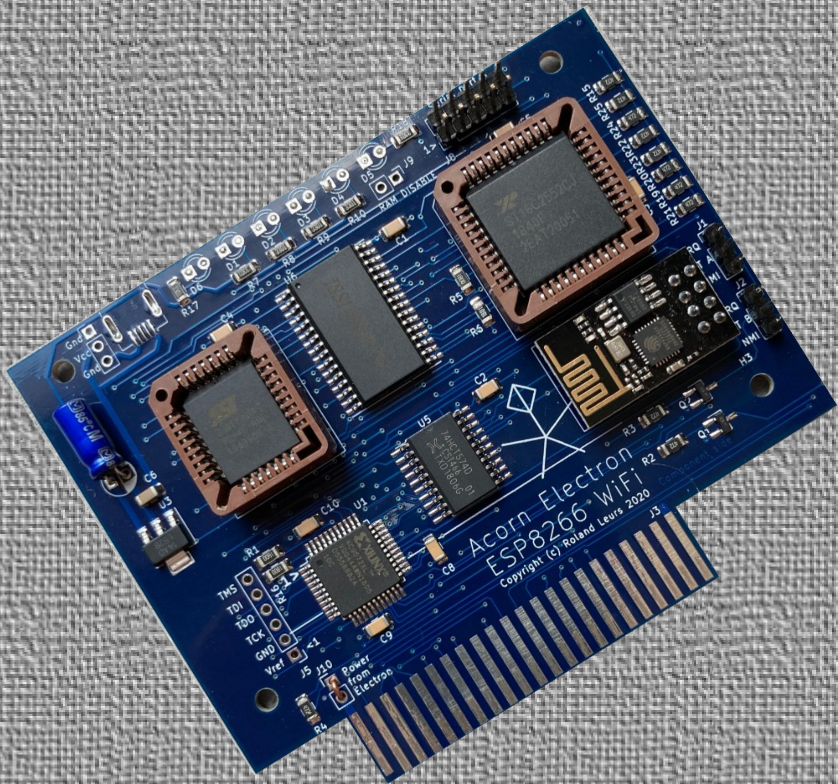
Acorn Electron WiFi

User Manual

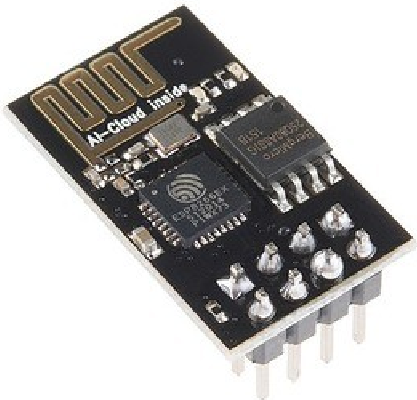


Roland Leurs

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.

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

WARNINGS

Before I continue I have two important warnings:

1. Security

The Acorn Electron 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 Electron 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).

Hardware description

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

Design considerations

The Electron is a wonderful, small and very sophisticated computer. To keep it small and affordable, Acorn used a very unusual method of accessing the memory. Instead of using 32k x 8 memory chips they used 64k x 4 chips. This means that accessing the memory by the CPU needs two read cycles of 4 bits. This is all taken care of by the ULA, from the programmers point of view, it just behaves like 8 bit memory. A drawback is that accessing the RAM is slow. I did some tests and even at 1200 baud I suffered lost data from the serial interface.

Another issue is that the Electron has not much memory. In the lowest text mode it has only about 20kB of RAM available. This implies that the received data must be processed and stored as soon as it comes in. Even an 8 MHz 6502 is not fast enough to do all the necessary checks and actions within 80 µs (the time that a complete byte is received at 115,200 baud). So we need a buffer that is fast and large. There are two ways to add more memory:

- Add sideway RAM in a bank at &8000-&BFFF
  This would either imply bank switching during data transfer or add both RAM and ROM to the bank that is used for the software. The memory is also limited to a maximum of 16kB. And that is not very much.

- Add Paged RAM at &FD00-&FDFF
  Acorn reserved a 256 byte page for memory expansion. This memory is “chopped” in small banks of 256 bytes. With a paged RAM register the program can select a bank. Since this register is 8 bit wide, we have 256 pages. The total memory can be 256 x 256 = 65535 bytes: 64kB.
  Since this memory is a real 8 bit memory it is large and fast enough. So I added this type of memory to my module.

The paged RAM register should be both writable and readable. I could do this by adding an extra buffer that is enabled when a read cycle occurs at &FCFF (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 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 &FCFF is just a normal register that can be written and read.

If your Electron already has paged RAM at &FDxx then you can disable the memory on the WiFi board by adding a jumper.

To make the WiFi software available without loading it from tape, disk or memory card I also added an EEPROM to the board. The WiFi is available directly after powering on the Electron.

Both the RAM and the EEPROM are 128kB and although memory is cheap these days it is also a pity if we can’t use it to the max. For both memory chips I connected the Multi Function output of the serial ports to an address line (A15 of the EEPROM, A16 of the RAM). This way the usable memory capacity is doubled without any extra hardware. It is important to know that as soon as MFA (the extra address line of the EEPROM) is changed, the sideway ROM banks also change without notifying the operating system. This can lead to a hanging Electron when the operating system knows that there was a ROM during boot time. So use this with extreme caution. It is most unlikely that switching will damage your Electron.

Circuit description

The circuit has the following blocks:

1. **Power supply**
   The board gets its +5V from the edge connector and so it is powered by the Electron. Therefore J10 should be closed. Normally this is not a jumper but a fixed wire bridge. It is possible to power the board from an external power supply, e.g. via an USB smartphone loader but before mounting the connector you should remove the link in J10!

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 edge connector and converted to the necessary control signals for accessing the devices on the board.

3. **RAM**
   The RAM is directly controlled from the Plus-1 (signal: nPGFD). The RD and WR signals are controlled by the CPLD. Address lines A0-A7 are also directly connected to the edge connector and A8-A15 are connected to the paged RAM register. A16 is connected to the MFB output of the UART to add another 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 &FCFF.

   The onboard RAM can be disabled by closing J9. This disables only the RAM, the paged RAM register is still in use, just like the shadow copy in the Scratch Pad Register[1].

4. **EEPROM**
   The EEPROM is controlled by the CPLD. However, it is addressed to &8000-&BFFF and the ROMQA signal is connected to A14. In this setup it provides two banks of sideway ROM. A15 is connected to MFA which makes it possible to activate another set of two banks. This should be used with care, like described in the *design considerations*.

5. **UART: the serial device**
   The UART is the interface between the CPU and the ESP8266. It

1 Note to myself: this disable signal can be connected to pin 1 of the CPLD. This way the CPLD can also disable the paged RAM register.

has two independent channels.
Channel A is unused by the WiFi board and can be used as a normal serial device. J8 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 Funcion output A is used as an extra address line for the EEPROM (see above). Channel A is located at &FC38-&FC3F (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.

6. **ESP8266**
   The module will be fitted as an add-on to the board. To prevent a physical clash between the ESP8266 and the Plus-1 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 Electron. Plug the WiFi board into a free slot of the Plus-1 and power on your Electron. It should start with a banner like “Acorn Electron WiFi”. A special WiFi symbol will be displayed whenever the module is enabled. If the WiFi symbol does not occur you can enable the WiFi module with the command

*WIFI ON

If you don’t see the WiFi banner at all then reinstall the module.

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. The driver functions are described later in this manual. Please not that you cannot access the driver functions directly; there is an OSWORD call for accessing these functions from your own applications.

DATE display date

Syntax: *DATE

This commands fetches the current date from the Internet. It fetches a special page from http://acornatom.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

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 Electron's memory!

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.

PRD dump contents of paged ram

Syntax: *PRD <address> <bank nr>

PRD stands for “Paged RAM Dump”. This command is used to inspect the contents of the paged RAM. The address is a 16 bit value with the most significant byte being the page number and the least significant byte is the offset within the page. The bank number is either 0 or 1 and selects the bank that you want to dump. The current bank number is always saved and restored after the dump command is finished. You can stop the dump by pressing the <ESC> key.

REWIND reset UEF pointer

Syntax *REWIND

Resets the pointer of an UEF file to the beginning of the file. See chapter “Working with UEF files” for more information.

TIME display current time

Syntax: *TIME

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

UPDATE check for updates

Syntax: *UPDATE [-R]

To facilitate easy updates of the Electron WiFi ROM you can use the *UPDATE command. If there is a newer version of the ROM available then you can download and install it with a single key press. When a newer version is found on the server then you will be prompted “There is an update available. Do you want to install it (y/N/R)?”. To install the update press ‘y’ (in lower case!) to start the update. Then, the newer ROM image will be downloaded and if the CRC16 of the downloaded image matches the CRC16 on the server then the EEPROM (only the WiFi part) is erased. After erasing, the new version will be “burned” into the EEPROM. Then routine ends with a simulated hard reset of the Electron to re-initialize the new ROM.

Warning: the new version will be downloaded in the memory area &2000 - &5FFF. So save any data before downloading the update!

Do not power off or interrupt the update process as this might leave your ROM in an unusable state. If you brick your ROM then see appendix 2 for a possible fix.

Please note that this software does not support any kind of update of the ESP8266 module, nor can the CPLD be updated with this tool.

If you use the optional parameter -R you will see a text file with the release notes for the latest versions. This overview is also shown when you press R after the question if you want to update. With the -R option the ROM will not be updates.

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 U A P S] <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 Electron'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 Electron's 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 the Electron's 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.

-U the file is an UEF file. This file will be loaded into the second 64k bank of the paged RAM.

-S the file a a ROM file and will be loaded directly into sideways RAM. You should specify the RAM bank number (0 – F) as the last parameter; the default is bank 0.

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.

WICFS Activate WiCFS

Syntax: *WICFS

WiCFS is an emulated cassette filing system using an UEF file as data source. This commands sets PAGE to &E00 and sets vectors FILEV, FSCV, FINDV and BGETV. See chapter “Working with UEF files” for more information.

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

Driver functions

The communication with the ESP8266 is done with a few standard functions. So there is no need to use any of the AT commands in user applications. If you write an application that needs some features that are not in the driver then please report this and it can be added to the driver. This way we can assure compatibility whenever another WiFi device will be used.

The driver supports these functions[2]:

Function	AT command	Short Description
00	AT+RST	Initializes the ESP8266 (soft reset)
01	n/a	Hard reset
02	AT+GMR	Get firmware information
03	AT+CWLAP	Get list of access points
04	AT+CWJAP	Join access point
05	AT+CWQAP	Quit access point
06	Not implemented	Set access point parameters
07	AT+CWMODE	Set device mode
08	AT+CIPSTART	Connects to host
09	AT+CIPMUX	Activate multiplex (up to five channels)
10	Not implemented	List joined interfaces
11	Not implemented	Set buffer address (old Atom driver)

2 Well, supports almost all of these functions. Some functions are not implemented yet.

Function	AT command	Short Description
12	AT+CIPSTATUS	Get TCP/IP connection status
13	AT+CIPSEND	Send data to remote host
14	AT+CIPCLOSE	Close connection to remote host
15	Not implemented	Set as server
16	Not implemented	Set time out
17	AT+CIOBAUD	Get baud rate (set baud rate is not implemented)
18	AT+CIFSR	Get IP and MAC address
19	Not implemented	Get firmware update
20	IPD	Transfer data
21	AT+CSYSWDTENABLE	Enable watchdog timer
22	AT+CSYSWDTDISABLE	Disable watchdog timer
23	n/a	Get multiplexer channel
24	n/a	Enable/disable WiFi device
25	AT+CWLAPOPT	Set options for LAP function
26	AT+CIPSSLSIZE	Set SSL Buffer size

Function 00 Soft reset

Parameters: *none*

This function sends a software reset command to the WiFi device. The device will reinitialize itself.

Function 01 Hard reset

Parameters: *none*

This function performs a hardware reset by toggling the RTS line from high to low and back to high.

Function 02 Get firmware information

Parameters: *none*

This function retrieves the firmware version from the WiFi device.

Function 03 Get list of access points

Parameters: *none*

This function retrieves a list of all wireless SSID’s in the neighbourhood. The response is like:

+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

See function 25 for LAP options. The ESP8266 might receive additional parameters for the AT+CWLAP command but these are not supported by this driver.

Function 04 Join access point

Parameters: *X points to the high byte of a parameter block*
Y points to the low byte of a parameter block

The parameter block holds two adjacent strings which are terminated with a &0D byte. The first string is the SSID of the access point and the second string is the password for that access point. Both strings should not contain quotes and are case sensitive.

On success the WiFi device is connected to the wireless network and gets an IP address. This connection is permanently stored in the device so it connect automatically to this network after the next (power on) reset.

Function 05 Quit access point

Parameters: *none*

This functions disconnects the WiFi device from the wireless network.

Function 07 Set device mode

Parameters: *Y = mode*

The ESP8266 has three operation modes:
1 → station mode
2 → SoftAP mode
3 → SoftAP and station mode

Note that these values are not binary values but the ASCII values. If Y = &00 then it will query the device for the current 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.

Function 08 Connect to remote host

Parameters: *X = hi byte param block*
Y = low byte param block

The parameter block contains the following information, presented as strings terminated by &0D:

<link ID>	(but only when multiplexing is active)
<type>	(UDP, TCP or SSL)
<remote IP>	(or hostname)
<remote port>	

No quotes are allowed around these parameters. On success the device responds with CONNECTED crlf crlf OK crlf crlf. On failure it will respond with an error message, such as *already connected* or *DNS failure* etc.

Function 09 Multiplexing control

Parameters: *Y = multiplexing on (1) or off (0)*

This function initializes the multiplex workspace for the driver and sends the multiplexer value to the ESP8266. This value might be both the ASCII value or the binary value.

Function 13 Send data to remote host

Parameters: *X points to data block in zero page*
Y = channel (note to myself: this is not implemented yet!)

This data block holds two addresses:

- two bytes start address of data (low byte / high byte)
- two bytes length (low byte / high byte)

Function 14 Close connection to host

Parameters: *Y = channel (only when multiplexing is active, otherwise ignored)*

Some servers or services close the connection after responding to function 13. So it is advised to always call this function after a transfer has completed. If the connection is already closed by the server then this function will end with an error that might be ignored.

Function 18 Get IP and MAC address

Parameters: *none*

This function retrieves the current IP address and MAC address of the ESP8266 device.

Fucntion 23 Get multiplexer channel

Parameters: *none*

You can use this function to request a free channel from the driver. Multiplexing must be enabled with function 09. This function responds with one of the following:

Carry cleared:Y = 255; multiplexing is switched off
Carry set: Y = 255: no free channel
Carry set: Y <> 255: assigned channel number

Function 24 Enable/disable wifi device

Parameters: *X = 0: disable WiFi, X = 1: enable WiFi*

This function causes the UART to pull DTR low and so it disables the ESP8266. It also configures the CPLD to prevent that the UART is reset after pressing the BREAK key. So, if WiFi is disabled it will be disabled even after a (hard) reset. After a power off/power on reset it will be enabled again.

Function 25 Set options for LAP function

Parameters: *to be determined….*

This sets the options which fields will be returned by the LAP function:

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

Function 26 Set SSL Buffer size

Parameters: *none*

This function sets the SSL buffer size to a fixed value of 4096.

The “receive data” routine is written in a way that it does not use any main memory of the Electron. It avoids instructions that change the stack (like JSR and PHA). So it only uses the ROM and paged RAM. This way, the CPU can run at full 2 MHz and is fast enough to read and store the incoming data. For graphical modes 0 to 3 it also disables interrupts, since these will make the CPU access the stack. In modes 4 to 6 the UART FIFO is large enough to buffer the incoming data during the execution of the interrupt service routine. So the WiFi functions can be used in any graphics mode.

Using the driver: OSWORD &65

Since you cannot access the WiFi driver functions directly, you can use this operating system call to get access to the driver.

Like any other OSWORD call the A register contains the function number (&65) and the X (lsb) and Y (msb) registers point to a parameter block. This parameter block is always three bytes and contains the parameters for the driver functions:

first byte:	the driver function number (A)
second byte:	the value for the X register
third byte:	the value for the Y register

Example:
You want to connect to a wireless network. So you store the ssid and password in memory at &680. The parameter block will be stored at &600.

To connect to the network do:

```
LDA #$04      \ load driver function
```

```
STA &600       \ write to first byte of driver
                              parameter block
LDA #&06       \ load X value for function
STA &601       \ write to second byte of driver
                              parameter block
LDA #&80       \ load Y value for function
STA &602       \ write to third byte of driver
                              parameter block

LDX #&00       \ load X register with low byte of
                              parameter block
LDY #&06       \ load Y register with high byte of
                              parameter block
LDA $&65       \ load OSWORD function number
JSR &FFF1      \ do OSWORD call
```

Working with UEF files

Unified Emulator Format (UEF) is a container format for the compressed storage of audio tapes, ROMs, floppy discs and machine state snapshots for the 8-bit range of computers manufactured by Acorn Computers. First implemented by Thomas Harte's ElectrEm emulator and related tools, it is now supported by major emulators of Acorn machines. Martin Barr wrote UPCFS which is a tool that reads UEF files from a PC using an UPURS connection.

UEF attempts to concisely reproduce media borne signals rather than simply the data represented by them, the intention being an accurate archive of original media rather than merely a capability to reproduce files stored on them. A selection of metadata can be included, such as compatibility ratings, position markers, images of packaging and the text of instruction manuals. UPCFS only supports file data.

With WiCFS I ported UPCFS to the WiFi ROM. Only a few changes were necessary, mainly fetching the data from memory rather than from a serial connection.

Downloading an UEF file

Before you can use an UEF file you have to download it from a (local) web server. Use the *WGET command for this with the -U switch, for example:

```
*WGET -u http://acornatom.nl/ddd.uef
```

This will download the UEF file and store it in the second bank of 64k paged RAM. The uncompressed UEF should not exceed 65533 bytes (the last two addresses are used to store the length of the UEF file).

Starting the WiCFS filing system

After loading the UEF file you can start the WiCFS filing system with the command:

```
*WICFS
```

This command activates *TAPE but changes the vectors FILEV, FSCV, FINDV and BGETV to its own routines. Commands like *CAT, *LOAD and *RUN are intercepted and redirected to the WiFi ROM. Page is set to &E00 as this is required for many games.

Also the read-pointer is reset to the beginning of the data.

Rewinding

Just like a real tape you might sometimes need to “rewind” the data in memory. Of course we are not moving the data around but we have to move the read-pointer to the beginning of the data in memory.

After you have *CAT-ted an UEF file you must reset the read-pointer to the beginning of the file before you can load a file into the main memory of your Electron. That’s all what the

```
*REWIND
```

command does.

Tape analogies

To place a tape into the recorder: *WGET -U <url>

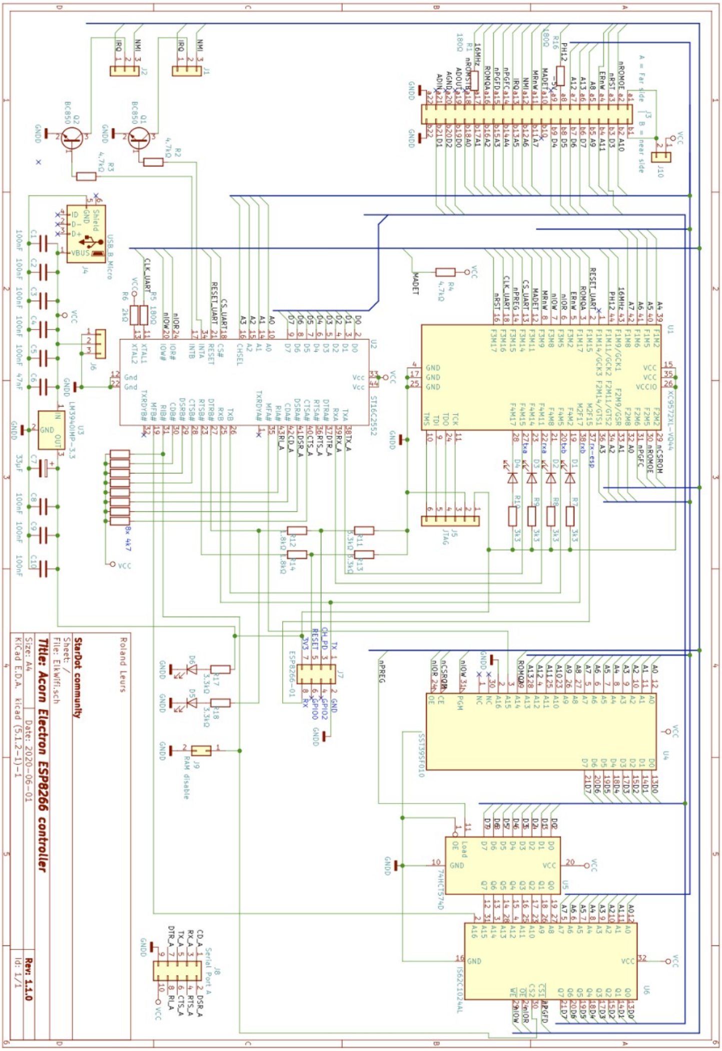
Switch on the tape recorder: *WICSF

Rewind the tape: *REWIND

Just like tapes, you might switch to another UEF file by issuing another WGET -U command. And just like real tapes you have to rewind it with *REWIND before accessing files otherwise you might get read errors.

You can compare the read-pointer at the UEF with the tape position just below the magnetic head of your cassette recorder.

Appendix I: circuit diagram



Appendix II: recovering a bricked WiFi ROM

In case the update process is interrupted or crashed, you can manually try to re-program the ROM. You need three files, that you can download from a web server:

http://acornelectron.nl/wifi/elkwifi-latest.bin
this is the 16kB ROM image, load at &2000

http://acornelectron.nl/wifi/erase.bin
this is an erase tool, load at &1E00

http://acornelectron.nl/wifi/program.bin
this is a programming tool, load at &1F00

It’s your own challenge to get these files into your Electron’s memory without a WiFi connection. At some point in time I will provide a SSD image that can be opened with one of the many MMC storage solutions.

If these three files are loaded into your Electron’s memory do:

CALL &1E00	to erase the EEPROM bank (break may be needed after flashing)
CALL &1F00	to program the latest image into the EEPROM bank

Two important remarks:

1. **Load all the three files into the Electrons memory if you need to transfer them over WiFi**

2. **This procedure only works with the cartridge in the rear slot of the Plus-1**

Appendix III: compatibility

This cartridge is tested with an Acorn Electron and a standard Acorn Plus-1 expansion module. It is also tested with an Acorn Plus-1 and a Pres Plus-1 ROM. Both configurations work perfectly.

Known issues:

- With a Pres AP6 ROM in a Plus-1 there is a slight corruption on the screen after a hard reset. The cartridge still works as expected.
- The CPLD is configured to detect whether it is installed in a BBC Master computer. However, the software does some access to page &FExx so there are some compatibility issues with this card in a BBC Master computer.
- Neither is any compatibility tested with a disc system, the Tube and Econet interface. They probably won't work correctly together.
- This board clashes also with the AP5 extension for the Electron as both boards drive (part of) page &FC and page &FD.
- Not all games are compatible with WiCFS. A list of known working games is at: https://www.retro-kit.co.uk/page.cfm/content/UPCFS-Working-titles/

You can send your feedback to: roland@acornatom.nl

Appendix IV: Memory usage

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

&FC30-&FC37	UART Port B (used by WiFi)
&FC38-&FC3E	UART Port A (available to user)
&FC3F	read back of paged RAM register
&FCFF	paged RAM register
&FD00 - &FDFF	paged RAM

The software uses the following addresses:

&0090 - &0096	permanent storage in page 0
&00B0 - &00CF	temporary workspace in page 0
&0100 - &0140	temporary workspace for error handling
&0398 - &03FF	used by WiCFS
&07A4 - &07FF	used by WiCFS
&0900 - &0AFF	temporary workspace for WiFi commands
&8000 - &AFFF	sideway ROM in either slot 0 or 2