Air Mouse

Project Report
Embedded Systems
Fall – 2005

By
Mamta Venugopal
Reshmi Mitra
Mohammed Amaan -Ul Quadir

Index

[1]Abstract:

There has been tremendous research in the implementation of the functionalities of hardware devices during the last decade. For instance, research on mouse has led to designing it in different flavors like serial mouse, PS/2 mouse, bus mouse, USB mouse etc. To improvise the mouse functionalities further, 'Air Mouse' has been taken as project work.

Air Mouse is a three dimensional mouse that takes the user's finger acceleration as input and moves the mouse to the desired position on the computer. But due to problem of drivers, we are only trying to display the x, y and z analog to digital converted values of the accelerometer on the hyper terminal.

[2]Introduction:

Air Mouse is designed to lessen the inconvenience caused to the computer user, who has to switch back and forth to use both the keyboard and mouse. Whenever the user has to move the cursor, he will simply press a button that is already fixed to the glove, worn on the hand and point his finger at the screen and move it. Air mouse is comfortable to wear and does not significantly inhibit typing. It functions completely as a two button serial mouse, and even implements variable rate vertical scrolling.

[3]Motivation:

To work with an accelerometer sensor was our prime motivation. So we improvised a mouse project that would incorporate an accelerometer for moving the cursor to desired position.

[4]Hardware Design

Requirements:

To implement this project, the equipment needed:

1) White board

2) Analog Device Accelerometer – 2(ADXL311)

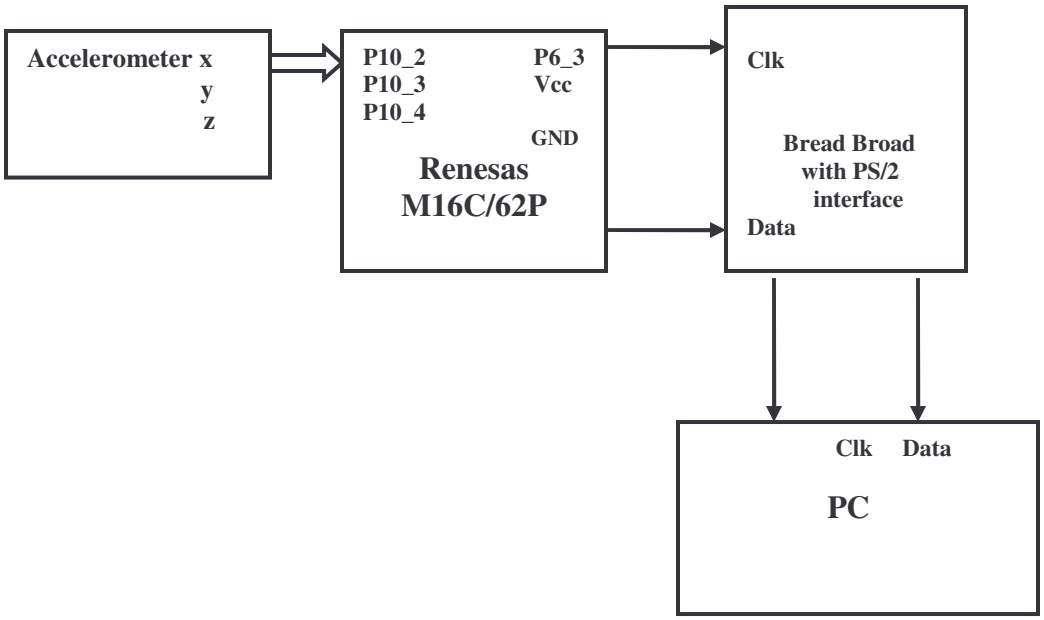
3) 9 volts DC Power Supply

4) 7400 nand chip – 2

5) PN2222 transistor - 2

6) Renesas – M16C/62P

Design Methodology:



Block Diagram for the circuit

The block diagram of the circuit has been drawn after analyzing the requirements. The working principle of a general PS/2 mouse was studied and described as follows:

- The accelerometers detect the mouse movement and button switches sense the button states. The controller reads the state of this sensor and button and takes into account the current mouse position.

- When this information changes, the controller makes calculations based upon the changed values of x, y and z scroll up and sends a packet of data to the computer data interface controller.
- The mouse driver in the computer receives that data packets and decodes the information from it and does actions based on the information.

Accelerometer:

1) Accelerometer ADXL311 is selected because it has high sensitivity (195mV/g), low cost, low power and dual axis.

2) It measures acceleration with a full-scale range of ± 2 g and the analog output voltages are proportional to acceleration.

3) This accelerometer uses tilt measurement i.e. it uses the force of gravity as an input vector to find the orientation of an object in space.

Renesas:

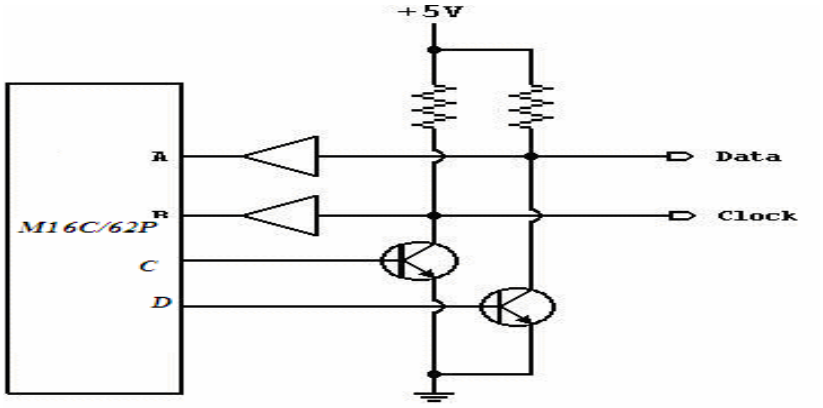
1)This board has 382KB of user ROM area and 31KB of user RAM for processing and storing values the received values in our present project.

2) It also has 10 bit ADC for converting the measured analog X , Y and Z voltages into digital values. It also has UART 's for sending the data serially to RS232.

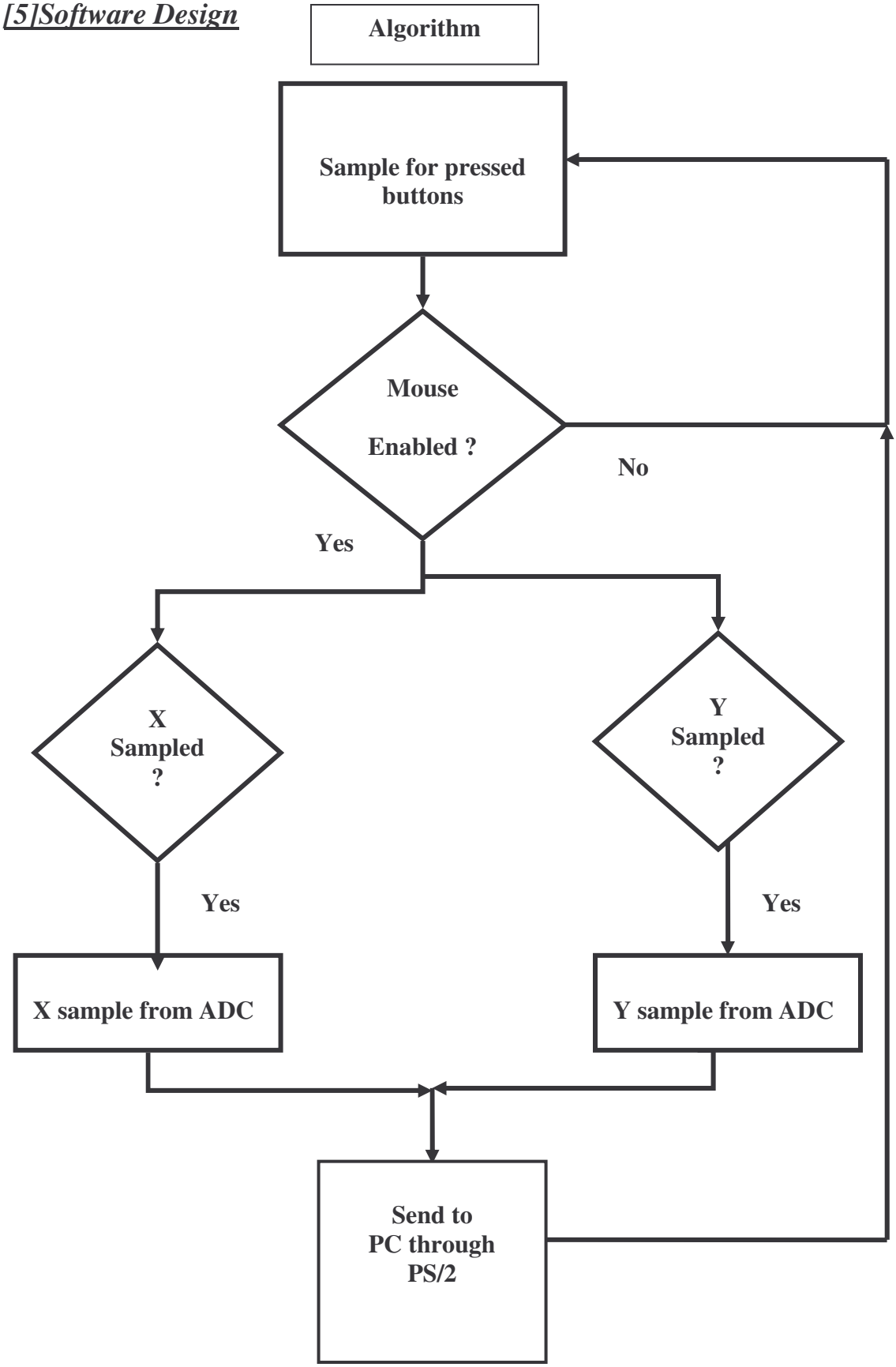
3) After having gone through the user and hardware manual of M16C62, we have decided to use this board for this project.

PS/2 :

PS/2 mouse implements a bidirectional synchronous serial protocol. The data from the accelerometers are collected and processed by **M16C/62P. This processed data is sent to the computer through an open-collector interface.**



[5]Software Design



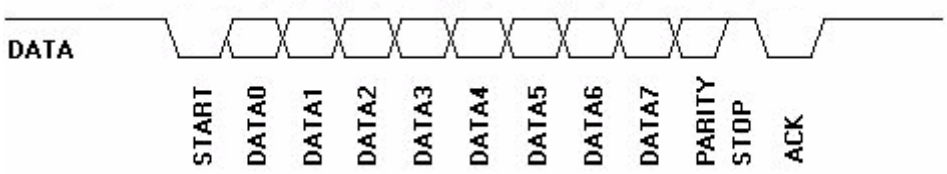
Ref: http://www.computer-engineering.org/ps2protocol/

The data and clock line are both open-collector with pull resistors to Vcc. A and B pins are set as input while C and D pins are set as outputs. A and C are input and output lines for the data while B and D are input and output lines for clock respectively.

Protocol Description:

When both data line and clock lines are high (open collector), the bus is "idle". When the data line is high and clock line is low, the communication is inhibited. When the data line is low and clock line is high, the hosts (computer) request the data to be sent. The data is sent by synchronizing with the clock, generated by the device (mouse).

Data is transmitted one byte at a time and each byte is sent in a frame consisting of 11-12 bits.



a)start bit = 0

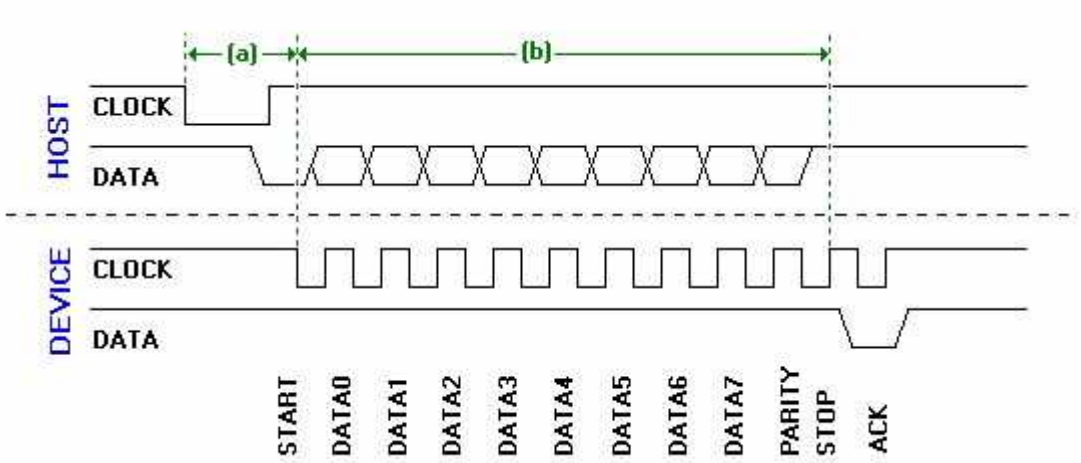
b) 8 data bits, least significant bit first
c) parity bit(odd parity)
d) stop bit = 1
e) Acknowledge bit(host-to-device communication only)

Data sent from the device to the host is read on the falling edge of the clock signal while data sent from the host to the device is read on the rising edge. The clock frequency is 10-16.7 kHz.

Device to Host: The clock line must be high for atleast 50microsecond before the device can decide to send data.

Host to Device:
The host may inhibit communication at any time by pulling the clock line low for atleast 100microseconds. If the transmission is inhibited before the 11^{th} clock pulse, the device must abort the transmission and resend it once again. If the new data is created that needs to be transmitted, it has to be buffered until the host releases clock.

When the host wants to send data, it has to pull the data line low and release the clock line.



Program Code

```
/*************************************************************************
/*  NAME       :Mamta Venugopal; Reshmi Mitra; Mohammed AmaanUl Quadir
/*  FILE       :main.c
/*  DATE       :December , 2005
DESCRIPTION :This program converts the sampled values of
             Accelerometer and sends to Hyper-terminal
/*************************************************************************/

#include "skp_bsp.h"    // include SKP board support package

/* Prototype declarations */
void mcu_init(void);    // MCU initialization
void main(void);
char * IntToAsciiHex(char * dest_string,int min_digits,unsigned int value);
char * IntToAsciiDec(char * dest_string,int min_digits,unsigned int value);

//global variables
int count_timer=0;
void timera1_isr(void);
char lcd_text[9],lcd_text1[21];
unsigned int x_input, y_input,x;
unsigned int i,flag1=0;
unsigned int admux=0;

//Define queue
#define Q_SIZE (32)

typedef struct {
```

```
  unsigned char Data[Q_SIZE];
  unsigned int Head; // points to oldest data element
  unsigned int Tail; // points to next free space
  unsigned int Size; // quantity of elements in queue
} Q_T;

Q_T tx_q, rx_q;

int Q_Empty(Q_T * q) {
  return q->Size == 0;
}

int Q_Full(Q_T * q) {
  return q->Size == Q_SIZE;
}

int Q_Enqueue(Q_T * q, unsigned char d) {
  // if queue is full, abort rather than overwrite and return
  // an error code
  if (!Q_Full(q)) {
    q->Data[q->Tail++] = d;
    q->Tail %= Q_SIZE;
    q->Size++;
    return 1; // success
  } else
    return 0; // failure
}

unsigned char Q_Dequeue(Q_T * q) {
  // Must check to see if queue is empty before dequeueing
  unsigned char t=0;
  if (!Q_Empty(q)) {
    t = q->Data[q->Head];
    q->Data[q->Head++] = 0; // empty unused entries for debugging
    q->Head %= Q_SIZE;
    q->Size--;
  }
  return t;
}

void Q_Init(Q_T * q) {
  unsigned int i;
  for (i=0; i<Q_SIZE; i++)
    q->Data[i] = 0;  // to simplify our lives when debugging
  q->Head = 0;
```

```
  q->Tail = 0;
  q->Size = 0;
}
/*****************************************************************************
Name       : timera1_isr
Parameters : none
Returns    : nothing
Description: Timer A ticks for every 0.05 seconds.
*****************************************************************************/
#pragma INTERRUPT timera1_isr
void timera1_isr(void)
{
 RED_LED^=1;
count_timer++;
}

/*****************************************************************************
Name       : main
Parameters : none
Returns    : nothing
Description: Main template
*****************************************************************************/
void main(void)
{
  mcu_init();               /* Initialize MCU */
  ENABLE_IRQ;    /*Enable interrupts*/
  ENABLE_SWITCHES /*Switch initialization-macro defined in skp_bsp.h*/
  ENABLE_LEDS       /* LED initialization - macro defined in skp_bsp.h */

  //ADC conversion initialization
     adcon1=0x21;//Vref is connected,AN0-AN3 are selected
     adcon2=0x01;//sample hold enabled

  // Setting port 10 pins 2, 3, 4 as inputs
     pd10_2 = 0x00;
     pd10_3= 0x00;
     pd10_4= 0x00;

 // Uart0 initialization
    u0brg=77; // for having baud rate of 9600, n=77
    u0mr=0x45; //8 data bits, one stop bit,one odd parity bit,LSB first
    u0c0=0x10; //Disabled CTS/RTS
    u0c1=0x05; //Transmission and reception enabled

//Timer initialisation
```

```
   ta0mr = 0x00; //Timer A0 is in timer mode
   ta1mr = 0x01; //Timer A1 is in event mode
   ta0   = 0x1770;
   ta1   = 0x64;  // Timer A is set for 0.05 seconds
   trgsr = 0x0002;//TA0 overflow is selected
   ta1ic = 0x03;  //Timer A1 interrupt level is set
   udf   = 0x00;  //Down count for both timers
   ta0s  = 1;     //Start timer A0

   Q_Init(&tx_q); //Initialise queues for transmitter
   InitDisplay();   //Clear Display

while(1)
      {
      if(!S1)
      //When Switch 1 is pressed  enable mouse
      // Check if timer is enabled. If yes, then print the message "MOUSE ENABLED"
            {
            if(ta1s==0)
                        {
                        lcd_text1[13]=0xB;
                         lcd_text1[12]='D';
                         lcd_text1[11]='E';
                         lcd_text1[10]='L';
                        lcd_text1[9]='B';
                        lcd_text1[8]='A';
                        lcd_text1[7]='N';
                        lcd_text1[6]='E';
                        lcd_text1[5]='  ';
                        lcd_text1[4]='E';
                        lcd_text1[3]='S';
                       lcd_text1[2]='U';
                       lcd_text1[1]='O';
                       lcd_text1[0]='M';

                       for(i=0;i<14;i++)
                            {
                           if(!Q_Enqueue(&tx_q,lcd_text1[i])){}
                                 while(!ti_u0c1);
                                       u0tbl = Q_Dequeue(&tx_q);
                           }//END for
                      } //END if
            ta1s = 1;
            GRN_LED^=1;
      } //END if for switch SW1
```

```
if(!S2)
  //When Switch 2 is pressed disable mouse
 // Check if timer is enabled. If yes, then print the message “MOUSE DISABLED”
 // Disable the timer

{
 if(ta1s==1)
              {
              lcd_text1[13]=0xB;
              lcd_text1[12]='E';
              lcd_text1[11]='L';
              lcd_text1[10]='B';
              lcd_text1[9]='A';
              lcd_text1[8]='S';
              lcd_text1[7]='I';
              lcd_text1[6]='D';
              lcd_text1[5]='  ';
              lcd_text1[4]='E';
              lcd_text1[3]='S';
              lcd_text1[2]='U';
              lcd_text1[1]='O';
              lcd_text1[0]='M';

             for(i=0;i<14;i++)
                    {
                    if(!Q_Enqueue(&tx_q,lcd_text1[i])){}
                          while(!ti_u0c1);
                                u0tbl = Q_Dequeue(&tx_q);
                    }//END for
           } //END if
           ta1s = 0;
 GRN_LED^=1;
 } //END if

if(count_timer==10) //Values are updated after every 0.5 seconds
              {
              YLW_LED^=1;
              count_timer=0;
              if(admux==0)  //detect x movement
                        {
                         adst=1;
                        x_new=ad2;  // read x-coordinates from AN2 (port 10 pin 2)
                        if(x_new == x_previous)
                                  {
```

```
                x_input=0;
                }
            if(x_new > x_previous)
                {
                if((x_new -x_previous)>2)
                        {
                        x_input=5;
                         lcd_text1[2]='+';
                        }
                 }
            if(x_new < x_previous)
                 {
                 if((x_previous-x_new )>2)
                        {
                        x_input=5;
                        lcd_text1[2]='-';
                         }
                }
             x_previous=x_new;
            //Convert measured AN2 (x-coordinate) value for LCD.
             IntToAsciiDec(lcd_text,4,x_input);
            //Now write value to LCD starting on the 1st position on line 1
             DisplayString( (char)(LCD_LINE1 ), lcd_text);
             lcd_text1[7]=0x20;
             lcd_text1[6]=lcd_text[3];
             lcd_text1[5]=lcd_text[2];
             lcd_text1[4]=lcd_text[1];
             lcd_text1[3]=lcd_text[0] ;
             lcd_text1[1]='=';
             lcd_text1[0]='X';

             for(i=0;i<7;i++)
                    {
                   if(!Q_Enqueue(&tx_q,lcd_text1[i])){}
                            while(!ti_u0c1);
                                    u0tbl = Q_Dequeue(&tx_q);
                   }//END for
            admux=1;   // read for y-coordinate
    }// END if admux



if (admux==1) // detect y movement
        {
        adst=1;
```

```
 y_new=ad3;   // read y-coordinates from AN3 (port 10 pin 3)
 if(y_new == y_previous)
        {
        y_input=0;
        lcd_text1[2]=' ';
         }
if(y_new > y_previous)
        {
       if((y_new -y_previous)>2)
              {
              y_input=5;
              lcd_text1[2]='+';
              }
        }
        if(y_new < y_previous)
              {
              if((y_previous-y_new )>2)
                    {
                     y_input=5;
                    lcd_text1[2]='-';
                    }
               }
         y_previous=y_new;
        //Convert measured AN2 (y-coordinate) value for LCD.
        IntToAsciiDec(lcd_text,4,y_input);
        //Now write value to LCD starting on the 1st position on line 2
        DisplayString( (char)(LCD_LINE2 ), lcd_text);
        lcd_text1[7]=0x20;
        lcd_text1[6]=lcd_text[3];
        lcd_text1[5]=lcd_text[2];
            lcd_text1[4]=lcd_text[1];
        lcd_text1[3]=lcd_text[0] ;
       lcd_text1[1]='=';
        lcd_text1[0]='Y';

        for(i=0;i<7;i++)
              {
              if(!Q_Enqueue(&tx_q,lcd_text1[i])){}
                    while(!ti_u0c1);
                          u0tbl = Q_Dequeue(&tx_q);
              }//END for
       admux=2;  //read for z - coordinate
}// END else
```

```
if(admux==2)  // d e t e c t  z  m o v e m e n t
            {
            adst=1;
            z_new=ad4; // read z-coordinates from AN4 (port 10 pin 4)
            if(z_new == z_previous)
                        {
                        z_input=0;

                        lcd_text1[2]=' ';
                        }
                       if(z_new > z_previous)
                                {
                                if((z_new -z_previous)>2)
                                            {
                                           z_input=5;
                                           lcd_text1[2]='+';
                                            }
                                 }
                                 if(z_new < z_previous)
                                           {
                                           if((z_previous-z_new )>2)
                                                      {
                                                      z_input=5;
                                                      lcd_text1[2]='-';
                                                       }
                                            }

                                z_previous=z_new;

                                //Convert measured AN2 (y-coordinate) value for LCD.
                                IntToAsciiDec(lcd_text,4,z_input);

                                 //Now write value to LCD starting on the 5th position on line 2
                                 DisplayString( (char)(LCD_LINE2 + 4 ), lcd_text);


// after displaying the x,y, z movements delete the contents of the hyper terminal screen

// show fresh detections
                                 //lcd_text1[29]=0x08;
                                //lcd_text1[28]=0x08;
                                lcd_text1[27]=0x08;
                                lcd_text1[26]=0x08;
                                lcd_text1[25]=0x08;
                                lcd_text1[24]=0x08;
                                lcd_text1[23]=0x08;
                                lcd_text1[22]=0x08;
                                lcd_text1[21]=0x08;
                                lcd_text1[20]=0x08;
                                lcd_text1[19]=0x08;
                                lcd_text1[18]=0x08;
                                lcd_text1[17]=0x08;
                                lcd_text1[16]=0x08;
                                lcd_text1[15]=0x08;
                                lcd_text1[14]=0x08;
                                lcd_text1[13]=0x08;
```

```
                                        lcd_text1[12]=0x08;
                                        lcd_text1[11]=0x08;
                                        lcd_text1[10]=0x08;
                                        lcd_text1[9]=0x08;
                                        lcd_text1[8]=0x08;
                                        lcd_text1[7]=0x08;
                                        lcd_text1[6]=lcd_text[3];
                                        lcd_text1[5]=lcd_text[2];
                                        lcd_text1[4]=lcd_text[1];
                                        lcd_text1[3]=lcd_text[0] ;
                                        lcd_text1[1]='=';
                                        lcd_text1[0]='Z';

                                    for(i=0;i<28;i++)
                                            {
                                            if(!Q_Enqueue(&tx_q,lcd_text1[i])){}
                                                    while(!ti_u0c1);
                                                            u0tbl = Q_Dequeue(&tx_q);
                                            }//END for
                                     admux=0;         //read for x - coordinate
                        }// END if admux

                    }//END if counter

                }//END while(1)

        }//END main()



        /*****************************************************************************
        Name      : IntToAsciiHex
        Parameters: dest_string
                Pointer to a buffer will return the string that will reside
                in   min_digits. Specifies the minimum number of characters
                the output string will have. Leading zeros will be written
                as '0' characters.
        Returns:    A pointer to the string's NULL character in the string that was just created.
        Description:This function is used to convert a passed unsigned int into
                a ASCII string represented in base Ascii format.
        *****************************************************************************/
        char * IntToAsciiHex(char * dest_string,int min_digits,unsigned int
        value)
        {
                        unsigned int i, total_digits = 0;
                        char buff[4];

                        for(i=0;i<4;i++)
                        {
                                        buff[i] = (char)(value & 0x0F);
```

```
                value = value >> 4;
                if( buff[i] <= 9)
                        buff[i] += '0';
                else
                        buff[i] = (char)(buff[i] - 0xA + 'A');

                if(buff[i] != '0')
                        total_digits = i+1;
        }

        if( total_digits <      min_digits)
                total_digits = min_digits;

        i = total_digits;
        while(i)
        {
                *dest_string++ = buff[i-1];
                i--;
        }

        *dest_string = 0;

        return dest_string;
}
/*****************************************************************************
Name    : IntToAsciiDec
Parameters: dest_string
        Pointer to a buffer will return the string that will reside
        in  min_digits. Specifies the minimum number of characters
        the output string will have. Leading zeros will be written
        as '0' characters.
Returns:  A pointer to the string's NULL character in the string that
        was just created.
Description:This function is used to convert a passed unsigned int into
        a ASCII string represented in base 10 decimal format.
*****************************************************************************/
char * IntToAsciiDec(char * dest_string,int min_digits,unsigned int
value)
{
        const unsigned long base10[] = {1,10,100,1000,10000,100000};

        unsigned int tmp;
        unsigned int i, total_digits = 0;
        char buff[5];
```

```
        for(i=0;i<5;i++)
        {
                tmp = (int)( value % base10[i+1] );
                value -= tmp;

                buff[i] = (char)( tmp / base10[i] );
                buff[i] += '0';

                if(buff[i] != '0')
                        total_digits = i+1;
        }

        if( total_digits <      min_digits)
                total_digits = min_digits;

        i = total_digits;
        while(i)
        {
                *dest_string++ = buff[i-1];
                i--;
        }

        *dest_string = 0;

        return dest_string;
}
```

[6]Problems:

1) The amplifier and filter circuit that we designed for our project was not giving the desired output spikes. So we directly incorporated the accelerometer with the microcontroller and we found that the results were accurate.

2) Due to unavailability of Microsoft Serial mouse drivers in Windows XP, we were unable to implement the movement of the cursor.

3) The sensitivity of the accelerometer is too high and hence it is difficult to show the exact positional change of the sensors. The slightest movement of the hand was causing some output at the hyperterminal. Hence, we have scaled down the inputs and showed the positive and negative movements.

[7]Future Implementation:

Once the drivers are loaded into the computer, we can implement the movement of the cursor. The algorithm for this has been designed and so code can be written easily.

Implementation of Software Program:

1) The working principle of Microsoft Serial Protocol that describes how cursor has to move when packets are sent. For instance, we were planning to use "MZ" Extended Microsoft protocol, mouse wheel mode

	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Packet 1	'1'	L	R	Y7	Y6	X7	X6
Packet 2	'0'	X5	X4	X3	X2	X1	X0
Packet 3	'0'	Y5	Y4	Y3	Y2	Y1	Y0
Packet 4	'0'	'0'	M	Z3	Z2	Z1	Z0

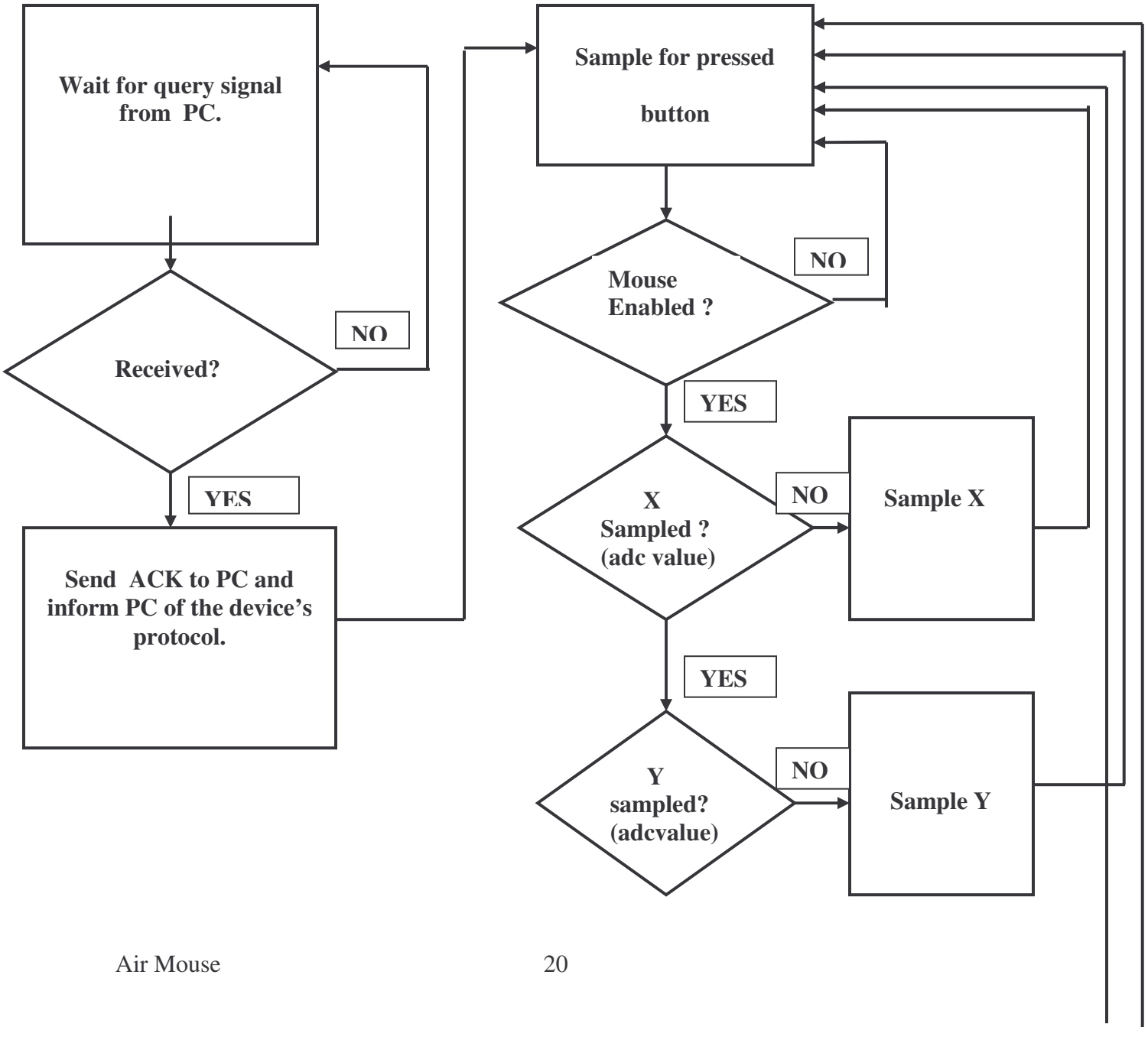
2) Microsoft serial mouse protocol uses an 8-bit two's complement to send data to the computer. Therefore the numbers output by Renesas's ADC has to be converted to two's complement format.

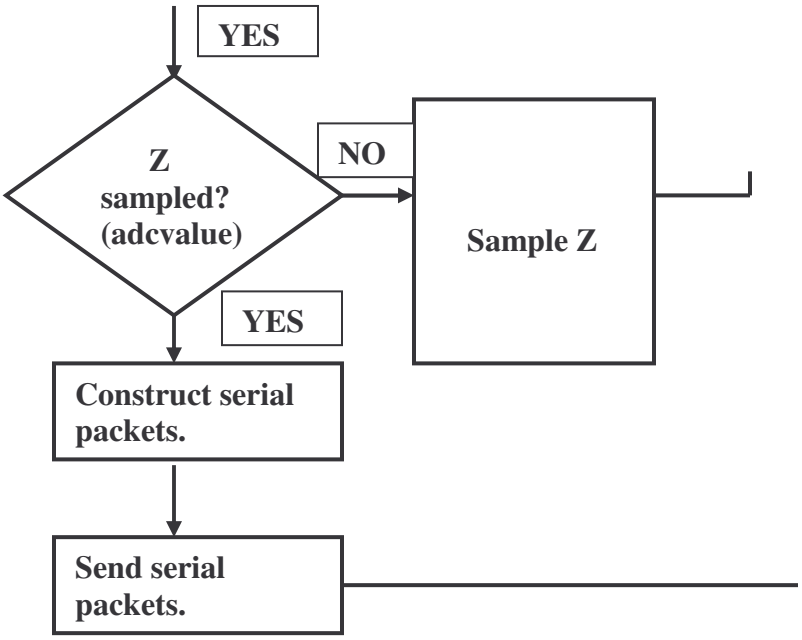
3) Designed an algorithm for it.

4) This algorithm can be converted into code.

5) This code when executed will give the desired output of moving the cursor to a desired position.

ALGORITHM


Wait for query signal from PC.
Received?
NO
YES
Send ACK to PC and inform PC of the device's protocol.
Sample for pressed button
Mouse Enabled ?
NO
YES
X Sampled ? (adc value)
NO
Sample X
YES
Y sampled? (adcvalue)
NO
Sample Y


YES
Z
sampled?
(adcvalue)
NO
Sample Z
YES
Construct serial
packets.
Send serial
packets.

[8]Conclusion:

X , Y values of the accelerometer has been sent to the micro-controller. From the micro-controller the converted analog to digital value is sent to the hyper- terminal.

[9]References:

Datasheets :

1) Renesas M16C62 Hardware Manual

2) ADXL311 Accelerometer

3) LM358P Dual Operational Amplifier

4) MAX233A RS232

Websites:

1) http://freedos-32.sourceforge.net/showdoc.php?page=sermouse

2) http://users.tkk.fi/~then/mytexts/mouse.html