Keeping Other Axes Running During An Axis Fault

Abstract

Some applications require the machine to keep running after an axis fault with only the faulted axis being disabled. Whenever an axis fault occurs, MotionBASIC® opens the NO FAULT relay triggering an E-STOP. This tech note shows you how to keep your machine running after an axis fault without sacrificing the safety provided by proper use of the NO FAULT relay.

Note: You should also read tech note tn020 which describes a Simple MotionBASIC® Error Handler.

Wiring Description

For safety reasons ORMEC recommends all applications use a contactor to disconnect the servo drives from their source of bus power. The coil of this contactor should be wired in series with the NO FAULT relay. This will disconnect bus power any time the NO FAULT relay opens.

If you want to keep un-faulted axes running after an axis fault, you will need to provide one contactor for each servo drive and wire them as shown in Figure 1.

The Off-Delay relay turns on immediately after the NO FAULT relay closes and turns off a fixed delay after the NO FAULT relay opens. This time delay gives your program time to react to an axis fault and take the actions needed to keep the machine running before the E-STOP circuit is opened. The two control relays allow each servo to be disconnected from the servo bus power separately.

Example Application

The example program has two axes which perform a simple one revolution index followed by a one second dwell. If axis two faults, axis one must keep running. Axis two can then be restarted without affecting axis one.

If axis one faults, or any other type of fault occurs, both axes perform an E-STOP.

The program is designed to run on ORMEC’s standard two axis demonstration systems.

Program Description

The example program is an adaptation of the Simple MotionBASIC® Error Handler described in Tech Note #20. You will find it helpful to read and understand the error handler described in tn020 before trying to understand this program.

The design goals relating to fault handling are as follows:

- To re-close the NO FAULT relay as quickly as possible after a fault that shuts down only one axis. The example program accomplishes this in about 60ms.
- To RESUME to the program as quickly as possible after a fault that shuts down only one axis so that additional faults are handled properly. The example program accomplishes this in about 110ms.
• To be fail-safe so that if your program has errors or the controller malfunctions while processing a fault, the NO FAULT relay will still open and E-STOP the machine.

![E-STOP Circuit](image)

**Figure 1, E-STOP Circuit**

**RESTART:**

The restart routine prints a heading on the screen then enters a WHILE/WEND loop. Within the loop, the program continuously checks to see which of the axes are enabled and sets ENABLED.AXES~ accordingly.

It then checks to see if all the enabled axes have completed the index sequence by testing DSP.DONE@(ENABLED.AXES~). If they are done, another index/dwell sequence is commanded.
Finally, it tests to see if the "E" key has been pressed. If it has, it sets AXIS to AXIS2~ and calls the ENABLE routine to attempt to re-enable axis 2.

ERROR.HDLR:

When an error occurs, ERR.HDLR: tests to see if it is an axis fault and to make sure axis 1 is not among the axes with faults. If it is not an axis fault or axis 1 is among the axes with faults, it continues with the standard error handling process.

If it is an axis fault and axis 1 is not among the axes with faults, it does the following:

- Sets AXIS to AXIS.FLT1@ in order to remember which axis faulted.
- Calls GET.FAULTS to store the specific axis fault codes.
- Halts and disables the axis that faulted and opens its bus power relay.
- Clears AFAULT@ and FAULT@ in order to close the NO FAULT relay contacts before the Off-Delay Relay times out.
- Calls SHOW.STATUS to display the faults for whichever axis faulted.
- Clears the control stack and then resumes at the RESTART routine.

ENABLE:

The enable routine operates on the axis designated by the variable AXIS. First it checks to make sure the axis is not already enabled. If it is not, it closes the axis’s bus power relay then waits for the bus to stabilize. It then clears any axis faults and re-enables the axis.

The routine then calls GET.FAULTS to recover any fault codes that may still exist for that axis and SHOW.STATUS to update the axis’s status on the screen.

GET.FAULTS:

This routine operates on the axis designated by the variable AXIS. It simply recovers the values of AFAULT@ and ALARM@ and stores them in convenient variables for later use. By using this routine, ERROR.HDLR is able to quickly recover the fault codes then clear them to re-close the NO FAULT relay without the delay which would occur if it tried to print them before re-closing NO FAULT.

SHOW.STATUS:

This routine simply displays whether the axis designated by the variable AXIS is enabled, has faulted or is disabled.

Geared Applications

In geared applications, motion information is transmitted from DSP to DSP using MotionDATA. This information is transmitted at each DSP loop update. When one axis on a dual axis DSP card is enabled (by setting MODE@ to a non-zero value), the other axis may miss one packet of incoming MotionDATA information. If this happens while that axis is executing a motion statement, the axis will get out of synch with other axes in the machine by one DSP loop update. If this axis is providing it’s own actual or commanded position to MotionDATA, axes further down the MotionDATA chain will also be out of synch.
General Comments:

As shown by this program, keeping other axes running when one or more other axes fault is not difficult. The more complex part is re-entering your application program and continuing the process after an axis faults. The problems are too application specific for a tech note to show you exactly how to accomplish it, however here are some techniques that might help you.

- Set flags that tell you where you are in your program sequence and use these flags to figure out where you should go after re-entering RESTART;
- Test MODE@ before executing motion commands on axes that may be faulted to avoid creating another fault which will E-STOP the machine.

Program Listing

```bas
' AUTO 0
Routine name: POWERUP
Abstract: Program entry point. Performs all the needed initialization then puts the machine in the E-STOP state.
Routines called: *MP.CONFIG, CLEAR.FAULTS, ERROR.HDLR, INIT.DIO
                INIT.EVENTS, INIT.FUNCTIONS, INIT.VARS, RESTART
Variables used: ESTOP.OK@

POWERUP:
  MP.CONFIG 'run the unit/axis configuration routine.
               ' You will need to add this routine to your program using MotionPRO’s configuration menus.
  INIT.DIO 'initialize the discrete I/O
  INIT.VARS 'initialize your program variables
  INIT.EVENTS 'initialize any ON EVENT routines
  INIT.FUNCTIONS 'initialize any user defined functions
  CLS
  ON ERROR GOTO ERROR.HDLR
  IF ESTOP.OK@ THEN
    CLEAR.FAULTS
    RESTART
  ELSE
    ERROR 1910
  ENDIF
END

Routine name: INIT.EVENTS
Abstract: Initialize any EVENT subroutines.
Routines called: None
Variables used: None

INIT.EVENTS:
  ' Any ON EVENT ... GOSUB statements, go here.
  RETURN
```
Routine name: INIT.FUNCTIONS
Abstract: Initialize any user defined functions
Routines called: None
Variables used: None

INIT.FUNCTIONS:
'Any DEF FN ... statements go here.
RETURN

Routine name: INIT.VARS
Abstract: Initialize any variables your program uses
Routines called: None
Variables used: AXIS1~, AXIS2~, INITIALIZED

INIT.VARS:
IF NOT INITIALIZED THEN
AXIS1~={1}
AXIS2~={2}
'Things that only need to be initialized once
'when the program is first run, go here.
ENDIF
'String variables and other variables you need to
'initialize each time the program is run, go here.
INITIALIZED=TRUE
RETURN

Routine name: INIT.DIO
Abstract: Initialize any DIO points your program uses
Routines called: None
Variables used: A1, A2, FAULT.LIGHT, IO.MODE@()

INIT.DIO:
A1=2 :IO.MODE@(A1)="O"
A2=4 :IO.MODE@(A2)="O"
FAULT.LIGHT=6 :IO.MODE@(FAULT.LIGHT)="O"
'Additional DIO initialization statements go here.
RETURN

Routine name: ARM.EVENTS
Abstract:
Routines called: None
Variables used: None
""

ARM.EVENTS:
'any EVENT DIO(..) ON statements go here. Depending on your application
'some ENENTS may need to be armed at specific places in your program

EVENT ON 're-arm EVENT TRAPPING
RETURN
'

Routine name: DISARM.EVENTS
Abstract:
Routines called: None
Variables used: None

DISARM.EVENTS:
'EVENT DIO@() OFF statements go here

'Any error will suspend EVENT trapping. You should use this routine
to explicitly turn EVENT trapping off for DIO@() EVENTS you do not want
automatically re-enabled when you resume from the error handler
and execute the EVENT ON statement in ARM.EVENTS.

RETURN

Routine name: ESTOP.STATE
Abstract: Wait here for transition of ESTOP.OK@ from false
to true to clear faults and restart
Routines called: CLEAR.FAULTS, DISARM.EVENTS, DISPLAY.FAULTS
RESTART
Variables used: DIO@(), ESTOP.FLAG, ESTOP.OK@, EXIT, FAULT.LIGHT

ESTOP.STATE:
DISARM.EVENTS 'turn off any EVENT trapping that you do not
want automatically re-enabled.
DIO@ (FAULT.LIGHT)=ON
ESTOP.FLAG=NOT ESTOP.OK@
DISPLAY.FAULTS
PRINT
COLOR 15,0
PRINT "Cycle the ESTOP OK input to clear faults and restart...";
COLOR 7,0
EXIT=FALSE
WHILE NOT EXIT
   IF ESTOP.OK@ AND ESTOP.FLAG THEN
      EXIT=TRUE
   ELSEIF NOT ESTOP.OK@ THEN
      ESTOP.FLAG=TRUE
   ENDIF
WEND
CLS
CLEAR.FAULTS
STACK CLEAR
GOTO RESTART
'RESTART your main program
END
Routine name:  DISPLAY.FAULTS

Abstract: Waits for a time set in the DELAY variable (ms) checking to see if the ESTOP input has been while it waits.

Routines called: None

Variables used: AFAULT@(), ALARM@(), AXIS.FAULT@, AXIS.FLT1@

DISPLAY.FAULTS:
COLOR 7,0
CLS
PRINT "Controller fault(s): ";FAULT@
PRINT
IF FAULT@={} THEN PRINT " No Faults"
IF {1}*FAULT@ THEN PRINT " RAM Checksum Error"
IF {2}*FAULT@ THEN PRINT " Battery Failure"
IF {3}*FAULT@ THEN PRINT " Invalid Unit ID"
IF {4}*FAULT@ THEN PRINT " Internal Error"
IF {5}*FAULT@ THEN PRINT " Axis Module Failed"
IF {6}*FAULT@ THEN PRINT " E-STOP Input Open"
IF {7}*FAULT@ THEN PRINT " Axis Fault"
IF {8}*FAULT@ THEN PRINT " Program Generated Fault"
IF {9}*FAULT@ THEN PRINT " String Space Fault"
IF {10}*FAULT@ THEN PRINT " L.E.M. Card Fault"
IF (NOT {1,2,3,4,5,6,7,8,9,10})*FAULT@ THEN PRINT " Unknown Fault(s)"
PRINT
PRINT "Program error(s): "
PRINT
IF STORED.ERR THEN
PRINT USING " Code ####;& at line #####";STORED.ERR,ERR$(STORED.ERR),ERL
ELSE
PRINT " No program errors"
ENDIF
PRINT
PRINT "Axis fault(s):"
PRINT
IF AXIS.FLT1@ THEN
PRINT " Axis ";AXIS.FLT1@;" was the first to fault"
ELSE
PRINT " Fault was not caused by an axis"
ENDIF
FOR TMP~ WITHIN AXIS.FAULT@
PRINT " Axis ";TMP~;" - fault code";AFAULT@(TMP~);"(";
IF AFAULT@(TMP~)=0 THEN
PRINT "No faults"
ELSEIF AFAULT@(TMP~)=1 THEN
PRINT "Max position error"
ELSEIF AFAULT@(TMP~)=2 THEN
PRINT "Servo drive alarm code";ALARM@(TMP~);")"
ELSEIF AFAULT@(TMP~)=3 THEN
PRINT "Open wire, channel A"
ELSEIF AFAULT@(TMP~)=4 THEN
PRINT "Open wire, channel B"
ELSEIF AFAULT@(TMP~)=5 THEN
PRINT "Pos. cmd. overspeed"
ELSEIF AFAULT@(TMP~)=6 THEN
PRINT "Pacer overspeed"
ELSEIF AFAULT@(TMP~)=7 THEN
PRINT "Axis encoder overspeed"
ELSEIF AFAULT@(TMP~)=8 THEN
PRINT "MotionDATA error"
ELSEIF AFAULT@(TMP~)=9 THEN
PRINT "LOOP.RATE@ too high"
ELSEIF AFAULT@(TMP~)=10 THEN

PRINT "Pacer rate too high")
ELSEIF AFAULT@(TMP~)=11 THEN
PRINT "Missing MotionDATA")
ELSE
PRINT "Unknown fault")
ENDIF
NEXT TMP~
RETURN

Program: TN023.BAS

Routine name: RESTART

Abstract: Application restart location. This is where your application program takes over

Routines called: ARM.EVENTS, ENABLE

Variables used: AXIS, AXIS1~, AXIS2~, DSP.DONE@(), ENABLED.AXES~ MODE@()

RESTART:
ARM.EVENTS 're-arm any EVENT trapping
PRINT @1,1;"Press E to re-enable axis {2} ":
WHILE TRUE

'This is where the application program really starts. For this example we have a simple routine that indexes each un-faulted axis in turn. It also attempts to re-enable an axis when you enter an axis number at the keyboard

'find out which axes are enabled and set ENABLED.AXES~
IF MODE@(AXIS1~) THEN ENABLED.AXES~=AXIS1~ ELSE ENABLED.AXES~={}
IF MODE@(AXIS2~) THEN ENABLED.AXES~=ENABLED.AXES~+AXIS2~

IF DSP.DONE@(ENABLED.AXES~) THEN
MOVE ENABLED.AXES~ FOR 360 IN 1000
MOVE ENABLED.AXES~ FOR 0 IN 1000
ENDIF

IF UCASE$(INKEY$)="E" THEN AXIS=AXIS2~ :ENABLE
WEND
END

Program: TN023.BAS

Routine name: ERROR.HDLR

Abstract: React to a fault or error

Routines called: ESTOP.STATE, GET.FAULTS, RESTART, SHOW.STATUS

Variables used: AFAULT@, AXIS, AXIS.FAULT@, AXIS.FLT1@
AXIS.LIST@, AXIS1~, DIO@(), DSP.DONE@(), FAULT@
MODE@(), routine, STORED.ERR, the

ERROR.HDLR:
STORED.ERR=ERR 'remember the error code number
'A CTRL-C stops the program for debugging purposes. Normally you should comment this line out after completing your development.

IF STORED.ERR=1805 THEN CLOSE: ON ERROR GOTO 0

'if it is an axis fault and axis 1 has not faulted
IF STORED.ERR=1911 AND NOT AXIS1~ WITHIN AXIS.FAULT@ THEN
AXIS=AXIS.FLT1@
CLEAR.FAULTS:
AXIS.SET@=AXIS.LIST@
OTL.FWD@=0 :OTL.REV@=0 :AFAULT@=0 :FAULT@=0 :WAIT 300
DIO@(FAULT.LIGHT)=OFF
STORED.ERR=0
CLS
AXIS=AXIS1~ :ENABLE
AXIS=AXIS2~ :ENABLE
RETURN

GET.FAULTS:
AFAULT=AFAULT@(AXIS)
ALARM=ALARM@(AXIS)
RETURN
Routine name: ENABLE

Abstract: Attempt to enable AXIS

Routines called: GET.FAULTS, SHOW.STATUS

Variables used: AFAULT@(), AXIS, DIO@(), MODE@()

ENABLE:
    IF MODE@(AXIS)=OFF THEN
        DIO@(AXIS*2)=ON
        WAIT 300
        AFAULT@(AXIS)=FALSE
        MODE@(AXIS)=5
    ENDIF
    GET.FAULTS
    SHOW.STATUS
RETURN

Routine name: SHOW.STATUS

Abstract: Show the status of AXIS

Routines called: None

Variables used: AFAULT, ALARM, AXIS, MODE@()

SHOW.STATUS:
    IF MODE@(AXIS)=5 THEN
        PRINT @ AXIS*3,5;"Axis {" STR$(AXIS) "} enabled"; :CLREOL
    ELSEIF AFAULT THEN
        PRINT @ AXIS*3,5;"Axis {" STR$(AXIS) "} fault code" AFAULT; CLREOL
        IF AFAULT=2 THEN
            PRINT " alarm code " ALARM
        ELSE
            PRINT
        ENDIF
    ELSE
        PRINT @ AXIS*3,5;"Axis {" STR$(AXIS) "} disabled"; :CLREOL
    ENDIF
RETURN