ParaPort v3.0 PC Parallel Port Controller

ParaPort is used to control 5 'real' inputs and 8 'real' outputs using a PC's Parallel port and may be configured to use LPT1 or LPT2. It runs on Windows 98se and Windows ME and XP. It should run on Windows 2000 or NT but this has not been tested.

ParaPort is programmable using an easy-to-learn language and may be left running to monitor up to 5 real-world inputs (such as a high temperature switch, etc) and to control up to 8 real-world outputs (such as turning on a fan, etc). The hardware used is up to you but a simple circuit with 5 input switches and 8 LED outputs and which simply plugs into the parallel port is shown later.

ParaPort is designed to work with a standard (ie inexpensive) I/O card so a spare card from virtually anything from an old 386 upwards should be suitable. I could have designed ParaPort round a more modern ECP port (which would have provided at least 8 inputs) but it would have required more complicated external hardware. These cards should still work with ParaPort but you will not be able to take advantage of their bi-directional data lines.

Basic Operation

When ParaPort is first started, you will see the Controller's "front panel" which, at this stage is somewhat bare-looking. Across the top of the panel (just below the main menu bar) is a row of "Tabs" labelled, from left to right:

```
Controller |Program Code |Labels & Events|Timers & Counters |Daily |Dates |I/O Tester |Quick Help
```

To get a general idea of how ParaPort works, click on the `I/O Tester` Tab. The right hand window shows a row of 'simulated' LEDs and a row of Buttons. Clicking on any of the buttons will turn on the LED and the appropriate output line will go 'High' (ie +5v). Clicking the button again will turn the output off.

The two boxes below the buttons show the decimal and hexadecimal values represented by the outputs. You may also enter values in these boxes and, when you press the `Send Dec` or `Send Hex` button, the appropriate LEDs will light and the corresponding outputs will go high.

The left hand panel shows the current state of the five inputs. You can't do anything with this panel unless you have real inputs connected to the parallel port. If you do have switches connected, the 5 LEDs will indicate which switches are open and which are closed. Note that, because of the way the PC's hardware works, these inputs are *high* when the switches are open so the switches should connect the inputs to Ground (0v) when they are closed.

Under the Main Menu Item 'Port' there is an option for ParaPort to reverse this 'internally'. Because 'Active Low' inputs can be confusing, the default action of the controller is to reverse the action to 'Active High'. Throughout this Readme file, I will refer to an input as being *'True'* or *'not True'*. When operating as 'Active High', *True* would indicate that the switch is, in fact, open.

Under the same Menu Item, you will also find where to select LPT1 or LPT2. These are assumed to be at the PC standard addresses of 0378H and 0278H respectively.

Also under this Menu Item is the option to Disable the outputs. This is to help when testing a Controller Program you have written. The Controller will still respond to inputs and will display the 'status' of the outputs on its front panel but will *not* send real outputs to the parallel port.

Programming the Controller

In order for the controller to do any useful work - monitoring inputs (or a combination of inputs) and setting outputs as required - you need to program the unit using a simple programming language. The following explanation assumes that you have some inputs suitably connected to the parallel port - otherwise you will be unable to test the examples.

There are 5 real inputs from the Parallel Port - **input 1** to **input 5** but each input can be used any number of times in a program. Each input can also be "inverted" by prefixing it with **not.** For example: **not input 3**

There are 8 real outpus to the Parallel Port - **out 1** to **out 8**. Do not use outputs more than once in a program otherwise you will get unpredictable results.

Click on the '`Program Code`' Tab. Then click on the top line inside the programming window and type the following on a single line:

input 1 out 1

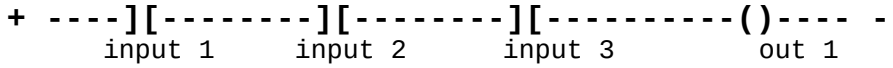
Now click on the '`Controller`' Tab and press the '`Run`' button. When you make switch one *True* output 1 will come on. Not very inspiring yet!

Click on the 'Program Code' Tab to return to the programming window and you will notice that you should not be able to make any changes to the code. This is because the program is still running so the controller is preventing you from making any changes.

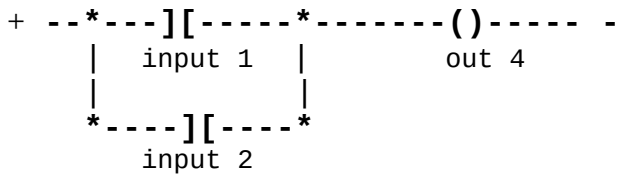
Press the '`Stop`' button - you can now type in the window. Change the line of code to:

input 1 and input 2 and input 3 out 1

Go back to the `Controller` window and press '`Run`' again. This time, switches 1 and 2 and 3 will need to be *True* before the output comes on. It's helpful to imagine the equivalent electrical circuit which, in this case, would be three switches in series:-



The following circuit would need input 1 **or** input 2 to be *true* to bring on the output:-



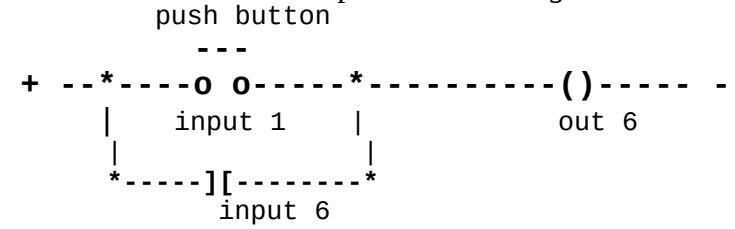
That would be programmed into the controller as:- **input 1 or input 2 out 4**

If you wanted the output to come on when, for example, input 1 was *true* but input 2 was *not true*, you would use the program line:- **input 1 and not input 2 out 4**

Valid "real" outputs are **out 1** to **out 8** but, because there are no real inputs from the parallel port higher than **input 5**, **out 6, out 7** and **out 8** can be used as 'virtual' inputs. That is, if **out 6** is energized, **input 6** will be *true*.
The same applies to **out 7** and **out 8.**

Latching Circuits

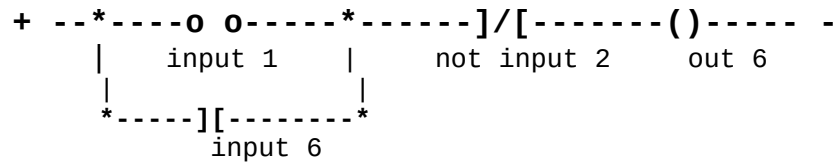
A common use for 'virtual' inputs is in *latching circuits.* Consider the following circuit:-



The output **out 6** will energize when **input 1** is *true* but will *remain* energized even after **input 1** is no longer *true* because the 'virtual' input, **input 6** will now retain the circuit. (Think of **out 6** as a relay with **input 6** as one of its contacts).

The equivalent programming line would simply be:- **input 1 or input 6 out 6**

Normally, a method of resetting the latch is needed:-



Input 2 is normally *true* (otherwise the output would *never* energize) but making it not *true* briefly will reset the latch by de-energizing **out 6** and making the 'virtual' **input 6** not *true.*

The programming line would be:- **input 1 or input 6 and not input 2 out 6**

Latching with 'virtual' outputs

There are times when a latching circuit is required but you do not want to use a real output (**out 6 - out 8**) for the purpose. To cater for this, the Controller has 7 additional 'virtual' outputs - **out 9** to **out 15.** These work in exacty the same way as **out 6** to **out 8**, in that they can be used as **inputs,** except there is no real output to the parallel port.

Timers

Sometimes, instead of *latching* a fleeting input, such an input needs to be *ignored.* For example, if an input was arranged to be *true* if the ambient lighting dropped below a pre-set level (perhaps with an output to turn on a light), it would be useful to ignore the input if it was only *true* briefly when the lighting level dropped due to a passing cloud..

For this purpose, the Controller provides four **Timers** each with separately adjustable time delays of between 1 and 65 seconds. (Actually between 0 and 65,500 milli-seconds!).

A Timer is programmed in the same way as 'virtual' outputs but is in the number range: **out 21** to **out 24.** The Timer starts timing when the **inputs** to it are *true* but the inputs must *remain true* until the delay is complete. If the inputs do not remain *true* during the delay, the Timer resets and no output occurs. When the inputs go *true* again, the whole time delay starts again. Once the Timer output has occured, it will remain on until the Timer's input becomes *not true* again.

The actual time delays are programmed by clicking on the `'Timers & Counters'` Tab and entering the times required in milli-seconds (ie 1000 = 1 second).

Inputs **input 21** to **input 24** will become *true* when the time delay is complete. The following programming lines show a simple example:-

input 1 and input 2 out 21
input 21 out 3

If inputs **input 1** and **input 2** remain *true* for the whole duration of the Timer, **input 21** will become *true* and the real output **out 3** will be energized.

The following code shows how to program a simple flashing output, in this case **out 3**:-

not input 22 out 21
input 21 and not input 22 out 22
input 21 out 3

Instead of flashing the real output **out 3**, it can sometimes be more useful to flash a virtual output, such as **out 9.** You could then include **input 9** in any program line that required a flashing output.

not input 22 out 21
input 21 and not input 22 out 22
input 21 **out 9**
input 9 and input 1 out 2
input 3 or input 4 and input 9 out 4

Counters

The controller provides 4 counters. Counters are incremented by activating outputs in the range **out 31** to **out 34.** For example, in the following line of code, counter 31 will increment each time **input 5** becomes *true.*

input 5 out 31

The counters can be set with pre-set values under the '`Timers & Counters`' Tab and, when the pre-set value is reached, **input 31** to **input 34** will become *true.* In the following example, if the pre-set count for Counter 31 is set to 5, **input 2** will need to become *true* 5 times before **out 8** comes on.

input 2 out 31
input 31 out 8

Unlike Timers, a Counter will remain energized once its preset count is reached. In order to reset the Counter back to its initial value, a 'reset' needs to be output to **out 41** to **out 44.** (That is, **out <counter + 10>)** For instance, to reset the above example:-

input 2 out 31 (Input 2 must become *true* the preset number
input 31 out 8 times before out 8 comes on)

input 4 out 41 (input 4 will reset the counter whether or not it has finished counting)

In addition to simply counting an input, Counters can be used to extend the range of Timers. Although a Timer has a maximum delay setting of about 1 minute, the maximum count for a Counter is 999. By arranging for a Timer to repeatedly re-run until a Counter reaches its count, times of over 16 hours can be acheived - although it is doubtful that it would be very accurate over this length of time.

Logging Events

The main window on the Controller's front panel can be used to record events. Pressing the `Start` and `Stop` buttons automatically logs the times of those events but other events can be logged as well. Events are logged by activating outputs **out 51** to **out 55**. The text which is to be displayed is entered into the Text boxes marked 'Event Text' under the '`Labels & Events`' Tab.

To log an event, use a line such as, for example:-

input 1 or input 9 out 51

To log a switch becoming *true* and *not true* again, use lines such as:-

input 4 out 51
not input 4 out 52

Up to 200 events may be logged before the window is cleared and logging re-starts. The log may be saved to a file under the Main File Menu.

Sound Event

Under the '`Labels & Events`' Tab you will find a Text input box labelled '`Sound Event`' with '`Browse`' and '`Test`' buttons. You can use this to load a Wave File (.wav) for one special event.

This event is triggered when **out 91** is activated. The event is also recorded in the main event window on the Controller's front panel. If no sound has been selected, the controller will play the Windows default sound.

Built-In Flash

To avoid having to use two of the **Timers** to create a flashing output, version 1.2 now includes a built-in flashing input: **input 99**. For example:

input 99 out 1

will make output **out 1** flash at an interval set under the `Timers & Counters` Tab. Obviously, **input 99** can be used anywhere that any other **input** can be used.

Dates and Times

Version 2.0 onwards includes 8 new 'Date & Time' Events: **input 61** to **input 68.**

Inputs **input 61** to **input 68** may be pre-set to any Date and Time to over 100 years into the future (!!!). When the *current* date and time equals the pre-set values, **input 61 - input 68** will become *True.* The input will then remain *True* for a time determined by the setting of the Event's "On For (Mins)".

The dates and times are set under the `Dates` Tab. Click inside the 'Date & Time Event' that you want to change and then use the 'Set Date' and 'Set Time' controls to change the values. The Controller will not accept a date or time that is 'earlier' than the current date and time.

For example, if the 'Date & Time Event 61' is set to 28/11/2001 14:35 the following program code line:-

input 61 out 1

... would turn on **out 1** on 28th November 2001 at 14:35

The inputs **input 61** to **input 68** become *not true* at the Date and Time set in the 'Off' box.

If you want the input to remain *true* indefinately, check the corresponding checkbox.

To make the 'Set Date' and 'Set Time' controls easier to use, they will initially show the *current* date and time but they will not update while they are being displayed. To set them to the current date and time at any time, press the '`Time Now`' button.

Daily Timers

Added in version 2.1 (and modified in version 2.2) is the '`Daily`' Tab. Use this Tab to enter inputs in the range **input 71** to **input 78** to set the times for events which will occur every day. Like the Dates & Times Events, the length of time for which they remain *true* is set in the associated "On For (Mins)" box. As these Events operate every

day, it is not possible to set the On time beyond 23:58 with the minimum "On For" time of 1 minute.

For events that you want to occur every day, but which span across midnight, it's easy to arrange for a *latch* to "bridge the gap". For example:-

The following code will switch on **out 1** between 23:30 on one day and 01:10 on the next:-

Set **input 71** to 23:30 with a 29 Minute "On For" time.
Set **input 72** to 00:01 with a 70 Minute "On For" time.

.

input 71 or input 9 and not input 72 out 9
input 9 or input 72 out 1

It works as follows:-

input 71 or input 9 .. .. **out 9**	Set the latch with **out 9**
.. .. **and not input 72**	Reset the latch when **input 72** becomes *True.*
input 9 or input 72 out 1	Turn on **out 1.**

With various combinations of **inputs** from Date & Time and Daily Events, the actual dates on which the Daily events occur can be controlled. Quite complex control schemes are possible for things such as home security.

The Labels & Events Tab

Input Labels

When the controller is running, it is helpful if the 5 inputs on the front panel have meaningful labels, such a "Low temperature" etc. Enter labels in the 5 text-entry boxes.
Below the text-entry boxes the two buttons (`Input Colour` and `Output Color`) allow you to change the background colour of the labels.

Output Labels

As for the **Input Labels** you can type descriptive labels for each output.

Event Text

If your program wants events (such as an input switching on, etc) to be logged on the main front panel window, enter the text for each event in these boxes.

Sound Event

Use the `Browse` button to select a WAVE file that will be played if **out 91** is energized. Use the '**play**' button to test if the WAVE file is what you want. You can use any .wav file -- either the existing Windows files or one you have recorded yourself.

Programming Issues

- Once you have written a program and tested it, you can Save it for later use by selecting the `Save All` option under the Main File Menu. The Controller will then attempt to re-load the same program automatically the next time it is started up. Under the Main File menu, there is also the option to `Load All`

- In the programming window, all the usual Windows clipboard options (Undo, Cut, Copy, Paste, etc are available by right-clicking in the window so no separate 'Edit' option is provided on the Main Menu bar.

- **Tip:-** If you get confused whether you should use **input** or **out**, think of it as **input *(from)*** and **out *(to)*.** For example, you energize a Timer or Counter with **out *(to)*** and read whether it has completed its "task" with **input *(from)*.**

  You check whether a Date & Time event has occured with **input (*from*).** Date & time are running all the time so there is no **out *(to)*** to start it! Similarly, the Flasher (**input 99**) is flashing all the time so there is no need to start it.

- Because I've kept the programming 'language' simple, there can sometimes be confusion in lines of code that contain more than two inputs. For example, **input 1 and input 2 or input 3 out 1** could be interpreted in one of two ways -- does **or input 3** apply to both other inputs or just **input 2** ?

  As the controller works sequentially from left to right, **or input 3** in the above example 'over-rides' *both* **input 1** and **input 2.** If **input 3** is *true*, the state of the other two inputs doesn't matter. In cases of confusion, the simplest thing is to try it and see!

Unexpected results can often be resolved by re-arranging the order of the inputs in the line of code. If that fails, the problem can usually be solved by splitting the line into two or more simpler ones by using a 'virtual' output (**out 9 - out 15**) to "carry over" the result of one line to the next.

- Currently, the Controller allows for a maximum of 100 lines of code. I'm sure that will be enough! Each line of code must be all on one line and each line *must* end with **out** `<number>`**.** Ending a line of code with **not out** <number> is ***not*** allowed. Although the Controller does not prevent you from using outputs any number of times, you should use each output only once otherwise the results are unpredictable.

- When the Controller is running, it will only show the status of the real inputs **(input 1 - input 5)** if your Program uses one (or more) of them. This is done to speed up the Controller so it isn't unnecessarily checking the inputs if they're not being used in your program.

- If you select the 'Disable Outputs' menu option while any real outputs are on, they will stay on even when the Controller Program tries to turn them off. The 'lights' on the Controller's front panel can only indicate what status they *would* have if it had control of the real outputs.

- Although the Controller checks for most typing errors in your program lines, it cannot know if the actual 'logic' of your lines is correct. Pressing the '`Stop`' button or the '`Close`' button will *usually* bring a 'rogue' program to a halt but, occasionally, the controller may get into such a tight 'loop' doing nothing that you will have to resort to the Windows Program Manager to 'End the Task'. (Press Ctrl + Alt + Del, highlight 'ParaPort' and click 'End Task').

- Because the Controller is operating in the Windows environment, it has to pass control back to Windows on a regular basis so that Windows can deal with any other programs you may have running. For this reason, *very* short pulses on the Controller inputs may be missed at times.

Files Created By The Controller

Each program you build in the Controller creates three files:-

1. The main **Controller Program File** (with the extension **.prg**) contains information about the colour and text of the Input and Output Labels, any text you have entered for the Events, the values you have set for the Timers and Counters, the name of the wave file you have selected and a filename for the code you have entered into the Program Code window.

2. The **Program Code File** (with the extension **.cde**). This is saved as a separate file so that you can try different code without having to re-configure all the Timers etc. It should be saved with the same filename as the main Controller Program File but with the .cde extension. If you use the `Save All` option, it will automatically be saved with the same filename the the **.prg** file.

3. The Controller also saves a file called **paraport.ini** . This file simply contains the name of the last-used **.prg** file and will automatically attempt to re-load it next time. This file is ***only*** saved when you save the main Controller Program File.

The Event-window log may be saved with any filename and will be given the extension **.log**

I've included a couple of demonstration **.prg** files which do not require any real inputs from the parallel port. Use the `File | Load All..` menu option to load these files to try if you don't have anything connected to the parallel port.

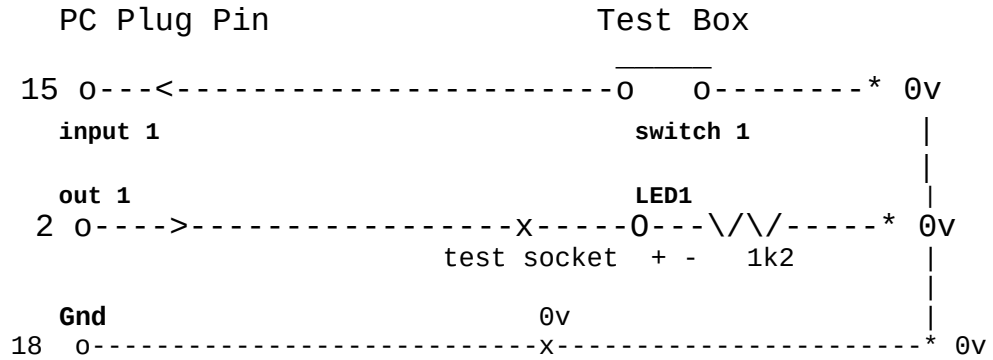
The `Load All` option will Load the entire program - that is the **.prg** file and the **.cde** file. The `Save All` option will save those files and **paraport.ini**.
The `Load Code` and `Save Code` options will just Load or Save the **.cde** file.

Hardware

To test the Controller, I plugged a second I/O card into a PC ISA slot and configured the parallel port for LPT2. As far as the controller is concerned, the IRQ used doesn't matter but the address *must* be 0378H for LPT1 or 0278H for LPT2 -- ie the standard PC addresses for those ports.

I made up a simple box with 5 switches for the inputs and 8 sub-miniature LED's (with 1k2 series resistors) and 8 miniature sockets for voltage measurements etc. The box is connected to the PC with a standard printer lead (with the printer end cut off and the lead connected to my box instead!).

I'll only show one input and one output in the following drawing so as not to complicate it. (I 'm hopeless at ASCII drawing!) :-



All the PC Plug Pins are as follows:-

Pin 15	**input 1**	**Pin 2**	**Output 1**
Pin 13	**input 2**	**Pin 3**	**Output 2**
Pin 12	**input 3**	**Pin 4**	**Output 3**
Pin 10	**input 4**	**Pin 5**	**Output 4**
Pin 11	**input 5**	**Pin 6**	**Output 5**
		Pin 7	**Output 6**
		Pin 8	**Output 7**
Pin 18	**0v**	**Pin 9**	**Output 8**

(To catch the unwary, Pin 11 is inverted inside the PC. The controller allows for this and inverts it back!)

The simple circuit above does not provide any electrical isolation between the PC and your own hardware. Take great care not to short any output to Ground or 0v.

It's been working fine for me for several weeks during the development of the Controller but playing with your PC's hardware MUST be AT YOUR OWN RISK.

If you install an I/O card at a different address, it is possible to set up ParaPort 'manually'. Firstly, start up Paraport and select either LPT1 or LPT2. Then select "Save All" and save the file with a name such as **test.prg**. Open the resulting file in NotePad (or other text editor) and, at the top, you will see the lines:

```
[ParaPort]
Port=LPT2
```

Edit the second line to the Base Address of your I/O card in Hexadecimal - including the '$' sign. For example - **ie, an example only! -**

```
[ParaPort]
Port=$0200
```

ParaPort will then use that address next time **test.prg** is loaded. It will use 0200H as the output address and 0201H as the input address - but note that only the 5 most significant bits of the input address are used.

I will be pleased to answer any questions and consider additions to ParaPort if you contact me at any of the addresses below.

The current version of ParaPort is 3.0 on 29 Dec 2004

Amateur Radio Packet Mail: G4VWL@GB7OAR

Email: john@vwlowen.demon.co.uk

www.vwlowen.demon.co.uk
