5 V-24 VDC Input Module

M/N 57C419

Instruction Manual J-3632-2

RELIANCE ELECTRIC

The information in this user's manual is subject to change without notice.

WARNING

THIS UNIT AND ITS ASSOCIATED EQUIPMENT MUST BE INSTALLED, ADJUSTED AND MAINTAINED BY QUALIFIED PERSONNEL WHO ARE FAMILIAR WITH THE CONSTRUCTION AND OPERATION OF ALL EQUIPMENT IN THE SYSTEM AND THE POTENTIAL HAZARDS INVOLVED. FAILURE TO OBSERVE THESE PRECAUTIONS COULD RESULT IN BODILY INJURY.

WARNING

INSERTING OR REMOVING THIS MODULE OR ITS CONNECTING CABLES MAY RESULT IN UNEXPECTED MACHINE MOTION. POWER TO THE MACHINE SHOULD BE TURNED OFF BEFORE INSERTING OR REMOVING THE MODULE OR ITS CONNECTING CABLES. FAILURE TO OBSERVE THESE PRECAUTIONS COULD RESULT IN BODILY INJURY.

CAUTION

THIS MODULE CONTAINS STATIC-SENSITIVE COMPONENTS. CARELESS HANDLING CAN CAUSE SEVERE DAMAGE.

DO NOT TOUCH THE CONNECTORS ON THE BACK OF THE MODULE. WHEN NOT IN USE, THE MODULE SHOULD BE STORED IN AN ANTI-STATIC BAG. THE PLASTIC COVER SHOULD NOT BE REMOVED. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN DAMAGE TO OR DESTRUCTION OF THE EQUIPMENT.

Table of Contents

Appendices

List of Figures

1.0 INTRODUCTION

The products described in this instruction manual are manufactured or distributed by Reliance Electric Company or its subsidiaries.

The 5V-24V Input Module will accept up to a maximum of 32 low level D-C input signals. The input signals may range from 5 volts through 24 volts. The inputs may be either NPN open collector outputs (sinking) or contact closures. Both configurations require an external power supply. Four of the inputs can be programmed to latch pulses of .7 msec duration or longer. These four inputs can also be used to generate an interrupt. Input signals have 5000 volt isolation to logic common. The module contains 8 isolated commons, one for each group of four inputs.

Typically, this module is used to input on/off signals from devices such as thumbwheel switches, relay contacts, limit switches, push-buttons, selector switches, open collector TTL, buffered LSTTL, and buffered high speed CMOS.

This manual describes the functions and specifications of the module. It also includes a detailed overview of installation and servicing procedures, as well as examples of programming methods.

Related publications that may be of interest:

- J-3600 DCS 5000 ENHANCED BASIC LANGUAGE INSTRUCTION MANUAL
- J-3601 DCS 5000 CONTROL BLOCK LANGUAGE INSTRUCTION MANUAL
- J-3602 DCS 5000 LADDER LOGIC LANGUAGE INSTRUCTION MANUAL
- J-3630 AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL VERSION 1.0
- J-3649 AutoMax CONFIGURATION TASK MANUAL
- J-3650 AutoMax PROCESSOR MODULE INSTRUCTION MANUAL
- J-3675 AutoMax ENHANCED BASIC LANGUAGE INSTRUCTION MANUAL
- J-3676 AutoMax CONTROL BLOCK LANGUAGE INSTRUCTION MANUAL
- J-3677 AutoMax LADDER LOGIC LANGUAGE INSTRUCTION MANUAL
- J-3684 AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL VERSION 2.0
- J-3750 ReSource AutoMax PROGRAMMING EXECUTIVE INSTRUCTION MANUAL VERSION 3.0
- IEEE 518 GUIDE FOR THE INSTALLATION OF ELECTRICAL EQUIPMENT TO MINIMIZE ELECTRICAL NOISE INPUTS TO CONTROLLERS FROM EXTERNAL SOURCES

2.0 MECHANICAL/ELECTRICAL DESCRIPTION

The following is a description of the faceplate LEDs, field termination connectors, and electrical characteristics of the field connections.

2.1 Mechanical Description

The input module is a printed circuit board assembly that plugs into the backplane of the DCS 5000/AutoMax rack. It consists of a printed circuit board, a faceplate, and a protective enclosure. The faceplate contains tabs at the top and bottom to simplify removing the module from the rack. Module dimensions are listed in Appendix A.

The faceplate of the module contains a female connector socket and 32 LED indicators that show the status of the inputs. Input signals are brought into the module via a multi-conductor cable (M/N 57C375; see Appendix D). One end of this cable attaches to the faceplate connector, while the other end of the cable has stake-on connectors that attach to two 20 point terminal strips for easy field wiring.

On the back of the module are two edge connectors that attach to the system backplane.

2.2 Electrical Description

The input module contains 32 input circuits for 5-24 volt logic signals. Each group of four circuits shares a single isolated common. Input signals have 5000 volt isolation to logic common. Refer to the block diagram in Appendix B.

Each input circuit consists of a current limiting resistor, a zener diode to control the switching level, an optical isolator, and a filter to eliminate spurious signals. The input filter time constant is 0.7 msec for the latching inputs (bits B17-B20) and 3.3 msec for all other inputs. A circuit diagram is shown in figure 2.1.

Inputs connected to terminal strip connections B17, B18, B19, and B20 can also be programmed to generate interrupts on positive or negative transitions.

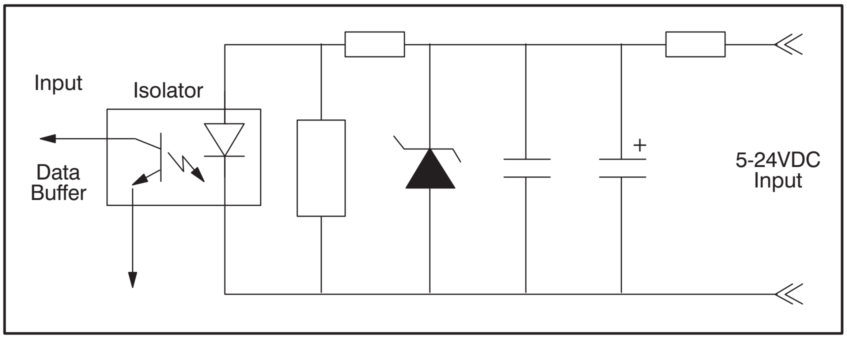


Figure 2.1 - Typical Input Circuit

There are 32 LED indicators on the faceplate of the module. The LED indicators display the status of the logic level circuitry. Proper operation of an LED indicates that both the input circuit and the logic level circuitry are operating correctly.

The LEDs are arranged as four groups of eight and are numbered from 0-31. See figure 2.2. LEDs numbered from 0-15 correspond to the inputs in register 0. LEDs numbered from 16-31 correspond to the inputs in register 1.

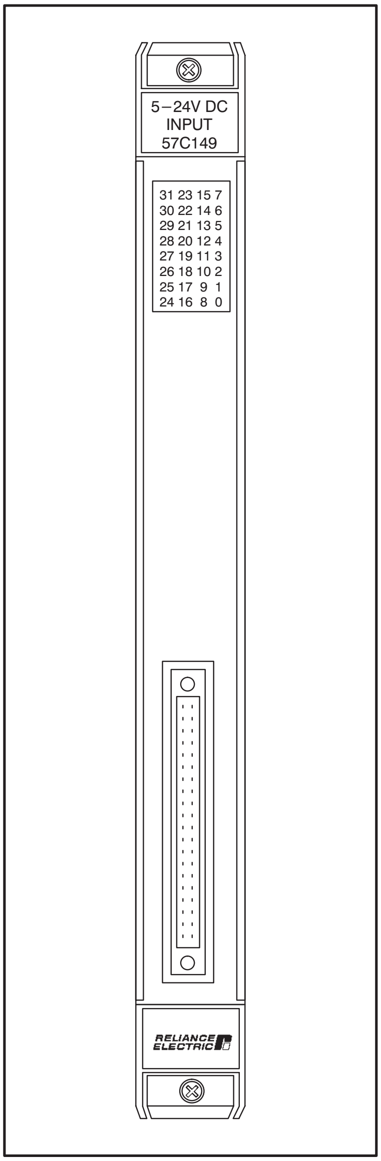


Figure 2.2 - Module Faceplate

3.0 INSTALLATION

This section describes how to install and remove the module and its cable assembly.

3.1 Wiring

The installation of wiring should conform to all applicable codes.

To reduce the possibility of electrical noise interfering with the proper operation of the control system, exercise care when installing the wiring from the system to the external devices. For detailed recommendations refer to IEEE 518.

3.2 Initial Installation

Use the following procedure to install the module:

Step 1. Turn off power to the system. All power to the rack as well as all power to the wiring leading to the module should be off.

Step 2. Mount the terminal strips (M/N 57C375) on a panel. The terminal strips should be mounted to permit easy access to the screw terminals on the terminal strips. Make certain that the terminal strips are close enough to the rack so that the cable will reach between the terminal strips and the module.

Step 3. Fasten field wires to the terminal strips. Typical field connections are shown in figures 3.1, 3.2, and 3.3.

Refer to Appendix C for the arrangement of terminal board connections. Make certain that all field wires are securely fastened.

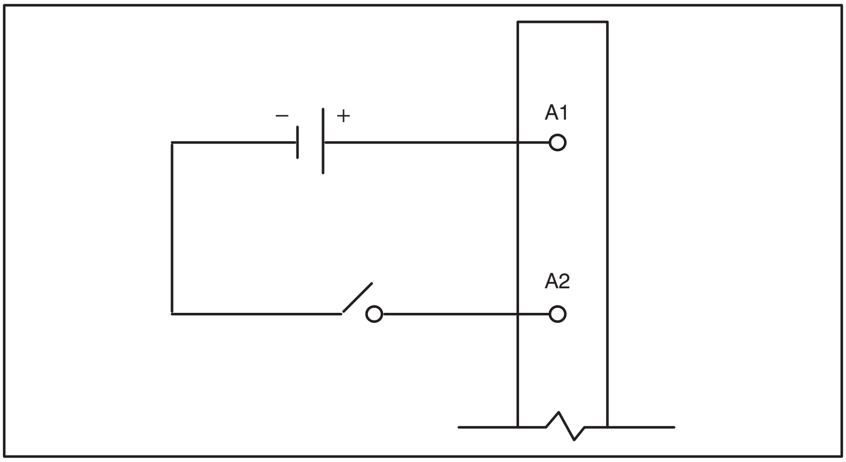


Figure 3.1 - Typical Field Connections for Contact Closures

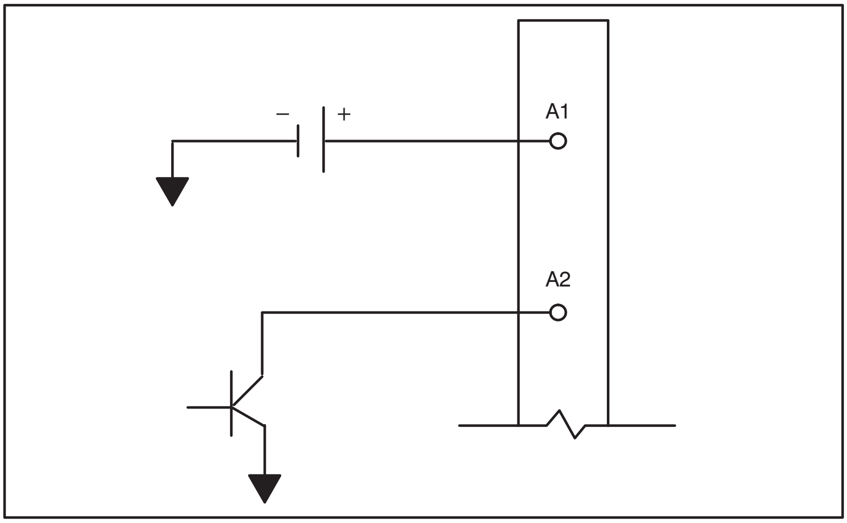


Figure 3.2 - Typical Field Connections for Open Collector TTL

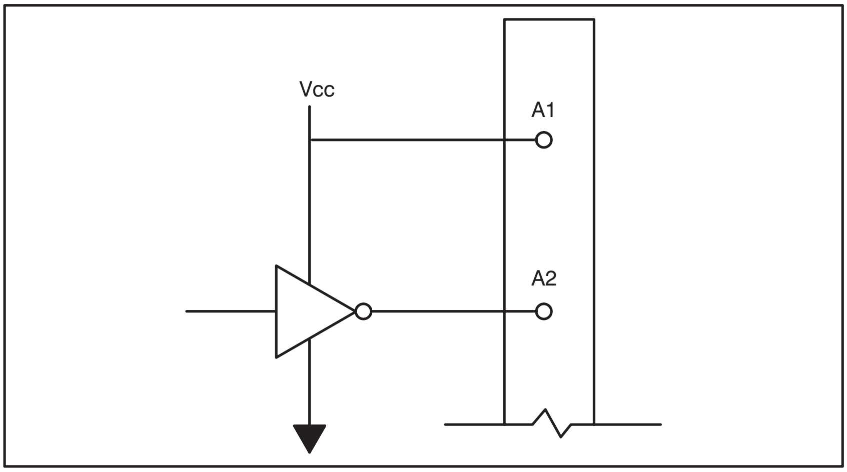


Figure 3.3 - Typical Field Connections for LSTTL or CMOS Buffer

Step 4. Take the module out of its shipping container. Take it out of the anti-static bag, being careful not to touch the connectors on the back of the module.

Step 5. Insert the module into the desired slot in the rack. Refer to figure 3.4. Use a screwdriver to secure the module into the slot.

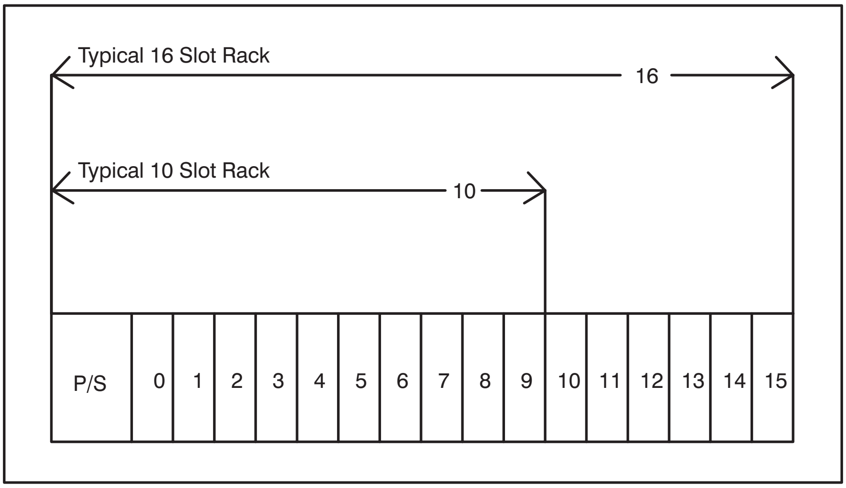


Figure 3.4 - Rack Slot Numbers

Step 6. Attach the field terminal connector (M/N 57C375) to the mating half on the module. Make certain that the connector is the proper one for this module. Use a screwdriver to secure the connector to the module.

Step 7. Turn on power to the system.

Step 8. Verify the installation by connecting the programming terminal to the system and running the ReSource Software.

Stop all programs that might be running.

Use the I/O MONITOR function. If the module is in the local rack, enter the input module slot number and register (0-1).

If the module is in a remote rack, enter the slot number of the master remote I/O module, remote I/O drop number (also called the remote rack number), input module slot number, and register (0-1).

One at a time, toggle each of the input devices connected to the input module to verify that the installation has been completed correctly.

3.3 Module Replacement

Use the following procedure to replace a module:

Step 1. Turn off power to the rack and all connections.

Step 2. Use a screwdriver to loosen the screws holding the connector to the module. Remove the connector.

Step 3. Loosen the screws that hold the module in the rack. Remove the module from the slot in the rack.

Step 4. Place the module in the anti-static bag it came in, being careful not to touch the connectors on the back of the module. Place the module in the cardboard shipping container.

Step 5. Take the new module out of the anti-static bag, being careful not to touch the connectors on the back of the module.

Step 6. Insert the module into the desired slot in the rack. Use a screwdriver to secure the module into the slot.

Step 7. Attach the field terminal connector (M/N 57C375) to the mating half on the module. Make certain that the connector keys are in the proper position for this module. Use a screwdriver to secure the connector to the module.

Step 8. Turn on power to the rack.

4.0 PROGRAMMING

This section describes how data is organized in the module and provides examples of how the module is accessed by the application software. For more detailed information, refer to the individual programming language manuals listed in section 1.0.

4.1 Register Organization

The data in the module is organized as four 16 bit registers. The first two registers (0 and 1) contain the current state (on or off) of the input data. The software allows you to define the module as a single long register of 32 bits, two separate registers of 16 bits each, or as 32 separate bits. These two registers are read only. Refer to figure 4.1.

There are also two registers (2 and 3) used to control the four latching inputs, which can be programmed as interrupts. The information in these registers is typically referenced as individual bits. Refer to figure 4.1.

Registers 0 and 1 - Input Data Status

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
register 0	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
register 1	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Register 2 - Interrupt and Status Control

register 2	–	–	–	–	r	r	r	r	–	rw	rw	rw	rw	–	–	–

Bit 11 - Latch status, B20	Bit 6 - Interrupt enable, B17
Bit 10 - Latch status, B19	Bit 5 - Interrupt enable, B18
Bit 9 - Latch status, B18	Bit 4 - Interrupt enable, B19
Bit 8 - Latch status, B17	Bit 3 - Interrupt enable, B20

Register 3 - Latch Reset and Edge Definition

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
register 3	–	–	–	–	rw	rw	rw	rw	–	–	–	–	rw	rw	rw	rw

Bit 11 - Latch reset, B20 *1	Bit 3 - Latch edge, B20 0 - off to on 1 - on to off
Bit 10 - Latch reset, B19 *1	Bit 2 - Latch edge, B19
Bit 9 - Latch reset, B18 *1	Bit 1 - Latch edge, B18
Bit 8 - Latch reset, B17 *1	Bit 0 - Latch edge, B17

*1 0, 1, 2, 7, and 15 of register 2 are controlled by the operating system and must not be defined or set by the user.

A latch must be acknowledged by writing a “0” to the proper bit after the latch has occurred.

B17-B20 refer to inputs. See Appendix C for more information on input connections.

Figure 4.1 - Organization of Register Bits

4.2 Configuration

Before any application programs can be written, it is necessary to configure the definitions of system-wide variables, i.e. those that must be globally accessible to all tasks.

For DCS 5000 and AutoMax Version 2.1 and earlier, you define system-wide variables by writing a Configuration task. For AutoMax Version 3.0 and later, you define system-wide variables using the AutoMax Programming Executive. After these variables are defined, you can generate the configuration file automatically, which eliminates the requirement to write a configuration task for the rack. If you are using AutoMax Version 2.1 or earlier, refer to Appendix E for examples that show how to define variables in the configuration task.

If you are using AutoMax Version 3.0 or later, see the AutoMax Programming Executive (J-3750) for information about configuring variables.

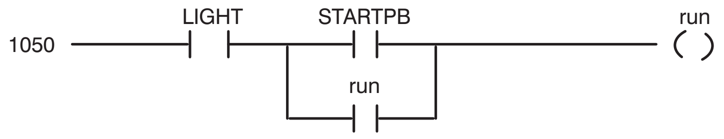
4.3 Reading And Writing Data In Application Tasks

In order for an input module to be referenced by application software, it is necessary to assign symbolic names to the physical hardware. In AutoMax Version 2.1 and earlier, this is accomplished by either IODEF or RIODEF statements in the configuration task. In AutoMax Version 3.0 and later, you assign variable name using the Programming Executive.

Each application program that references the symbolic names assigned to the module must declare those names COMMON.

The frequency with which tasks read their inputs and write their outputs depends on the language being used. Ladder logic and Control Block tasks read inputs once at the beginning of each scan and write outputs once at the end of scan. BASIC statements in BASIC tasks or Control Block tasks read an input and write an output for each reference throughout the scan.

4.3.1 Ladder Logic Task Example



The symbolic names LIGHT and STARTPB reference the input modules that were defined in the rack configuration. The trailing at symbol “@” is not used in Ladder Logic tasks. The symbolic name “run” is local to the Ladder Logic task and does not have I/O associated with it.

4.3.2 BASIC Task Example

```
1000    COMMON LIGHT@                  \!Fault Light
1010    COMMON STARTPB@                \!Start Push-button
2000    LOCAL RUN@                     \!Line run
3000    !
4000    !
5000    RUN@ = NOT LIGHT@ AND   &
          (  STARTPB@ OR RUN@)
5500    !
6000    END
```

The symbolic names LIGHT@ and STARTPB@ reference the input modules that were defined in the rack configuration. The symbolic name RUN@ is local to the BASIC task and does not have I/O associated with it.

4.3.3 Control Block Task Example

```
2400    COMMON  STARTPB@              \!Start Push-button
2500    LOCAL MOMENTARY@              \!Momentary output
4000    !
5000    CALL TRANSITION (  INPUT=STARTPB@, &
           OUTPUT=MOMENTARY@)
5500    !
6000    END
```

The symbolic name STARTPB@ references the input module that was defined in the rack configuration. The symbolic name MOMENTARY@ is local to the control block task and does not have I/O associated with it.

4.4 Using Interrupts in Application Tasks

Interrupts are used to synchronize software tasks with the occurrence of a hardware event. The input module has four inputs that can be programmed to generate an interrupt. The module allows you to synchronize real-world events with application tasks to a minimum of 1.2 msec., depending on the priority level of the task receiving the interrupt.

In order to use interrupts on the input module, it is necessary to assign symbolic names to the interrupt control bits and the interrupt status and control register (2). In AutoMax Version 2.1 and earlier, this is accomplished with IODEF statements in the configuration task. See Appendix E for an example. In AutoMax Version 3.0 and later, symbolic names are assigned using the Programming Executive. Note that interrupts cannot be used with input modules located in remote racks.

Only one task may act as a receiver for the interrupt generated by an input module. That task should declare the symbolic names assigned to the interrupt control register and bits on the input module as COMMON.

4.4.1 BASIC Task Example

The following is an example of a BASIC task that handles interrupts from inputs B17, B18 and B20.

Note that the 'timeout' parameter in the EVENT statement is disabled since interrupts from this module do not generally occur on a timed basis. The same consideration requires care in using time-based statements in loops used to read inputs from the module.

```
1000   COMMON ISCR%                       \!Interrupt Status/Control Register
1001   COMMON LATCH_EDGE_B17@             \!Latch edge, input B17
1002   COMMON LATCH_EDGE_B18@             \!Latch edge, input B18
1003   COMMON LATCH_EDGE_B20@             \!Latch edge, input B20
1005   !
1010   COMMON LATCH_STATUS_B17@           \!Latch status, input B17
1011   COMMON LATCH_STATUS_B18@           \!Latch status, input B18
1012   COMMON LATCH_STATUS_B20@           \!Latch status, input B20
1015   !
1020   COMMON INTRP_ENABLE_B17@           \!Interrupt enable, input B17
1021   COMMON INTRP_ENABLE_B18@           \!Interrupt enable, input B18
1022   COMMON INTRP_ENABLE_B20@           \!Interrupt enable, input B20
1025   !
1030   COMMON LATCH_RESET_B17@            \!Latch reset, input B17
1031   COMMON LATCH_RESET_B18@            \!Latch reset, input B18
1032   COMMON LATCH_RESET_B20@            \!Latch reset, input B20
1050   !
1060   LOCAL B17_CNT%                     \!Interrupt counter, input B17
1070   LOCAL B18_CNT%                     \!Interrupt counter, input B18
1080   LOCAL B20_CNT%                     \!Interrupt counter, input B20
2000   !
2001   !    Define the edge transition that will generate an interrupt
2002   !
2010   LATCH_EDGE_B17@ = FALSE            \!Off to on
2011   LATCH_EDGE_B18@ = FALSE            \!Off to on
2012   LATCH_EDGE_B20@ = TRUE             \!On to off
3000   !
3001   !    The following statement connects the name HW_EVENT to the
3002   !    Interrupt defined in ISCR%. The event name chosen should
```

```
3003    !    be as meaningful as possible. The watchdog timeout has been
3004    !    disabled because the interrupt is not periodic.
3005    !
3006    !
3007    !
3010    EVENT A NAME=HW_EVENT, INTERRUPT_STATUS=ISCR%,&
3011    TIMEOUT=DISABLED
4000    !
4001    !    The following statements reset the latch status bits
4002    !
4010    LATCH_ACK_B17@=FALSE
4011    LATCH_ACK_B18@=FALSE
4012    LATCH_ACK_B20@=FALSE
5000    !
5001    !    The following statements enable interrupts
5002    !
5010    INTRP_ENABLE_B17@=TRUE
5011    INTRP_ENABLE_B18@=TRUE
5012    INTRP_ENABLE_B20@=TRUE
5500    !
5600    !    complete the remainder of task initialization here
5700    !
6000    !
6001    !    The next statement synchronizes the task with the external
6002    !    event via the interrupt. Task execution will be suspended
6003    !    until the interrupt occurs. When the interrupt occurs, if this
6004    !    task is the highest priority task waiting to execute, it will
6005    !    become active. If it is not the highest priority task, it will
6006    !    remain suspended until all higher priority tasks have executed,
6007    !    at which point it will become active.
6008    !
6010    WAIT ON HW_EVENT
7000    !
7001    !    The next statements determine which bit generated the interrupt
7002    !    by examining the latch status bit. If a latch is
7003    !    found, it is reset. The interrupt service routine is
7004    !    then executed.
7005    !
7010    IF NOT LATCH_STATUS_B17@ THEN GO TO 8010
7020    LATCH_RESET_B17@ = FALSE
7030    B17_CNT% = B17_CNT% + 1              \!Interrupt service routine for B17
7040    !
8000    !    Test B18 interrupt
8005    !
8010    IF NOT LATCH_STATUS_B18@ THEN GO TO 9010
8015    LATCH_RESET_B18@ = FALSE
8020    B18_CNT% = B18_CNT% + !              \!Interrupt service routine for B18
8050    !
9000    !    Test B20 interrupt
9005    !
9010    IF NOT LATCH_STATUS_B20@ THEN GO TO 10010
9015    LATCH_RESET_B20@ = FALSE
9020    B20_CNT% = B20_CNT% +                \!Interrupt service routine for B20
10020   !
10010   GO TO 6010
15000   !
18000   !
20000   END
```

4.4.2 Control Block Task Example

The following is an example of a Control Block task that handles an interrupt from input B19.

Note that the ‘timeout’ parameter in the BASIC statement EVENT is usually disabled since interrupts from this module do not generally ocur on a timed basis. The same consideration requires care in using time-based statements in loops used to read inputs from the module. The ‘timeout’ parameter is not disabled in the following example because the interrupt is expected to happen in a specified time period.

```
1000   COMMON ISCR%                          \!Interrupt Status/Control Register
1010   COMMON LATCH_EDGE_B19@                \!Latch edge, input B19
1020   COMMON LATCH_STATUS_B19@              \!Latch status, input B19
1030   COMMON LATCH_ENABLE_B19@              \!Interrupt enable, input B19
1040   COMMON LATCH_RESET_B19@               \!Latch reset, input B19
1050   LOCAL B19_CNT%                        \!Count of interrupts
2000   !
2001   !    Define the edge transition that will generate an interrupt.
2002   !
2010   LATCH_EDGE_B19@ = FALSE               \!Off to on
3000   !
3001   !    The following statement connects the name HW_EVENT to the
3002   !    Interrupt defined in ISCR%. The event name should be
3003   !    as meaningful as possible. The watchdog timeout has been
3004   !    set to 1800 clock ticks (9.9 sec.). 1 tick equals
3005   !    .0055 seconds. If the time between
3006   !    interrupts exceeds this value, a severe error will be declared
3007   !    and the system will be stopped.
3008   !
3009   !
3010   EVENT NAME=HW_EVENT, LATCH_STATUS=ISCR%, &   TIMEOUT=1800
4000   !
4001   !    The following statement resets the latch status bit
4002   !
4010   LATCH_RESET_B19@ = FALSE
5000   !
5001   !    The following statement enables interrupts
5002   !
5010   INTRP_ENABLE_B19@ = TRUE
5200   !
5300   !    complete the remainder of task initilization
5500   !
6000   !
6001   !    The next statement synchronizes the task with the external
6002   !    event via the interrupt. Task execution will be suspended
6003   !    until the interrupt occurs. When the interrupt occurs, if this
6004   !    task is the highest priority task waiting to execute, it will
6005   !    become active. If it is not the highest priority task, it will
6006   !    remain suspended until all higher priority tasks have executed.
6007   !    at which point it will become active.
6008   !
6010   CALL SCAN_LOOP(  TICKS=1200, EVENT=HW_EVENT)
7000   !
7001   !    The next statements determine which bit generated the interrupt
7002   !    by examining the latch status bit. If an interrupt is
7003   !    found, it is reset. The interrupt service routine is
7004   !    then executed.
7005   !
7500   !
```

```
7800    !
8000    LATCH_RESET_B19@ = FALSE
8010    B19_CNT% = B19_CNT% + 1                  \!Interrupt service routine B19
9000    !
10000   END
```

4.5 Restrictions

This section describes limitations and restrictions on the use of this module.

4.5.1 Writing Data to Registers

The input registers (0 and 1) on this module are read only. Attempts to write to them will cause a bus error (severe system error). The following are examples from programs that write to the module and should therefore be avoided:

a. Referencing an input from the coil in a Ladder Logic task.

b. Referencing the module on the left side of an equal sign in a LET statement in a Control Block or BASIC task.

c. Referencing an input as an output in a Control Block function.

4.5.2 32 Bit Register Reference

WARNING

IF YOU USE DOUBLE INTEGER VARIABLES IN THIS INSTANCE, YOU MUST IMPLEMENT A SOFTWARE HANDSHAKE TO ENSURE THAT BOTH THE LEAST SIGNIFICANT AND MOST SIGNIFICANT 16 BITS HAVE BEEN TRANSMITTED BEFORE THEY ARE READ BY THE RECEIVING APPLICATION PROGRAM. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

32 bit register references should be used with caution when this module is placed in a remote rack. The remote I/O system does not always transfer registers greater than 16 bits as a unit. As a result, it is possible for an application program to read the least significant 16 bits of a new value and the most significant 16 bits of the previous value as a 32 bit register reference.

4.5.3 Interrupts in Remote I/O Racks

When this module is in a remote rack, the interrupt mode cannot be used.

4.5.4 Dynamic Modification of Latch Edge Transition

Whenever the edge transition bit is toggled, the corresponding latch status bit will be reset, regardless of whether an interrupt was pending. For this reason, edge transition bits should not be modified by the user after the module has been initialized.

5.0 DIAGNOSTICS AND TROUBLESHOOTING

This section explains how to troubleshoot the module and field connections.

5.1 Incorrect Data

Problem: The data is either always off, always on, or different than expected. The possible causes of this are a module in the wrong slot, a programming error, or a malfunctioning module. It is also possible that the input is either not wired or wired to the wrong device. Use the following procedure to isolate the problem:

Step 1. Verify that the input module is in the correct slot and that the I/O definitions are correct.

Refer to figure 3.4. Verify that the slot number being referenced agrees with the slot number defined in the configuration. Verify that the register number and bit number are correct.

For remote I/O installations, also verify that the master slot and drop number are defined correctly.

Step 2. Verify that the input is wired to the correct device.

Confirm that all connections at the terminal strip are tight. Connect a voltmeter to the proper points on the terminal strip and toggle the device. The voltmeter should alternate between 0 and the D-C power supply voltage (5-24 volts). If this does not happen, there is a problem with either the external device, the D-C power supply, or the wiring to the terminal strip.

Check the cable for continuity between the faceplate connector and the terminal strip.

Step 3. Verify that the input circuit on the module is working correctly.

Toggle the input device. Verify that the LED associated with the particular bit is also toggling. If it is not, the input module is malfunctioning.

Step 4. Verify that the module can be accessed.

Connect the programming terminal to the system and run the ReSource Software. Use the I/O MONITOR function. Toggle the input device to determine whether the bit is changing state.

If the I/O MONITOR is able to read the input, the problem lies in the application software (proceed to step 5). If the I/O MONITOR cannot read the inputs, the problem lies in the hardware (proceed to step 6).

Step 5. Verify that the user application program is correct.

Verify that the application program that references the symbolic names assigned to the module has declared those names COMMON in application tasks.

Verify that the symbolic name in question is being referenced in the application program. This can be done indirectly by monitoring the name with the VARIABLE MONITOR function in the ReSource Software.

If operation is intermittent, verify that the task reading the module is executing fast enough to catch all of the input changes.

Step 6. Verify that the hardware is working correctly.

Verify the hardware functionality by systematically swapping out modules. After each swap, if the problem is not corrected, replace the original module before swapping out the next module.

- To test local I/O, first replace the input module. Next, replace the Processor module (s). If the problem persists, take all of the modules except one Processor module and the input module out of the backplane. If the problem is now corrected, one of the other modules in the rack is malfunctioning. Replace the other modules one at a time until the problem reappears. If none of these tests reveals the problem, replace the backplane.
- To test remote I/O, first verify that the remote I/O system is communicating with the drop that contains the input module being tested. Next, determine whether the input module is the only module that is not working. If more than one module is not working correctly, the problem most likely lies in the remote I/O system. If the problem does not lie in the system, it probably involves the remote rack.
- To test the remote rack, connect a dumb terminal or a personal computer running terminal emulation software such as Kermit to the slave remote I/O module RS-232C port. Set the port parameters on the terminal or computer to 8 bits, 1 stop bit, no parity and a baud rate of 1200. Connect the remote I/O module (C<CR> with Kermit). See the Remote I/O instruction manual (J-3606) for how to test the module.

  If you cannot determine the problem, replace the input module. Next, replace the slave remote I/O module. If the problem persists, take all of the modules out of the remote backplane except the slave remote I/O module and the input module. If the problem is now corrected, one of the other modules in the rack is malfunctioning. Reconnect the other modules one at a time until the problem reappears. If the problem proves to be neither in the remote I/O system nor in the remote rack, try replacing the backplane.

5.2 Bus Error

Problem: A “31” or “51” through “58” appears on the Processor module’s LED. This error message indicates that there was a bus error when the system attempted to access the module. The possible causes of this error are a missing module, a module in the wrong slot, or a malfunctioning module. It is also possible that the user is attempting to write to the wrong registers on the module. Use the following procedure to isolate a bus error:

Step 1. Verify that the input module is in the correct slot and that the I/O definitions are correct.

Refer to figure 3.4. Verify that the slot number being referenced agrees with the slot number defined in the configuration task. Verify that the register number and bit number are correct.

For remote I/O installations, also verify that the master slot and drop number are defined correctly.

Step 2. Verify that the module can be accessed.

Connect the programming terminal to the system and run the ReSource Software. Use the I/O MONITOR to monitor the four registers on the input module. If the I/O MONITOR is able to monitor the inputs, the problem lies in the application software (proceed to step 3). If the I/O monitor cannot monitor the inputs, the problem lies in the hardware (proceed to step 4).

Step 3. Verify that the user application program is correct.

Registers 0 and 1 of the input module cannot be written to. If a BASIC task caused the bus error, the error log will contain the statement number in the task where the error occurred. If a Ladder Logic or Control Block task caused the error, you will need to search the task for any instances where you used an input as a ladder logic coil or wrote to it in a Control Block task.

Step 4. Verify that the hardware is working correctly.

Verify the hardware functionality by systematically swapping out the input module, the Processor module(s) and the backplane. After each swap, if the problem is not corrected, replace the original item before swapping out the next item.

To test the remote rack, connect a dumb terminal or a personal computer running terminal emulation software such as Kermit to the slave remote I/O module RS-232C port. Set the port parameters on the terminal or computer to 8 bits, 1 stop bit, no parity and a baud rate of 1200. Connect the remote I/O module (C<CR> with Kermit). See the Remote I/O instruction manual J-3606 for how to test the module.

If you cannot determine the problem, replace the input module. Next, replace the slave remote I/O module. If the problem persists, take all of the modules out of the remote backplane except the slave remote I/O module and the input module. If the problem is now corrected, one of the other modules in the rack is malfunctioning. Reconnect the other modules one at a time until the problem reappears. If the problem proves to be neither in the remote I/O system nor in the remote rack, try replacing the backplane.

5.3 Interrupt Problems

Problem: No interrupts at all, or too many (unexpected) interrupts, signified by error codes being displayed on the faceplate of the Processor module. Go through the following steps first before going on the the more specific troubleshooting steps:

Step 1. Verify that the input module is in the correct slot.

Refer to figure 3.4.

Step 2. Verify that the I/O definitions are correct.

Verfiy that the configuration task contains the proper interrupt control definitions. Refer to the example in section 4.6.

Step 3. Verify that the user application program is correct.

Verify that the application program that uses the symbolic names defined in the configuration task has defined those names as COMMON.

Compare your interrupt task with the examples given in sections 4.4.1 and 4.4.2. Make sure that the actions shown in the examples are performed in the same order in your task.

5.3.1 No Interrupts

Problem: The task does not execute but no error codes are displayed on the Processor module faceplate. If interrupts are never received and the “timeout” parameter in the event definition was disabled, the task will never execute. Use the following procedure to isolate the problem:

Step 1. Verify that the user application program is correct.

Verify that your interrupt response task is checking the proper latch status bit to determine which bit caused the interrupt. Confirm that when an interrupt has been located, the latch reset bit is being reset.

Compare your interrupt task with the examples given in sections 4.4.1 and 4.4.2. Make sure that the actions shown in the examples are performed in the same order in your task.

Step 2. Verify that the input is wired to the correct device.

Confirm that all connections at the terminal strip are tight. Connect a voltmeter to the proper points on the terminal strip and toggle the device. The volt- meter should alternate between 0 and the D-C power supply voltage (5-24 volts). If this does not happen, there is a problem with either the external device, the D-C power supply, or the wiring to the terminal strip.

If the device generates a pulse output, use a scope and verify that the pulse width is at least .7 msec.

Step 3. Verify that the input circuit on the module is working correctly.

Toggle the input device. Verify that the LED associated with the particular bit is also toggling. If it is not, the input circuit on the module is malfunctioning.

Step 4. Verify that the module can be accessed.

Connect the programmer to the system and run the ReSource Software. Use the I/O MONITOR function to display register 1 and 2. Toggle the input device to determine whether the bit is changing state. If it is not, the input circuit on the module is malfunctioning.

Step 5. Verify that the hardware is working correctly.

Systematically swap out the input module, the processor module(s), and the backplane. After each swap, if the problem has not been corrected, replace the original item before swapping out the next item.

5.3.2 Hardware Event Time-Out

Problem: All tasks in the chassis are stopped and error code “12” appears on the faceplate of the Processor module. The interrupt has either never occurred or is occurring at a slower frequency than the value specified in the “timeout” parameter in the event definition. Use the following procedure to isolate the problem:

Step 1. Verify that the timeout value is set correctly.

Check the value specified, if any, in the “timeout” parameter in the event definition. The unit is ticks. Each tick is equal to 5.5 msec. The timeout value should be at least 2 ticks greater than the interrupt frequency. It can reasonably range up to 1.5 times the interrupt frequency. Note that the ‘timeout’ parameter is usually disabled since the interrupt doesn’t usually happen on a timed basis.

Step 2. Check for no interrupt.

Refer to section 5.3.1.

5.3.3 Hardware Event Count Limit Exceeded

Problem: All tasks in the chassis are stopped and error code “1B” appears on the faceplate of the Processor module. A hardware interrupt has occurred but no task is waiting. Use the following procedure to isolate the problem:

Step 1. Verify that the user application program is correct.

Verify that your interrupt response task contains either a “WAIT ON event” or “CALL SCAN_LOOP” statement that will be executed. Check carefully to determine whether a higher priority task is preventing the interrupt response task from running.

Make sure that the ordering of your statements agrees with the examples in section 4.4.

Step 2. Verify that the signal from the external device is clean.

Connect a scope to the input terminals and monitor the pulse waveform from the external device. The waveform should have a clean transition, i.e. no overshoot or undershoot.

Step 3. Verify that the hardware is working correctly.

Verify the hardware functionality by systematically swapping out modules. After each swap, if the problem is not corrected, replace the original module before swapping out the next module.

To test local I/O, first replace the input module. Next, replace the Processor module(s). If the problem persists, take all of the modules except one Processor module and the input module out of the backplane. If the problem is now corrected, one of the other modules in the rack is malfunctioning. Reconnect the other modules one at a time until the problem reappears. If none of these tests reveals the problem, replace the backplane.

5.3.4 Illegal Interrupt Detected

Problem: All tasks in the chassis are stopped and error code “1F” appears on the faceplate of the Processor module. A hardware interrupt has occurred but no event has been defined.

Step 1. Verify that the user application program is correct.

Verify that your interrupt response task contains an “EVENT“ statement to be executed. Check carefully to determine whether a higher priority task is preventing the interrupt response task from running. Make sure that the ordering of your statements agrees with the examples in section 4.4.

Step 2. Verify that the hardware is working correctly.

Verify the hardware functionality by systematically swapping out modules. After each swap, if the problem is not corrected, replace the original module before swapping out the next module.

To test local I/O, first replace the input module. Next, replace the Processor module(s). If the problem persists, take all of the modules except one Processor module and the input module out of the backplane. If the problem is now corrected, one of the other modules in the rack is malfunctioning. Reconnect the other modules one at a time until the problem reappears. If none of these tests reveals the problem, replace the backplane.

Appendix A

Technical Specifications

Ambient Conditions

- Storage temperature: −40ºC - 85ºC
- Operating temperature: 0ºC - 60ºC
- Humidity: 5-90% non-condensing

Maximum Module Power Dissipation

- 22.6 Watts

Dimensions

- Height: 11.75 inches
- Width: 1.25 inches
- Depth: 7.375 inches

System Power Requirements

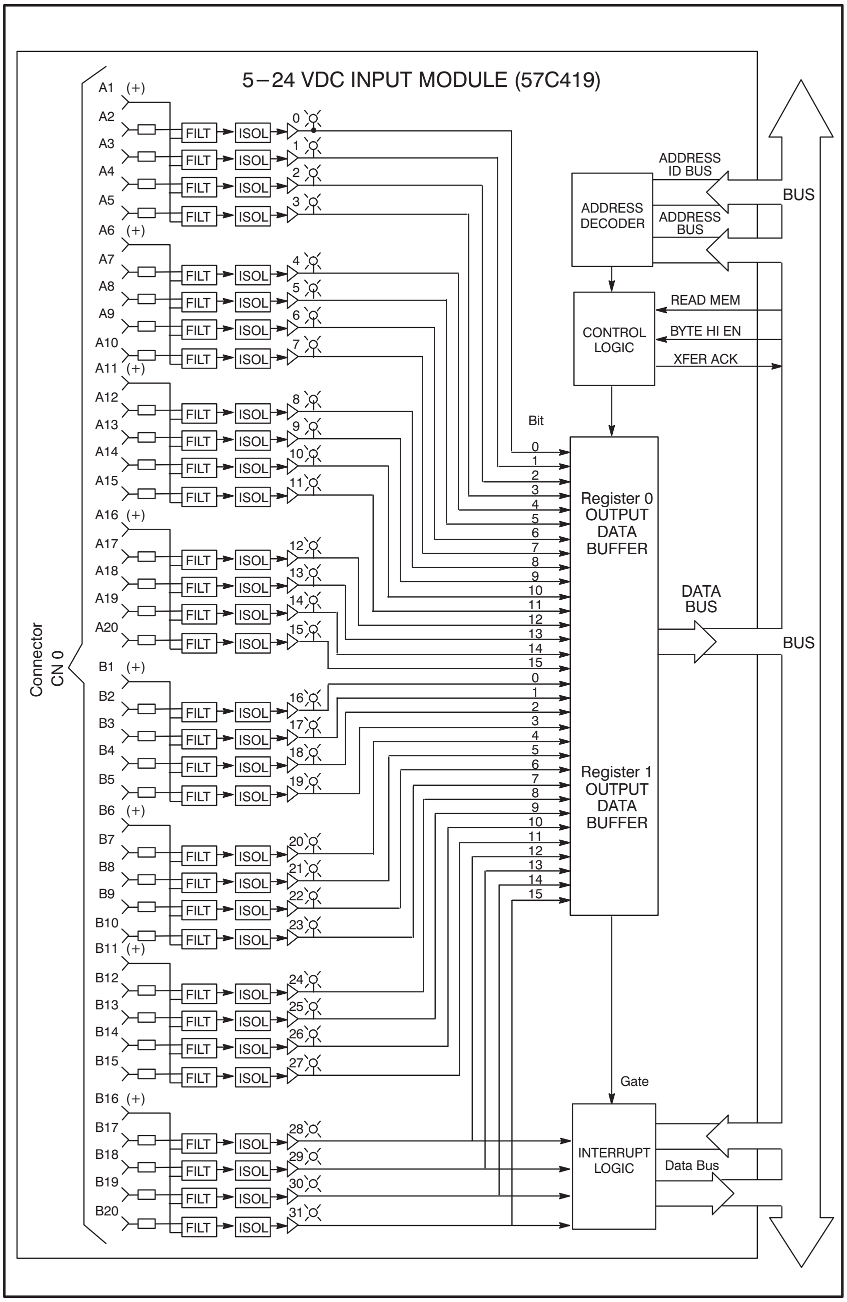
- +5 volts: 700 mA

Input Circuit

- Number of inputs: 32, 4 of which can be programmed to latch and interrupt
- Maximum turn-on time: 3.3 msec for normal inputs
  0.7 msec for latch inputs
- Maximum operating voltage: 24 volts D-C
- Minimum operating voltage: 2.7 volts D-C
- Maximum ON current: 4.95 mA at 5 volts D-C
  23.70 mA at 24 volts D-C
- Four inputs per isolated common
- 5000 volt isolation between logic common and input power

Appendix B

Module Block Diagram



Appendix C

Field Connections

Register 0				Register 1			
Conn Pin No.	Wire Color Code	Bit No.	LED No.	Conn Pin No.	Wire Bit Color Code	LED No.	No.
A1(+)	black			B1(+)	black/w.s.		
A2	white	0	0	B2	red/w.s.	0	16
A3	red	1	1	B3	d green/w.s.	1	17
A4	green	2	2	B4	yellow/w.s.	2	18
A5	yellow	3	3	B5	brown/w.s.	3	19
A6(+)	brown			B6(+)	blue/w.s.		
A7	blue	4	4	B7	purple/w.s.	4	20
A8	purple	5	5	B8	gray/w.s.	5	21
A9	gray	6	6	B9	pink/w.s.	6	22
A10	pink	7	7	B10	lt. green/w.s.	7	23
A11(+)	white/b.s.			B11(+)	white/r.s.		
A12	red/b.s.	8	8	B12	d green/r.s.	8	24
A13	d green/b.s.	9	9	B13	yellow/r.s.	9	25
A14	yellow/b.s.	10	10	B14	brown/r.s.	10	26
A15	brown/b.s.	11	11	B15	blue/r.s.	11	27
A16(+)	blue/b.s.			B16(+)	purple/r.s.		
A17	purple/b.s.	12	12	B17	gray/r.s.	12	28
A18	gray/b.s.	13	13	B18	pink/r.s.	13	29
A19	pink/b.s.	14	14	B19	lt. green/r.s.	14	30
A20	lt. green/b.s.	15	15	B20	orange/r.s.	15	31

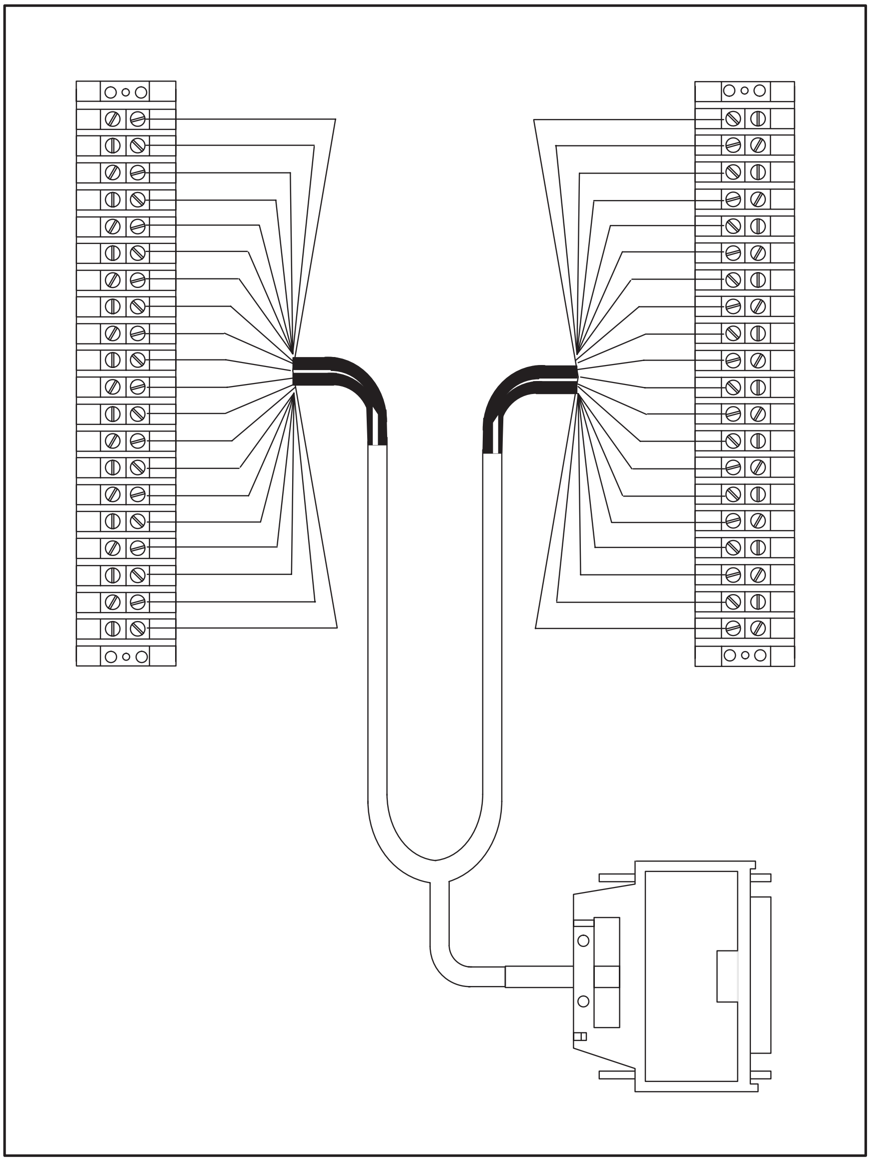
b.s. = black stripe   w.s. = white stripe   r.s. = red stripe

Appendix D

Related Components

57C375 – Terminal Strip/Cable Assembly

This assembly consists of two terminal strips, a cable, and a mating connector. It is used to connect field signals to the faceplate of the input module.

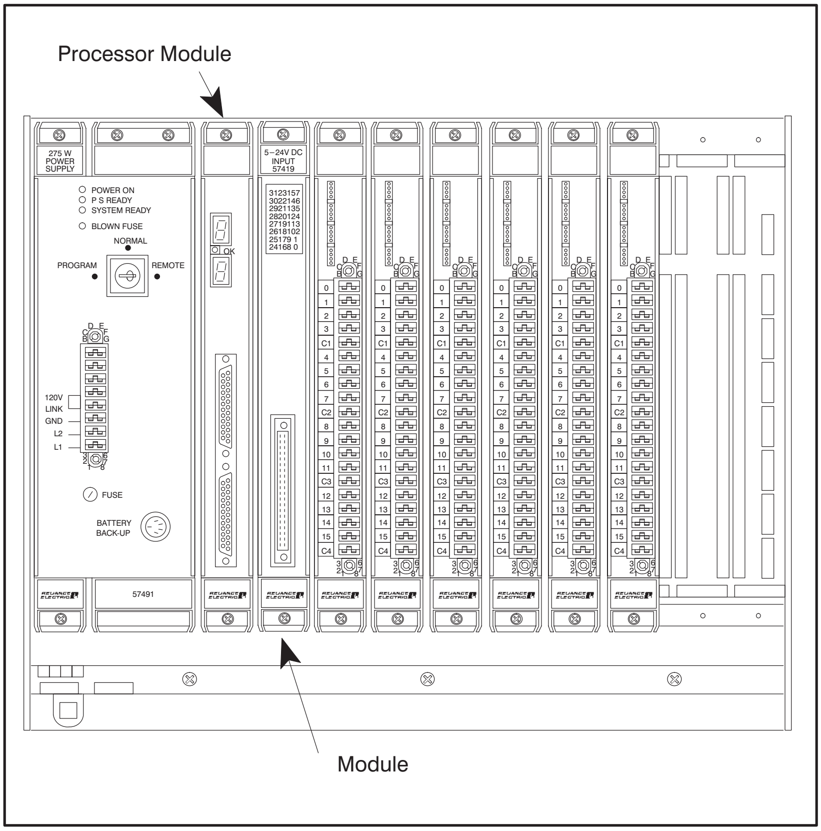


Appendix E

Defining Variables in the Configuration Task

Local I/O Definition

This section describes how to configure the input module when it is located in the same rack as the processor module that is referencing it. Refer to the figure below. Note that this procedure is used only if you are using the Programming Executive software version 2.1 or earlier.



Module in a Local Rack

32 Bit Register Reference

WARNING

IF YOU USE DOUBLE INTEGER VARIABLES IN THIS INSTANCE, YOU MUST IMPLEMENT A SOFTWARE HANDSHAKE TO ENSURE THAT BOTH THE LEAST SIGNIFICANT AND MOST SIGNIFICANT 16 BITS HAVE BEEN TRANSMITTED BEFORE THEY ARE READ BY THE RECEIVING APPLICATION PROGRAM. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

Use the following method to reference all 32 inputs as a single register. Only one statement is necessary. The symbolic name of each register should be as meaningul as possible:

nnnnn IODEF SYMBOLIC_NAME![SLOT=s, REGISTER=0]

When referenced as a long register of 32 bits, register 0 becomes the high order 16 bits.

16 Bit Register Reference

Use the following method to reference a 16 bit register as a single input. For this module, a maximum of four statements can be included in the configuration task (one for each register). The symbolic name of each register should be as meaningful as possible:

nnnnn IODEF SYMBOLIC_NAME%[SLOT=s, REGISTER=r]

Bit Reference

Use the following method to reference individual inputs on the module. For the entire module, a maximum of 48 statements can be included in the configuration task (one for each bit that can be read or written by the user). The symbolic name of each bit should be as meaningful as possible:

nnnnn IODEF SYMBOLIC_NAME@[SLOT=s, REGISTER=r, BIT=b]

where:

nnnnn - BASIC statement number. This number may range from 1-32767.

SYMBOLIC_NAME! - A symbolic name chosen by the user and ending with (!). This indicates a long integer data type and all references will access registers 0 and 1.

SYMBOLIC_NAME% - A symbolic name chosen by the user and ending with (%). This indicates an integer data type and all references will access register “r”.

SYMBOLIC_NAME@ - A symbolic name chosen by the user and ending with (@). This indicates a boolean data type and all references will access bit number “b” in register “r”.

s - Slot number that the module is plugged into. This number may range from 0-15.

r - Specifies the register that is being referenced. For long integers this number must be zero. For all other references this number may range from 0-3.

b - Used with boolean data types only. Specifies the bit in the register that is being referenced. This number may range from 0-15.

Examples of Local I/O Definitions

The following statement assigns the symbolic name WINDOW! to the input module located in slot 11:

```
1000  IODEF  WINDOW![ SLOT=11, REGISTER=0]
```

The following statement assigns the symbolic name POSITION% to register 0 of the input module located in slot 4:

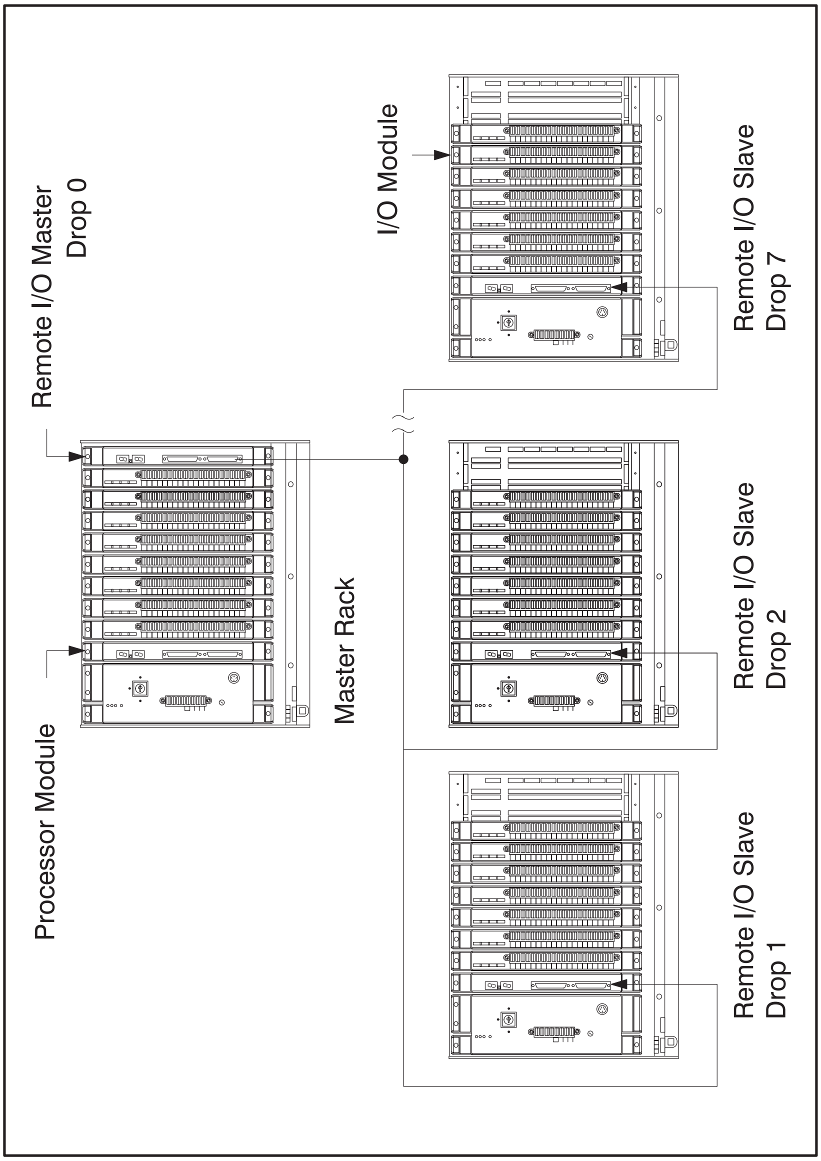
```
1020   IODEF   POSITION%[  SLOT=4, REGISTER=0]
```

The following statement assigns the symbolic name LIGHT@ to bit 9 of register 1 on the input module located in slot 7:

```
2050  IODEF  LIGHT@[  SLOT=7, REGISTER=1, BIT=9]
```

Remote I/O Definition

This section describes how to configure the module when it is located in a rack that is remote from the processor module that is referencing it. Refer to the figure below. Note that when this module is in a remote rack, the interrupt mode cannot be used.



Module in a Remote Rack

32 Bit Register Reference

WARNING

IF YOU USE DOUBLE INTEGER VARIABLES IN THIS INSTANCE, YOU MUST IMPLEMENT A SOFTWARE HANDSHAKE TO ENSURE THAT BOTH THE LEAST SIGNIFICANT AND MOST SIGNIFICANT 16 BITS HAVE BEEN TRANSMITTED BEFORE THEY ARE READ BY THE RECEIVING APPLICATION PROGRAM. FAILURE TO OBSERVE THIS PRECAUTION COULD RESULT IN BODILY INJURY OR DAMAGE TO EQUIPMENT.

Use the following method to reference all 32 inputs as a single register. Only one statement is necessary. The symbolic name of the register should be as meaningful as possible:

```
nnnnn  RIODEF SYMBOLIC_NAME![ MASTER_SLOT=m,
      DROP=d, SLOT=s, REGISTER=0]
```

When referenced as a long register of 32 bits, register 0 becomes the high order 16 bits. A 32 bit register reference over remote I/O should be used with care since the remote I/O system cannot guarantee that the entire 32 bit value will be moved as a single operation. For more information refer to the DCS 5000 Remote I/O Instruction Manual (J-3629).

16 Bit Register Reference

Use the following method to reference a 16 bit register as a single input. A maximum of four statements can be included in the configuration task (one for each register). The symbolic name of each register should be as meaningful as possible:

```
nnnnn  RIODEF SYMBOLIC_NAME%[  MASTER_SLOT=m,
    DROP=d, SLOT=s,  REGISTER=r]
```

Bit Reference

Use the following method to reference individual inputs on the module. For the entire module, a maximum of 48 statements can be included in the configuration task (one for each bit that can be read or written by the user). The symbolic name of each bit should be as meaningful as possible:

```
nnnnn  RIODEF SYMBOLIC_NAME@[  MASTER_SLOT=m,
      DROP=d, SLOT=s,  REGISTER=r, BIT=b]
```

where:

nnnnn - BASIC statement number. This number may range from 1-32767.

SYMBOLIC_NAME! - A symbolic name chosen by the user and ending with (!). This indicates a long integer data type and all references will access registers 0 and 1.

SYMBOLIC_NAME% - A symbolic name chosen by the user and ending with (%). This indicates an integer data type and all references will access register “r”.

SYMBOLIC_NAME@ - A symbolic name chosen by the user and ending with (@). This indicates a boolean data type and all references will access bit number “b” in register “r”.

m - Slot number of the master remote I/O module.

d - Drop number of the slave remote I/O module that is in the same rack as the input module. This number may range from 1-7.

s - Slot number that the module is plugged into. This number may range from 0-15.

r - Specifies the register that is being referenced. For long integers this number must be zero. For all other references this number may range from 0-3.

b - Used with boolean data types only. Specifies the bit in the register that is being referenced. This number may range from 0-15.

Examples of Remote I/O Definitions

The following statement assigns the symbolic name UPPER_LIMIT! to the input module located in slot 10 of remote I/O drop 7. This remote drop is connected to the remote I/O system whose master is located in slot 9 in the master rack:

```
1000  RIODEF  UPPER_LIMIT![ MASTER_SLOT=9, DROP=7,
      SLOT=10,REGISTER=0]
```

The following statement assigns the symbolic name LEVEL% to register 1 on the input module located in slot 4 of remote I/O drop 3. This remote drop is connected to the remote I/O system whose master is located in slot 15 in the master rack:

```
1020  RIODEF  LEVEL%[ MASTER_SLOT=15, DROP=3,
      SLOT=4,REGISTER=1]
```

The following statement assigns the symbolic name STARTPB@ to register 0 bit 9 on the input module located in slot 7 of remote I/O drop 2. This remote drop is connected to the remote I/O system whose master is located in slot 6 in the master rack:

```
2050  RIODEF  STARTPB@[ MASTER_SLOT=6, DROP=2,
      SLOT=7,REGISTER=0, BIT=9]
```

Sample Configuration Task Defining Interrupts

The following is an example of a configuration task for an input module defining interrupts:

```
1000   !
1001   !    Interrupt status and control register (used by
1002   !    the operating system)
1005   IODEF ISCR%[ SLOT=4, REGISTER=2]
1010   !
1011   !    Interrupt enables (one per bit) 0=disable, 1−enable
1012   !
1015   IODEF INTRP_ENABLE_B17@[SLOT=4, REGISTER=2, BIT=6]
1016   IODEF INTRP_ENABLE_B18@[SLOT=4, REGISTER=2, BIT=5]
1017   IODEF INTRP_ENABLE_B19@[SLOT=4, REGISTER=2, BIT=4]
1018   IODEF INTRP_ENABLE_B20@[SLOT=4, REGISTER=2, BIT=3]
1020   !
1021   !    Latch status (one per bit) 1=being asserted
1022   !
1025   IODEF LATCH_STATUS_B17@[SLOT=4, REGISTER=2, BIT=8]
1026   IODEF LATCH_STATUS_B18@[SLOT=4, REGISTER=2, BIT=9]
1027   IODEF LATCH_STATUS_B19@[SLOT=4, REGISTER=2, BIT=10]
1028   IODEF LATCH_STATUS_B20@[SLOT=4, REGISTER=2, BIT=11]
1030   !
1031   !    Latch edge transition selection (one per bit)
1032   !    0 = Off to on, 1 = On to off
1033   !
1034   IODEF LATCH_EDGE_B17@[SLOT=4, REGISTER=3, BIT=0]
1036   IODEF LATCH_EDGE_B18@[SLOT=4, REGISTER=3, BIT=1]
1037   IODEF LATCH_EDGE_B19@[SLOT=4, REGISTER=3, BIT=2]
1038   IODEF LATCH_EDGE_B20@[SLOT=4, REGISTER=3, BIT=3]
1040   !
1041   !    Latch reset (one per bit)
1042   !    write 0 = reset status
1043   !
1045   IODEF LATCH_RESET_B17@[SLOT=4, REGISTER=3, BIT=8]
1046   IODEF LATCH_RESET_B18@[SLOT=4, REGISTER=3, BIT=9]
1047   IODEF LATCH_RESET_B19@[SLOT=4, REGISTER=3, BIT=10]
1048   IODEF LATCH_RESET_B20@[SLOT=4, REGISTER=3, BIT=11]
32767  END
```

This configuration defines all of the information available on the module. If fewer than four interrupts are used, the unused definitions should be deleted.

For additional information
1 Allen-Bradley Drive
Mayfield Heights, Ohio 44124 USA
Tel: (800) 241-2886 or (440) 646-3599
http://www.reliance.com/automax

Publication J-3632-2 - June 1991