is not recommended.** For reference purposes, the storage format used by constat is included in GetEEP's description (see GetEEP for more information).

run PutEEP (Array, FromSlot)

where: **Array** =This parameter is the data structure from which the data will be taken by PutEEP and written to the EEPROMs. It makes best sense to DIM as an array of BYTES. An array to hold all Testhead board EEPROMs could be DIM a(2048):BYTE. Each EEPROM has 128 bytes of data.

FromSlot =The starting slot number to begin reading from (-1 to 15). The numbers 0 to 15 correspond to the slots in the Testhead, 0 being the PS Controller, 14 being the MDE, and 15 being the SelfTest Assembly. The slot -1 is used to access a second EEPROM on the SelfTest Assembly not used by CONSTAT. Writing begins from this slot and continues as long as there is data in Array to write.

EXAMPLE PROGRAM:

```
PROCEDURE Write
DIM info(2048):BYTE
REM Initialize the Array to Zeroes
FOR LP= 1 to 2048 \ info(LP)=0 \ NEXT LP
run PutEEP(info, 0)
REM All bytes of all EEPROMs on all Testhead boards are now zero.
```

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

SECTION - 2.17 MEMORY TEST (MEMTEST)

BASIC VERSION

The MEMTEST Functional Call is used to test RAM. It is called with a starting address, and an ending address to test. It returns the address that failed test, or the end address plus one if no failures occurred.

run MEMTEST (StartAddr, EndAddr)

where: **StartAddr** =An integer variable set to the address at which testing is to begin. This is also the return parameter and is set to the address that fails test, or EndAddr+1 if no failures occur.

EndAddr =The end address at which testing stops. Can be an integer variable or integer constant.

NOTE: There are two programs MTST1MB and MTST.25MB that are set up to automatically test the appropriate locations of the respective memory boards. To function, these tests require MEMTEST to be loaded into memory or available in the CMDS directory.

EXAMPLES:

run MEMTEST(\$200000,\$2FFFFFF)

Test memory addresses from \$200000 to \$2FFFFFF

This is what is done by MTST1MB

run MEMTEST(\$8000,\$0BFFFF)

Test memory addresses from \$8000 to \$0BFFFF

PROGRAMMING THE ANALOG FUNCTIONAL CALLS

These are the parameters passed by MTST.25MB

This section will present some fundamentals that maintenance personnel should be familiar with. This includes general information and commonly used commands. If you desire more information about OS9 or Microware BASIC, you can refer either to the sections in this manual titled "Learning the OS9 Operating System" and "Learning BASIC", or directly to the appropriate Microware reference manual. This section is meant only as a quick reference.

OS9 FUNDAMENTALS

When in the OS9 Operating System, you will see a '\$' prompt on the terminal screen. This level is also referred to as the "Shell". The Shell is a "command line interpreter". After the '\$' prompt you type whatever command you wish to execute and end the command line with a <CR> (Carriage Return). The <CR> is labeled "RETURN" on many terminals. After you type a command and a <CR> in the Shell, the operating system will attempt to do whatever you instructed it to do. If an error results, an appropriate message will be displayed.

Whenever an error occurs, an error message is displayed along with the associated error code number. To find out what this code means, you can either refer to a listing of error codes (Appendix D) or use the PERR utility.

- 1) To generate an error code listing, make sure your system has a printer and enter the following Shell command:

```
"$ LIST /S0/SYS/ERRMSG >/P <CR>"
```

- 2) To use the PERR (print error) utility, enter the following Shell command:

```
"$ PERR <ERRNUM> <CR>"
```

where <ERRNUM> is the error code you wish to know.

If you have an error code such as 099:102, you can enter:

```
"$ PERR 99 102 <CR>"
```

Other OS9 Shell commands that are used very commonly include the following:

BUILT IN SHELL COMMANDS

CHD <path> -Change Current DATA Directory.

CHX <path> -Change Current EXECUTION Directory.

SYSTEM MAINTENANCE

MAINTENANCE LEVEL OS9 AND BASIC COMMANDS

SECTION - 6.2

COMMON UTILITY COMMANDS

Page numbers refer to OS9 User's Manual

COPY	-Copy data from one path to another.	Page 8-17
DATE	-Display System date and time.	Page 8-20
DEL	-Delete a file.	Page 8-25
DELDIR	-Delete directory & all files in it.	Page 8-26
DIR	-Display contents in a directory.	Page 8-27
LIST	-List contents of a text file.	Page 8-61
MAKDIR	-Create a directory file.	Page 8-67
PD	-Print Current Directory.	Page 8-85
RENAME	-Change a file name.	Page 8-92
SETIME	-Activate and set system clock.	Page 8-97

You should refer to each of these commands in the OS9 User's Manual and familiarize yourself with them. Page numbers are shown for quick reference.

BASIC FUNDAMENTALS

There are four modes associated with BASIC.

- 1) System Mode
- 2) Edit Mode
- 3) Debug Mode
- 4) Execution Mode

Here are the various commands that are valid in each of the above modes.

SYSTEM MODE

\$ < SHELL COM >	-SHELL Command.
LIST < proc name >	-LIST Procedure.

BYE	-EXIT Basic.
DIR or <CR>	-Display Directory of Workspace.
E/EDIT	-Enter Edit Mode.
KILL	-Delete Procedure from Workspace.
LOAD <path>	-LOAD Basic Source from <path>.
CHD/CHX	-Change directory (Data/Exec).
MEM < # nnnnk >	-Display/Request Workspace Mem.
PACK <proc name>	-Extra compiler pass on procedure.
RENAME <from> <to>	-Rename process from <from> to <to>.
RUN <proc name>	-Execute a procedure.
SAVE <proc>	-Write source to file.

NOTE: The above are applicable when in the SYSTEM MODE of BASIC (i.e. the "B:" prompt). For complete information on usage refer to Part II - Chapter 4 of the BASIC Users Manual.

EDIT MODE

<CR>	-Move edit pointer forward 1 line.
+n	-Move forward.
-n	-Move backward.
<space> <text>	-Insert Unnumbered Line.
<space> <ln#> <CR>	-Insert or Replace numbered line.
<line #> <CR>	-Move to numbered line.
c	-Change string.
d	-Delete line.
l	-List line.
q	-Quit Editing.
s	-Search for string.

NOTE: The above are applicable when in the EDIT MODE of BASIC (i.e. the "E:" prompt). For complete information on usage refer to Part 2 - Chapter 5 of the BASIC Users Manual.

DEBUG MODE

\$<SHELL COM>	-SHELL Command.
BREAK	-Set Breakpoint.
CONT	-Continue execution.
DEG/RAD	-Select DEGree/RADian units.
DIR	-Display workspace directory.
LET	-Assignment statement.
LIST	-List current procedure.
PRINT	-Print present value of a variable.
Q	-Quit debug mode.
STATE	-List calling order of procedures.
STEP <n>	-Execute single/<n> steps at a time.
TRON/TROFF	-Trace mode on/Trace mode off.

NOTE: The above are applicable when in the DEBUG MODE of BASIC (i.e. the "D:" prompt). For complete information on usage refer to Part II - Chapter 7 of the BASIC Users Manual.

EXECUTION MODE

The execution mode is entered when a program is RUN from the system mode. There are no commands used "within" this mode.

System certification ensures that system standards that are used for SelfTest and autocalibration are accurate and can be traced to National Bureau of Standards (NBS). There are two elements of the Series 2030 Functional Test System that need to be certified: 1) The TDAC high precision digital to analog converter on each SelfTest Assembly, and 2) The TCXO precision oscillator on the Time Measurement System (TMS) in the 2030's testhead.

To carry out certification the following equipment is needed:

- 1) A Digalog Calibration Station (cables are inside).
- 2) A Digalog SelfTest Assembly.
- 2) A floppy disk containing Cert2030 certification software.
- 3) A printer, if a hardcopy certificate is desired.

The Calibration Station contains a precision digital multimeter and a precision counter/timer. Both instruments are equipped with a GPIB interface so that the certification process (in terms of instrument setup and measurement processing) is completely automatic.

Each Calibration Station is shipped with two floppy disks containing certification software. One disk contains "Cert2000" which is certification software for pre-2030 Series systems. For Series 2030 testers you MUST use the "Cert2030" software or errors will result.

SETUP

To get started with certification, the certification program must be executed. To do this:

- 1) Place the floppy containing Cert2030 in /s1 (the floppy drive) and close the door on the drive.
- 2) From OS9's Shell (the Operating System with the '\$' prompt) you should enter:
 - a) \$ LOAD /S1/CERT~~2000~~ <CR>
 - b) \$ CERT~~2000~~ <CR>

You should now be running the certification software where information on how to use the program is found. A number of connections between the 2030 and the Calibration Station must be made before continuing. Information about these connections is available from the main menu in the certification program and includes:

- 1) Installation of the SelfTest Assembly.
- 2) Power connections for the Calibration Station.
- 3) GPIB IEEE-488 communication link between the 2030 and the Calibration Station.
- 4) Connections between the TDAC in the SelfTest Assembly to the multimeter.
- 5) Connections between the 10 MHz output of the TCXO on the testhead and the counter/timer.

The SelfTest Assembly is installed the same way for certification as it is for SelfTest and autocalibration. After removing any patch panel that may be installed on the testhead, lower the SelfTest Assembly on the testhead and lock the cam to engage. Also make sure that the data cable coming from the Selftest Assembly is plugged into the SelfTest access connector immediately behind the patchboard.

The Calibration Station requires one 110 VAC outlet to supply power to both instruments inside it. There are usually unused outlets on the utility strip at the rear of the 2030 where you can plug it in. If not, a grounded outlet nearby will suffice.

The communication between the CPU and the instruments in the Calibration Station occurs over the GPIB cable (supplied with the Calibration Station). One end should be connected securely to the GPIB connector on the front panel of the Calibration Station. The other end must be connected to an identical connector on the rear of the CPU rack of the 2030. To make this connection it may be necessary to lift and slide the equipment bay cover off if a GPIB cable is not already installed, there by permitting external access.

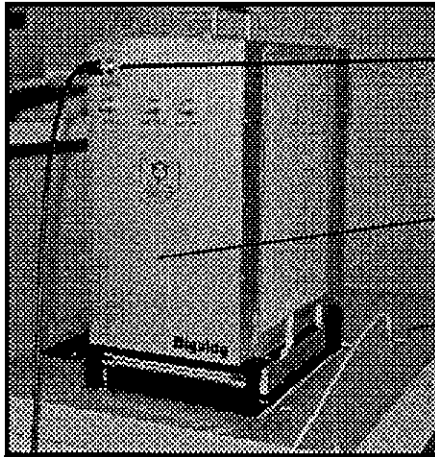
The connection between the TDAC and the multimeter is made with a coaxial cable with BNC type connectors on both ends. One end mates with the VCAL/MON jack on the front of the SelfTest Assembly (it is the upper left connector on the front panel of the SelfTest Assembly). The other end mates with an adapter (supplied) that plugs into the multimeter's inputs (labelled HI and LO). Note that one edge of the adapter is tabbed indicating the GND pin. This input corresponds to the LO input of the multimeter.

The last connection between the 10 MHz output on the testhead and the counter/timer is made with a BNC to mini-mate cable (supplied). The mini-mate end is connected to the 10 MHz output which is found on the top of the testhead near the SelfTest access. It is the jack furthest to the left as you look from the front of the system. The BNC end is connected to the Channel A input on the counter/timer.

With all connections made, you are now ready to carry out the certification of the system. Selecting the 'C' option in the program will do this. You will be asked for various information about the system being certified as well as the Calibration Station being used to perform the certification. This information is merely for printing on the certification documents for traceability purposes and is optional.

For the portion of the test dealing with the TDAC, a variety of voltage points are programmed and the output voltage of the TDAC is read at each point. At any given point, the measured value must not differ from the programmed voltage by more than 1.2207 mV or a failure will result. If the TDAC does not pass certification, a manual adjustment is necessary. Contact Digalog Systems in such an event.

For the portion of the test dealing with the TCXO an initial frequency measurement is taken to see if the output is even in the ballpark. This is done to avoid going into a settling delay if connections, for example, are incorrect. If the pre-settled measurement of the TCXO fails, the problem is most likely an improper connection. Review the instructions above and try again. Once the settling delay is begun, nothing will happen for the next 25 to 30 minutes to allow the TCXO and the counter/timer's oven oscillator to settle. If both are warm, the delay can be aborted by pressing any key on the keyboard. This will cause the frequency measurement to be taken instantaneously. The tolerance on the measurement is +/- 5 Hz. A TCXO out of range and requires manual adjustment. Contact Digalog Systems in such an event.



2030 CERTIFICATION SETUP

- A - VCAL/MON output on SelfTest Assembly
- B - SelfTest Assembly (installed)
- C - 10 MHz output (1st from left)
- D - Multimeter signal input (with adapter)
- E - Counter/timer input (Chan A)
- F - Calibration Station power cord (UK shown)
- G - Calibration Station GPIB IEEE-488 connector
- H - Digalog Systems Calibration Station

Calibration on the Series 2030 Functional Test System is completely automated. All calibration is based on high precision references built into the system testhead and the SelfTest Assembly. A high precision Temperature Compensated Crystal Oscillator (TCXO) on the Time Measurement System Board (TMS) is used as a frequency/time reference. A precision digital to analog converter (TDAC) contained within the SelfTest Assembly is used as a voltage reference for system calibration. The section titled "System Certification" describes how traceability of these references to secondary NIST (formally NBS) standards is accomplished. Given a complete 2030 tester with a certified TCXO and a certified TDAC, "Auto-Calibration" of the 2030 can begin.

PURPOSE OF CALIBRATION

Calibration of the functional tester is necessary to obtain gain and offset terms for all signal paths that exist within the system, and to determine timing for the MDE's (Measurement Display Electronics) sweeps, delays, triggers, and measurement mark. Since potentiometers are used in only a few instances where factory calibration is required, the majority of the calibration is done by taking measurements and calculating calibration terms which can be referenced at a later time. These terms are used by the Functional Hardware Calls to account for gains, offsets, and timing discrepancies, and their use is transparent to the user.

In all calibration programs, limits are set on what values any given calibration term may take on. If a calculated value is outside of the allowable range, a failure will result and be displayed/printed to the output device selected in the Test Executive.

CALIBRATION PROGRAMS AND CALIBRATION TERM STORAGE

In the previous sections the generation of calibration terms by calibration programs was discussed. It was mentioned that these terms are stored so that they can be referenced at a later time. With only one exception, the terms are placed in arrays that are stored in OS9 data memory modules. These modules are saved in the directory /S0/CMDS so that if the computer is reset, the calibration data is not lost. The only exception is POWER_C, the calibration program responsible for calculating calibration terms for the UUT Product Power Supplies. For each UUT Product Power Supply, the calculated terms are stored in an EEPROM on the configuration card of the UUT PPS Controller for that supply. This information is also retained after reset/power-up.

Below is a list of the current family of calibration programs and the calibration memory modules they generate.

CALIBRATION PROGRAM	CAL TERMS STORED IN
AMS_c	OS9 Memory Module "AMC" —
DA_v_c	OS9 Memory Module "DAC" —
ARB_v_c	OS9 Memory Module "ABC" —
DIO_c	OS9 Memory Module "RSC" —
MDE_c	OS9 Memory Modules "SWC", "DYC", "MKC"
POWER_c	UUT PS Controller EEPROM(s)

For additional information about these programs, refer to their header information which is available through Option 4 of the Test Executive's main menu. For more information on the Test Executive, refer to the chapter on the Test Executive.

WHEN TO CALIBRATE

SelfTest (see next chapter) is the best guide to determine if calibration is required. The battery of SelfTest programs is designed to test all functions of the 2030 to their specifications. If any failures occur during SelfTest (especially borderline failures), calibration should be carried out before proceeding.

Running through SelfTest periodically can tell you if calibration is or is not required. If you find that all SelfTest programs run without failures, there is no need to recalibrate any portion of the tester. The recommended guideline is to calibrate every 3 months.

Also, whenever a temperature change of more than five (5) degrees Celsius occurs or a testhead board swap is made, the 2030 should be recalibrated.

HOW TO CALIBRATE

System calibration programs are organized within the Test Executive. To be able to carry out calibration (and/or SelfTest) you will need to be familiar with the Test Executive's use. Please refer to Chapter 4 for more information on the Test Executive if necessary.

NOTE: Calibration (and SelfTest) programs may be run outside the Test Executive. When this is done, ALL data (i.e. pass or fail) is directed to the standard output path (typically the terminal). See OS9 Users Manual on Standard Input/Output.

Within the Test Executive, calibration program names are contained in the test menu named "CALIBRATE". This must be your current test menu in order to execute calibration programs. Option 5 in the main menu will enable you to change test menus. Use Option 7 "Execute Test Menu Level" to allow execution of desired calibration programs.

In order to calibrate the system, you need to follow these steps.

- 1) Install the SelfTest Assembly on the patchboard interface.
- 2) Change your Data Directory to "/S0/SelfTest". This is done by typing "CHD /S0/SELFTEST" in the Shell ('\$' prompt) or in BASIC ('B:' prompt).
- 3) Execute the Test Executive. This is done by typing "EXECUTIVE #50k" in the Shell ('\$' prompt). In BASIC you may enter "run EXECUTIVE" to start. If unfamiliar with the Test Executive, you should refer to Chapter 4 - The Test Executive before proceeding.
- 4) You will then be using the Executive's Execute Test Menu Level to execute the calibration programs. Again, Chapter 4 contains the information you need to do this.

NOTE: System Calibration should be performed with all covers installed on the 2030. Also, the system should have been switched on for at least thirty (30) minutes with all covers in place prior to calibrating. Calibrating without doing so will cause calibration terms to be stored that are not consistent with normal operating conditions. System SelfTest may pass initially after a "cold calibration", then fail after warming up.

SECTION - 6.4

WHAT IF CALIBRATION FAILS

Any type of failure in calibration could indicate a hardware problem with the 2030 being calibrated. Another attempt at calibration might verify that a problem truly exists. If repeated attempts at calibration fail, selftest programs that verify digital communication between the CPU and the various testhead boards should be executed (these programs have names ending in "_dig_f" for digital-functional test and are found in the test menu "FUNCT" -- see next chapter for information about SelfTest).

Other common problems that occur include product power supplies that aren't switched on, SelfTest assembly data cable not connected, and/or testhead power supply fault (refer to indicator LEDs on testhead top panel -- this requires turning the system off and on again).

Beyond these common problems, the following list may guide you to the most likely hardware culprit(s). In the following list, the cause(s) listed assume that other calibration programs in the menu passed.

<u>FAILING CAL PROGRAM</u>	<u>PROBABLE CAUSE ???</u>
AMS_c	Bad RMUX, AMS.
DA_v_c	Bad ASB.
ARB_v_c	Bad ASB.
DIO_c	Bad DIO.
MDE_c	Bad TMS, MDE.

The above is only a brief guide and further troubleshooting using system SelfTest will probably be necessary.

SECTION 6.5

SelfTest on the Series 2030 Functional Test System, like calibration, is completely automated. Like system calibration programs, the programs that comprise system SelfTest are organized under the Test Executive. The test menu containing names of SelfTest programs is called "FUNCT".

SelfTest programs exercise the hardware functions available to the user through the Functional Hardware Calls, and in many cases check function of hardware at more fundamental levels than can be accessed through the Functional Hardware Calls.

PURPOSE OF SELFTEST

SelfTest exists to check the functionality of all modules of the test system to their printed specifications. Additionally, it is used to troubleshoot hardware problems that arise. The SelfTest programs attempt to utilize all options available through all of the Functional Hardware Calls.

In all SelfTest programs, limits are set on every measurement that is made. If any measurement is outside of the allowable range, a failure will result and will be displayed/printed.

SELFTEST PROGRAMS

There are currently over thirty SelfTest programs found in the test menu "FUNCT". To see a current list of the SelfTest programs bring up the Test Executive and examine the test menu. Information about each of the SelfTest programs is available through Option 4 of the Test Executive's Main Menu ("Print SelfTest Program Helpfiles"). This information includes the program's revision level and describes what the program is responsible for testing. See chapter on the "Test Executive" for additional information on this topic.

WHEN TO SELFTEST

SelfTest can be performed whenever you want to verify that the system is functioning properly. SelfTest is the best guideline as to whether system calibration is necessary. The recommended time between calibrations is three months (refer to section on "Calibration"). Executing SelfTest periodically during the time between calibrations will verify that the system is functioning within specifications given current calibration terms. If failures occur during SelfTest program execution, calibration should be performed and then functionality should be checked again by running through SelfTest.

HOW TO SELFTEST

System SelfTest programs are organized within the Test Executive. To be able to carry out SelfTest (and/or calibration) you will need to be familiar with the Test Executive's use. Please refer to the chapter "Test Executive" for more information if necessary.

NOTE: SelfTest (and calibration) programs may be run outside the Test Executive. When this is done, ALL data (i.e. pass and fail) is directed to the standard output path (typically the terminal -see OS9 Users Manual on Standard Input/Output).

Within the Test Executive, SelfTest program names are contained in the test menu named "FUNCT". This must be your current test menu in order to execute SelfTest programs from the Test Executive. Option 5 "Menufile Editor" in the main menu will enable you to change test menus. Use Option 7 "Execute Test Menu Level" to allow execution of desired SelfTest programs.

In order to selftest the system, you need to follow these steps.

- 1) Install the SelfTest Assembly on the patchboard interface.
- 2) Change your Data Directory to "/S0/SELFTEST". This is done by typing "CHD /S0/SELFTEST" in the Shell ('\$' prompt) or in BASIC ('B:' prompt).
- 3) Execute the Test Executive. This is done by typing "EXECUTIVE #50k" in the Shell ('\$' prompt). In BASIC you may enter "run EXECUTIVE" to start. If unfamiliar with the Test Executive, you should refer to the chapter "Test Executive" before proceeding.
- 4) You will then be using the Executive's option "Execute Test Menu Level" to execute the calibration programs.

The "FUNCT" test menu has been organized so that tests that are closely related are blocked together in groups.

Digital tests are grouped together and typically are the first block of tests in the menu since they are the most fundamental. If the digital SelfTest programs fail, chances are very great that other SelfTest programs found elsewhere in the menu will fail as well. Calibration does not affect the function of digital SelfTest programs. Failure(s) in any of these programs indicates a hardware problem. Current digital SelfTest programs include:

TRLY_dig_f INST_dig_f ARLY_dig_f RMUX_dig_f
AMS_dig_f TMS_dig_f ARB_mem_f

Other SelfTest programs found in the menu are named such that the primary module being tested is the first part of the program name. For example, AMS_modes_f is a SelfTest program primarily responsible for testing the function of the various measurement modes of the Amplitude Measurement System.

SelfTest programs may be executed in any order (i.e. they do not depend on one another).

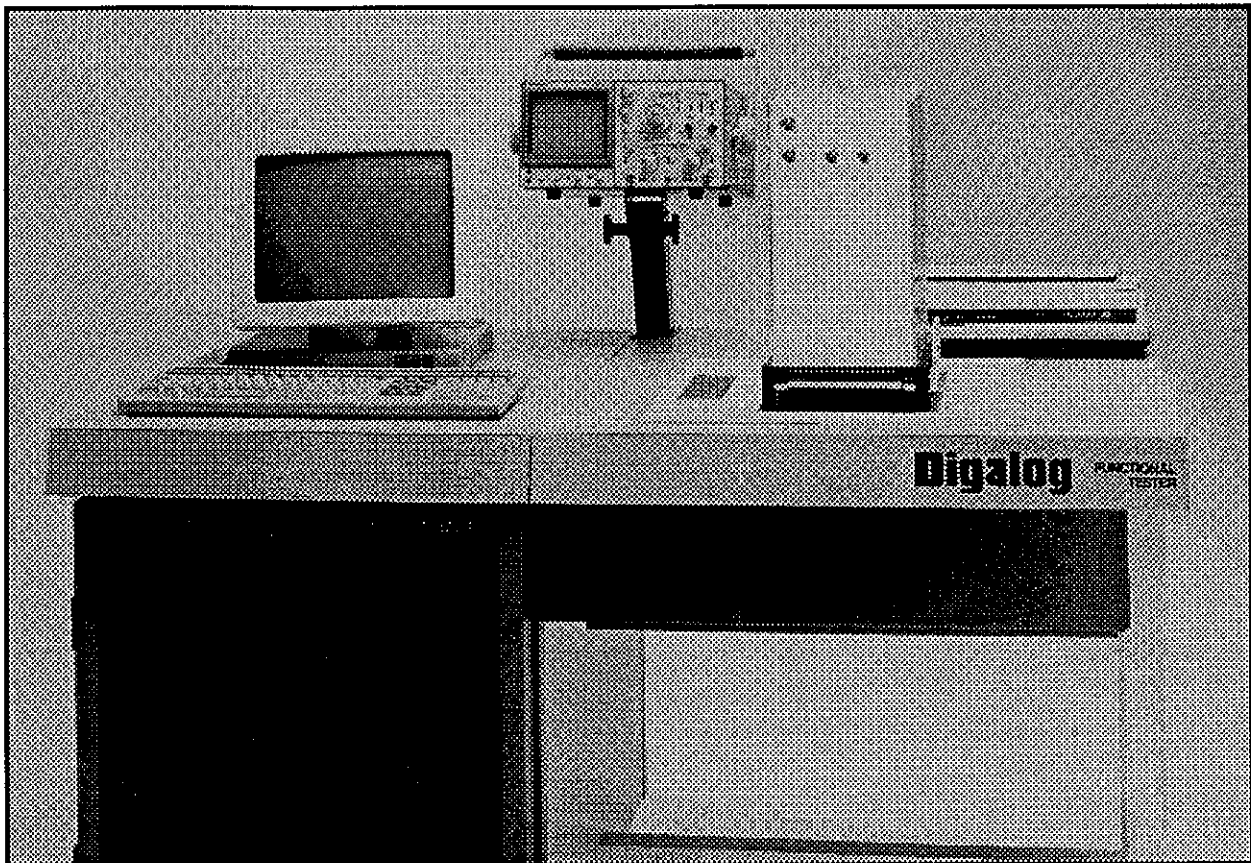
WHAT IF SELFTEST FAILS

If selftest programs fail, you should first check to see that one of these common problems is not at fault.

- 1) **SelfTest Assembly Improperly Installed.** Check to make sure the assembly is mounted properly on the patchboard. The cam should be engaged. Also ensure that the SelfTest data cable is installed correctly.
- 2) **UUT Power Supplies OFF.** Check to see if the UUT power supplies are switched on.
- 3) **Testhead Power Supply Fault.** Sometimes a Testhead power supply fault will occur and a variety of problems will result. Check the indicators on the testhead top to see that all Testhead power supplies are up (LEDs ON). If any are not, the system needs to be powered down and then switched on again. (See Section 6.9 on Testhead power supplies)

After the above common problems have been ruled out, a failure in selftest could indicate a hardware problem. However, often errors are due to the system needing calibration. Attempt to recalibrate the system and rerun any selftest programs.

If after recalibrating the system, selftest still fails, chances are a hardware problem exists. Diagnosing what failures are occurring in which selftest programs should indicate where hardware problems exist. Refer to Section titled "SelfTest Failure Analysis".



Vertical Configuration Shown

SYSTEM SELFTEST

Install the SelfTest adapter by rotating the receiver handle forward and positioning the adapter rear corner first in the receiver. Rotate the adapter forward so that it mates with the receiver. Reposition the handle, locking the adaptor in place. The flat cable in the rear of the adapter must be plugged in to the SelfTest access connector just behind the receiver.

SECTION - 6.6

This section is designed to aid in troubleshooting failures that may occur during the Series 2030 AutoCal and SelfTest Functional Tests. It is assumed that all cables are properly installed, and the internal Power Supplies are operational (see section of manual on System SelfTest).

CALIBRATION TEST MENU

All of the AutoCal programs have been designed to calibrate individual modules or sections of the system independent of all other sections. It should be noted that sections of the system may be made up of more than one board (ex. AMS and RMUX). If failures occur during AutoCal, it is recommended that the Functional SelfTest programs be executed in order to determine the source(s) of the problem.

The following is a list of AutoCal programs located in the Calibration section of SelfTest with the three most likely boards or assemblies that could cause failures.

TEST NAME	#1	#2	#3
2) AMS_c	AMS	RMUX	ISA
4) DA_v_c	ASB	ISA	AMS
5) ARB_v_c	ASB	ISA	AMS
7) DIO_c	DIO	TH CONT/CPU	SELFTEST PB.
8) MDE_c	MDE	TMS	AMS
10) POWER_c	UUT CONTROLLER	AMS/RMUX	SELFTEST PB.

FUNCTIONAL TEST MENU

The menu of Functional tests in SelfTest has been structured to start out by testing the basic digital circuitry and progress towards testing entire functional blocks and modules.

Tests numbered 1 through 7 are the digital tests and give an indication that the computer can communicate with boards in the system. Failures in these tests could cause numerous failures in the remainder of tests in the menu. Cables and power supplies should be checked before proceeding on to the next tests.

Failures in the tests numbered 10 through 40 could give an indication that the system is out of calibration if the readings are slightly out of tolerance.

The following page is a list of Functional SelfTests with the three most likely boards or assemblies that could cause failures.

SYSTEM MAINTENANCE
SELFTTEST FAILURE ANALYSIS

SECTION - 6.6

	TEST NAME	#1	#2	#3
1)	TRLY_dig_f	SELFTTEST PB.	TH CONT/CPU	P/S CONT.
2)	INST_dig_f	ISA	TH CONT/CPU	P/S CONT.
3)	ARLY_dig_f	ARLY/AFET	TH CONT/CPU	P/S CONT.
4)	RMUX_dig_f	RMUX	TH CONT/CPU	TH P/S CONT.
5)	AMS_dig_f	AMS	TH CONT/CPU	TH P/S CONT.
6)	TMS_dig_f	TMS	TH CONT/CPU	TH P/S CONT.
7)	ARB_mem_f	ASB	TH CONT/CPU	TH P/S CONT.
9)	SERIAL_port_f	IO CONT/CPU	TH P/S CONT.	P/S DIST ASSY
10)	DIO_f	DIO	SELFTTEST PB.	TH P/S CONT.
11)	DIO_ext_f	DIO	SELFTTEST PB.	TH P/S CONT.
12)	TMS_test_f	TMS	MDE	TH P/S CONT.
13)	TMS_event_f	TMS	MDE	AMS
15)	RMUX_f	RMUX	AMS	SELFTTEST PB.
16)	RMUX_prot_f	RMUX	AMS	MDE
17)	ARLY_f	ARLY/AFET	RMUX	AMS
19)	AMS_modes_f	AMS	ISA	ASB
20)	AMS_sig3_f	AMS	ISA	TH PS CONT.
21)	MDE_permeas_f	MDE	TMS	AMS
22)	MDE_freq_f	MDE	TMS	AMS
23)	MDE_delttime_f	MDE	TMS	AMS
24)	MDE_delmodes_f	MDE	TMS	AMS
25)	MDE_markplace_f	MDE	TMS	AMS
26)	MDE_measmark_f	MDE	TMS	AMS
27)	MDE_triglev_f	AMS	MDE	TMS
28)	MDE_trigfilt_f	AMS	MDE	TMS
30)	INST_f	ISA	ASB	AMS
31)	INST_filt_f	ISA	ASB	AMS
33)	ARB_mon_f	ASB	AMS	ISA
34)	ARB_burst_f	ASB	AMS	MDE
35)	ARB_ext_f	ASB	AMS	MDE
36)	ARB_freq_f	ASB	AMS	MDE
37)	ARB_ref_f	ASB	AMS	MDE
39)	POWER_mon_f	POWER SUPPLY	TH P/S CONT.	ISA
40)	PTEST_f	UUT CONTROLLER	AMS/MUX	SELFTTEST PB.