Z-100 LifeLine
Since 1989

March 2022

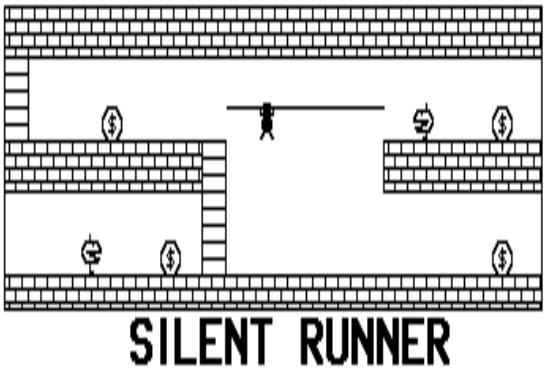
#WEB

This article was first published in issue #55, February 1998

Joystick on the Z-100

This probably caught you by surprise, as it did me. So, let me start with some background.

Way back in issue #54, I did a review of some AVES (Audio Visual Entertainment Software) Games, which included one called "**Silent Runner**" which included the capability to use an Atari joystick on the Z-100.



SILENT RUNNER

SILENT RUNNER has the player climbing ladders, crossing ropes, and digging holes in order to collect the gold sacks distributed around the various display screens. There are also fierce creatures that are out to get you and prevent you from collecting the gold. Play can continue through 99 different screens, plus an editor allows you to design your own custom screens. You can control the speed of the game and high scores are saved to disk.

To avoid these fierce creatures, you can dig a hole and jump through it before it closes, or trap the monsters in them temporarily. Monsters will get stuck for a little while and, if you time it right, they will be destroyed - but only to come back out their little door again. This has the sole effect of delaying them slightly, or maybe moving them to another part of the board. If you get stuck in a hole, you will suffocate, lose a man, then start the level over. You can use ropes; the monsters can't.

The joystick works like you would expect it to, with up, down, left, and right all in their normal places.

However, since there is only one button on a joystick, you control the hole digging a bit differently. When the joystick button is depressed, a hole is dug to whatever side (left or right) the joystick is directed toward. This will dig a hole in front of you if you are running and hit the button.

Hint: For accurate digging, center the joystick, press the dig button, and then move the joystick in the appropriate direction.

The joystick must be the digital (switch) type and connected to the Z-100 parallel port (J3).

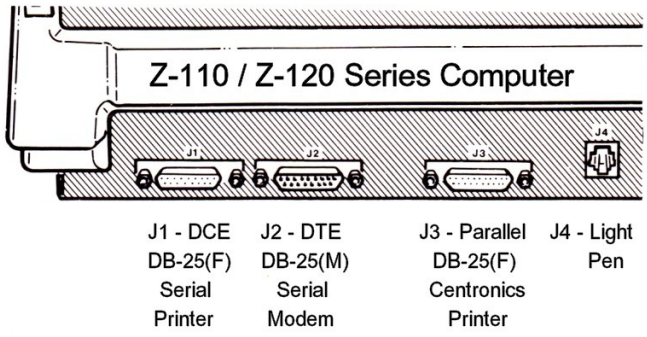


Figure 1.
Z-100 Parallel Port (J3)

No device driver is needed and joystick operation does not disable the keyboard - a much slicker operation than the arrangement in STARHAWK (another AVES game).

Joystick operation requires a special adaptor cable. See the next section.

Some final comments:
- The button on my joystick in SILENT RUNNER was very erratic on my computer. In some areas it dug a hole, in others, it would not - where the keyboard would work fine (flaky joystick?).
- I have not tried a mouse on either game, but I think it would be less natural in operation than the joystick.

Joysticks on the Z-100!?

So, what is all this foofaraw over operating a joystick on the Z-100's parallel port?

Well, the Z-100's parallel port is output only, except for certain status signals that the computer must recognize from the printer or other peripheral device. You know the signals - busy, off, out of paper, etc.

Well, the people of Audio Visual Entertainment Software have cleverly used these same signal lines to operate the joystick!

So, let's look at the hardware connection first.

The joystick must be a digital (switch) type joystick, similar to the joysticks that came with the early ATARI games that played on a television screen. These came with a 9-pin connector that must be adapted to the Z-100's 25-pin parallel port, J3.

The joystick's cable is somewhat short for comfortable use on the Z-100, so the adaptor cable could also serve as an extension cord (about 2 foot long, is best).

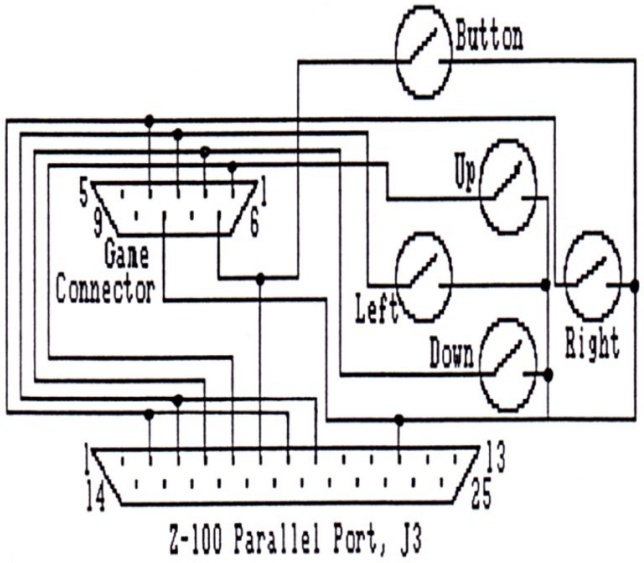


Figure 2.
Z-100 Joystick Circuit

According to the Z-100 Technical Manual, the printer port is a parallel port with handshaking capabilities. The pinout of the parallel port, J3, is defined as:

PIN:	SIGNAL NAME:	FUNCTION:
1	STROBE	A pulse that clocks data
2	PDATA1	Data to the peripheral
3	PDATA2	Data to the peripheral
4	PDATA3	Data to the peripheral
5	PDATA4	Data to the peripheral
6	PDATA5	Data to the peripheral
7	PDATA6	Data to the peripheral
8	PDATA7	Data to the peripheral
9	PDATA8	Data to the peripheral
10	ACKNLG	Acknowledge signal from the printer
11	BUSY	Printer not ready, when this signal is high
12	GROUND	Ground
13-14		(not used)
15	ERROR	Error signal from printer, when this signal is low
16	INIT	Pulse signal that initializes printer
17-25	GROUND	Ground

As can be seen from Figure 2, the switches of the joystick short the respective data line to pin 11, the BUSY signal line.

According to the manual, the BUSY line asserts if the printer cannot accept a data byte at the time a STROBE signal occurs (when data is ready to be sent to the printer). The BUSY line causes an interrupt which stops the printing until the BUSY line goes to its inactive state.

The Z-100's 68A21 Parallel port uses addresses E3, E2, E1, and E0. The BUSY signal is identified as bit zero of port E2.

Now, we had stretched my knowledge to the limit and I had to start stretching to reach three possible theories of operation:

First, somehow, each data line and switch could form a certain bit pattern to comprise the BUSY signal byte on pin 11. This byte could then be tested to figure out which switch is closed. This is doubtful, however, because the best I can figure out, the BUSY signal itself is just a Bit with two levels or values, high or low. Scratch this one.

Second, the 68A21 device's data lines can be programmed as input, as well as output. However, these data lines are coupled to the connector pins through a 74LS244 device, which limits the lines to output only. Scratch this one, also.

Third, it could be possible to program an output strobe signal to send to the joystick via the parallel port's pins 2 thru 6 (or more) in order, such that if the BUSY line goes high at any time, the output line at the time would have caused the change in the BUSY signal. Knowing this, the program could then recognize which switch was closed and take the appropriate action.

This third method would cause delays in the program over the other methods, but this would have minimal impact on the type of program like SILENT RUNNER where screen activity is already relatively slow.

Normally, eight bits of data are sent to the printer via signal lines PD1-PD8, pins 2 thru 9 of the 25 pin connector. PD1 and PD2 are controlled by bits 0 and 1, respectively, of port address E0. While PD3 thru PD8 are controlled by bits 2 thru 7, respectively, of port address E2.

Could this strobe signal be as simple as sending a 00000001 to port E0, a 00000010 to port E0, a 00000100 to port E2, a 00001000 to port E2, and a 00010000 to port E2 (for 5 switches), in a series, and waiting for a BUSY signal after each? After each series of output/input, take time to do any other activity, then start the series again?

Food for thought, and worth experimenting with. If any of you have other ideas, let me know. I hope to have more information on joysticks next time. And that is where I ended it...

In the last issue, we discussed the use of a digital joystick from the popular Atari game computer on the Z-100's parallel port.

Of course, the Z-100's parallel port is output only, except for certain status signals that the computer must recognize from the printer or other peripheral device, such as busy, off, out of paper, etc.

But these status signals can be used for the Z-100 to sense which switch of the joystick is closed and process the signal accordingly.

As we saw from Figure 2, the switches of the joystick short the respective data line to pin 11, the BUSY signal line.

Haven't Got an Atari Joystick?

John Anderson published an article in the October 1982 issue of "*Creative Computing*" that describes an excellent alternative, using pushbuttons.

This has several advantages over the conventional joystick:

- On the Z-100, you can eliminate the need to play with a 9-pin connector. Wire it directly to a 25-pin connector using the diagram above.

- You can make the cable longer and it will be more reliable because of the fewer cable connections.

- On some types of games, the pushbuttons are more effective than a joystick. The Asteriods, Space Invaders, Galaxian, and other games in arcades actually used buttons rather than joysticks.

John used Radio Shack momentary contact push-button switches positioned in a Radio Shack project case.

There are two common button configurations. The first is the "classic" Asteriods format.

```
                (U)
      (L)   (R)         (D)   (T)
```

The second is more like the positioning relationship of a joystick, a "clock-directional" format:

```
            (U)
      (L)         (R)         (T)
            (D)

      Where:
            (T) = Trigger
            (L) = Left
            (R) = Right
            (D) = Down
            (U) = Up
```

Theory of Operation:

As can be seen from Figure 2, the wire common to all the switches is connected to the parallel port's BUSY signal.

The Z-100's 68A21 Parallel port uses addresses E3, E2, E1, and E0. Ports E0 to E3 are 224 to 227 decimal.

The various signals of the 68A21 Parallel Port are shown in Figure 3. Of concern to us are:

```
    CA1 = LTPNSTB (Light Pen Strobe)
    CA2 = QVIDINT (Latched Vertical Sync)
    CB1 = ACK (Printer Acknowledge Signal)
    CB2 = BUSY (Printer Busy Signal)
```

The 68A21 and associated circuitry perform three functions:

```
      * Parallel Printer Port
      * Light Pen Port
      * Couples video retrace signal to CPU
```

The 68A21 is configured as a parallel printer port. The CPU programs the 68A21 and controls it during data transfer.

This printer port uses portions of both port A and port B in the 68A21. The eight bits of data out to the printer, PD1-PD8 and pins 2 thru 9 of the 25 pin connector, are assigned to port address E0, bits 0 and 1, and to port address E2, bits 2 through 7 respectively.

Data is latched at the printer by pulsing the STROBE signal (Port A, bit 2). The printer may respond by activating the BUSY signal, which can be interrogated for a "level" (zero) condition by reading Port B, bit 0, or for a transition by appropriate use of the CB2 input and control bits. The printer may also respond by pulsing the ACK line, which may be detected through use of the CB1 input and the CB1 control bits.

The printer error signal, ERROR, is read by Port B, bit 1, but will not be used here.

So, in short, the BUSY line asserts when data is sent to the printer port via PD1 thru PD8. The BUSY line is identified as bit zero of port E2 (226 decimal).

By strobing an output strobe signal to send to the joystick via the parallel port's pins 2 thru 6 (or more) in order, such that if the BUSY line goes level (zero) at any time, the output line at the time would cause a change in the BUSY signal. Knowing this, the program could then recognize which switch was closed and take the appropriate action.

68A21 Parallel Port (E0-E3)

Port Address	7	6	5	4	3	2	1	0	
E0(CRA2 = 1)	$\overline{\text{CLPHT}}$	$\overline{\text{LPSWT}}$	$\overline{\text{CVINT}}$	VIDINT	$\overline{\text{INIT}}$	STROBE	PD2	PD1	Peripheral Register A
E0(CRA2 = 0)	1	0	1	0	1	1	1	1	Data Direction Register A
E1	IRQA1	IRQA2	CA2 Control			CRA2	CA1 Control		Control Register A
E2(CRB2 = 1)	PD8	PD7	PD6	PD5	PD4	PD3	$\overline{\text{ERROR}}$	BUSY	Peripheral Register A
E2(CRB2 = 0)	1	1	1	1	1	1	0	0	Data Direction Register B
E3	IRQB1	IRQB2	CB2 Control			CRB2	CB1 Control		Control Register B

Figure 3.
68A21 Parallel Port (J3)

~~~~~~

This time consuming method slows a program considerably, but this would have minimal impact on the type of program where screen activity is already relatively slow, such as our "SILENT RUNNER".

Software:

Our first job is to see what happens at port address 226, bit 0, when we send various output signals, with the joystick attached. I am using a ZBASIC program, but other languages could be used just as easily.

```
100 PRINT "Test of the parallel port."
110 PRINT "Runs until a CTRL-C is typed."
120 REM Send a number out and read port 226 (E2).
200 OUT 226,#: REM Where # is the decimal number
    to send to port E2
210 A=INP(226): REM Check BUSY bit
300 PRINT A;:GOTO 200
```

This simple routine would send a number (#), of your choosing to port address 226 and return a number from the command INP(226) as a value of A. It would then print this value. If a switch of the joystick was closed at any point, the value of A would drop one unit as a reaction.

This worked great for checking the functions of Up, Down, and Fire. So we knew the input signal, INP(226) was reacting to different input numbers and joystick functions. The BUSY signal went to zero if certain operation of the joystick occurred.

But Left and Right did not work, because these used the first two bits of port E0 (Address 224).

Modifying line 200 to:

```
200 OUT 224,#
```

produced the desired results, but had a serious problem. It seems that any number output to port 224 causes the cursor to freeze upon exiting the routine! It also caused the value of INP(226) to react differently:

For example, for OUT 224,#:

Out DEC#:	Binary Value:	Normal Value of A:	Binary Value of A:
1	00000001	255	11111111

Moving left alone caused A to drop to 254 (11111110)

2	00000010	255	11111111

Moving right alone caused A to drop to 254 (11111110)

However, upon testing each number and exiting, the cursor would reappear and then freeze somewhere on the screen. And, even though you could type in commands and list the routine, the cursor never moved and the commands or routine would display elsewhere on the screen, in
~~~~~~

columns, but not at the left margin! The only fix was to reboot!

I have NOT been able to resolve this problem. I would be interested in knowing if this is experienced in other languages, or if it is a bug to ZBASIC alone.

A workaround was simple. In the wiring diagram of the 25-pin connector, I jumpered pin 2 to pin 7 and pin 3 to pin 8, allowing us to use only port 226 for all our joystick functions. It also left one pin left, pin 9, for an additional function switch, if you were building your own joystick.

I left pins 2 and 3 connected to permit using the joystick with the AVES games discussed earlier.

Using the above test routine now gave the results in the following paragraphs. Only one number can be tested at a time. But, to shorten the listed results, I'm going to group the numbers. Except for the first three numbers output, 1-3, the numbers are in groups of four, 4-7, 8-11, etc. The numbers in each group had the same result.

Out DEC#:	Binary Value of First #:	Normal Value of A:	Binary Value of A:
1-3	00000001	3	00000011

Pressing the Fire button or moving in any direction caused A to drop to 2 (00000010).

4-7	00000100	7	00000111

All but Down caused A to be 6 (00000110). Thus remember 4 for "Down".

8-11	00001000	11	00001011

All but Up caused A to be 10 (00001010). Thus remember 8 for "Up".

12-15	00001100	15	00001111

All but Up or Down caused A=14 (00001110).

16-19	00010000	19	00010011

All but Fire caused A=18 (00010010). Thus remember 16 for "Fire".

20-23	00010100	23	00010111

All but Fire & Down caused A=22 (00010110).

24-27	00011000	27	00011011

All but Fire & Up caused A=26 (00011010).

The tests continued until we found that:

```
Number 32 singled out Right
Number 64 singled out Left
```

These gave us our 5 functions of the joystick. Number 128 would single out another function, if one existed.

It was easy to then write a program to check out the theory discussed above and demonstrate joystick use.

```
100 PRINT "Test a joystick routine."
110 PRINT "It will run until a CTRL-C is typed."
120 REM Strobe each output in turn.
130 REM Get BUSY bit; AND with 1; and check status.
200 OUT 226,0: REM Clear port E2
210 A=INP(226) AND 1: REM Check BUSY bit
220 IF A=1 GOTO 600
300 OUT 226,4: REM Output 00000100 to port E2
310 A=INP(226) AND 1: REM Check BUSY bit
320 IF A=1 THEN PRINT "Down": GOTO 600
350 OUT 226,8: REM Output 00001000
360 A=INP(226) AND 1: REM Check BUSY bit
370 IF A=1 THEN PRINT "Up": GOTO 600
400 OUT 226,16: REM Output 00010000
410 A=INP(226) AND 1: REM Check BUSY bit
420 IF A=1 THEN PRINT "Fire": GOTO 600
500 OUT 226,32: REM Output 00100000
510 A=INP(226) AND 1: REM Check BUSY bit
520 IF A=1 THEN PRINT "Right": GOTO 600
550 OUT 226,64: REM Output 01000000
560 A=INP(226) AND 1: REM Check BUSY bit
570 IF A=1 THEN PRINT "Left"
600 GOTO 200
```

For some reason, lines 200-220 are needed to clear the port. There is a problem with some spurious signals, but I have not found the reason.

Let's see what happens with a practical example.

This program begins with a box on the screen. The joystick controls the movement of the box, left/right and up/down. With the addition of a second button, it could also control the size of the box, but we only have one button. So I've made it a toggle - between up/down movement and size adjustment. The result is an excellent example of how a joystick can be used.

```
10  CLS: DEFINT A-Z
20  PRINT "This is a test of a joystick routine."
30  PRINT "Runs until {Q} (Quit) or {E} (End)."
40  PRINT:PRINT "The box moves in the direction of
    the joystick."
50  PRINT "The FIRE button toggles between up/down
    & fore/aft movement."
100 X=200:X1=X:Y=200:Y1=Y:Z=4:Z1=Z:D=0
110 A$="U20R20D20L20" 'Make a box
150 IF Z<1 THEN Z=1 '"S" cannot be zero
160 PSET (X1,Y1),0:DRAW "S"+STR$(Z1)+A$ 'Erase box
170 PSET (X,Y),7:DRAW "S"+STR$(Z)+A$ 'Display new box
200 'Strobe each output in turn.
210 'Get BUSY bit; AND with 1; and check status.
220 OUT 226,0 'Clear port E2
230 A=INP(226) AND 1 'Check BUSY bit
240 IF A=1 GOTO 600
300 OUT 226,4 'Output 00000100 (DOWN) to port E2
310 A=INP(226) AND 1 'Check BUSY bit
320 IF A=1 AND D=0 THEN Y=Y+10: GOTO 600 'Moves down
330 IF A=1 AND D=1 THEN Z=Z-1: GOTO 600 'Decrease size
350 OUT 226,8 'Output 00001000 (UP)
360 A=INP(226) AND 1 'Check BUSY bit
370 IF A=1 AND D=0 THEN Y=Y-10: GOTO 600 'Box moves up
380 IF A=1 AND D=1 THEN Z=Z+1: GOTO 600 'Increase size
400 OUT 226,16 'Output 00010000 (FIRE)
410 A=INP(226) AND 1 'Check BUSY bit
420 IF A=1 AND D=0 THEN D=1: GOTO 600
430 IF A=1 AND D=1 THEN D=0: GOTO 600
500 OUT 226,32 'Output 00100000 (RIGHT)
510 A=INP(226) AND 1 'Check BUSY bit
520 IF A=1 THEN X=X+10: GOTO 600
550 OUT 226,64 'Output 01000000 (LEFT)
560 A=INP(226) AND 1 'Check BUSY bit
570 IF A=1 THEN X=X-10
```

```
600 E$=INKEY$: IF E$="" GOTO 150
610 IF E$="q" OR E$="Q" GOTO 700
620 IF E$="e" OR E$="E" GOTO 700
630 GOTO 150
700 END
```

Well, this program works well, but it still has a problem with spurious inputs from somewhere.

The AVES games that use the joystick do not seem to have a problem using port 224 or with spurious inputs, although "SILENT RUNNER" does not always seem to recognize a selection from the joystick.

I suspect that my joystick is erratic and perhaps this would work better with a different one. I hope one of you can confirm my issues, or provide ideas on how to make the joystick operation more efficient?

Have fun with this project.

If you have any questions or comments, please email me at:

z100lifeline@swvagts.com

Cheers,

Steven W. Vagts