FIRMWARE DEVELOPMENT IMPROVES SYSTEM EFFICIENCY

E. James Chern and David W. Butler
Materials Branch / Code 313
NASA Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

Most manufacturing processes require physical pointwise positioning of the components or tools from one location to another. Typical mechanical systems utilize either stop-and-go or fixed feed-rate procession to accomplish the task. The first approach achieves positional accuracy but prolongs overall time and increases wear on the mechanical system. The second approach sustains the throughput but compromises positional accuracy. A computer firmware approach has been developed to optimize this pointwise mechanism by utilizing programmable interrupt controls to synchronize engineering processes "on the fly". This principle has been implemented in an eddy current imaging system to demonstrate the improvement. Software programs were developed that enable a mechanical controller card to transmit interrupts to a system controller as a trigger signal to initiate an eddy current data acquisition routine. The advantages are: (1) optimized manufacturing processes, (2) increased throughput of the system, (3) improved positional accuracy, and (4) reduced wear and tear on the mechanical system.

INTRODUCTION

Many industrial production processes such as machining, cleaning, assembling, labeling, inspections, etc., require mechanical maneuvering of components or assemblies from one position to another. Typical state-of-the-art mechanical systems use machine tool or robotics motion controllers with stop-and-go or fixed feed-rate mechanisms to meet positioning requirements. The stop-and-go approach achieves positional accuracy but prolongs overall processing time and increases wear on the mechanical system. The fixed-rate sampling approach sustains the throughput but compromises positional accuracy. We have employed a motion controller board and developed software programs to achieve both positional accuracy and sustained throughput. Interrupts generated by the motion controller firmware can be used to synchronize engineering processes "on the fly" for those processes that require minimum or no dwell time at given locations.

The hardware and associated software have been successfully implemented in an eddy current imaging system which consists of a system controller, an eddy current instrument, a mechanical motion controller card, and a two-axis x-y linear table with incremental encoders. The scan routine programs the mechanical controller card with the scan parameters and pre-determined measurement positions. The mechanical controller card's microprocessor constantly compares the real-time probe position from the encoder feedback with the preset acquisition coordinates. An interrupt is generated by the mechanical controller and is used by the system as the trigger signal to initiate the data acquisition routine. Although the concept is demonstrated on an eddy current image system, the interrupt control mechanism can be applied to many other engineering processes.

In this paper, we reviewed hardware and software requirements for implementation of firmware interrupt controls in an eddy current imaging system and other practical applications. Laboratory setup and experimental procedures for the benchmark comparison of the current and conventional approaches using the eddy current imaging system are also described. The results from the parametric experiments clearly demonstrate the improvement in system efficiency. Essential C-language source codes for the interrupt control routines are provided in the Appendix for user reference.

BACKGROUND AND APPROACH

Recent advances in personal computer and electronic technology have enabled the development and operation of various stand-alone concurrent engineering stations. These advancements also facilitated the development of evaluation engineering, nondestructive evaluation (NDE), signal acquisition, image processing, and data presentation techniques. NDE methods are widely utilized in manufacturing and service

industries for quality assurance and related applications[1,2]. Image-oriented data presentation which directly correlates acquired engineering parameters with component coordinates is generally the preferred way of displaying results.

Based on the underlying physical principles of NDE imaging methods, images can be acquired in two forms depending on whether or not the sensor or specimen is manipulated with respect to the other. Imaging systems such as ultrasonic C-scan and eddy current imaging, have to rely on a mechanical scanner to physically maneuver the probe relative to the specimen point by point over the area of interest to acquire images. A typical pointwise NDE imaging system consists of three major components: a system controller to control instruments, command movements, and acquire data; instrumentation to excite the sensor and measure desired signal parameters; and a mechanical scanner to relatively scan the sensor over the area of interest on the specimen. The block diagram and a sketch of a typical eddy current imaging system is shown in Figure 1.

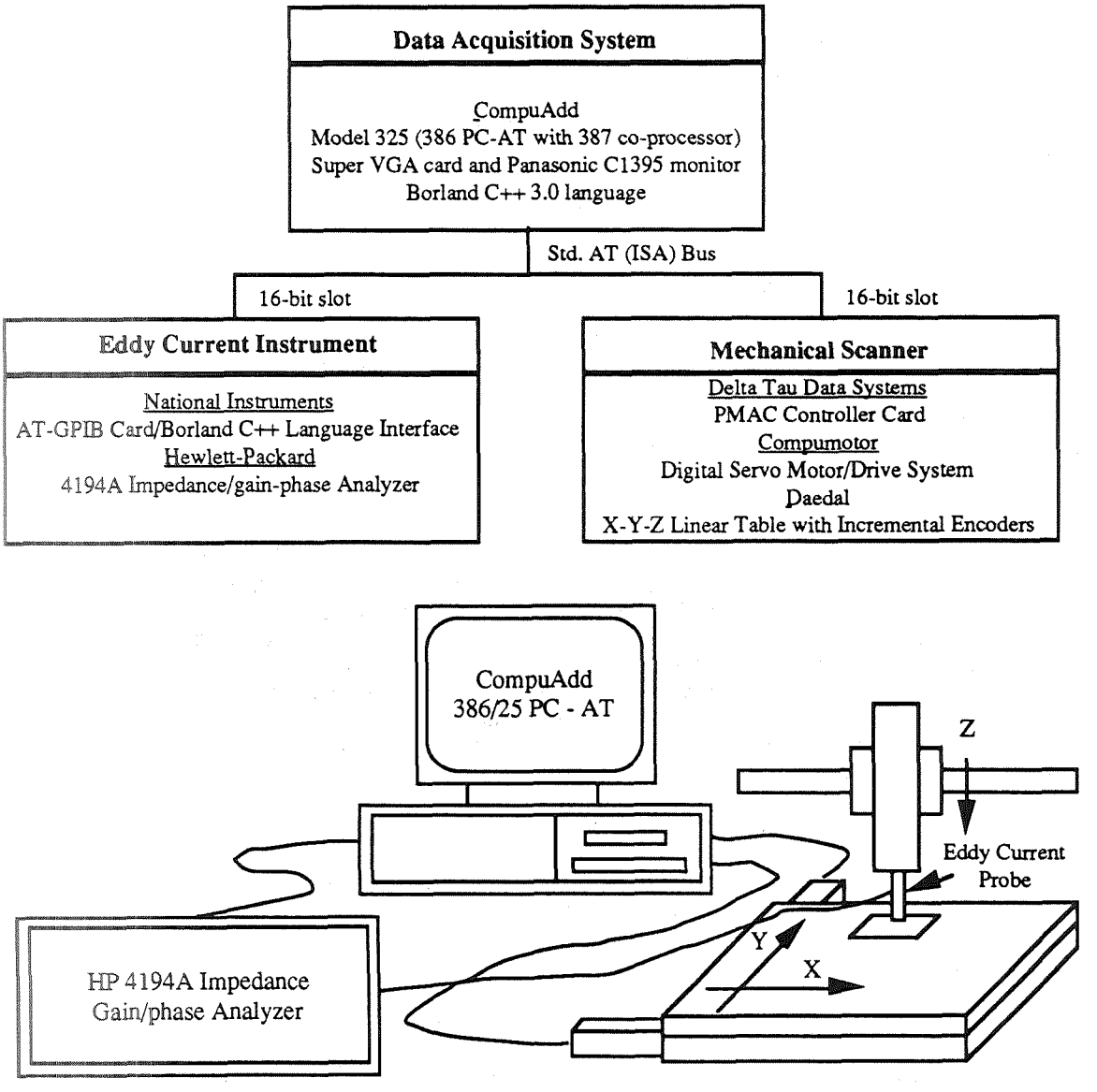

Figure 1. The block diagram and the sketch of the prototype interrupt based eddy current imaging system.

The ideal pointwise imaging system is to command the scanner to scan at a desired speed and fetch measurements at the designed positions "on the fly". However, due to hardware and software constraints, data acquisition is commonly accomplished by either stop-and-go or fixed rate sampling. The principle of the improvement is to utilize the newly available microprocessor based motion controller card as an intelligent controller which initiates and controls the data acquisition process.

The specific approach is to develop firmware routines which enable the motion controller card's microprocessor to constantly compare the real-time probe position from the encoder feedback with the preset acquisition coordinates[3]. An interrupt signal is transmitted to the system controller as a trigger to initiate a data acquisition routine when the positional conditions are met. We have devised a position-driven closed-loop mechanical system for NDE applications. The system uses interrupts generated by the mechanical system at the designed positions, to trigger and initiate the data acquisition routine for the measurements[4].

SYSTEM CONFIGURATION

The improved eddy current imaging system consists a CompuAdd 325 as the system controller, a Hewlett-Packard 4194A Impedance/gain-phase Analyzer as the signal drive and measuring instrument, a Delta Tau Data Systems PMAC motion controller card, Compumotor Plus motors, and a Daedal X-Y linear table with incremental linear encoders as the mechanical scanner. The system controller interfaces with the scanner using the PMAC mechanical controller card through the industry standard architecture (ISA) PC-bus and acquires eddy current impedance data from the HP 4194A through an IEEE-488 interface bus.

The system controller commands the mechanical system such that, the probe traverses the area of interest in a raster pattern. The impedance of the probe is acquired by the impedance analyzer during the scan. Firmware was developed to enable the PMAC to generate interrupts in the system controller as trigger signals to initiate data acquisition sequences. The interrupt structure between host controller and peripheral PIC is shown in Figure 2. The C source code of the interrupt routine is listed in the Appendix.

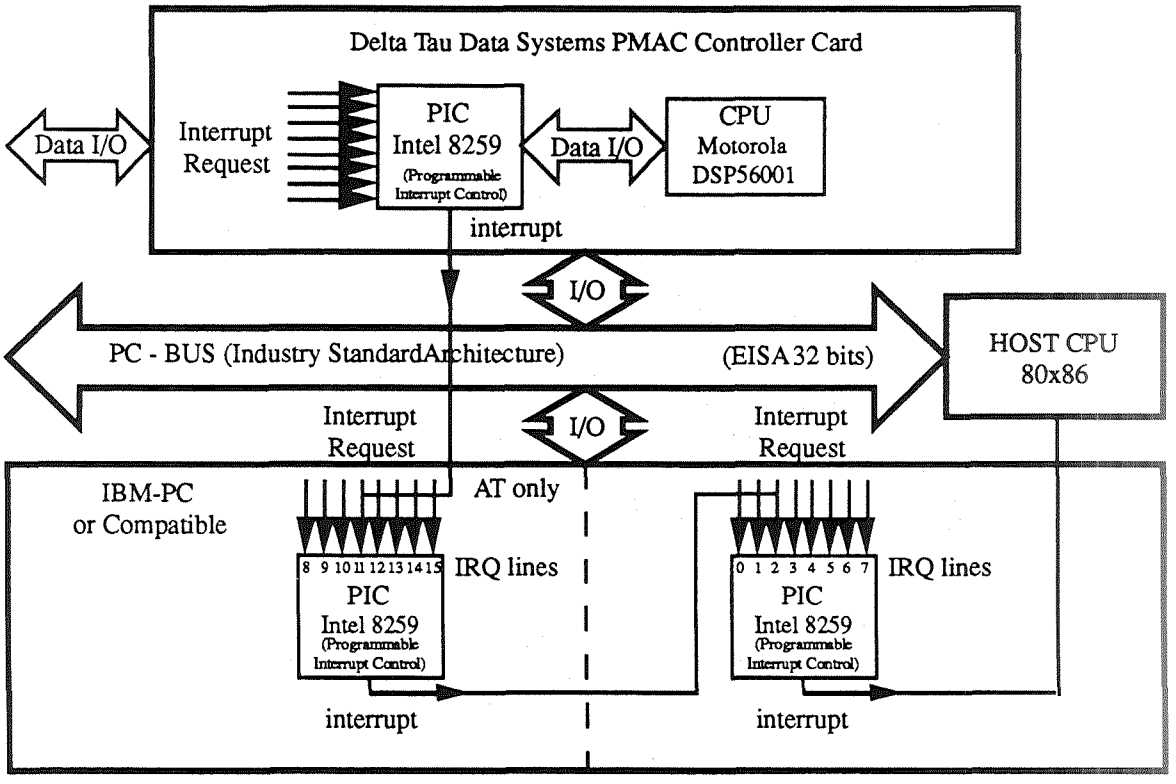

Figure 2. The interrupt structure for the Host PC and peripheral PICs of the prototype interrupt-based eddy current imaging system.

The mechanical scan subroutine programs the PMAC with the necessary scan parameters such as the home position, distances, velocities, and accelerations as well as pre-determined positions where measurements are to be performed. The PMAC microprocessor constantly compares the real-time probe position from the encoder feedback with the preset acquisition coordinates. An interrupt signal is generated by a Programmable Interrupt Controller (PIC) on the PMAC card when the positional conditions are met. This signal is then received by another PIC located in the host controller. This PIC subsequently generates an interrupt in the host CPU. This interrupt is used by the host CPU as the trigger signal to initiate the data acquisition routine and synchronize other events. The PMAC PIC continually generates interrupts until the scan subroutine is completed.

Since the linear encoders are independent of the mechanical drives, the interrupts are generated precisely at the desired coordinates. The only constraint is that the speed of the scanner is limited by the time needed to complete the data acquisition routine and transfer the data through the interface bus. Also, this improvement can only apply to engineering processes that do not require prolonged dwell time. Figure 3 is an eddy current image of an impact damaged composite test piece to demonstrate one of the practical uses of the approach. The scan area is 2.5 inch (6.35 cm) by 0.5 inch (1.27 cm). The data acquisition interval is 0.01 inch (0.254 mm) for both x and y axis, i.e. 250 points by 50 points. Although the current application is for eddy current data acquisition and image generation, the approach can be easily applied to ultrasonic imaging and other engineering systems. The only modification necessary is to substitute the eddy current measuring subsystem with the desired engineering instrumentation.

EXPERIMENTS AND RESULTS

The time needed for a given engineering process at a desired location depends only on the process itself. This time is the same regardless of mechanical approaches. The tangible benefit is the decreased scan time in the interrupt-based approach versus the point-to-point approach. The main effect for the improvement is that the interrupt driven scan maintains a constant speed along the scanning axis during data acquisition, while the point-to-point scan must stop at designated intervals. Thus the experiments and data acquisition software have been setup to enable the recording and comparison of the time required for both the interrupt and point-to-point approaches. Identical scan parameters such as scan speeds and index sizes were used for both approaches for the comparisons.

The test specimen used for the bench mark tests is a 3 inch (76.2 mm) by 4 inch (101.6 mm) aluminum block. The scanning velocity for x-axis and y-axis is set to be 0.5 inch (12.7 mm) per second. Three scan configurations are used for the experiments: (1) x-step size of 0.025 inch (0.635 mm) and y-step size of 0.025 inch (0.635 mm); (2) x-step size of 0.025 inch (0.635 mm) and y-step size of 0.050 inch (1.27 mm); and (3) x-step size of 0.050 inch (1.27 mm) and y-step size of 0.050 inch (1.27 mm). Identical scans are performed at least two times to ensure a proper estimate of scan times. The scan time for the described tests are recorded and compared. The average scan time variation is approximately ten seconds or less. The results from the tests are tabulated in Table 1.

The ratio of the two scan times, point-to-point scan time over interrupt scan time, is calculated for each of the three tests conducted. This ratio is used as the measure of the improvement factor. As shown in Table 1 there is approximately a factor of two improvement in scan time for the interrupt scan as compared to the point-to-point scan. This improvement factor is a function of the scan configuration such as scanning speed, data acquisition interval, and specimen size, etc. However, the dominate factor is the number of acquisition points along the scanning axis.

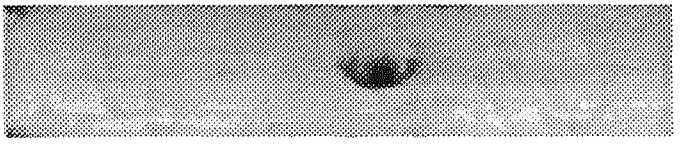

Figure 3. A typical eddy current image of an impact damaged composite test piece (2.5" x 0.5" scan envelop).

Table 1. Comparison of the experimental results from point-to-point and interrupt scans.

Test Configuration Scan Area = 4" x 3" X velocity = 0.5"/sec Y velocity = 0.5"/sec	Point-to-Point	Interrupt	Improvement Factor
x-step size = 0.025" y-step size = 0.025" (160 pts x 120 pts)	2375.49 sec	1013.52 sec	2.34
x-step size = 0.025" y-step size = 0.050" (160 pts x 60 pts)	1192.42 sec	514.01 sec	2.32
x-step size = 0.050" y-step size = 0.050" (80 pts x 60 pts)	1015.27 sec	526.10 sec	1.93

CONCLUSION AND DISCUSSION

In summary, we have (1) proposed a new approach by using an interrupt control mechanism to improve pointwise engineering systems; (2) performed experiments to prove the concept, and (3) verified the practical application aspect by implementing the concept in an eddy current imaging system. Technical improvements include (1) optimized operating parameters; (2) reduced wear and tear on the mechanical system; (3) increased throughput; and (4) improved accuracy for data acquisition and image generation. These improvements translate to increased productivity and reduced cost in engineering operations.

IBM-PCs and their compatibles are gaining in popularity as system controllers and host computers for many mechanical control, instrument control, and signal processing boards. IBM-PC based manufacturing and test/measuring systems thus are routinely being developed, introduced and implemented in various industries. This new approach of using interrupts to initiate and synchronize engineering events has immense commercial potentials; it can be applied to engineering systems in manufacturing, testing, evaluation, and monitoring such as material dispensing, packaging, sorting, and many other industrial applications.

In conclusion, we have demonstrated the use of an interrupt control mechanism for PC-based eddy current NDE data acquisition and image generation. IBM-PCs and their compatibles are gaining in popularity as the system controllers and hosts for mechanical and instrument control boards used in many manufacturing and measuring systems. This new approach of using interrupts to initiate and synchronize engineering events has tremendous commercial potential; it can be applied to systems in manufacturing, evaluation, and monitoring. Specific examples are material dispensing, packaging, sorting, and many other applications.

REFERENCES

1. E. J. Chern, *Materials Evaluation*, Vol. 49, No. 9, September 1991, p1228.

2. M. J. Golis, *An Introduction to Nondestructive Testing*, American Society for Nondestructive Testing, Inc., 1991.

3. E. J. Chern and D. W. Butler, *NASA Tech Briefs*, Vol. 16, No. 9, September 1992, p44.

4. E. J. Chern, to be published in the proceedings of the 1992 *Review of Progress in Quantitative NDE*, Vol. 12, Plenum Press, New York, 1993.

APPENDIX

```
/* Program: INTERRUPT.C */

/*******************/
/* ~do_interrupt_scan */
/*****************************************************************************
This is the main scanning routine in which the data will be recorded using interrupts generated on the actual
position of the X-Y scanning table. Global 2-D array, Data_array, will be sized and allocated with the
values defined by the user. If there is not enough available memory to store the data for the entire scan, the
user will be asked to reduce the size of the scan or abort. Array indexing will be done by incrementing the
array pointer whenever a position interrupt occurs. Since these events are controlled as to occur only at
valid data acquisition points within the scan, data for each scan line is stored sequentially in the array. The
data associated with each scan line in the array can be decoded by knowing the total number of data points
for each scan line. This number will vary depending on the size of the scan and the increment size. The
velocity of the scan has to be limited in such a way as to allow the impedance meter enough time to take a
valid reading.
*****************************************************************************/

void do_interrupt_scan (int scan_type)
{
DATA_VAL huge *mem_ptr ;                    /* Huge pointer for traversing data array */
char data_buf [MAX_LEN] ;                   /* Response string for impedance meter */
char cmd_str  [MAX_LEN] ;                   /* Command string for PMAC */
int abort = FALSE ;                         /* Flag to abort scan */
int done = FALSE ;                          /* Flag for normal exit of scan */
int key ;                                   /* Key pressed on keyboard */
/* initialize global flags */
Breq_flag = 0 ;                             /* Interrupt flag for PMAC */
Equ1_flag = 0 ;                             /*   "       "      "   */
Inpos_flag = 0 ;                            /*   "       "      "   */
Endofscan = FALSE ;                         /* Flag to indicate end of scan */
/* Allocate memory for 2-D data array */

if ((Data_array = define_data_array (Num_Y_pts, Num_X_pts)) == NULL)
{
      display_mem_error () ;
      exit_program (ERROR) ;
}
mem_ptr = Data_array ;                      /* Set pointer to beginning of data array */
/* Send PLC programs and definitions to PMAC */
dnload_pmac_defines () ;
dnload_PLC_0 () ;
dnload_PLC_1 () ;
/* Setup instruments and position the probe */
setup_HP4194() ;
display_positioning_msg () ;
/* Initialize PMAC and the host PC to accept interrupts */
init_interrupt_mode () ;
/* Begin data acquisition */
send_pmac_cmd ("R") ;                       /* Send RUN command to PMAC card */
while (!done)
{
/* Now process the interrupts */
if(Equ1_flag)                               /* At data measurement point */
      {
      Equ1_flag = 0 ;                       /* Reset Interrupt flag */
      /* Get reading from impedance meter and store it in memory */
```

```
        Receive (BRD, Imp-meter, data_buf, MAX_LEN, STOPend) ;
        mem_ptr->val_1 = atof (data_buf) ;          /* store impedance value */
        mem_ptr++ ;
        }
if (Breq_flag)                                      /* PMAC ready to receive next command */
        {
        Breq_flag = 0 ;                             /* Reset interrupt flag */
        send_next_pmac_move () ;                    /* Send a move sequence to PMAC */
        }
if (Inpos_flag)                                     /* All motion and move timer stopped */
        {
        Inpos_flag = 0 ;                            /* Reset interrupt flag */
         if (Endofscan)                             /* Terminate loop if scan complete */
        done = TRUE ;
        }
if (kbhit ())                                       /* Process user requests */
        if (getch() == ESC)                         /* Check for ABORT request */
        {
        send_pmac_cmd ("H") ;                       /* Halt the scan */
        done = abort = TRUE ;                       /* Terminate loop */
        }
}
/* End of data acquisition loop */
/* Restore the original PC interrupt vectors and store acquired data to disk */
restore_interrupt_vector () ;
if (!abort)
        {
        write_data_to_disk (Data_file) ;
        create_header_file (Data_file, scan_type) ;
        }
/* Free the memory used for data storage */

farfree ((void *) Data_array) ;
}
/**  End of do_interrupt_scan  **/


/*********************/
/*  ~init_interrupt_mode  */
/*****************************************************************************************
This routine will setup the Programmable Interrupt Controller (PIC) on both the PMAC card and the IBM-
PC.  PMAC will be interrupting the IBM_PC on one of the interrupt request lines (IRQ).  The IRQ line is
defined by PC_IRQ in EC_DEFIN.H.  Interrupt request levels 1 (IR1) and 5 (IR5), which correspond to the
buffer request and equ1 lines on PMAC's PIC, will be used to generate interrupt pulses that are to be sent to
the IBM-PC, on the selected IRQ line.  When these interrupt pulses are acknowledged by the PC's PIC,
interrupt service routines will be activated to record data and send motion commands to PMAC's rotary
buffer.
IMPORTANT: in order for this routine to function properly, jumpers on the PMAC card must be installed
to reflect the PC IRQ line that will be used to interrupt the host PC.  A jumper at E65 must also be
installed to electrically connect the equ1 line to PMAC's PIC.
*****************************************************************************************/
void init_interrupt_mode (void)
{
/* Save original interrupt vector for PC IRQ line, so it can be restored when the program terminates.  If
this is not done, the default handler for this IRQ will not function without rebooting. */
disable () ;                                        /*  disable interrupts until done  */
Old_int_vector = getvect (PC_INT) ;                 /*  save original interrupt vector */
setvect (PC_INT, pmac_isr) ;                        /*  write in new interrupt vector  */
```

```
/* Save original mask value for the PC PIC's interrupt request register and enable interrupt request level
used by PMAC */
#ifdef PC_PIC_2
Old_int_status = inportb (PC_PIC2_ODD) ;          /* Save old mask value of PIC #2 */
outportb (PC_PIC2_ODD,(Old_int_status & PC_MASK)) ;          /* Enable IRQ used by
                                                    PMAC */
#else
Old_int_status = inportb (PC_PIC1_ODD) ;          /* Save old mask value PIC #1 */
outportb (PC_PIC1_ODD,(Old_int_status & PC_MASK)) ;     /* Enable IRQ used by PMAC */
#endif
/* Set up PMAC's PIC so it can generate interrupt pulses when the BREQ or EQU1 lines go high. */
outportb (BASE, FLUSH);                           /* Flush PMAC's interrupt control register */
outportb (PMAC_PIC_EVEN, EDGE_TRIG) ;             /* Set edge triggered mode (ICW1) */
outportb (PMAC_PIC_ODD, BUS_VECTOR) ;             /* Vector for data bus (ICW2) */
outportb (PMAC_PIC_ODD, MODE_8086) ;              /* Set up for 8086 mode (ICW4) */
outportb (PMAC_PIC_ODD, PMAC_MASK) ;              /* Unmask IR1 (BREQ) & IR5 (EQU1) (OCW1) */
outportb (BASE, DSP_READ) ;                       /* Enable PMAC DSP read */
enable () ;                                       /* enable interrupts, done! */
}
/** End of init_interrupt_mode **/


/*************/
/* ~pmac_isr */
/*****************************************************************************************
This is the interrupt service routine for a PMAC generated hardware interrupt in the host. PMAC will be
interrupting the host PC on one of IRQ lines. In order for this routine to function, a jumper must be set on
the PMAC card to indicate which IRQ line is being used, as defined by PC_IRQ in EC_DEFIN.H. A
jumper must also be installed at E65 on the PMAC card to connect the compare-equals signal (EQU1), to
PMAC's PIC (8259A). This routine will test the In Service Register (ISR) of PMAC's PIC to determine
which event has triggered an interrupt, and then set a flag in the host to indicate that event. Currently the
events that are to be tested for are: the buffer request (BREQ), EQU1 signal, and the in position (IPOS)
signal.
*****************************************************************************************/

static void interrupt far pmac_isr (void)
{
char isr_val ;                                    /* Value read from PIC's in service register */
disable () ;                                      /* Prevents other interrupts */
outportb (PMAC_PIC_ACK, PMAC_NOP) ;               /* Rising edge of 1st INTA pulse */
outportb (PMAC_PIC_EVEN, PMAC_NOP) ;              /* Trailing edge of 1st INTA pulse */
outportb (PMAC_PIC_ACK, READ_ISR) ;               /* Setup to read PIC's ISR */
isr_val = inportb (PMAC_PIC_EVEN) ;               /* Read PIC's ISR value, generates  rising edge of
                                                     pulse */
2nd INTA
if (isr_val & PMAC_BREQ)                          /* If BREQ event, set a flag */
Breq_flag = 1 ;
if (isr_val & PMAC_EQU1)                          /* If EQU1 event, set a flag */
Equ1_flag = 1 ;
if (isr_val & PMAC_IPOS)                          /* If in position, set a flag */
Inpos_flag = 1 ;
outportb (PMAC_PIC_EVEN, PMAC_NOP) ;              /* Trailing edge of 2nd INTA pulse */
#ifdef PC_PIC_2
outportb (PC_PIC2_EVEN, PC_EOI) ;                 /* Send end of interrupt byte to PC PIC #2 */
#endif
outportb (PC_PIC1_EVEN, PC_EOI) ;                 /* Send end of interrupt byte to PC PIC #1 */
enable () ;                                       /* Re-enables other interrupts */
}
/** End of pmac_isr **/
```

```
/*****************/
/*  ~dnload_PLC_0  */
/*****************************************************************************
This routine will down load a PLC program to PMAC to generate interrupts when the X axis encoder
reaches set positions at even increments in both scanning directions. The position-compare-function of
PMAC's DSP-Gate array is utilized. Positions at which interrupts are to occur are calculated and then
preloaded into the compare register, thus generating an interrupt in the host computer when the encoder
value and compare register value are equal.
*****************************************************************************/

void dnload_PLC_0 (void)
{
send_pmac_cmd ("OPEN PLC 0") ;                  /* Open buffer */
send_pmac_cmd ("CLEAR") ;                       /* Clear it */
send_pmac_cmd ("IF(M116=1)") ;                  /* ENC1 EQU flag bit set? */
send_pmac_cmd ("IF(P1=0)") ;                    /* Negative scan direction ? */
send_pmac_cmd ("WHILE (M101=M105)");            /* Wait for update of position */
send_pmac_cmd ("ENDWHILE") ;
send_pmac_cmd ("M103=M105") ;                   /* Load next EQU position */
send_pmac_cmd ("M105=M105-P101") ;              /* Calc. following EQU position */
send_pmac_cmd ("IF(M105=P201-P101+P301)") ;     /* End of neg. dir. scan line? */
send_pmac_cmd ("M105=M105+P101") ;              /* Prepare pos. dir. position */
send_pmac_cmd ("P1=1") ;                        /* Set positive direction flag */
send_pmac_cmd ("ENDIF") ;
send_pmac_cmd ("ELSE") ;                        /* Moving in positive direction */
send_pmac_cmd ("WHILE (M101=M105)") ;           /* Wait for update of position */
send_pmac_cmd ("ENDWHILE") ;
send_pmac_cmd ("M103=M105") ;                   /* Load next EQU position */
send_pmac_cmd ("M105=M105+P101") ;              /* Calc. following EQU position */
send_pmac_cmd ("IF(M105=P301+P101)") ;          /* End of pos. dir. scan line? */
send_pmac_cmd ("M105=M105-P101") ;              /* Prepare neg. dir. position */
send_pmac_cmd ("P1=0") ;                        /* Set negative direction flag */
send_pmac_cmd ("ENDIF") ;
send_pmac_cmd ("ENDIF") ;
send_pmac_cmd ("M111=0") ;                      /* Clear and set latch control bit */
send_pmac_cmd ("M111=1") ;                      /* to clear latched flag */
send_pmac_cmd ("ENDIF") ;
send_pmac_cmd ("CLOSE") ;
while (getline(Response_buf)) ;                 /* Clear PMAC's data register */
}
/** End of dnload_PLC_0 **/


/*****************/
/*  ~nload_PLC_1  */
/*****************************************************************************
This routine will down load a PLC program to PMAC to activate the DSP-Gate array registers on the
PMAC card. The DSP-Gate array will be used in PLC program dnload_PLC_1 and initialize those registers
with the proper values to start the interrupt generating sequence. This PLC program is executed only once
and then disables itself.
*****************************************************************************/

void dnload_PLC_1 (void)
{
send_pmac_cmd ("OPEN PLC 1") ;                  /* Open buffer */
send_pmac_cmd ("CLEAR") ;                       /* Clear it */
```

```
send_pmac_cmd ("M111=0") ;                    /* Make sure ENC1 EQU flag latch */
send_pmac_cmd ("M111=1") ;                    /* control bit is reset */
send_pmac_cmd ("M112=1") ;                    /* Enable EQU output */
send_pmac_cmd ("M113=0") ;                    /* Set EQU output to high true */
send_pmac_cmd ("P1=0") ;                      /* Clear direction flag */
send_pmac_cmd ("P301=M101") ;                 /* Get starting position from ENC1 */
send_pmac_cmd ("M105=M101") ;                 /* Init. counter to staring position */
send_pmac_cmd ("M103=M105") ;                 /* Load first EQU position */
send_pmac_cmd ("M105=M105-P101") ;            /* Calc. following EQU position */
send_pmac_cmd ("ENABLE PLC 0") ;
send_pmac_cmd ("DISABLE PLC 1") ;
send_pmac_cmd ("CLOSE") ;
while (getline(Response_buf)) ;               /* Clear PMAC data register */
}

/** End of dnload_PLC_1 **/
```