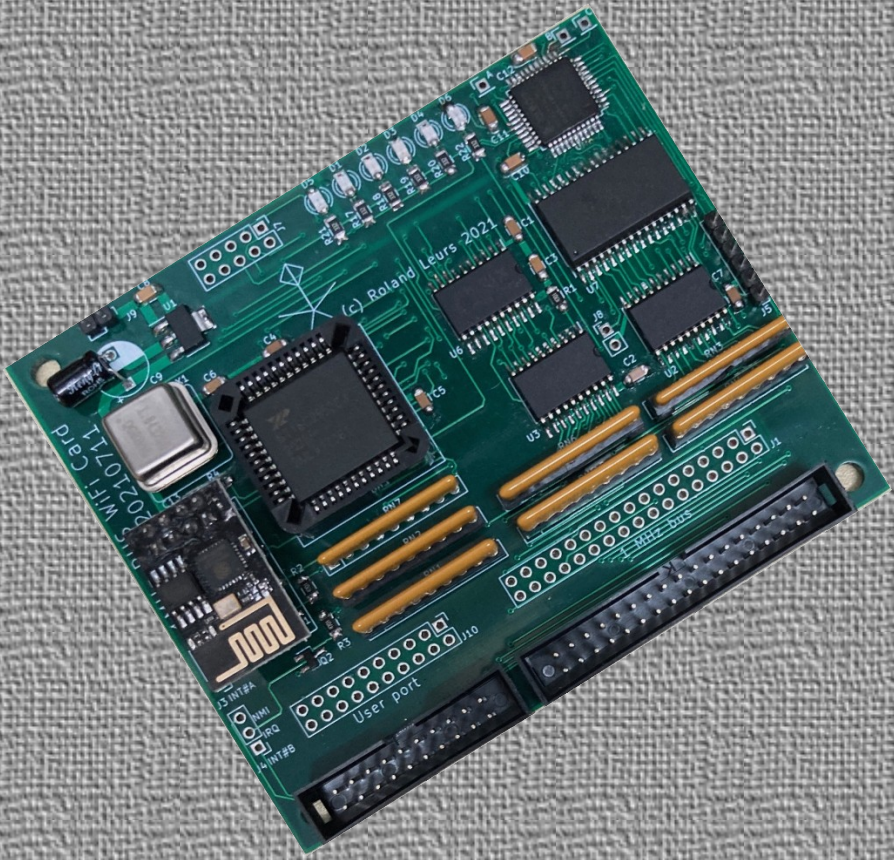
Acorn Atom WiFi

User Manual



Roland Leurs

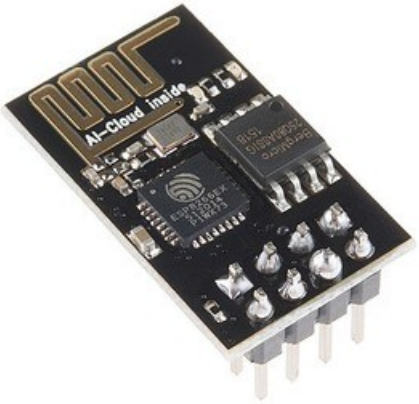
April 2022

Table of contents

Introduction

The ESP8266-01 is (literally) a small device with great possibilities. It enables classic computers and modern micro-controllers to connect to a WiFi network and transmit data via a simple serial interface. Just like bitd it accepts a set of AT commands and performs the necessary actions that are needed for the network communication. The device holds a complete TCP/IP stack, albeit IPv4 only.

So all you need to connect to a wireless network is just a standard serial interface, an ESP8266 and some software.



When I started this project on my Acorn Atom I used the serial port of the Godil and although it basically worked, data transfers were mostly not completed. I never figured out what the reason was, I still suspect my first ESP module (which was another type) was not quite good.

I decided to start a new project with another, even smaller, device (the one you see in the picture above) and I also added an extra serial device. For my Atom 2k18 I could use some additional serial ports since it needs one for communicating with the on-board Pi-Zero and now also for the WiFi device. I picked the 16C2552 which is a dual UART (Universal Asynchronous Receiver and Transmitter) which has two independent communication channels, a 16 byte input / output buffer (FIFO, first in first out) and can transmit up to 4 Mbps. And most important: it is affordable and available from reliable suppliers.

For the Atom I wrote a WiFi driver which can be used by user applications and commands. A basic set of commands include LAP (list access points), JOIN (join a network), TIME/DATE and WGET (to download a file from a web server). All these commands use the same driver. And this setup works nice.

About the time I finished this project the question “Acorn Electron online – any such hardware” popped up on the StarDot forum. And as I had such hardware and software for the Atom, I decided to port both to the Electron. The board was a great success and soon the request for a BBC (Master) version arose. After finishing most of the software for the Elk WiFi I started to work on a version that connects to the 1 MHz bus of the BBC Micro.

While developing the hardware and software for the Electron and BBC Micro I realized that the Atom WiFi board lacks “something”. Especially the paged RAM for temporarily storing the incoming data and a location for storing the Wifi driver and ROM. In an Atom 2k15 or FPGAtom there’s enough memory but not for an unexpanded Atom with only 12k+12k. So I build an adapter board for connecting the Beeb WiFi module to the Atom. This adapter board provides a 1 MHz bus[1] and an User Port compatible interface for the Atom. This adapter board is documented in this manual.

So here it is, the Beeb WiFi module for the Acorn Atom. An easy to install and use board to connect your Acorn Atom to the Internet. I hope you enjoy this device as much as I do.

1 Only when the Atom runs at 1 MHz, on higher clock speeds the bus speed also increases.

WARNINGS

Before I continue I have some important warnings:

1. Security

The Acorn Atom is in no way a secure device, nor is the software for the WiFi module. When using the commands or driver, **usernames and passwords may be kept in memory**.

2. Baud rate changes may brick the ESP8266

The default transmission speed for the ESP8266 is 115,200 baud. Both the Atom and the BBC Micro can handle data transfers at this speed. Do not try to change this speed because the ESP8266 might accept your command but does not always perform it correctly. You might end up with a module that communicates at an unknown serial speed. The only remedy to fix that is to flash the device (or replace it, after all, they cost only about £1.00).

3. Only for personal, domestic use

Both the hardware and software are not designed for medical- and health use nor for mission critical, industrial and automotive purposes. Use it at your own risk.

Hardware description

The complete diagram is included at appendix I. I will describe my design considerations and the components in this chapter.

Design considerations

There were two options to make the WiFi board available to the Acorn Atom:

1. Create an interface board that creates an Electron compatible slot or 1 MHz bus for the Atom.
2. Design a complete new board with the same core components.

Since there are a few other 1 MHz bus expansions that are also applicable for the Atom (like BeebOPL and the 1 MHz bus FPGA board) I decided to build a third version[2] of a 1 MHz bus Adapter with room for an eprom.

The core components of the WiFi board are:

- ESP8266 WiFi module
- 16C2552 Dual Channel UART
- 128 kB RAM
- Xilinx XC9572XL CPLD for logic

The paged RAM register should be both writable and readable. I could do this by adding an extra latch that is enabled when a read cycle occurs at #BDFF (the address of the paged RAM register). But the UART has two scratch pad registers. These behave like a normal memory address. So I use the scratch

2 The first board is a board that provided all BBC Micro ports to the Atom and the second board is an expander board for the FPGAtom. That board is however also usable for normal Atoms. Both boards are not widely in use.

pad register of the A-channel (which is available for generic serial applications) as a read/write copy of the paged RAM register. The decoding is simply done in hardware so for a programmer the address #BDFF is just a normal register that can be written and read.

If your Atom already has paged RAM at #BCxx then you can disable the memory on the WiFi board by adding a jumper. To use the full 128kB of the RAM I connected the Multi Function B output of the serial port to address line A16 of the RAM.

Although it was more evident to map “Fred” (page &FC) to page #BC and “Jim” (page &FD) to page #BD in the Atom I did not because there are already some expansions addressed in page #BD so this is not competely free. There are no known expansions located in page #BC so that makes page #BC suitable for the paged RAM.

Circuit description

The circuit has the following blocks:

1. **Power supply**
   The board gets its +5V from either the User Port (a connector is provided on the board) or you can connect an external power supply to J9. Only the power lines of the User Port are connected. No other pins are used by the WiFi board. A voltage regulator provides the necessary 3.3V.

2. **CPLD: Control logic and level shifting**
   The ESP8266 is a 3.3V device but it is connected to the UART which is a 5V device. This is no issue for the receiving line but the transmitting line must be converted from 5V to 3.3V. So the RX input of the ESP8266 is not directly connected to the UART but to the CPLD.

   The CPLD also controls the RX and TX LEDs. Since the data transfer is quite fast, the human eye might miss some visual feedback. The CPLD is

triggered by a level change at each RX and TX input and will light the according LED for 0.10 seconds.

The most important function of the CPLD is the control logic. Most inputs are directly taken from the 1 MHz bus connector and converted to the necessary control signals for accessing the devices on the board.

Three unused pins of the CPLD are connected to extra pads on the board so they can be used for future features.

3. **RAM**
   The RAM CS, RD and WR signals are controlled by the CPLD. Address lines A0-A7 are also directly connected to the 1 MHz bus connector and A8-A15 are connected to the paged RAM register. A16 is connected to the MFB output of the UART to access the second bank of 64kB RAM.

   Selecting a page is as easy as writing to the paged RAM register. After a (power on) reset the paged RAM register and the copy in Scratch Pad Register B are probably out of sync. Before using the paged RAM the program should synchronize these by simply writing a value to #BDFF.

   The onboard RAM can be disabled by closing J8. This disables only the RAM, the paged RAM register is still in use, just like the shadow copy in the Scratch Pad Register.

4. **UART: the serial device**
   The UART is the interface between the CPU and the ESP8266. It has two independent channels.
   Channel A is unused by the WiFi board and can be used as a normal serial device. J7 has all the standard data and control signals plus an additional 5V. With a small extra interface you can directly connect a level converter for creating a real RS232 or RS432 interface. The Scratch Pad Register of channel A is used as a copy of the paged RAM register; Multi Function

output A is fed into the CPLD for future features. Channel A is located at #BD38-#BD3F (hence the choice of SPR#A for the register copy: all lower address lines are ‘1’).

Channel B is exclusively used for the WiFi interface. The RX and TX lines are for data communication. The other modem control signals are also used:

* DTR#B → enable/disable the ESP8266
* RTS#B → reset the ESP8266
* MF#B → extra address line for the paged RAM
* RI#B → paged RAM status: enabled or disabled

So it is possible to give the ESP8266 a real hardware reset, just in case it does not respond to any command. Just toggle the RTS line from high to low by writing a ‘1’ value to the corresponding bit in the modem control register. And of course, set is back to a high level by writing a ‘0’ to the modem control register. In a similar way it is also possible to disable the ESP8266. Just make the DTR output low by writing a ‘1’ to the corresponding bit in the modem control register. On a hardware reset of the UART this register is reset to all ‘0’ so the WiFi module becomes enabled again. To prevent this, you can write any value to the Line Status Register. This is unused in the UART device itself but it sets a flag inside the CPLD. This flag will prevent the CPLD to activate the RESET line to the UART. By writing any value to the Modem Status Register this flag will be cleared and reset signals are passed to the UART.

5. **ESP8266**

   The module will be fitted as an add-on to the board. For optimal stability it is soldered directly to the board. Exchanging the ESP8266 will be a bit complicated so remember not to change the baud rate (see chapter ‘warnings’).

Installation

Since the module is build and tested it is directly ready for use. Turn off your computer and connect the adapter board to the Atom Expansion connector. Then connect the Beeb WiFi board to the adapter board with the two flat cables. The User Port is only used to power the Beeb WiFi board. However, your Atom might need a small modification to use the EPROM and I/O space.

Please note that the Atom won't display a banner like the Electron and BBC Micro do because the software is not automatically initialized!

Standard Atom

The standard Atom is the most difficult to prepare for the WiFi module. As the 1 MHz bus adapter has room for an EPROM this hardware and software can be used on every Atom with the configuration as small as 8k ROM and 3k RAM (page #28 is used for WiFi workspace, so you need RAM from #2800 - #2BFF).

When your Atom has no ROM or RAM at block #Exxx you can best use the EPROM on the adapter board.

You'll need to open the bus driver (IC3, DP8304) when page #BC or page #BD are addressed. There are several ways to achieve this. This is explained in appendix ???

Atom with YARRB boards version 1 or 2

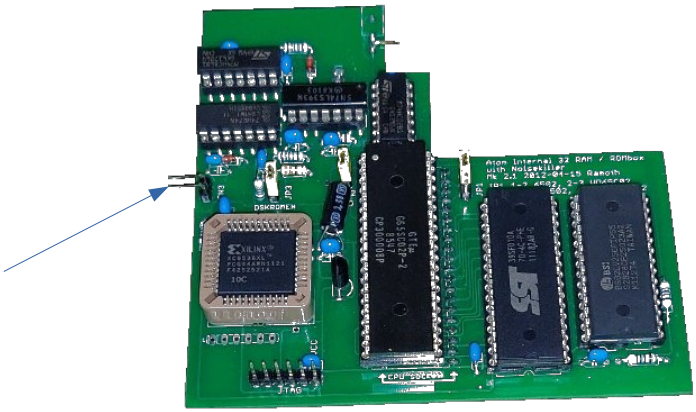
These Atoms already have ROM or RAM at block #Exxx so you can program the software into your ROM or load it into RAM. The bus driver is already taken care of by the CPLD on the Yarrb board.

Atom 2k14, 2k15 and FPGAtom

These Atoms also already have ROM or RAM at block #Exxx so you can program the software into your ROM or load it into RAM. The bus driver is already taken care of by the CPLD on the main board. FPGAtoms also might have two 64kB banks of paged RAMs but these are not used with the Beeb WiFi board and they won't conflict with each other.

Atom with Prime's RAMROM board

Note: only on versions Mk2.x or higher, if you have any add-on board that connects to PL6 or PL7 on the Atom, you will need to connect the supplied cable between either pin of CN3 and pin 8 of the IC5 socket .



Software

The ROM contains a set of new * commands and a driver for the ESP8266 module. In the next section I describe the new commands. In order to use these commands you have to initialize the ROM by typing either *DOS or LINK #E000[3]

Unlike the Electron and BBC Micro, the Atom has only 4 kB of ROM space available for the WiFi ROM, so a lot of features have been removed and only the basic functionality is provided.

DATE display date

Syntax: *DATE

This commands fetches the current date from the Internet. It fetches a special page from http://bbcmicro.nl to get the current time and date.

DISCONNECT disconnect from a server

Syntax: *DISCONNECT

You can use this command to disconnect from a server if the connection stays open due to an error condition. Most commands do automatically close their connection to the server but sometimes they stay open. When you get a message “Already connected” you can use *DISCONNECT to close the connection.

To disconnect from a wireless network use *LEAVE.

IFCFG display network information

Syntax: *IFCFG

3 You can also assemble the software for usage in another address space than block #Exxx. In that case you’ll have to initialize the ROM by LINKing to the appropriate start address of the ROM.

Use this command to show the current assigned IP address and the MAC address of your ESP8266. This command cannot be used to manually set your IP configuration.

JOIN connect to a wireless network

Syntax: *JOIN <ssid> [password]

In order to use the network functions you must first join a WiFi network. Use this command to join a network. The <ssid> is a required parameter. If you don't supply a password on the command line then you will be prompted to enter the password. Please keep in mind that both the ssid and the password are case sensitive and that the password might remain in the computer's memory!

If the ssid or the password of your network contains one or more spaces then you can put them between double quotes, like *JOIN "ACORN WIFI". You can type a space when you are prompted for the password. If your network has no password then you can simply enter an empty string.

The command *JOIN ? will show you to what network you are currently connected to.

LAP get a list of access points

Syntax: *LAP

This commands shows a list of access points. By default it shows this information for each access point:

+CWLAP:<ecn>, <ssid>, <rssi>, <mac>, <ch>, <freq offset>, <freq cali>

where
<ecn> = Encryption 0: OPEN, 1: WEP, 2: WPA_PSK, 3: WPA2_PSK, 4: WPA_WPA2_PSK
<ssid> = Network Id

<rssi> = Signal strength
<mac> = MAC address of access point
<ch> = Channel
<freq offset> = frequency offset of access point in KHz
<freq cali> = calibration for frequency offset

You can change this information with LAPOPT. The list of networks is always sorted by signal strength (rssi).

LAPOPT set options for lap command

Syntax: *LAPOPT <option>

The option is a binary value with each bit representing what field will be shown when you use *LAP:

bit 0 sets whether <ecn> will be shown
bit 1 sets whether <ssid> will be shown
bit 2 sets whether <rssi> will be shown
bit 3 sets whether <mac> will be shown
bit 4 sets whether <ch> will be shown
bit 5 sets whether <freq offset> will be shown
bit 6 sets whether <freq calibration> will be shown

So, for example, *LAPOPT 7 will only show the encryption type, the name (ssid) and signal strength (rssi) of the available WiFi networks.

LEAVE disconnect from current network

Syntax: *LEAVE

Disconnects you from the wireless network.

MODE select operating mode

Syntax: *MODE <1...3>

The ESP8266 can operate as a WiFi station (client) or as an access point (server) or both. With the *MODE command you can select the operation mode:

1 → station mode
2 → SoftAP mode
3 → SoftAP and station mode

The mode is also stored into the device's flash configuration and will remain until it is changed. When a new device is first powered on and it won't respond to commands it is probably not configured as a station. Setting mode to 1 will solve that issue.

With *MODE ? you can query the current mode of the device.

PING ping to a host

Syntax: *PING <host or ip-address>

You can test internet connectivity with this command or test whether a host is reachable. This command will send five "ping" packets and waits for the response from the remote host. When successful it will display the response time. If the host is not reachable then you will get a "No response from host" message. In case that the host does not exist or other failures you will see a "Host error" message.

TIME display current time

Syntax: *TIME

Like the *DATE command, you can also query the current time. See *DATE for additional information.

VERSION display firmware information

Syntax: *VERSION

This command retrieves the firmware version of the ESP8266 module.

WGET retrieve a file from the Internet

Syntax: *WGET [-T X A P] <url> [load address]

This tool downloads a file from a web server and either displays it on the screen (for example a text file) or stores it in the Micro's memory.

The tool has three parameters. The first one is an optional switch to indicate the file type. You may only specify one of these switches:

-T treat the file as a text file and display it on the screen after downloading the file. It will not be stored in the main memory.

-X this is the same as -T but it uses the code &0A as newline. Suitable to display Linux/Unix files.

-A the file will be downloaded into memory and has an ATM file header. If no load address is specified on the command line, the file will be stored on the load address in the header.

-P similar to the -A option, but now the file has an Atom-in-PC header. If no load address is specified on the command line, the file will be stored on the load address in the header.

The URL consists of a number of components:

protocol	(required, http and https are supported protocols)
hostname	(required)
port number	(optional)
path and filename	(optional)

For example: http://acornatom.nl:8080/path/to/file.htm

Although you can specify https as a protocol and the ESP8266 will connect to port 443 of the web server, it does not retrieve any data. This is probably related to either outdated encryption protocols or the ESP8266 might not support SNI.

The last parameter, load address, will override the load address that is in a header. If this parameter is omitted and the file has no header then it will be loaded at the current PAGE.

WIFI interface control

Syntax: *WIFI [on | off | sr | hr]

This command accepts one of these parameters:

on	enables the WiFi device
off	disables the WiFi device
sr	performs a software reset of the ESP8266 by issuing an AT+RST command
hr	performs a hardware reset of the ESP8266 by toggling the RTS line of the UART

Appendix I: Memory usage

The hardware uses the following addresses in the Atom memory map:

#BD30-#BD37	UART Port B (used by WiFi)
#BD38-#BD3E	UART Port A (available to user)
#BD3F	read back of paged RAM register
#BDFF	paged RAM register
#BC00 - #BCFF	paged RAM

The software uses the following addresses:

&0080 - &009F	temporary workspace in page 0
&2800 - &28FF	temporary workspace for heap and string buffer
&E000 - &EFFF	WiFi ROM (can be assembled to another space)

Please note: this is the memory usage as for the 4kB ROM at #Exxx. Other versions of the software may use different areas of memory.

APPENDIX II: ERROR MESSAGES

The error messages in the Atom WiFi ROM are very short, so here's a bit more explanation.

BUFFER FULL

The paged RAM buffer is full. This is mostly caused by a too large transfer. Keep in mind that there is some protocol overhead in each transfer so the maximum size per transfer is about 60 kB.

BUFFER EMPTY

The software reads a byte but all data is already read and processed. Mostly caused by a software or transfer error.

CONNECT ERROR

The requested connection could not be made. Try to *DISCONNECT or reset the ESP module with *WIFI HR to solve this error.

DEVICE?

The ESP device is not found. Check the board connections.

HTTP ERROR

The web server responded with an HTTP error. This error will also be displayed.

NO PAGED RAM

The paged RAM is not detected.

NO RESPONSE

There is no response received within the time out period. Either the (web) server takes too long to respond or the connection is not set up properly.

NOT IMPLEMENTED

The driver call is not implemented.

OPTION?

The specified command option is not known. Check your documentation.

PROTOCOL?

You specified an unknown protocol for the command or driver function. Check your documentation.

PARAMETER?

There is a bad parameter. Most likely by an unterminated string or bad hexadecimal value.

WIFI DISABLED

You are trying to use the WiFi board but is has been switched off. Use *WIFI ON to (re-) enable the WiFi board.

APPENDIX III: BUILDING THE ADAPTER BOARD

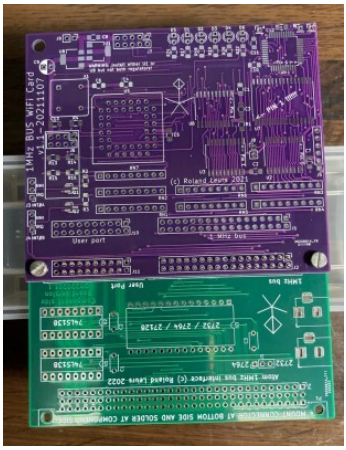
The adapter board is quite easy to build. It has only a very few components:

- A 64p AB connector, depending on your plans male or female.
- A 34p IDC header for the 1 MHz bus
- A 20p IDC header for the User Port
- 2 x 74LCT138 for address decoding, consider using IC sockets
- 2 x 100nF ceramic capacitor

Optional:

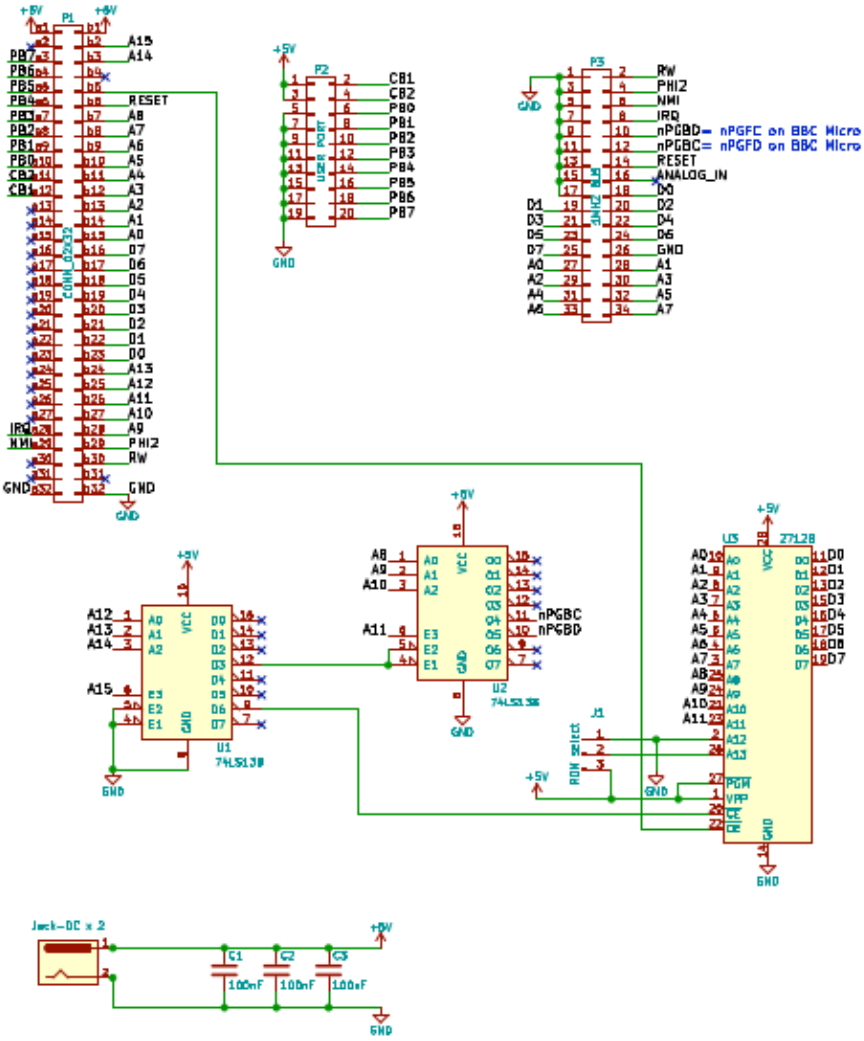
- A 2732, 2764 or 27128 EPROM, consider using a socket
- One or two barrel jack connectors for powering 1 MHz bus devices
- A three way jumper

Building the board is very straight forward. Just take care how you mount the 64p connector for the Atom. One of the rows is duplicated so you can fit the board both internally or externally. Make sure you mount this connector at the bottom of the board and solder it on the component side.



You might leave out the IDC header connectors when you decide to mount the adapter board and WiFi board permanently together like in the image on the right.

APPENDIX IV: 1MHz Bus Adapter diagram



APPENDIX V: ATOM BUS DRIVER CONTROL

If your Atom has no bus driver control like the Prime's RAMROM board (v2 and later), Atom2k14/15, FPGAtom or Yarbb 1 or 2 boards then you have to modify your Atom to enable the data bus driver when pages #BC or #BD are addressed. Consult the manual of your memory board for more information.

There are two way to do this:

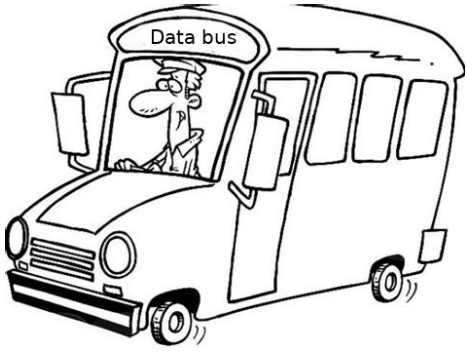
1. The very easy way: replace IC4 by wires.

This was done in the past by some people in the past. Simply remove this driver and place small pieces of wire between the A and B side. If the Beeb WiFi board is the only connected board to the Atom Expansion Connector and the flat cables are very short enough (less than 10cm / 4 inches) then this might work.

2. Provide an extra enable signal to IC5

First, let me explain how the Atom bus driver works. The Atom memory is designed in a way that all the memory and I/O on the main board is internally which means the bus driver IC4 (DP8304) is not activated. This goes for the following memory ranges:

- #0000 - #03FF
- #2800 - #3BFF
- #8000 - #DFFF
- #F000 - #FFFF



If one of these address blocks is addressed then IC5 (74LS30) will disable the bus driver. IC5 is an eight input NAND-gate. If one of the inputs goes low (0 volt) then the output (pin 8) goes to a high level (about 4 volt) and this will disable (switch off) IC4. If none of the inputs for IC5 is low then its output will be high and is IC4 enabled.

All of the inputs of IC5 are connected to the chip selects of the 2114 memory chip pairs. So if whatever 2114 (in the lower memory space) is addressed then one of the inputs will also go low and the bus driver is disabled.

But there's also block #Exxx which is by design external for the Atom Disc Pack. So when the DOS ROM is addressed the bus driver should also be disabled. The signal for this comes from IC8 (quad dual-input NAND gate). When A15 is high and #Exxx is not addressed (pin 9 of IC23 74LS138 is high) then the NAND gate will have a low output at pin 6 and so drive pin 11 of IC5 low which disables the bus driver. When #Exxx is addressed then pin 9 of IC23 goes low but A15 is still high so the output of the NAND gate goes high and that makes all inputs of IC5 high which means that the output goes low and that enables the bus driver.

So, block #Exxx is taken care for by design. However, we also need the bus driver be activated when we access our Wifi board. But the block #Bxxx does not enable the bus driver since A15 is high and block #Exxx is the only block in the upper part of the memory that can enable the bus. So we have to modify out Atom in a way that it will enable the bus when we access de WiFi board.

The I/O block is divided in four parts:

- #B000 - #B3FF (IC49, pin 4) 8255 PPI
- #B400 - #B7FF (IC49, pin 5) PL8 with Econet or AtoMMC
- #B800 - #BBFF (IC49, pin 6) 6522 VIA
- #BC00 - #BFFF (IC49, pin 7) unused

The last unused block is however decoded in the I/O address decoder (part 1 of IC49, 74LS139). So we can use this signal for our purpose, however, this will enable the bus driver for a bit larger address space than we need. I consider this as an advantage because other expansions can also benefit from this.

But how do we feed this signal to the bus driver enable circuit. It is seductive to feed the select signal from the 74LS139 directly into the 74LS30 but this will only be opposing since an active low input signal will disable the bus driver as we have read before.

The only way is to logical OR the signals #Exxx and #BCxx before they are “mixed” with A15. So we need to disconnect pin 9 of IC23 and combine it with pin 7 of IC49 just before we feed them into pin 4 of IC8. This seems a bit complicated but it’s just a bit fiddling with two small diodes (1N4148) and a 4k7 resistor. The best thing of all is that we don’t need to cut any tracks so no permanent damage will be done to the Atom!

Let’s do it:

1. Remove IC23 from its socket and bend pin 9 upwards
2. Place the first 1N4148 diode into the socket with the anode in the socket and the cathode upwards. The cathode is often marked with a circle on the diode. [*image 1*]
3. Place IC23 back in its socket and solder the cathode of the diode together with the upstanding pin 9. Cut the remaining wire off.
4. Solder the cathode of the second 1N4148 diode at pin 7 of IC49. Do not cut the lead of the anode. [*image 2*]
5. Remove IC8 from its socket and bend out pin 4. Reinsert IC8 into its socket.

6. Solder the lead of the second diode to pin 4 of IC8 and cut the remaining part of the lead. [*image 3*]
7. And then the final step: solder a 4k7 resistor between the diode and pin 16 of IC49; this is a pull up resistor for the open collector OR function that we have created with the two diodes. [*image 3*]

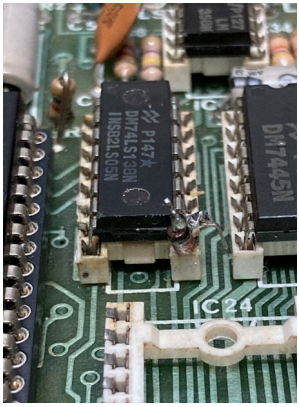


Image 1, IC23 pin 9

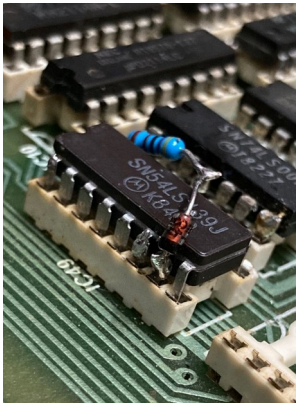


Image 2, IC49 pin 7

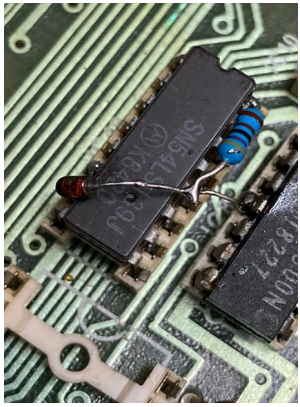
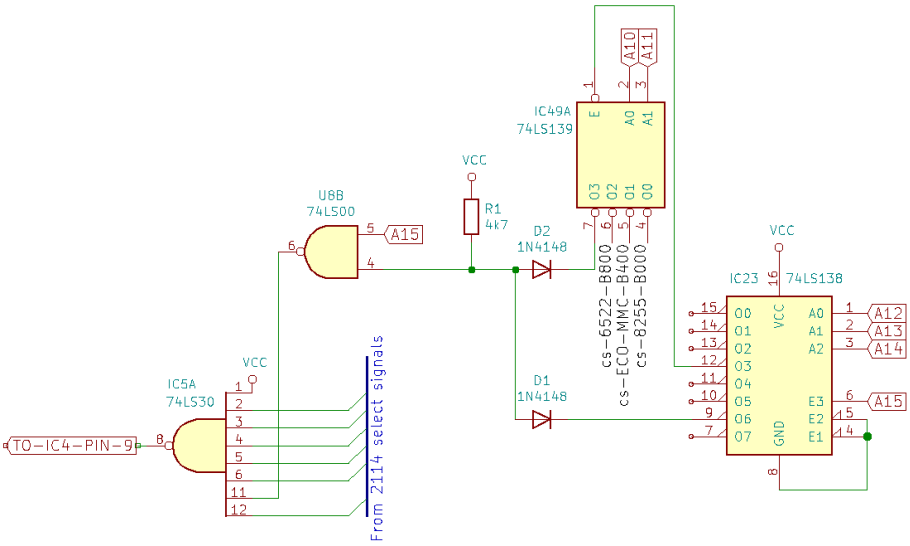


Image 3, IC8 pin 4

Now we have modified the address bus driver circuit to this schema:



Appendix VI: License

The license for this Work (i.e. both the hardware and software) is simple:

1. You have the right to use the Work in any way you want for non-commercial use. Commercial use is considered when you integrate the Work into your own products or replicate the Work and sell it.
2. You may create your own hardware and software based on the Work for non-commercial use. However, for deviated projects you must use the same license.

Warranties and Disclaimer

Except as required by law, the Work is licensed by the Licensor on an "as is" and "as available" basis and without any warranty of any kind, either express or implied.

Limit of Liability

Subject to any liability which may not be excluded or limited by law the Licensor shall not be liable and hereby expressly excludes all liability for loss or damage howsoever and whenever caused to You.