Tynemouth Software

TYNEMOUTH MINSTREL JOYSTICK

OVERVIEW

This is a single 9 way D Kempston compatible joystick port for Minstrel 2, Minstrel 3 or ZX81.

It is designed for the Minstrel Expansion Bus, but could be built with an edge connector for direct connection to one of those machines.

PARTS LIST

CAPACITORS – CERAMIC RATED 6.3V OR HIGHER

3 x 100nF axial *(usually marked 100n or 104)*

RESISTOR ARRAYS – ALL ¼W 5% OR BETTER

1 x 8 commoned 10KΩ resistors *(usually marked 9X-1-103LF)*
Dot on package and square pad on PCB indicate pin 1

SEMICONDUCTORS – NEW TEXAS INSTRUMENTS PARTS RECOMMENDED

1 x 74HC138
1 x 74HC240
1 x 74HC688

CONNECTORS / JUMPERS

2 x 23 way 0.1” straight pin header or 2x23 way 0.1” card edge connector
1 x 9 way D Male solder bucket or straight PCB pin connector
1 x 16 pin and 2 x 20 pin IC sockets *(Optional, turned pin recommended if fitted)*

ASSEMBLY

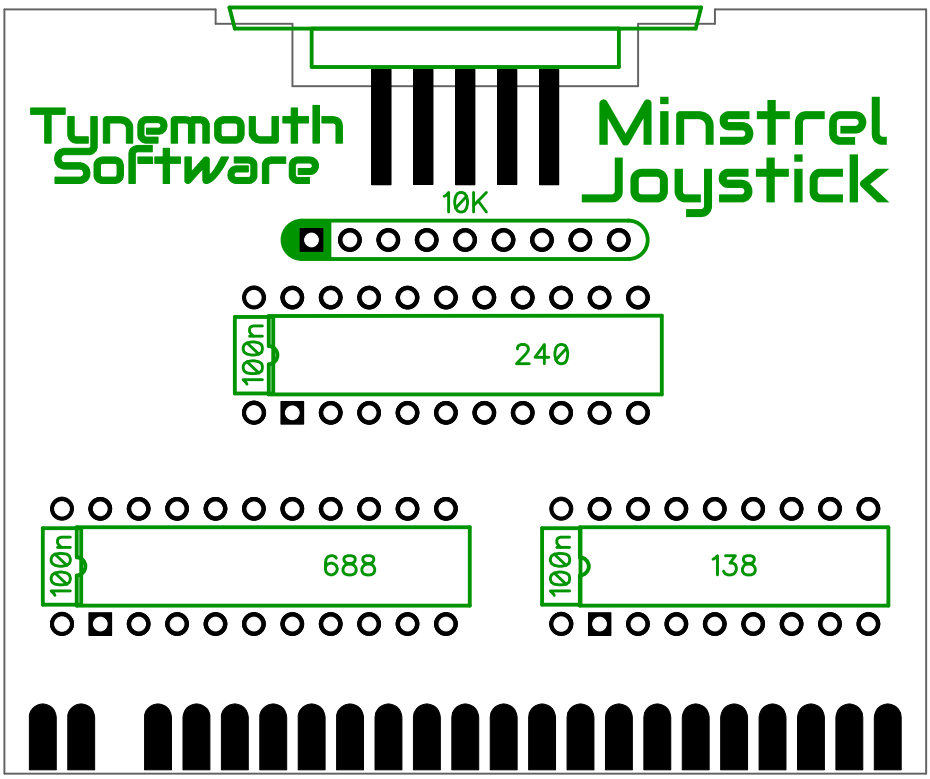
Assembly should be straight forward, starting with the axial capacitors, then ICs (or IC sockets if used), and resistor array.

The pin header or edge connector is soldered to the pads at the bottom of the board. In both cases, the third pin from the left, front and back, should be removed for polarisation.

The 9way D connector will fit into the PCB at the top. Solder pads are provided near the “ears” of the connector, and can be used to solder to the board for a stronger mechanical connection. You may need to use additional flux and turn up your iron to solder to the connector shell.

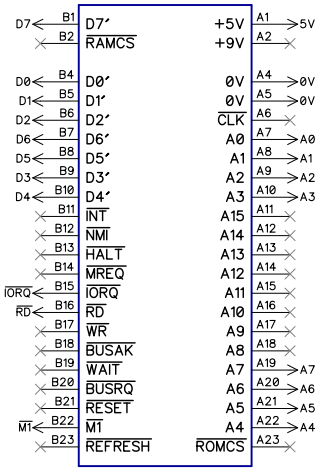
Tynemouth Software

COMPONENT PLACEMENT



SCHEMATIC

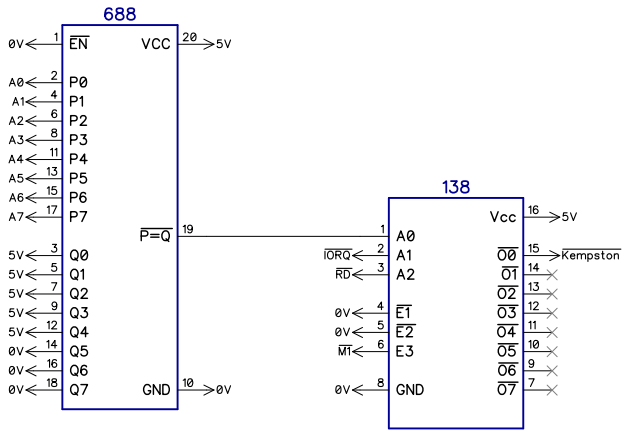
MINSTREL EXPANSION BUS



The signals shown are connected to the Minstrel Expansion Bus or ZX81 edge connector.

Tynemouth Software

ADDRESS DECODING



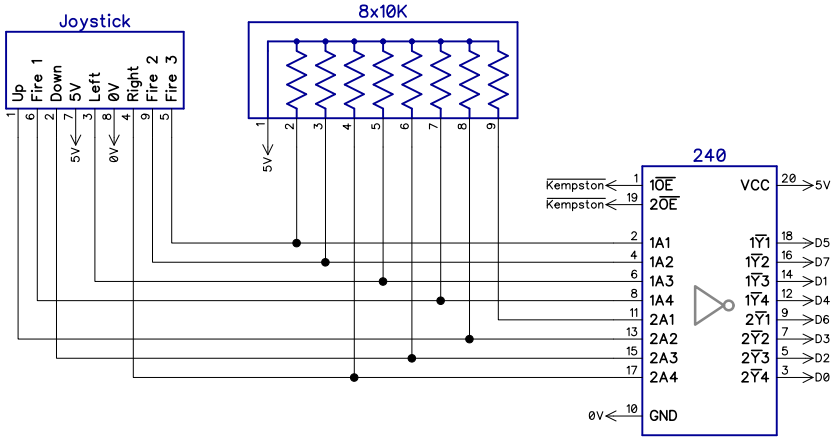
The standard address for a Kempston compatible joystick is 31 decimal, 0x1F hex.

This address is fully decoded by the 74HC688 to give a low signal when that address is accessed.

The 74HC138 will give a low output on the Kempston signal when the correct address is accessed, IORQ is low and RD is low and M1 is high.

This decodes as an IO read access to that address, which is not an interrupt acknowledge.

JOYSTICK INTERFACE

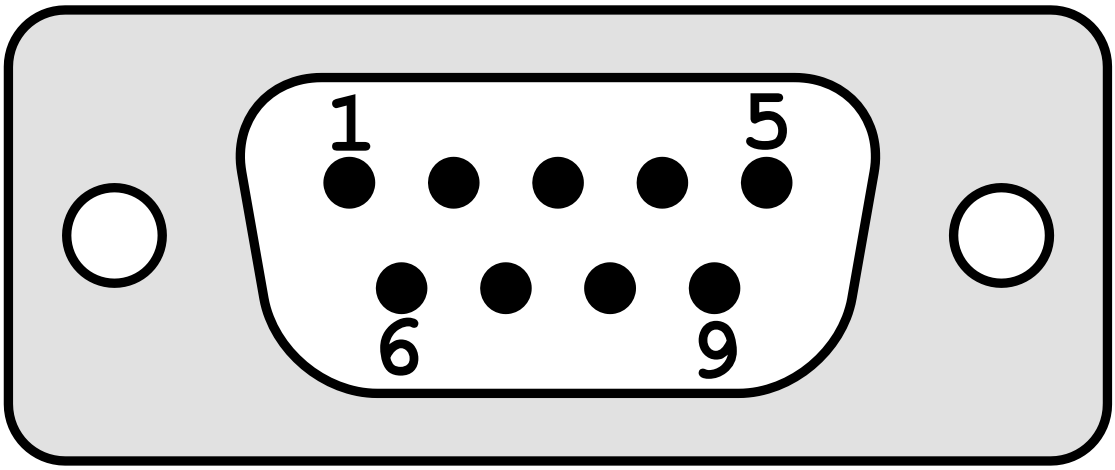


A simple 8 bit IO port is formed using the 74HC240. This is enabled by the decoded IO read from above. The 240 is an inverting buffer. The signals on it's inputs are pulled high by the resistor array and connected to the 9 way D joystick port.

When an attached Commodore / Atari style joystick is moved in a direction or fire is pressed, the corresponding signal is pulled low. The corresponding output pin is set to a 1, where the remaining bits read as a 0.

Tynemouth Software

9 WAY D JOYSTICK PORT



The joystick connector is the standard 9 way D pinout used on Atari and Commodore systems, and other Kempston interfaces etc. (note this is not the same as the Atari 5200, Spectrum +2 or Sega Genesis).

Pin	Signal	Bit	Value (Hex)	Value (Dec)
1	Up	3	08	8
2	Down	2	04	4
3	Left	1	02	2
4	Right	0	01	1
5	Fire #3	5	20	32
6	Fire #1	4	10	16
7	5V	-	-	-
8	0V	-	-	-
9	Fire #2	7	80	128
-	Pull Up	6	-	-

The original interfaces would usually not connect the upper three bits. Here bit 6 is only ever pulled high, so will always read a 0. Bits 5 and 7 are connected to the alternate fire button inputs fire#2 and fire#3. There are not connected in most joysticks, so will also normally read 0, but these signals are connected in case they are useful in some cases.

READING THE PORT

ASSEMBLER

The port can be read using the IN instruction.

```
IN A,(31)
```

The value can then be masked and processed as required.

The result will be a bitmask with 0 for any signal not currently active, and 1 for any direction or fire button input.

E.g. Down and Fire #1 will read as 0001 0100 or 0x14

BASIC

Accessing the port in BASIC is a little more difficult as the IN command was only added to the BASIC of the ZX Spectrum, it is not available in ZX81 or ZX80 BASIC.

To get around this, a small amount of assembler must be entered. 5 instructions, 6 bytes in total. The additional instructions setup the value to readable to BASIC as the result of a USR call.

```
XOR A
LD B,A
IN A,(31)
LD C,A
RET
```

There are various ways to do this. This example sets up a REM statement at line 1. The 6 bytes of assembler will be stored in the 6 characters of this statement.

You would normally use a loader program, but since this is only a few bytes, a series of POKE statements is easier.

The following code will place the assembled code into the REM statement when run.

```
 1 REM JSREAD
10 POKE 16514,175
20 POKE 16515,71
30 POKE 16516,219
40 POKE 16517,31
50 POKE 16518,79
60 POKE 16519,201
```

After running, you can see the letters JSREAD have been replaced by characters representing the values poked into memory.

```
 1 REM ■?<=3?TAN
10 POKE 16514,175
20 POKE 16515,71
30 POKE 16516,219
40 POKE 16517,31
50 POKE 16518,79
60 POKE 16519,201
```

Lines 10-60 can now be deleted. Line 1 must remain intact, and the first line of the program (as it must stay at memory address 16514).

```
 1 REM ■?<=3?TAN
10 PRINT USR 16514
```

In its simplest form, line 10 shows the command to call the assembler and print the result.

In most cases, you would assign the value read to a variable, here J. The value of J can then by tested to determine what the joystick is doing. Refer to the table on the previous page for the values that will be read.

```
 1 REM ■?<=3?TAN
10 LET J=USR 16514
20 PRINT AT 0,0;J;"  "
30 GOTO 10
```

If the user is moving diagonally or fire is pressed whilst moving, the value read will be the sum of all the active direction or fire values. It is left as an exercise for the reader to determine the best way to test for diagonals and the fire buttons being pressed at the same time as moving.