Programming to Receive Joystick Input in Dos

One of the steppng stones to utilizing a computer for data acquisition and device control is the ability to receive input from its ports. After becoming familiar with programming for a PC's ports, you can start to design programs and hardware to meet the needs of your applications.

MOTIVATION AND AUDIENCE

The focus of this tutorial is to demonstrate a method for reading joystick input from a computers game port using C in a DOS based application. This tutorial will teach you:

- ***The basics of the game port.***
- ***How to write code to receive input from a joystick.***
- ***How to utilize data received from the joystick.***

To do this, it is assumed that you already:

- ***Are familiar with programmng in C.***

The rest of the tutorial is presented as follows:

- **Parts List and Sources**
- **The Game Port**
- **Programming**
- **Final Words**

PARTS LIST AND SOURCES

All that is required for this tutorial are the following:

- ***A 286 or higher PC with a game port.***
- ***A C compiler. Turbo C can be found and downloaded at Programmers Heaven.***
- ***A 2 axis, 2 button joystick that utilizes the game port.***

This sensor was chosen because of its compactness and the wide range over which it can measure. It is also easily interfaceable with microcontrollers. The servo chosen is a standard servo, however, any servo that operates off of PWM input will do (timing may vary).

To construct the circuit, you will also need:

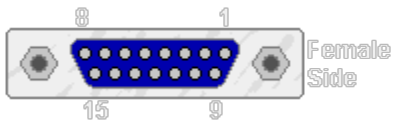
- ***a soldering iron with a fine point***
- ***materials for soldering (solder, flux, etc.)***
- ***small gauge wire***

- ***wire strippers***
- ***multimeter***
- ***DC power supply***

After downloading the compiler, install it nto the directory "C:\borlandc".

THE GAME PORT

The game port was originally designed to relive the erial port of devices like joysticks and MIDI devices. It eventually became the standard port for interfacing with a joystick or other game controller. The address for the game port is 0201 (Hex). The following diagram shows the pins for the game port and there respective functions:



Bit	Meaning
1	Joystick A, X Axis
2	Joystick A, Y Axis
3	Joystick B, X Axis
4	Joystick B, Y Axis
5	Joystick A, Button 1
6	Joystick A, Button 2
7	Joystick B, Button 1
8	Joystick B, Button 2

Pin No.	Function	Pin No.	Function
1	+ 5 Vdc	9	+ 5 Vdc
2	Joystick A, Button 1	10	Joystick B, Button 1
3	Joystick A, X Axis	11	Joystick B, X Axis
4	Ground	12	Ground
5	Ground	13	Joystick B, Y Axis
6	Joystick A, Y Axis	14	Joystick B, Button 2
7	Joystick A, Button 2	15	+ 5 Vdc
8	+ 5 Vdc		

The axis pins generally receive analog input from a pot on the joystick. The button pins receive a digital input. This information is primarily background, and will not greatly affect the actual programming.

PROGRAMMING

You can write your program with a standard text editor such as notepad (be careful with programs like MS Word that might add erroneous characters or header information). The code is presented in full below. Each portion of code will be addressed individually later in the tutorial.

Joytest.c

```
/*
* _outportb(port, byte) - outputs a byte to an I/O port
* _inportb(port) - inputs a byte from an I/O port
* _disable() - disables CPU interrupts
* _enable() - enables CPU interrupts
*/
```

```
#include dos.h
#include graphics.h

/* standard I/O port address for the PC joystick*/

#define JPORT 0x201
#define CMAX 9999

void joystick(int v[4], char b[4])

{ int i, j;
int mask = 15;
disable();
outportb(JPORT, 0);
for (i=1; mask && i < CMAX; i++) {
j = inportb(JPORT) ^ mask;
if (j & 1) { v[0] = i; mask ^= 1; }
if (j & 2) { v[1] = i; mask ^= 2; }
if (j & 4) { v[2] = i; mask ^= 4; }
if (j & 8) { v[3] = i; mask ^= 8; }
}
enable();
j = inportb(JPORT);
b[0] = !(j & 0x10);
b[1] = !(j & 0x20);
b[2] = !(j & 0x40);
b[3] = !(j & 0x80);
}

main(int argc, char **argv)
{
int v[4];
int color;
char b[4];
int gdriver = DETECT, gmode, errorcode;

/* start with zeros, indicating nothing there */

memset(v, 0, sizeof(v));

initgraph(&gdriver, &gmode, " ");

errorcode = graphresult();
if (errorcode != grOk)
  {
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt.");
  getch();
  exit(1);
  }

while (!kbhit())
  {
  joystick(v, b);
  if (b[0] & !b[1])
  {color=3;}
  else
  if (b[1] & !b[0])
  {color=4;}
  else
```

```
  if (b[1] & b[0])
  {color=6;}
  else
  {color=5;}
  setcolor(color);
  setfillstyle(1,color);
  circle(v[0],v[1],15);
  floodfill(v[0],v[1],color);
  delay(100);
  setcolor(0);
  setfillstyle(1,0);
  floodfill(v[0],v[1],0);
  }
closegraph();
}
```

INCLUDES AND CONSTANTS

The first portion of code defines the necessary header files that contain the functions we will be usng. After stating the includes, two constants are declared. One is the address for the game port. The other constant simply defines a maximum number of times to look for output from the axes before moving on.

```
#include dos.h
#include graphics.h

/* standard I/O port address for the PC joystick*/

#define JPORT 0x201
#define CMAX 9999
```

JOYSTICK FUNCTION

The next portion of code defines a function that, when called, reads in data from the joystick and returns an array of integers representing the position of the axes and the states of the buttons.

```
void joystick(int v[4], char b[4])

{ int i, j;
int mask = 15;
disable();
outportb(JPORT, 0);
```

This starts with the declaration of the function. The array v has four positions, each representing a value returned from one of two axes on one of two joysticks. Likewise, the b array stores the output from the joysticks buttons. The mask variable will be used to determine which axes have returned data. It starts off set as decimal 15, which is binary 1111. Disable() disables the PC's interupts, which givs more reliable readings. Finally, a value is sent to the joystick. This initializes the joystick for a reading (triggers an oscillator in the joystick that starts its outputting of values).

```
for (i=1; mask && i < CMAX; i++) {
```

```
j = inportb(JPORT) ^ mask;
if (j & 1) { v[0] = i; mask ^= 1; }
if (j & 2) { v[1] = i; mask ^= 2; }
if (j & 4) { v[2] = i; mask ^= 4; }
if (j & 8) { v[3] = i; mask ^= 8; }
}
```

This loop reads the axes of the joystick and stores their values in the v array. The loop begins by inputting values from the various axes. The values 1, 2, 4, and 8 represent the bits 1,2,3 and 4, respectively. If any of the bits are set, the value of i at that time is stored in v and the coresponding bit in mask is set to 0. This continues as long as mask is not 0000 and i is less than 9999.

```
enable();
j = inportb(JPORT);
b[0] = !(j & 0x10);
b[1] = !(j & 0x20);
b[2] = !(j & 0x40);
b[3] = !(j & 0x80);
}
```

UTILIZING INPUT FROM THE JOYSTICK

The remainder of the code is a simple application that places a circle on the screen who's position is directed by the joystick. The circle changes color based on which buttons are pressed.

```
main(int argc, char **argv)
{
int v[4];
int color;
char b[4];
int gdriver = DETECT, gmode, errorcode;
```

This sets up the arrays and the graphics driver.

```
memset(v, 0, sizeof(v));

initgraph(&gdriver, &gmode, " ");

errorcode = graphresult();
if (errorcode != grOk)
  {
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt.");
  getch();
  exit(1);
  }
```

v is filled with zeroes to ensure that it does not have a value to begin with. The graphics driver is then initialized and a standard error detection test follows.

```
while (!kbhit())
```

```
  {
  joystick(v, b);
  if (b[0] & !b[1])
  {color=3;}
  else
  if (b[1] & !b[0])
  {color=4;}
  else
  if (b[1] & b[0])
  {color=6;}
  else
  {color=5;}
  setcolor(color);
  setfillstyle(1,color);
  circle(v[0],v[1],15);
  floodfill(v[0],v[1],color);
  delay(100);
  setcolor(0);
  setfillstyle(1,0);
  floodfill(v[0],v[1],0);
  }
closegraph();
}
```

This loop is the main program. The joystick function is called and the values from the joystick are returned in v and b. Depending on which buttons are pressed, the color for teh circle is changed. A circle of that color is then drawn and filled in. After a short delay, it is drawn over in black (to erase it and make room for the next circle. This continues as long as a key on the keyboard is not hit.

FINAL WORDS

After completing this tutorial you should be familiar with the the gameport, be able to program in C to receive input from a joystick and apply that input to an application.

If you have questions about this tutorial you can email me at **Keithicus@drexel.edu**.