BV4627 User Guide

Rev	Change
Dec 2010	Preliminary
Jan 2011	Update Firmware version 1.1 (IR setup)
Jan 2011	Addition of Arduino Library
Nov. 2013	Added RPI I2C instructions, using notSMB

Resources (zip file) can be found on www.byvac.com/bv3 under the description of the 8 Way relay. This is in the category interface/output.

Introduction

This is an 8 way relay board that can be used for general purpose switching or timing. It has 4 interfaces so that is can be used with a PC or microcontroller or other electronic equipment in various ways. The relays have indicator lights to make servicing easier and are also capable of delayed switch on and off.

There are two versions of this board, the full version with USB, IR, Binary and I2C interfaces and a cut down version that does not have the USB and IR interfaces fitted.

Physical Interfacing

This device has no less than four interface options:

1) Binary interface
2) I2C interface
3) USB interface
4) IR interface

Only one interface at a time can be used and this is detected on reset. In the absence of all other interfaces the IR interface will prevail, provided that it has first been set up using the USB interface. This Guide will go through each interface in turn.

Binary Interface

This interface has been designed to make it easy to attach to another electronic device and control the relays. This will normally be a microcontroller but could just as easily be some discrete logic.

The interface is at the top of the board above the USB socket and consists of 6 pins.

Pin	Function
A3	Relay select
A2	Relay select
A1	Relay select

A0	Clock
GND	Ground
+5V	Power supply

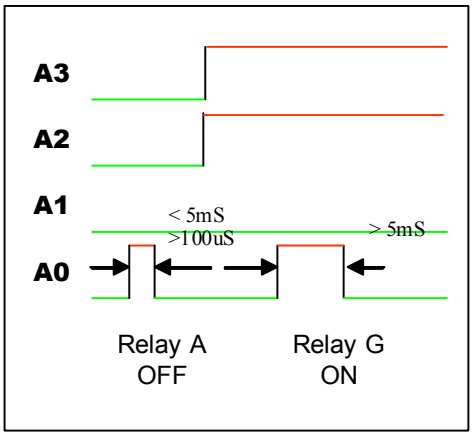
Power to the board is supplied by the +5V input. The logic will work down to 3V3 but the relays will not operate at that voltage to 5V is required.

Four output lines are required, three to select the required relay A through H and one for the Clock. The PCB is marked A0 to A3 and pins A1 to A3 are used to select the relay.

Rly>	**A**	**B**	**B**	**D**	**E**	**F**	**G**	**H**
A1	0	1	0	1	0	1	0	1
A2	0	0	1	1	0	0	1	1
A3	0	0	0	0	1	1	1	1

The clock line is A0 which is normally low and has a very simple scheme whereby if the clock is pulsed quickly (high / low in less then 5mS) this will turn the relay off. If longer then the relay will be turned on.

A0 Also has another important function; it must be held low during reset or power up in order to select the binary interface, one in this mode the device will stay in this mode until power cycled or reset.



The timing for the clock pulse is very flexible in that any length of clock less than 5mS will turn the relay off and any longer then 5mS (even 1 or 2 seconds) will turn the relay on.

Programming

To give an example of how this may be interfaces PIC32 Basic is used.

```
 3 constant A0$ "d1"
 4 constant A1$ "d3"
 5 constant A2$ "d5"
 6 constant A3$ "d7"
 7
 8 function r8_binit
 9     setport A0$ as out
10     setport A1$ as out
11     setport A2$ as out
12     setport A3$ as out
13     port A0$ 0
14 endf
```

Port lines D1,3,5 and 7 are used and they are all set to output. To make selection of the relays easier a large case statement is used.

This of course just selects which lines are on and off and so other arrangements could be made. For example to switch on relay 5 (F) the table requires that A1=1, A2=0, and A3=1 which is exactly what is selected in the case statement.

If consecutive ports were used say D0 to D2 then it would be a very simple matter of applying that value to the port, e.g PORTD=5. This also of course assumes that nothing else is connected to the port.

NOTE That the set up make sure that A0 is in the low state, this is the normal or idle state.

```
17 function r8_select(relay)
18     select relay
19         case 0
20             port A1$ 0: port A2$ 0: port A3$ 0
21         case 1
22             port A1$ 1: port A2$ 0: port A3$ 0
23         case 2
24             port A1$ 0: port A2$ 1: port A3$ 0
25         case 3
26             port A1$ 1: port A2$ 1: port A3$ 0
27         case 4
28             port A1$ 0: port A2$ 0: port A3$ 1
29         case 5
30             port A1$ 1: port A2$ 0: port A3$ 1
31         case 6
32             port A1$ 0: port A2$ 1: port A3$ 1
33         case 7
34             port A1$ 1: port A2$ 1: port A3$ 1
35     endselect
36 endf
```

The Clock again is quite straightforward.

```
38 function r8_bon(r1)
39     r8_select(r1)
40     port A0$ 1
41     wait 10
42     port A0$ 0
43 endf
44
45 function r8_boff(r1)
46     r8_select(r1)
47     port A0$ 1
48     port A0$ 0
49 endf
```

The function at line 38 will switch on the selected relays as follows: 'r1' is set to a value say 5 and this will select the relay by setting lines A1 to A3, in this case line 29 will select A1=1, A2=0 and A3=1.

Next A0 is taken high and held there for more than 5mS, in this case 10mS (line 41) just to be sure, it is then taken low again.

To switch the relay off there is no delay between taking A0 high and putting it back low again. This will switch the selected relay off.

The functions could be used something like:

```
r8_bon(1)  // switch on relays A,B and G

r8_bon(2)

r8_bon(7)

r8_boff(3) // switch off relays C and D

r8_boff(4)
```

Notes About the Binary Interface

You can only switch on and off relays, there are no timed facilities as there are on the other interfaces.

The interface is designed for simplicity and this takes priority.

The A0 line MUST be held low at start up in order to select the binary interface. There is a vey weak pull up resistor on this line (A0) at start up and so if it is left disconnected the binary interface will be ignored.

I2C Interface

This interface has been provided to give a standard microcontroller i/o to the device. The address of the interface is 0x64 (8 bit notation) 0x32 (7 bit notation). For more information about 7 & 8 bit addressing read this http://www.i2c.byvac.com/ar_trouble.php guide. The address can be changed via this or the USB interface if fitted.

This interface will allow timed delays to switch the relays adding flexibility and a degree of autonomy to the device.

The interface is at the bottom left of the board and consists of four pins.

Pin	Function
SCK	I2C Clock
G	Ground
SDA	I2C Data

+V	+5V Power

The power supply is provided for the board via the +5V and Ground. The Clock and data lines WILL require a pull up resistor somewhere on the I2C bus, this is normally provided by the master device.

Selection

This interface is selected by holding the SDA line high at switch on. This will happen normally if connected to an I2C bus as the pull up resistors on the bus will do this. If the interface is left unconnected at switch on there is a high value pull down resistor which will deselect this interface.

Programming

There are no special requirements for this device; it will act as a standard I2C slave. The device is command driven; following the address a command is expected. For a list of commands consult the data sheet.

It is also important to understand the number of bytes that are expected for each command as they differ. For example to switch a relay on requires 4 bytes after the address, the first to specify the relay, the second for on/off and the third and forth to specify a delay if any. When two bytes are used to represent a 16 bit number the high byte is always first. If for example 450 decimal was specified as a delay then convert to hex (using the Windows Calculator) = 0x1c2. This can now be sent as two bytes 0x01 and 0xc2.

There is a possible pitfall when using **commands 10 through 17** and that is the relays require a finite time to activate as they need to go through the timer even if the time is set to 1. If for example you set all of the relays to on and then immediately set a timed off the chances are that this will fail because the relays will not have had time to respond to the on command before receiving the off command. For this situation use command 19 or 20 to turn the relays on and then set the off time.

Examples Python - Windows and Raspberry Pi

Serial

This will only apply to the full version and both Windows and PRi will be the same. The only difference will be the name of the COM port.

I2C

Windows PC's do not normally have a built in I2C interface and so a BV4221_V2 device will be needed. For full details; see this link:

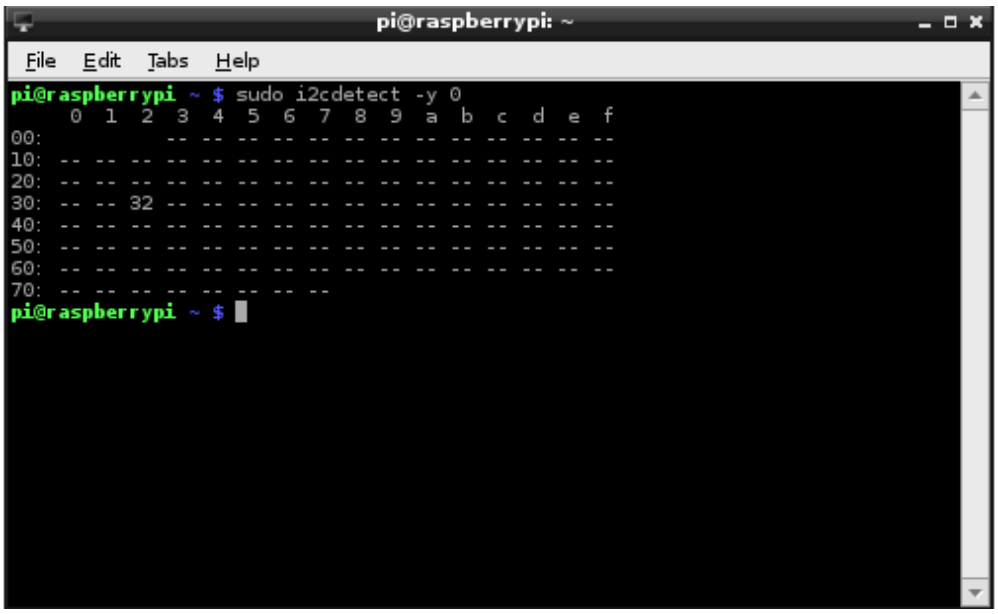
http://www.pichips.co.uk/index.php/Windows_I2C

I2C RPi

The RPi out of the box does not support I2C, this needs to be enables and software needs installing, full instructions for this are here:

http://www.pichips.co.uk/index.php/RPi_Software_Installation

Once installed and connected, open a console and type sudo **i2cdetect - y 0** OR sudo **i2cdetect -y 1**, the latter is for the newer RPi versions that use I2C bus 1



If you don't see '32' as above then something is wrong with either the connections or the software installations so check to try and find out what you have done wrong.

A simple test is now to download the test program 'bv4627.py' from the product documentation here:

http://doc.byvac.com/index.php5?title=Product_BV4627

and run it from the console. It will turn all of the relays on and then off in turn.

Examples PIC32 Basic

The following examples have been written in PIC32-Basic but this is an easy language to follow and so provides a good example.

```
19 // *********************
20 // send a command
21 // *********************
22 function r8_cmd(cmd)
23     i2cstart 1 ADR
24     i2cput(1,cmd)
25     i2cstop 1
26 endf
27
```

A useful function is the send command as most of the time a command is required. To turn all of the relays on for example would be r8_cmd(19). It works by first sending a start condition followed by the address which is stored in ADR (0x64). The red '1' is the I2C channel as there are two channels on the BV513 (which runs PIC32-Basic). The next step is to output the command to channel 1 and then issue the stop condition.

To turn all of the relays on, the I2C bus would see the following information:

s 0x64 19 p

Where s is the start condition and p is the stop condition.

Nearly all read commands will return 2 bytes, the high byte being first:

```
40 // *************************
41 // get value 16 bit
42 // *************************
43 function r8_get16(cmd)
44 dim rv
45     r8_cmd(cmd)
46     i2cstart 1 ADR+1
47     rv=i2cget(1,0)*256
48     rv=rv+i2cget(1,1)
49     i2cstop 1
50     result rv
51 endf
52
```

This is used in the form r8_get16(83), the command '83' is the get device ID that will return 4627. Line 45 will send the command to the device. Line 46 is the read address which is always 1+ the write address. At line 47 the byte received from the device with the i2cget command is multiplied by 256 because it represents the high byte. The next byte (low byte) is simply added to the first one. The result is a 16bit read from the I2C device.

The I2C bus would see this:

s 0x64 83 p s 0x65 r-2 p

Where 's' is the start condition, 'p' is the stop condition and 'r-2' gets two bytes.

```
52 // ***************************************************************
53 // turns relay on or off with a delay, set ron=1 for on and 0 for off
54 // ***************************************************************
55 function r8_rly(rly, ron, delay)
56 dim delh, dell
57     delh=and(rshift(delay,8),0xff)
58     dell=and(delay, 0xff)
59     i2cstart 1 ADR
60     i2cput(1,rly)
61     i2cput(1,ron)
62     i2cput(1,delh)
63     i2cput(1,dell)
64     i2cstop 1
65 endf
```

This is a slightly more complex example that uses commands 10 through 17 to give timed action. The delay is specified as two bytes, high byte first. The high byte is obtained form the number given (delay) by shifting 8 bits to the right and using that as the high value (line 57). The low byte (line 58) is simply the number with all of the other bits masked off.

The whole thing is then sent to the I2C bus. For relay 11, turned on with a delay of 300 the I2C bus would see:

s 0x64 11 1 0x1 0x2c p

300 split into high and low bytes is 0x1 and 0x2c.

Example Arduino

There is an Arduino library in the resource pack "Arduino_lib_bv4627.zip" which can be unzipped and placed into the Arduino environment, in the libraries folder. The

example uses the library to print out the Device ID, the Firmware version and a count down of the timer to relay 2.

The SDA line is connected to A4

The SCL line is connected to A5

The Arduino Nano +5V was used to power the device and 2 pull up resistors were used but these may not be necessary as the ATmega has pull up capabilities that are used on the port lines.

Upload and the library and run the sample, then press Serial Monitor which will restart the board and show the count down.

This has been tested on an Arduino Nano with the ATmege238 fitted.

Library functions:

BV4627 is the class and this is instantiated with the address of the device. Audrino uses 7 bit addressing and so divide the device address by 2, thus the address is 0x32.

click(char rly, char on, int del)

The word switch would have been better but this is a C keyword. This will turn on or off a specified relay 1 to 8 where 1 is relay A and 8 is relay H. Set on to 1 to turn the relay on and 0 to turn it off. The delay is only approximate and will slow down if there is a lot of serial activity but 13 will give approximately 1 second. The maximum value is 65535. Zero or one will give no delay.

int timer(char rly)

The value of the count down timer can be monitored (to see how much time is left before the relays switched). This will return the value of the relays count down timer.

off()

Turns all of the relays off

on()

Turns all of the relays on

bin(char rly)

This will set the relays on or off as determined by the supplied byte. A byte value of 0x45 will turn on relays A,C and G and all the rest off:

H,G,F,E,D,C,B,A
0,1,0,0,0,1,0,1

setaddress(char newAddress)

Sets a new I2C address for the device, the address will take effect when the device is next restarted and it will retain that address until changed again. NOTE: specify an 8 bit address for the new address. For example specifying a new address of 0x24 will give an Arduino address of 0x12.

int deviceid()

This will return a 16 bit number which is the device ID. For this device it will return 4627. This is useful for verifying the devices on the bus.

version(char *ver)

This returns the version number in the form ”major.minor” to the supplied buffer. The buffer must be gig enough to accommodate the string. Major and minor are numbers in string form.

reset()

Resets the device.

USB Interface

** The cut down version does not have the USB or IR interfaces fitted **

The USB interface is designed to be used with a PC and as such it presents itself as a COM port. It uses the FTDO chip from Future Technology (http://www.ftdichip.com/)

The latest drivers for this can be found on their website, the driver required is the Virtual COM Port VCP driver, choose the one for your operating system. When first plugged in the PC will ask for this driver. It is actually built in to some Linux systems.

This interface is also used to set up the IR section, for details on that see later on in the text.

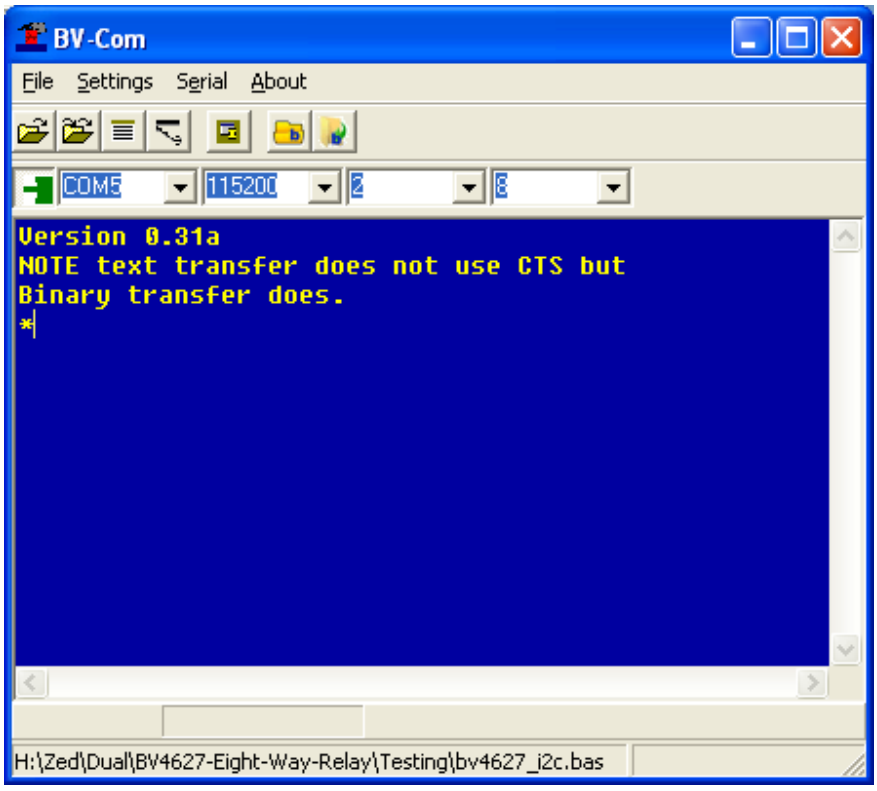
Baud Rate

The Baud rate is selected automatically as follows. When the USB is plugged in the device detects that the USB interface is being used and waits for a byte value 13 (0x0d). This is carriage return that will normally be issued when the enter key of the keyboard is pressed. In C this would be:

putch(13); or puts("\r");

It is important that this first character is correct as it will effect the operation of the device. Use BV-COMM (in the resources pack) to get started, set Echo on, 115200 Baud , 2 stop bits and 8 data bits.

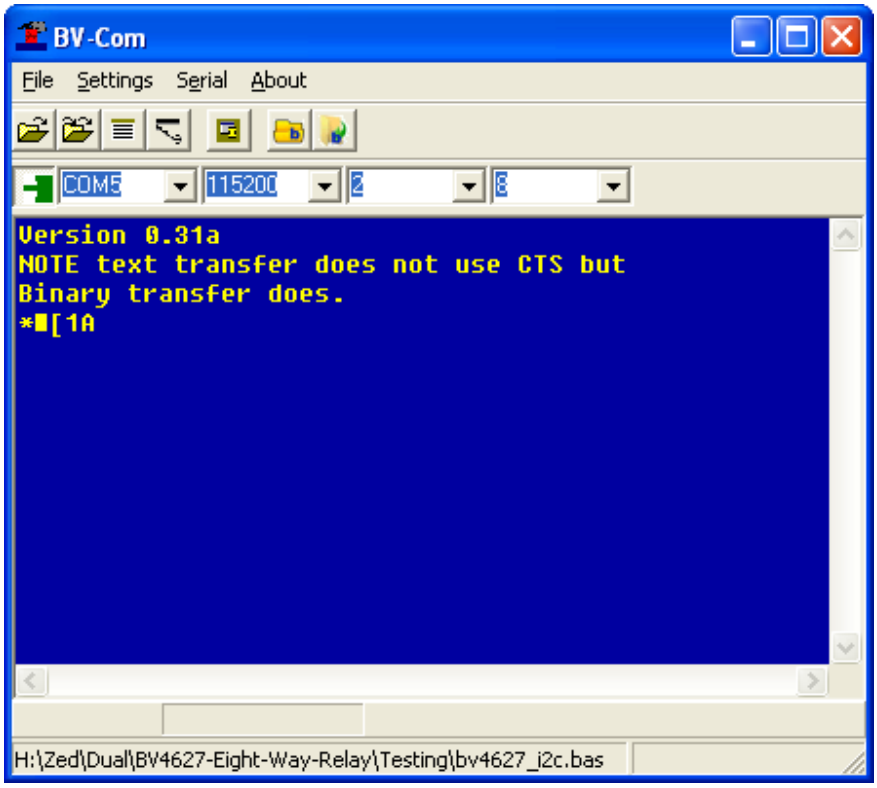
Connect the device, press return and '*' should appear:



The '*' is the response form the device to indicate that it is ready to accept commands.

To turn relay A on press the escape key, press the '[' key, press the 1 key and then press UPPER case A

e.g. esc[1A



As the escape is a none printable character it looks like a square on BV-COMM. It is a good idea to get used to the commands using BV-COMM before programming.

Programming

For these examples Just Basic will be used (http://www.justbasic.com/index.html). This is a free programming language that has good COM Port support and is easy to understand. A more advanced version is available if you like it. Read the web site for how to set it up and install. If you already have VB installed then use that.

The first thing to establish is opening the port. It is not printed here but it is in the source code. For this example we will turn on all of the relays and then turn them off one by one using the delayed switch off.

```
 7 ' ------------------------------------------------------------------
 8 ' start up
 9 ' ------------------------------------------------------------------
10    call  doComm "5"          ' open com port, change for your system
11    call doPause "500"        ' needed after opening comm
12    ' Only needed once at reset
13    print #comm, chr$(13);    ' only needed once
14    call doPause "500"
15    '
16    call  doCmd "[o"          ' all relays on
17     for j = 65 to 72
18         command$="[O,"+str$((j*5)-300)+chr$(j)
19         call doCmd command$
20         print command$ ' just shows o/p
21     next
22     close #comm
23 ' ------------------------------------------------------------------
24 ' send a command, just saves having to put chr$(27) in front
25 ' of everything
26 ' ------------------------------------------------------------------
27 sub doCmd cmd$
28     print #comm, chr$(27);
29     print #comm, cmd$
30 end sub
31
```

The above is in a different editor that has line numbers to make the example easier to explain…

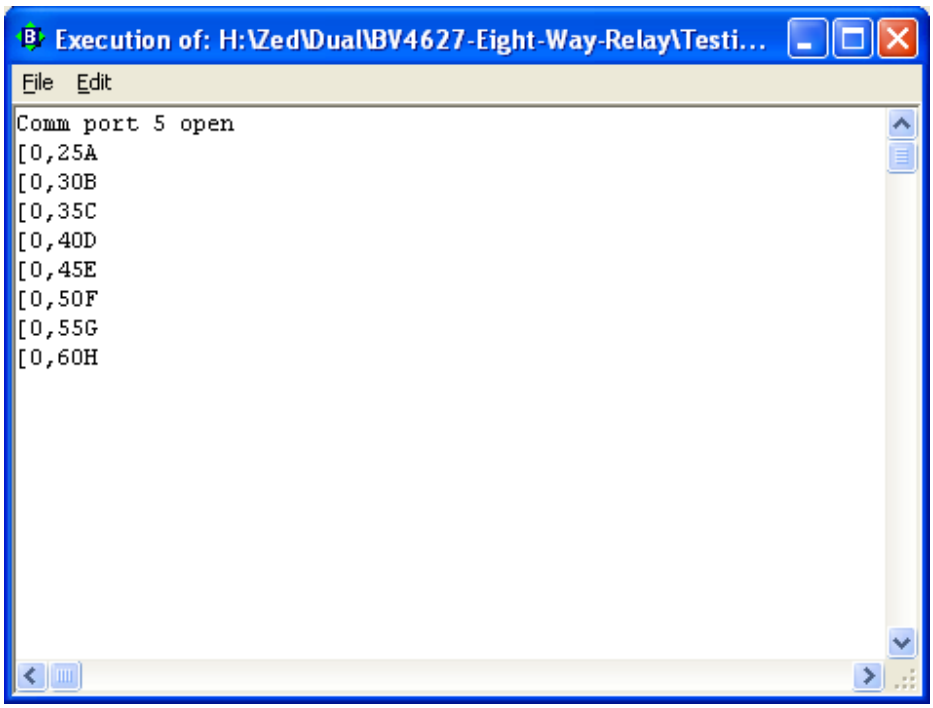
Line 10: This opens COM port 5. The actual routine to do this is not shown but it is in the supplied source code. It has also been found that a delay is required after opening the port.

Line 13: To establish the Baud rate with the device a byte value of 13 is sent. This only needs to be sent once, after reset but it is included here in case this is the first time that is has been used. This also needs a delay for the device to establish the correct rate.

Line 16: Command to turn all of the relays on

Line 17: To make it clear what is being sent, this is also printed out.

Line 27: simply sends a command to the COM port, as all commands are preceded with escape this just saves a bit of typing.



This is the output from the program.

ACK

The most difficult part of serial communication is the handshaking. This is not strictly necessary as 'waits' can be implemented instead but this will slow down the project to the slowest common denominator.

Hardware handshaking is best where the host RTS is connected to the device CTS but this can be the most troublesome. The USB port on this device has implemented this and so if it works okay in your system then no further action need be taken. When the device buffer gets full it will signal to the host to stop sending characters.

If this is not available then an ACK mechanism has been provided but this is switched off by default. To switch it on choose a byte value, say 42 ('*') and use the following command:

esc[42e

Now when any command is issued the device will carry out the command and output '*' (in this case) which can be detected by software.

Below is an example of coding this in Just Basic. It would be used in the form:

call doWait 42

after a command had been issued. It could in fact be combined with the doCmd subroutine to provide a complete command routine.

```
' ------------------------------------------------------------
' waits for comm buffer and looks for a specified character
' if found simply returns, if not prints error after waiting
' a number of iterations. Use this to wait for a response from
' external device ** Cant do zeroes **
' ------------------------------------------------------------
sub doWait value
    err = 200
    for j = 1 to err
        print j,
        if value = asc(input$(#comm,1)) then exit for
    next
    if j > err then print "ACK Error"
end sub
'
```

Infra Red

Setting up IR

The IR is set up using the USB interface and so this must be established first, see the instruction in that section. You will need a remote control handset handy to carry out the set up. The set up command is **esc[?32m.**

This will display a menu similar to:

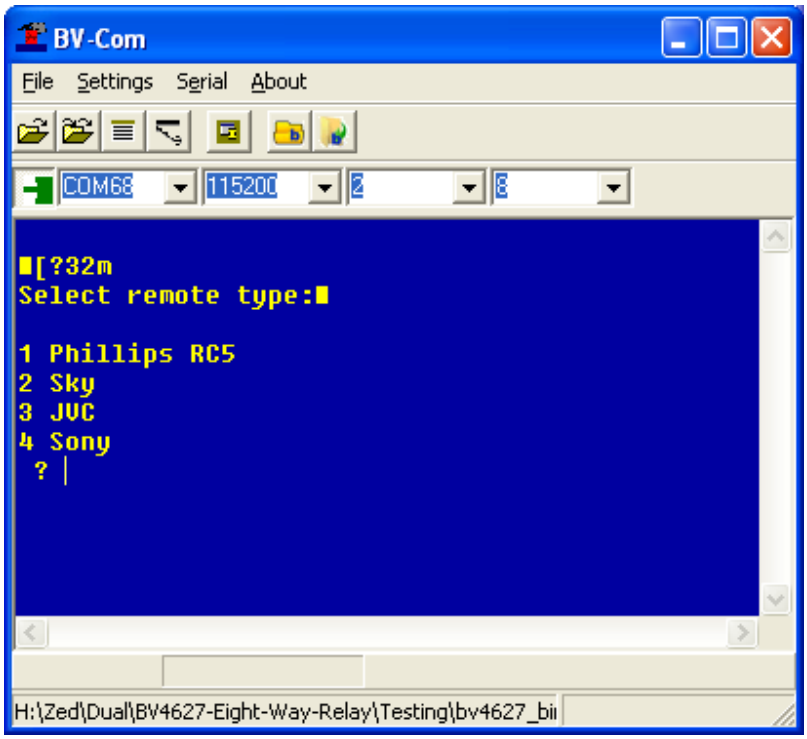


Figure 1 IR Setup Screen

The first job is to select the type of controller to use. Option 1 is the safest option as most remotes will do this in some shape or form. If it is an all-in-one, choose some Philips equipment as the code. When a remote type has been chosen another menu option will appear:

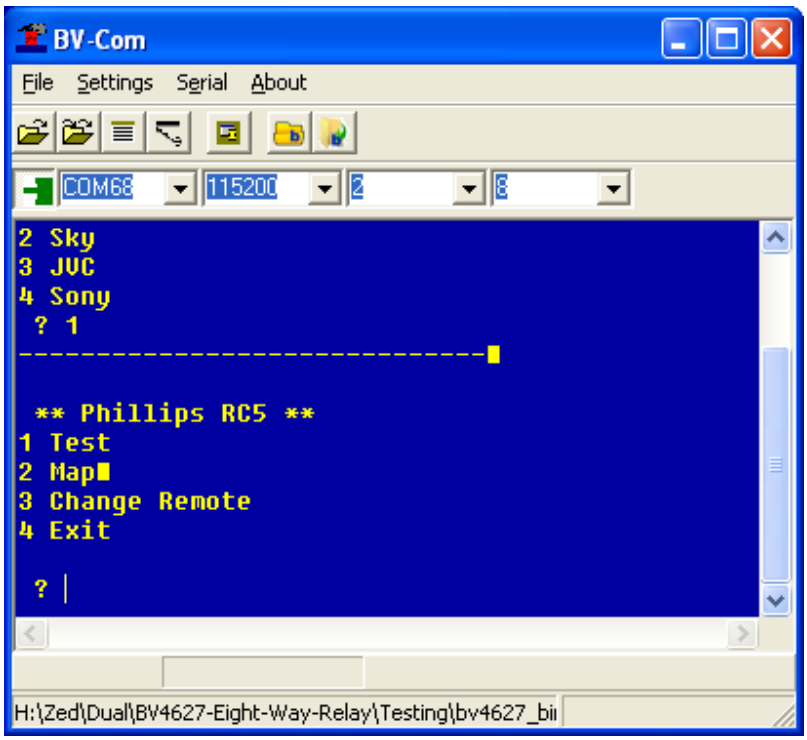


Figure 2 Second Menu

The "Test" option is used to test the remote for what codes it is sending out and how the interface is decoding them. The "Map" option is used to map a code to a relay and option 3 will allow a different type of remote to be selected.

Using Test

Select the test option and press any key on the remote. Just about any remote on any setting will produce some output here. If there is no output then the remote is using a frequency that is outside the receiver. The only solution to this is to use a different remote (but try a different code first). Figure 3 Shows a typical output, this output is unsuitable as we are looking for consistent values. Ideally just one value per button, but some remotes produce several values. This is okay providing they are consistently repeating and are different for as many relays you need to control.

To recap; you are looking for a different code for each key pressed, this can be several numbers but they must be consistent. If you choose incorrectly then simply do the procedure again. The Philips option is worth pursuing if you have a universal remote then try all the codes for a Philips TV

To **exit the test** press any key (on the keyboard not the remote).

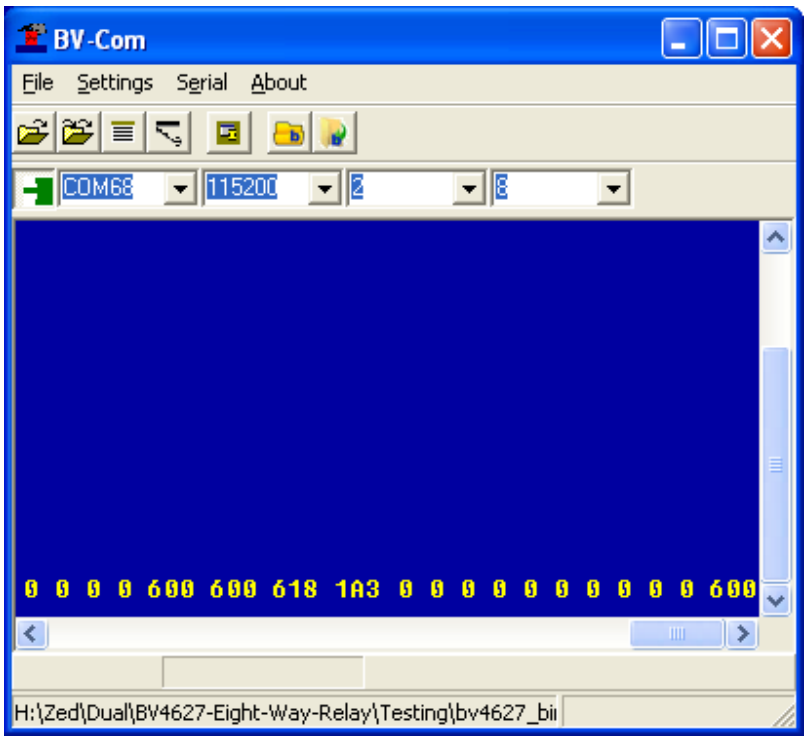


Figure 3 Output 1

A better output is shown in Figure 4. This is the result of pressing buttons 1,2 and 3. As can be seen button one is 3C, two is 3A and three is 38. The numbers are in hex but the actual value is unimportant as long as they are different.

It is perfectly possible to use an all-in-one type remote using the search facility, just keep pressing the search button until some suitable output is produced. The only way to test this is by mapping the keys and so this may be a time consuming but on-off task.

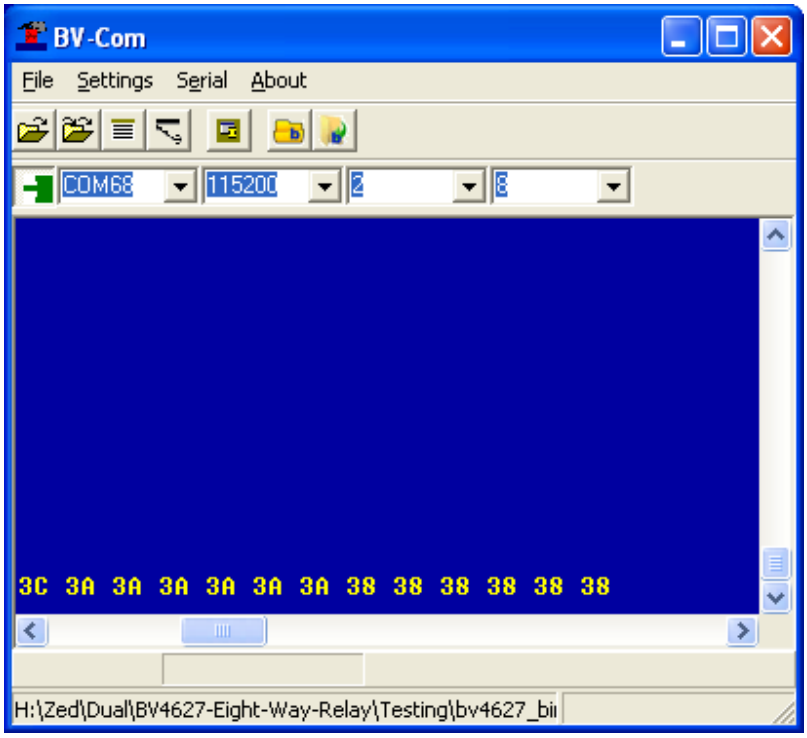


Figure 4 Better output

Key Mapping

Once a suitable output from the remote is obtained, the remote can be mapped to the relays. This means that any button on the remote can be mapped to any relay or relays. Not all relays need to be mapped.

The method is to create up to 20 slots, each slot represents a key on the remote. The slot can map to none, some or all of the relays in various ways. Each relay is given a number and this means the following:

```
0     The relay plays no part in the selected slot
1     The relay will come on when the slot is activated
2     The relay will go off when the slot is activated
3     The relay will toggle when the slot is activated
4     The relay will come on only when the slot is activated
```

The slot is activated by pressing a button on the remote that will give the slot value. In other words you can read button on remote=slot.

```
Slot    A   B   C   D   E   F   G   H
6       2   2   2   2   2   2   2   2
12      1   0   0   0   0   0   0   0
10      2   0   0   0   0   0   0   0
8       0   0   0   0   1   1   1   1
4       0   4   4   0   0   0   0   0

--------------------------------
N=new slot, E=edit, D=Delete, F=finished
```

Above is an example from the map output, slot value 6 was obtained when pressing the standby key on the remote. This has been set to turn off all of the relays. Slot 12 (button 1 on the remote) will turn on relay A, slot 10 turns relay A off. Slot 8 which was button 3 on the remote turns on relays E,F,G and H. Slot 4 which was the volume control on the remote turns on relays B and C, when the button is release the relays will go off.

Mapping Procedure

The best way to map the relays to the keys is to first create the slots, up to 20 can be created and so various combinations of relay operations can be used. For this demonstration the following is required:

Button	Operation
1	relay A toggle
2	relay B on
3	relay B off
4	relay C,D momentary on
5	Relay E on F off
6	Relay F on E off
Power off	All relays off

Seven slots are needed for the above.

From the USB interface enter the command **esc[?32m**. Select the remote control type (its is assumed that this has already been set up using the instructions in “Using Test” above).

From the menu select 2, which will bring us to the map menu:

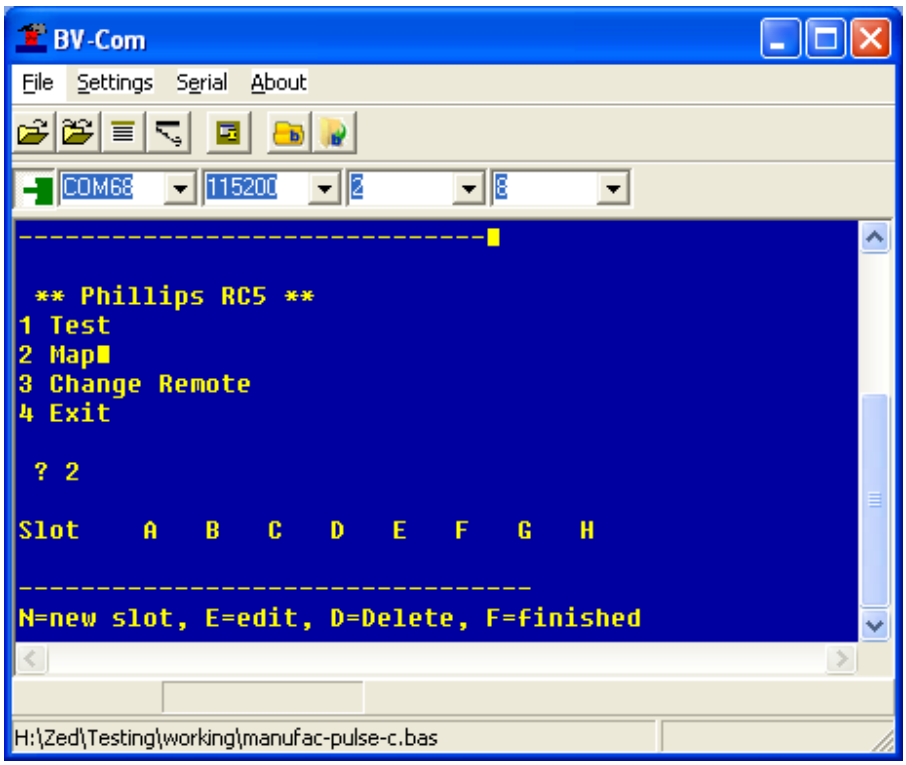


Figure 5 Key Mapping 1

After pressing menu option 2 you are presented with Figure 5. On a fresh system there may be no key mapping at all and so no slots will be seen. The factory test may leave behind a couple of slots, these can be deleted with the delete option.

Press N to create a new slot and on the remote, whilst holding the key down, stars will appear. The device is taking 10 samples of the output from the remote. When it has taken the samples, release the remote.

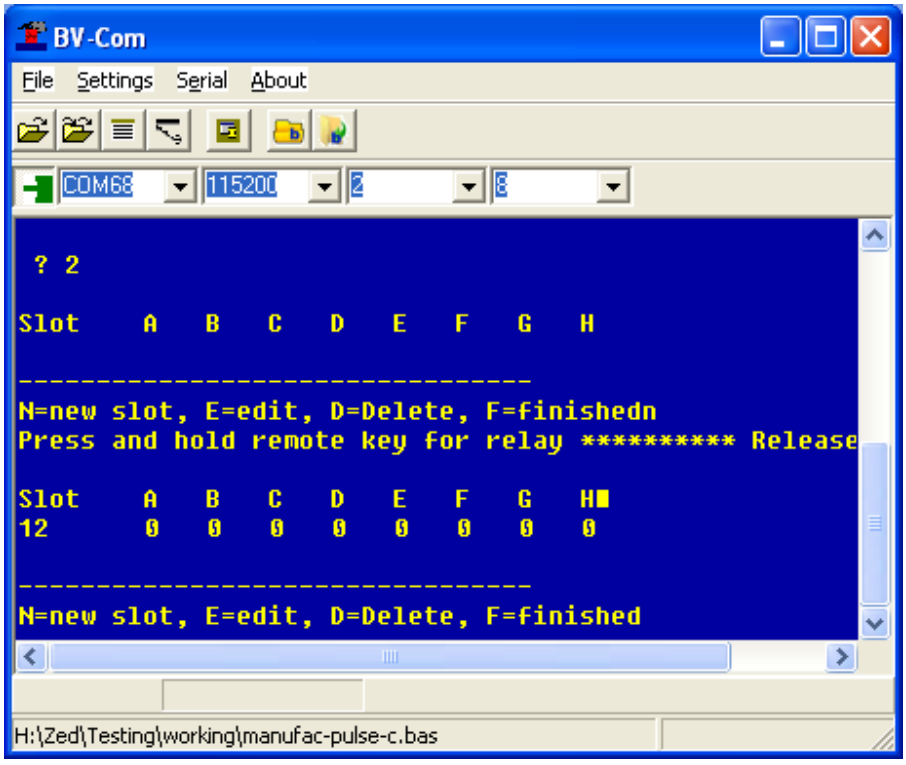


Figure 6 Slot has been created

A slot has now been created, the '12' in Figure 6 is the value that was read from the remote when that particular button was pressed. The value must be unique and if a repeated value is given it will be rejected.

Create another 6 slots using different buttons on the remote:

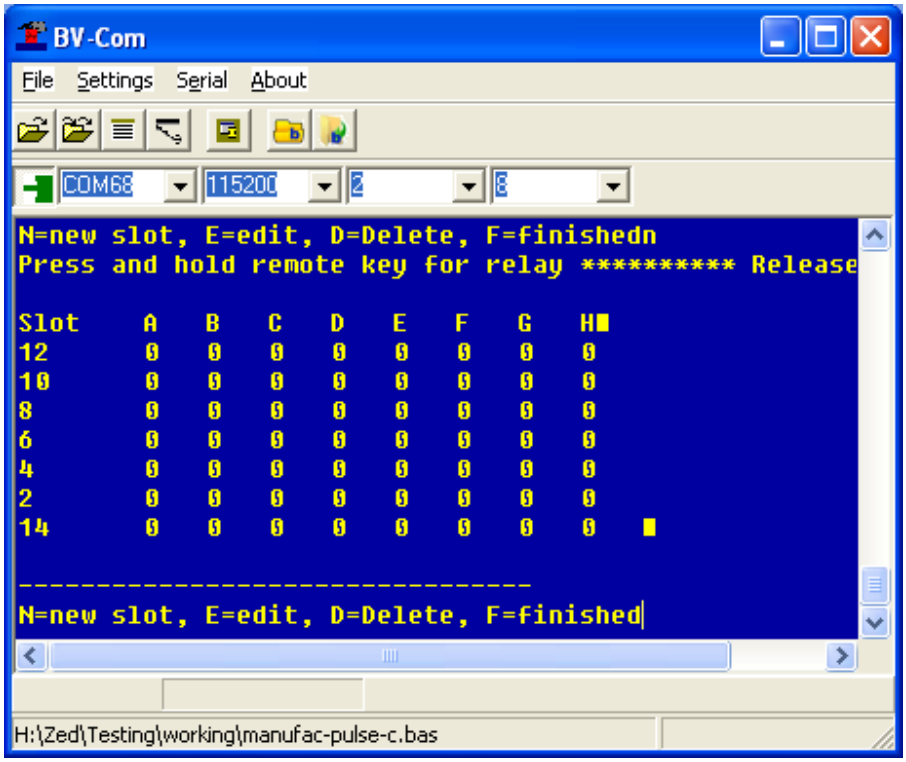
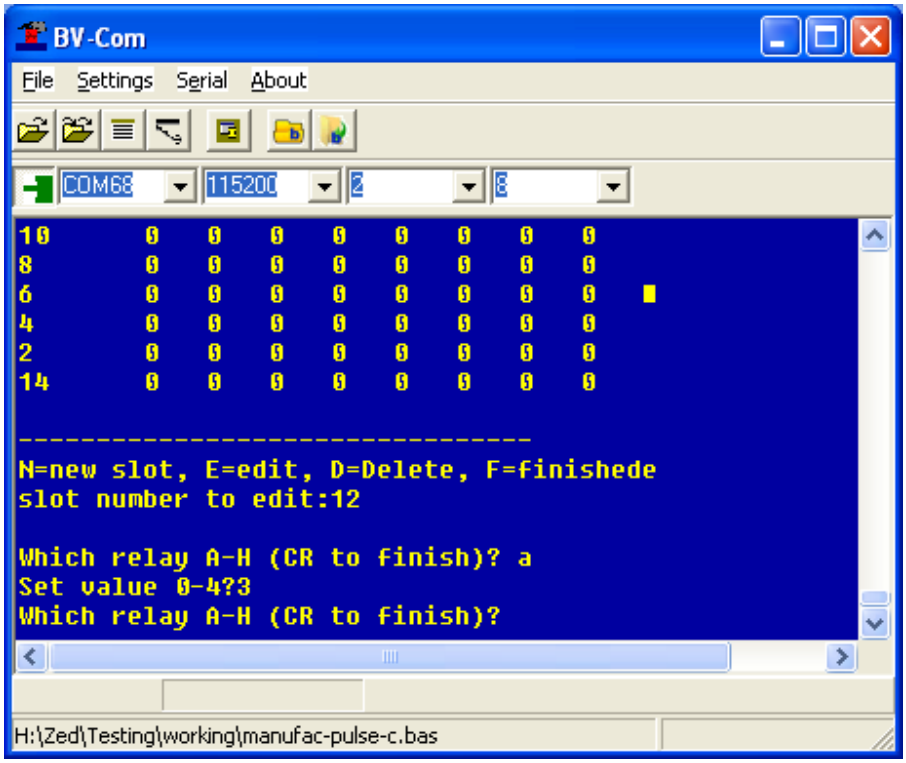
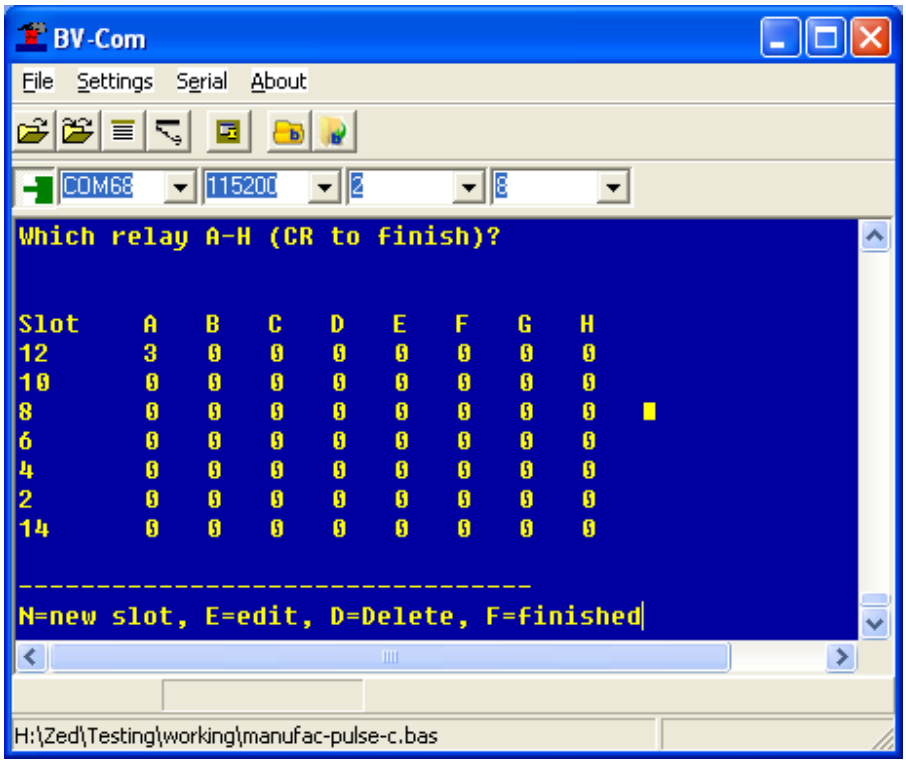


Figure 7 Slots created

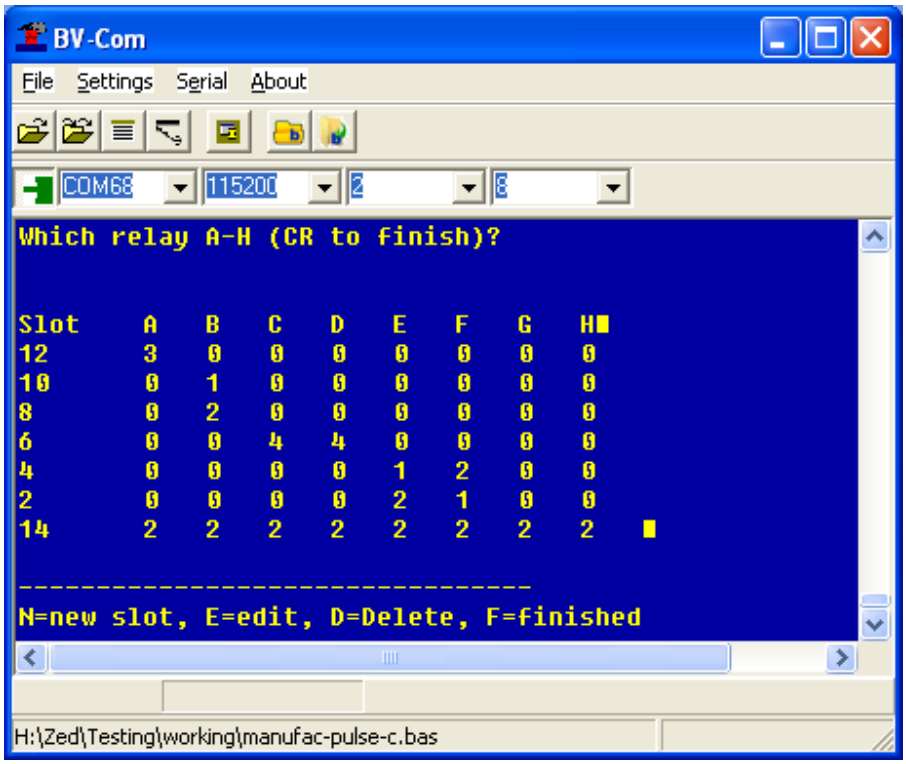
At the moment the slots have been created but they will not do anything because the relay values are set to 0. Slot 12 needs to be told to turn relay A on. Do this with the 'Edit' option, simply press 'e' for the edit menu.



The Edit function will ask which slot, this should be the slot number, in this case 12. when this has been entered you are asked for a relay, enter 'a' through 'h' to select the relay of interest and then put in a number form 0 to 4. The number 3 will have the effect of toggling this relay. As we want this button only to control one relay press CR and you will be taken back to the map menu:



See how relay A has now been set to 3. All of the other functions can be set up in this way



Press 'f' to finish and then 4 to exit the IR menu.

To test the IR with the USB plugged in the command is **esc[?32T** There is no way out of this command without a reset. When the USB is not plugged in and all of the other interfaces are disconnected. The IR is active and so no command is needed.