# BusRaider 1.7 Construction, Testing and Initial Setup and Use

Rob Dobson 2019

# BusRaider Functionality

The BusRaider is designed to plug into the bus of the RC2014 Retro-Computer (<https://rc2014.co.uk/>).

It performs two main functions:

* Emulation of the memory-mapped video and keyboard functionality of some 1970’s/80’s era computers such as the Sinclair ZX Spectrum and Tandy TRS-80
* Single-step debugging of Z80 programs (including breakpoints, register and memory inspection).

To achieve this it has two main tricks up its sleeve:

* Accessing the RC2014’s bus using the BUSRQ line. Once requested and acknowledged the Z80 “let’s go” of all the bus lines and allows the BusRaider to read/write memory and IO without interference.
* Detecting when the Z80 is performing a memory/IO operation or responding to an interrupt and then forcing the Z80 to wait so that the BusRaider can
  + respond to the memory/IO operation itself (emulating IO or memory)
  + single-step through a program
  + detect debug breakpoints and pause execution at that point

# Issues

While the hardware of the BusRaider is relatively simple the vast majority of effort in this project has been developing software for the Raspberry Pi Zero (W) to control the hardware in real-time to achieve the desired end-goals. While much effort has been made to ensure that this operates without error in all cases you should note that this is a hobby project and no guarantees can be made about the reliability and robustness of the code. If you do discover any problems please do let me know. Ideally this should be via GitHub’s issue tracking mechanism at https://github.com/robdobsn/PiBusRaider/issues

# Prerequisites

In order for the BusRaider to operate (in any operating mode) a few things need to be ensured:

* The BusRaider is in a PRO backplane (such as Spencer’s <https://rc2014.co.uk/modules/backplane-pro/> or Steve’s <https://www.tindie.com/products/tindiescx/modular-backplane-boards-for-rc2014/>) as additional bus lines BUSRQ, BUSAK, WAIT, NMI, RST2(Page) are used
* Jumpers need to be in place to enable communication between the ESP32 and Pi. The position of these jumpers is shown in figure [XXX]
* A Z80 processor board suitable for use with the PRO backplane – e.g. Spencer’s Z80 CPU 2.1 <https://rc2014.co.uk/modules/cpu/z80-cpu-v2-1/>

Optionally (depending on how you plan to use the BusRaider) you should also have the following:

* RAM and/or ROM cards for the RC2014 bus. These are actually optional because in emulated RAM mode there actually doesn’t need to be any physical memory at all. However, emulated RAM mode is much slower than using physical memory so it isn’t recommended for general use. Furthermore, if you wish to use opcode injection or the debugging facilities of the BusRaider (and are not using emulated RAM) then the RAM / ROM cards you choose should have paging support and the PAGE jumper should be fitted on the BusRaider – see Appendix 13.
* Z80 clock module. This is optional (but not recommended) because the BusRaider can generate an appropriate clock for the Z80 and this can be varied to approximate the speed of an original computer such as ZX Spectrum or TRS80. If a clock module is used then the CLK jumper on the BusRaider should not be fitted.
* Z80 serial port (68B50 / Z80 SIO / etc) card. This is optionally used if serial-based software is to be used with the BusRaider such as the BASIC or CP/M software provided in various forms with versions of the RC2014 ROM/CF-CARD/etc. If this is to be used the serial jumpers need to be installed to connect the Z80 serial port (1 or 2) to the ESP32 as described in Appendix 8
* USB keyboard and adapter for micro USB connector on the Pi Zero (W) – to interact with the target computer software and also set the WiFi settings
* HDMI monitor and adaptor for the small HDMI connector on the Pi Zero (W) – to see the main output from the BusRaider and memory-mapped graphics
* FTDI Serial cable (3V) – this can be used to set WiFi settings on the ESP32, reprogram the ESP32 and for diagnostic purposes as an initial check of board functionality or if there are operational issues
* Visual Studio add-on called Z80 Debug (<https://github.com/maziac/z80-debug>) created by Maziac which performs single-step debugging of Z80 programs

# Getting Started

If you haven’t built or tested the BusRaider yet then please refer to Appendix 10 Construction and/or Appendix 11 Inspection and Testing at this stage.

Continue with the Initial setup below once you have the BusRaider at the point where the HDMI output from the Raspberry Pi Zero (W) is showing the presence of the ESP32 – see below.

In addition, if you have a keyboard connected to the Raspberry Pi Zero (W) USB port, then you should see confirmation of this in the display as shown below.

[ note that the Pi HDMI display should show the presence of the ESP32 – version number, etc - also show keyboard presence ]

# Initial Setup

The first thing to do is to get the BusRaider onto your WiFi network. There are two ways to do this:

* Using the keyboard and HDMI display on the Raspberry Pi. First press F2 on the USB keyboard to enter “Immediate Mode”. You should see confirmation on the HDMI display. Then follow the instructions below.
* Using an FTDI serial connection cable as described in Appendix 9 with jumpers in place for FTDI to ESP32 as described in Appendix 8. Open terminal emulator software (Appendix 9) and make sure you can see the diagnostic messages described in the Appendix 12. Then follow the instructions below.

Enter (with either the USB keyboard or in the terminal emulator) your WiFi network SSID and password together with an optional “hostname” as follows:

w/YOURSSID/YOURPASSWORD/YOURHOSTNAME

Where:

* w starts the command to set up the wireless network
* / is used as a separator between the parameters to the command
* YOURSSID is the SSID of your network (doesn’t have to be all caps I’ve just done that for effect)
* YOURPASSWORD is the password to your network. At present this assumes the network is secure.
* YOURHOSTNAME is the hostname you want the BusRaider to appear as on your network. So that, for instance, if you chose the hostname busraider then when you are using the chrome browser (although maybe not other browsers) you can type just busraider into the address bar and see the Web UI for the BusRaider appear.

After you have entered the SSID wait a short time and you should see the HDMI display update with the IP address that the BusRaider’s ESP32 has been assigned by your network. If this doesn’t happen then you could check the output on the terminal emulator connected via the FTDI cable (if you have one) or just try again in case you entered an incorrect ssid or password.

Note that the WiFi support doesn’t work with all network security schemes (Enterprise security isn’t supported for instance) and there may be issues with spaces or other odd characters in the password or SSID.

# The BusRaider Web UI

Now that the BusRaider is on the WiFi network we can look at its WebUI. Open a browser (preferably chrome as it is generally able to use the hostname we set up with the WiFi settings) and enter one of the following:

* The hostname you chose when assigning WiFi settings (maybe you followed the example I gave and entered busraider)
* The IP address that your network has assigned to the BusRaider – this will be shown on the HDMI display in the yellow status text.

In either case you should see the WebUI which should look something like this:

[ WebUI ]

The main sections of this UI are as follows:

* Target machine type and settings
* Target commands and file drop area for immediate uploads
* File system contents and buttons to run programs, etc

# Selecting the Target Machine (and Options)

You will probably have a favourite machine that you would like to use at this stage so select this from the drop-down list. Also select the options you require from he checkboxes below the target machine. The options are described more fully in Appendix 4 and note that your hardware should match the options you select – specifically:

* if you select emulated RAM then there should be no physical RAM in your system
* if you select paging support then your RAM/ROM cards should support paging using the RST2 line on the RC2014 PRO bus, the PAGE jumper should be fitted on the BusRaider PCB and there must be a pull-up resistor on the RST2 line as described in Appendix 13

Once you have made your selection(s) you should see the HDMI display showing the output from your chosen machine. At this stage it is expected to be random garbage that was in the memory of the Z80 at power up, so something like one of these displays:

[ ZX Spectrum garbage screen]

[ TRS80 garbage screen]

# Running a Program

Now you are ready to run some software. First locate a ROM image file suitable for the machine you have chosen.

* If you have selected the ZX Spectrum then you can find a suitable ROM image here <http://www.shadowmagic.org.uk/spectrum/roms.html> and, specifically, the 48.rom file is the one to choose initially.
* For the TRS80 the there are some ROM images here <https://github.com/lkesteloot/trs80/tree/master/roms>, and the file level1.rom works well

Download the file to the computer that you have the WebUI running on and then drag and drop the file onto the middle section of the UI – where it says “Drop files to immediately run on the target”.

You should see something like the following:

[ Spectrum prompt ]

[ TRS80 prompt ]

Storing Programs to Run Later

To store a program for later use simply drag and drop the file in the section labelled “Drop files to upload to the BusRaider”

[ note that the above is currently wrong and text needs to change ]

Currently all programs are stored in the internal SPIFFS flash memory of the ESP32 but the SD card will be operational soon and when it is the SD card (when inserted) will take precedence over the SPIFFS files. So if an SD card is inserted on power up (dynamic insertion won’t be supported) then the files listed in the UI will be the files on the SD card and any files on the SPIFFS area will not be usable.

Demo Mode

Debugging a Program

# Appendix 1: Updating the ESP32 Firmware

The ESP32 firmware is contained in the folder BusRaiderESP32 in the GitHub repository. To build and run this code you will need the following:

\*

Visual Studio Code (https://code.visualstudio.com/) - VSCode

\*

PlatformIO - this can be installed from VSCode by clicking the Extensions icon in the left-hand column (it looks like a square within a square) and searching for PlatformIO - more details at https://platformio.org/

\*

Espressif 32 Platform - in the Home Page of PlatformIO (click the house icon in the bottom-left toolbar to get the PlatformIO Home Page) select Platforms and search for Espressif 32 then install it

\*

USB to Serial adapter such as 3V3 FTDI Cable https://www.amazon.co.uk/dp/B071WPW292 or equivalent "Prolific DebugCable" https://www.amazon.co.uk/gp/product/B01N4X3BJB,

Building the ESP32 Firmware

At this point you should be able to open the folder for the BusRaiderESP32 sub-project (it won't work if you open the parent folder for the whole of BusRaider) in VCCode and then click the tick icon in the bottom-left toolbar. This should build the BusRaiderESP32 project.

Programming the ESP32

First connect the serial cable (FTDI or similar) to the BusRaider PCB. This must be connected to the serial port at the top-right of the BusRaider PCB. Check the connections carefully and, if you are in any doubt about whether you really have a 3V3 cable, make sure the protection diode D1 is in place and connected the correct way around.

The connections for a genuine FTDI cable (black USB connector body) are:

\*

Black or Blue = GND (J1 Pin 1)

\*

Orange = FTDI TXD (J1 Pin 4)

\*

Yellow = FTDI RXD (J1 Pin 5)

And for the "Prolific Debug Cable" referenced above (blue USB connector body):

\*

Black = GND (J1 Pin 1)

\*

Green = FTDI TXD (J1 Pin 4)

\*

White = FTDI RXD (J1 Pin 5)

In addition two jumpers need to be in place to connect the FTDI Rx/Tx signals to the ESP32. These are on J5 (mid-way down the right side of the BusRaider PCB). You should put jumpers in place vertically to connect between Pi and ERSP1 for both the RX and TX columns.

If this is the first time you have connected the serial port it would be advisable to use a terminal emulator such as TeraTerm (described above) or the Serial Monitor built into the PlatformIO extension. The serial port settings are 115200, 8, N, 1 (but the baud rate 115200 is the only thing that generally needs setting). With the board powered up you should see a regular (once every 10 seconds) message from the ESP32 on the serial port indicating its status including WiFi connection, etc. If you don't see this (or other similar activity) then check the jumpers and the FTDI cable connections.

To get the firmware onto the ESP32 on the BusRaider you will need to make sure the settings in the platformio.ini file are correct. This file is in the root of the BusRaiderESP32 folder. The most important setting is the upload\_port. If on Windows this should be the COM port (e.g. COM3) that the FTDI serial port is using (you can find this out with a terminal emulator program like TeraTerm (https://ttssh2.osdn.jp/index.html.en) or through the device manager in windows. On Linux it will probably be /dev/ttyUSB0.

Next you need to put the ESP32 into download mode. To do this hold down the button marked DEMO on the BusRaider while pressing and releasing the ESP RESET button (the one closest to the ESP32 at the top of the PCB).

Finally press the right-arrow button on the bottom-right toolbar in PlatformIO to start programming the ESP32. If all goes well you should see the stages of the programming process and a success message.

The ESP RESET button has to be pressed again when programming is finished to start the new ESP32 firmware.

Appendix 2: Updating the Pi Software

Appendix 3: How it works

# Appendix 4 Memory Emulation and Paging Options

In the Web UI of the BusRaider a number of options can be set to enable some of the more advanced features. The items are listed under the machine selection drop-down and are:

* RAM emulation
* Paging
* Opcode injection

As expected the RAM emulation checkbox controls the BusRaider’s RAM emulation capability. In this case simply on/off control. If RAM emulation is on then paging is not used (no physical RAM/ROM should be connected to the target machine’s bus and hence paging is not required) and opcode injection is implicitly turned on as that is the only way to execute

The Paging checkbox controls whether the BusRaider generates a page signal on the RST2 line of the RC2014 bus when it wants physical memory to be deselected. Note that the PAGE jumper also needs to be fitted on the BusRaider and there must be a pull-up resistor on the RST2 line for this option to work – see Appendix 13 for more details.

# Appendix 5: Opcode Injection

One of the more interesting features of the BusRaider is its ability to insert opcodes (and associated data) onto the data bus of the target processor. This capability is used for two purposes:

* To execute software downloaded via the BusRaider, for instance with file formats such a SNA (snapshot) format for the ZX Spectrum which contain register values and execution address.
* For single-step debugging to read the values of the target processor’s registers at each step.

Operation in each of these cases depends on the use of wait-state generation to hold the target processor at the current execution point and then selectively releasing the wait-state while appropriate instructions are placed on the bus or data read from the bus as appropriate.

## Appendix 5.1 Program Execution with Opcode Injection

In the case of target program execution with opcode injection, the bus control sequence looks as follows:

[ image of target program execution ]

What we are seeing here is a view of the data bus and control bus while opcode insertion takes place for program execution. The actual instructions being injected are as follows:

[ opcodes injected for program execution ]

The purpose of this code is to set the registers to be as they were when a snapshot was taken (normally this information is extracted from a snapshot file format such as the SNA, Z80 or ZTX formats commonly used with the ZX Spectrum). In cases where only the program counter (execution address) needs to be specified (e.g. TRS80 CMD format) the same opcode injection is used but the other registers are initialised with 0 values.

## Appendix 5.2 Program Execution with Opcode Injection Off

When opcode injection is turned off (see appendix 4) the approach taken to program execution is to write code into two areas of the target machine’s memory to set registers and execute the program on the target. The code written is as follows:

* Jump instruction at the reset vector location (0 for Z80) which makes execution jump to the register setting code
* Register setting code which is a block of around 55 bytes which is written to a “safe” area of RAM in the target machine. In general this “safe” area will be part of the display RAM and, in that case, the code may show up on the memory mapped display if it is not quickly overwritten.

The actual code executed in this case is identical to that which is injected to set registers as described in appendix 5.1

## Appendix 5.3 Register Query Bus Control

The register query opcode injection mechanism works in the same way as described above but with different opcodes and read operations. The bus control sequence is as follows:

[ image of target reg query ]

The opcodes inserted in this case are as follows:

[ opcodes injected reg query ]

# Appendix 6 Hardware Overview

The BusRaider hardware performs the following functions:

* Generation of BUSRQ signal to capture target bus
* Setting address and control bus and writing/reading data bus to effect target memory access during bus capture
* Wait-state generation on memory and/or IO access to hold processor during a read/write cycle
* Read access to address and control buses and read/write data bus to enable IO/memory system emulation, opcode injection and single-step debugging
* IRQ, NMI and RESET signal generation
* Processor clock generation to allow matching of the machine speed of a specific retro computer
* Paging signal generation to “page out” physical RAM when emulating memory or injecting opcodes
* Access to target bus serial IO to enable terminal emulation mode and telnet support

There are two processors, a bunch of 74 series hardware and some discrete components on the BusRaider which accomplish these functions:

* Raspberry Pi Zero (or Zero W) referred to elsewhere as Pi
* ESP32
* Address bus registers and drivers [UUUUUU]
* Address bus read drivers [U]
* Data bus read/write driver [U]
* Control bus read/write driver [U]
* BUSRQ/BUSAK logic
* Wait state logic
* Serial port hardware and jumpers
* Power supply regulator

## Appendix 7 Raspberry Pi Software

The Raspberry Pi Zero (W) operates in bare-metal mode (i.e. without an operating system) and is responsible for all real-time operations on the target machine’s bus. It is responsible for:

* Configuring the hardware into the correct modes to operate as required for the user’s settings (target machine, RAM emulation, paging, opcode injection, target clock frequency, etc).
* Generating the BUSRQ signal used to gain access to the target machine’s program and memory-mapped graphics RAM. The rate at which this occurs is set in the descriptor-table for the machine being emulated – see Appendix 7.1 Machine Descriptors and Classes
* Handling the interrupts generated when an enabled wait-state occurs. This can be a memory or IO request and the interrupt behaviour is quite complex because different operating modes (RAM emulation, paging, opcode injection) are mainly handled in this activity – see Appendix 7.2 Wait-State Handling.
* Communicating with the ESP32 (and through that channel with the web-UI and/or Visual Studio add-on for single-step debugging). This is a bi-directional serial connection which currently operates at 500K baud.
* Generating the HDMI display signal for display on a connected monitor which is used to display the emulated graphics from the target machine and also used for diagnostic information, setting immediate-mode values such as WiFi SSID and password, etc.

## Appendix 7.1 Machine Descriptors and Classes

The Raspberry Pi software is responsible for emulation of the target machine type (ZX Spectrum, TRS-80, etc). Each machine that can be emulated requires a separate C++ class to define its behaviour and there is also a table of settings (called the descriptor table) that each machine exposes to the Pi software.

Appendix 7.1.1 A Quick Note about Classes

If you are familiar with classes (a part of object-oriented programming) then skip the following description but, if not, I hope the following will be a quick de-mystification of what I know is a testy topic for some people:

The idea of a class in C++ is actually very simple and this is an ideal example to describe its use. Imaging that you want to describe a number of different retro computers in a simple way to a stranger. One approach would involve “similarities and differences”.

On the subject of similarities, for example, you could say that both a TRS-80 and a ZX-Spectrum use the Z80 processor and have some memory which is read-only, some which can be written to generate a text/graphics output and some which is just read/write for general program and data use.

# Appendix 8 Serial Port Jumpers

# Appendix 9 FTDI Serial Cable Connector

Then run terminal emulation software such as TeraTerm (<https://ttssh2.osdn.jp/index.html.en> or screen) with baud rate and settings 115200, N, 8, 1.

# Appendix 10 Constructing the BusRaider

There is a construction video for BusRaider 1.7 <https://www.youtube.com/watch?v=akkhFuqRgis> and a series of videos for BusRaider 1.6 which are a little more detailed <https://www.youtube.com/watch?v=CyE8oFtVzxQ>

The key things to note when constructing are that the surface mount components should generally be attached first as they are lower to the PCB and sometimes awkward to add later. I generally find it easier to add components in height order with the least-high components being added first. This enables you to make use of gravity to hold components in place and avoids having to solder over a high component when trying to reach a lower one.

The use of sockets for the ICs is optional if you want to go down that path. I tend to prefer not to use them as they add an additional potential point of failure and there is little benefit unless you intend to experiment widely with different chip versions or similar.

# Appendix 11 Inspection and Testing

Once the board is fully populated it is time for inspection. I recommend using a magnifying glass or similar to inspect all soldered joints carefully looking for bridged tracks, dry-joints or similar. An awful lot of time and heartache can be saved by this simple step and getting into the habit of routinely inspecting your work before powering on is a really good thing to do. Take particular care with the power regulator area (because errors in this area can damage lower voltage components like the Pi and ESP32) and any area where soldered joints are very close together such as the 74LVC07 (U5), ESP32 (U12) and SD card socket.

Following a thorough inspection, you are read to test the board. I would initially recommend leaving the Raspberry Pi Zero (W) removed from its connector and initially concentrate on the ESP32 functionality.

If you have a USB to serial adapter cable (3V only) then connect this to the FTDI Serial Port connector (J1) as described in Appendix 9 and run your terminal emulation software (also described in Appendix 9).

It is now time to power up the board. Insert it into the RC2014 backplane (making sure it is the right way around (Pin 1 is marked with a small 1) and then turn on the power. Assuming you don’t see a flash of blue light and small acrid smoke (only kidding) then the BusRaider should be powered up.

If all is well, you should see something like the following appear on the terminal emulator:

[screen shot initial ESP32 status with no Pi]

If you don’t see this and you do have some basic electronics tools – like a multimeter – then check that you see 5V appear on the regulator (which is located on the far left of the PCB when looking at the component side)

[ image check 5V here ]

# Appendix 12 ESP32 Diagnotic Messages

During normal operation the ESP32 shows a regular diagnostic message when an FTDI cable is connected and ESP32 – FTDI jumpers are in place. This message is as follows:

[ ESP32 diagnostic message ]

# Appendix 13 Paging and the PAGE Jumper

# Appendix 14 Target Clock Generation and the CLK Jumper