skip to content

ECB USB FIFO

Introduction

The ECB-USB-FIFO board provides a high-speed interface for an EuroCard Bus machine to talk to a modern PC over USB. The board is based around the widely supported FTDI FT232H chip [http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232H.pdf] operating in "245-style asynchronous FIFO" mode. To avoid the need to solder surface-mount parts the Adafruit FT232H Breakout [https://www.adafruit.com/product/2264] board is used.

On the PC the interface is presented as a standard USB serial interface (a "Virtual COM port" on Windows, /dev/ttyUSB* on Linux) and can be used by any application normally used with standard serial ports. Note that although it presents as a serial port, the baud rate and other serial settings configured on the Virtual COM Port are ignored, it just provides an 8-bit clean path at whatever rate you can pump data in or out of it.

The FT232H has a separate FIFO (first-in-first-out queue) for the receive and transmit directions. Each FIFO is 1KB in size. Data is transferred to/from the FIFOs over the USB link in blocks of up to 512 bytes. A "send immediate" operation is supported which causes any queued data in the transmit FIFO to be sent over the USB link at the earliest opportunity, otherwise the FT232H may wait a short while to accumulate more data before performing a transfer over the USB link.

The board supports generating interrupts on either (or both) of two conditions: data waiting in receive FIFO, or space available in transmit FIFO. The interrupt can be routed to ECB lines NMI, INT, IR0–IR7. Enabling and masking of interrupts is under software control.

In addition to the 8-bit data path through the FIFO, there are two GPIO data lines (one input and one output) connected from the FT232H to the Status Register. These are not currently used but they were "free" to add to the design. To make use of them I believe you would need to use the FTDI MPSSE commands.

ECB I/O Registers

The ECB-USB-FIFO board decodes four adjacent I/O registers. The user can choose the top 6 bits (A2–A7) of the I/O address so it can be located anywhere in the I/O address space. The recommended base address is 0x0C (so the card occupies addresses 0x0C–0x0F).

The register use is as follows:

I/O address	Register use
BASE+0 (0x0C)	**Read:** Remove one byte of data from the receive FIFO **Write:** Insert one byte of data into the transmit FIFO
BASE+1 (0x0D)	**Read:** Read the Status Register **Write:** Write to the Status Register (note that not all bits are writable)
BASE+2 (0x0E)	**Read:** No effect (junk value is returned) **Write:** Send Immediate – any data queued in the transmit FIFO is sent over the USB at the earliest opportunity. Without this the data will be queued until a timeout (~17ms) expires. Note that the value written to the register is ignored, any value written to the register will trigger the Send Immediate signal.
BASE+3 (0x0F)	**Read:** No effect (junk value is returned) **Write:** No effect

The Status Register bits have the following meanings:

Bit (Value)	Meaning
0 (0x01)	**TX_FULL:** read-only 0 if the transmit FIFO has space available, 1 if the transmit FIFO is full. Any writes to the FIFO will be ignored while the transmit FIFO is full.
1 (0x02)	**INT_ENABLE:** read/write 0 if interrupts are disabled, 1 if interrupts are enabled.
2 (0x04)	**INT_TX:** read/write 0 if interrupts will not be generated to report transmit FIFO state, 1 if an interrupt will be generated when the transmit FIFO has space available (providing INT_ENABLE is also set).
3 (0x08)	**INT_RX:** read/write 0 if interrupts will not be generated to report receive FIFO state, 1 if an interrupt will be generated when the receive FIFO has space available (providing INT_ENABLE is also set).
4 (0x10)	**GPIO_IN:** read/write Single bit GPIO, output from ECB, input to FT232H
5 (0x20)	**GPIO_OUT:** read-only Single bit GPIO, output from FT232H, input to ECB
6 (0x40)	**IRQ:** read-only 0 if no interrupt would be requested providing INT_ENABLE is also set, 1 if an interrupt would be requested providing INT_ENABLE is also set.
7 (0x80)	**RX_EMPTY:** read-only 0 if the receive FIFO contains queued data, 1 if the receive FIFO is empty. Any reads from the FIFO while it is empty will return junk values.

Interrupts: Interrupt requests are controlled in two stages. The first stage is controlled by the **INT_TX** and **INT_RX** bits in the Status Register. Setting these bits to 1 causes an interrupt to be requested when the transmit FIFO has space available (**INT_TX**), or the receive FIFO contains data (**INT_RX**). The **IRQ** bit in the Status Register will be 1 only when either of these bits is set and the corresponding condition arises. The second stage is the **INT_ENABLE** bit: When the **INT_ENABLE** bit in the Status Register is set, and the **IRQ** bit is 1, the bus interrupt line selected with jumper **J3** will be asserted to interrupt the CPU.

For example, software which wants to receive an interrupt when data arrives in the receive FIFO over USB should set both **INT_RX** and **INT_ENABLE**. If the interrupt line is shared with multiple devices the interrupt service routine can then check the **IRQ** bit to determine if the USB-FIFO card is requesting the interrupt before calling the USB-FIFO interrupt handler code.

When the USB cable is disconnected, **TX_FULL** and **RX_EMPTY** will both be set, indicating that data cannot be transferred in either direction.

When the ECB machine is reset, the **INT_ENABLE**, **INT_TX**, **INT_RX** and **GPIO_IN** bits are all reset to 0. The USB connection is **not** reset (ie the virtual serial device will remain connected).

ECB Card Jumpers

J1: Reset input selector. Jumper 1-2 for legacy (DIN pin C31), or jumper 2-3 for Kontron (DIN pin C26). Kontron style (2-3) is recommended.

J2: I/O address selection. The top 6 bits of the I/O address can be set here. Presence of a jumper indicates a 0 bit, absence of a jumper indicates a 1 bit. Recommended configuration is to fit jumpers on A4, A5, A6, A7 to select I/O addresses 000011xx, ie 0x0C–0x0F.

J3: Interrupt line selection. Install at most one jumper to select the bus line to which the interrupt request signal will be routed. Recommended configuration is "IRQ" on Z80/Z180 machines, and "IR1" on 68K machines. If interrupts are not required the jumper may be omitted.

Schematic

PDF Schematic

Bill of Materials

Component	Description	Example Part
C1	47uF electrolytic capacitor, 0.2" lead spacing	Panasonic ECA-2AHG470 (Farnell order code 9694617)
C2–C11	0.1uF multilayer ceramic capacitor, 0.2" lead spacing	Multicomp MCRR50104X7RK0050 (Farnell order code 1216440)
C12	No component (see Notes on Construction)	-
RR1	22K bussed resistor network: 8 resistors, 9 pins	BOURNS 4609X-101-223LF (Farnell order code 9356924)
R1, R2	2.2K ohm 0.25W (or 0.125W) resistor	Multicomp MF25 2K2 (Farnell order code 9341536)
R3	Wire jumper (see Notes on Construction)	Use excess lead cut off from a resistor
J1	1×3 0.1" header	TE Connectivity part 5-826629-0: 1×50 contact breakaway header strip (Farnell order code 3418388)
J2	2×6 0.1" header	TE Connectivity part 5-826632-0: 2×50 contact breakaway header strip (Farnell order code 3418560)
J3	2×10 0.1" header	TE Connectivity part 5-826632-0: 2×50 contact breakaway header strip (Farnell order code 3418560)
U1, U3	74LS244	TI SN74LS244N (Farnell order code 1739688)
U9	74AHCT244 (preferred) or 74LS244	TI SN74AHCT244N (Farnell order code 1752769)
U2	74LS245	TI SN74LS245N (Farnell order code 1106085)
U4	74LS688	TI SN74LS688N (Farnell order code 1470949)
U5	74LS38	TI SN74LS38N (Farnell order code 1470758)
U6	74AHCT139 (preferred) or 74LS139A	TI SN74AHCT139N (Farnell order code 1741539)
U7, U11	74ACT32 (preferred), 74AHCT32 (untested but should work) or 74LS32	TI SN74ACT32N (Farnell order code 1739969), TI SN74AHCT32N (Farnell order code 1749941) or TI SN74LS32N (Farnell order code 1740030)
U8	74LS175	TI SN74LS175N (Farnell order code 1607731)
U10	Adafruit FT232H Breakout (product 2264)	Mouser order code 485-2264, Digikey order code 1528-1449-ND
U12	DIN 41612 Connector, Type C Plug, 96 pin	TE Connectivity 5650913-5 (Farnell order code 1557020)
(qty 5)	20 pin DIP IC socket	Multicomp SPC15501 (Farnell order code 2678574)
(qty 2)	16 pin DIP IC socket	TE Connectivity 1-2199298-4 (Farnell order code 2445622)
(qty 3)	14 pin DIP IC socket	TE Connectivity 1-2199298-3 (Farnell order code 2445621)

Notes on Construction

Important note on R3, C12: Footprints are provided for these components but they should not normally be required. **No component** should be fitted at **C12**. A **wire jumper** should be fitted at **R3**.

The FT232H /WRITE signal is active on the falling edge of the signal, instead of the standard rising edge. To ensure the data lines are steady for long enough to meet its setup time requirements, the write signal is delayed by routing it through a few gates in U3 and U7. R3 and C12 can form an RC circuit used to extend this delay if required. Alternatively the delay can be shortened by omitting the wire jumper at R3 and connecting U3 pin 17 to U3 pins 5, 7 or 13. Testing indicates that the standard delay should suit all current Retrobrew Computers systems and so the RC circuit components should not be required.

The Adafruit FT232H breakout board can be permanently mounted using either 0.1" breakaway header strips soldered to both boards, or it can be made removable using SIL pin strips and sockets (eg Harwin D01-9922046 and D01-9973246) which allows it to be re-used for other projects.

The FT232H breakout board contains a serial EEPROM which **must be programmed** to configure the FT232H to use the correct operating mode ("245 Asynchronous FIFO"). This can be done over USB from a PC. The EEPROM needs to be programmed one time only.

For Windows you need to use the FT_PROG [http://www.ftdichip.com/Support/Utilities.htm#FT_PROG] utility which can be freely downloaded from the FTDI web site. I've made a template that you can load into the FT_PROG program. The program is a bit counter-intuitive so here are some brief instructions: Connect the USB device, run FT_PROG, load the provided XML template. Tell FT_PROG to scan for devices, then right-click on the FT232H USB device (not the template) and tell it to apply the loaded template. This sets the desired configuration for the device. Click the "Program Devices" button in the toolbar, then click the "Program" button. This writes the configuration to the EEPROM.

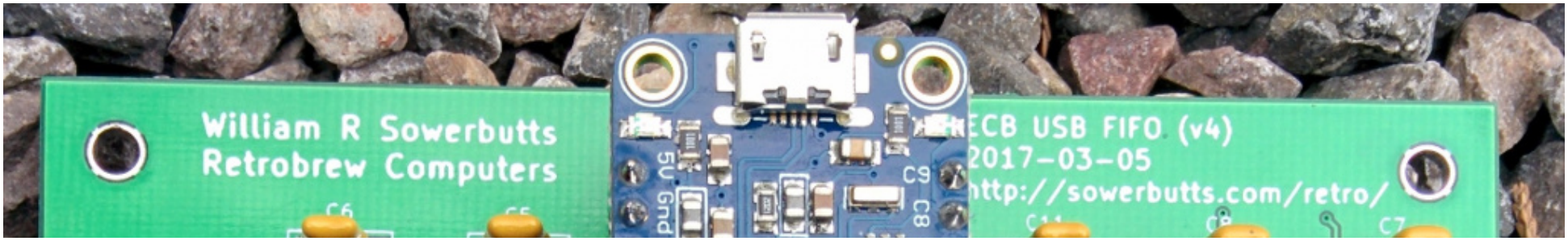
For Linux I have written a short C program which uses libftdi1 to program the EEPROM. The tarball contains both the source code and a binary with libftdi1 and libusb statically linked in, this should work on most recent Linux systems. Ensure the FT232H board is the only connected FTDI device (just in case it tries to overwrite the EEPROM in other devices!) and then run "sudo ./program-eeprom -w" to program the device. Running "sudo ./program-eeprom" (without the "-w" option) will report the current EEPROM contents.

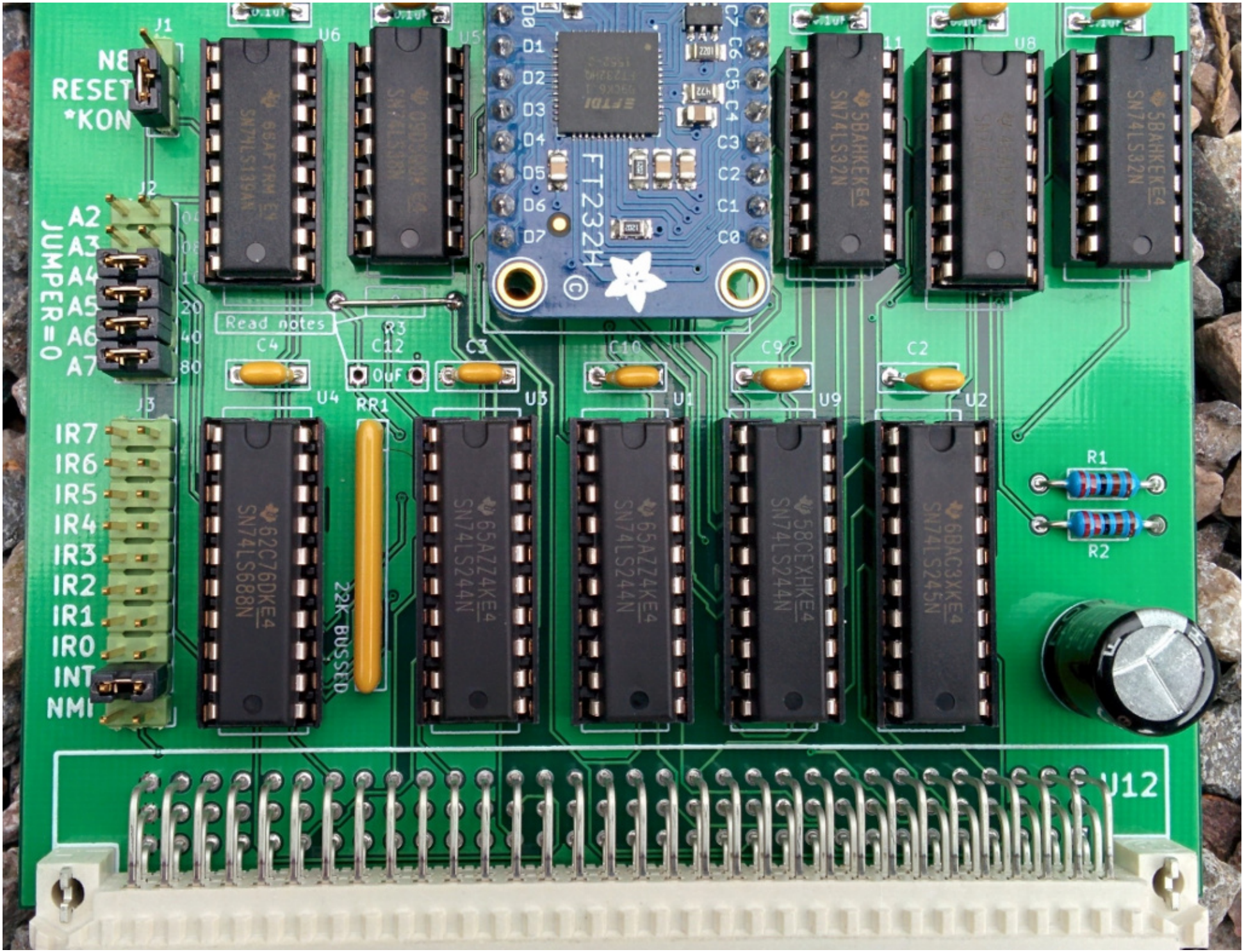
For MacOS I had success running FT_PROG in a virtual Windows machine (using Parallels Desktop) using the USB passthrough feature. In principle it should also be possible to compile libusb, libftdi1 and use my C program to program the EEPROM but I have not tested this.

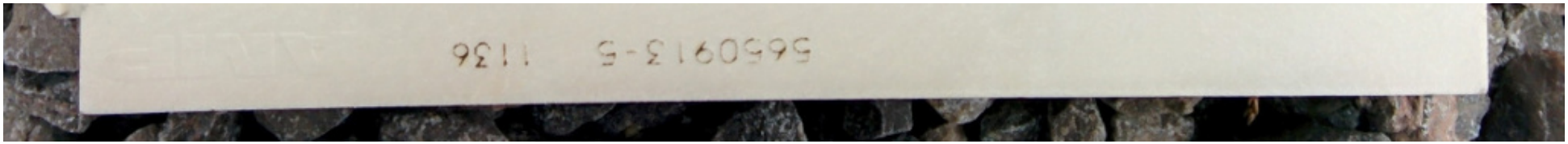
It is possible to build this board with entirely 74LS series logic, and this works well. However slightly higher performance can be achieved by substituting 74AHCT parts for U6 and U9, and 74ACT for U7 and U11. Testing on a Mark IV SBC at 36.864MHz, a board using only 74LS parts requires 2 I/O wait states, whereas with the 74AHCT parts at U6 and U9 the board operates correctly with only 1 I/O wait state. U7 and U11 can use 74ACT32 which also gives a further timing improvement. The forum thread [https://www.retrobrewcomputers.org/forum/index.php?t=msg&th=142&goto=2061&#msg_2061] has more information on how the timing improves with CMOS parts. Note that U1, U2 and U3 should always use 74LS as the bus interface benefits from having hysteresis on the inputs to improve noise rejection; the 74LS244 and 74LS245 parts have Schmitt trigger inputs.

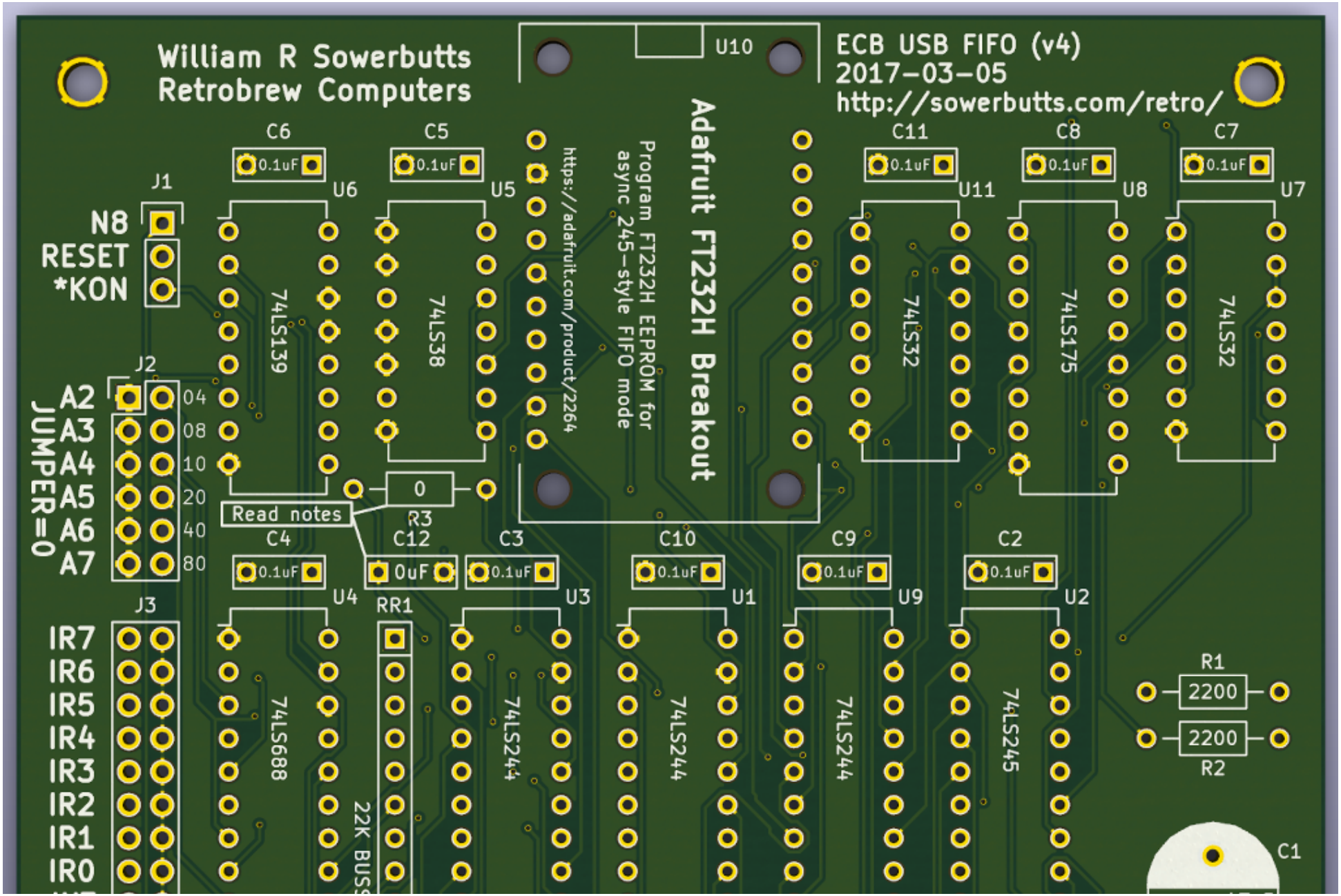
FT232H boards produced after Feb 2020 (using a USB-C connector instead of Micro-USB) have two additional pins for 3V and GND at the end of the board closest to the USB connector. These should be left disconnected.

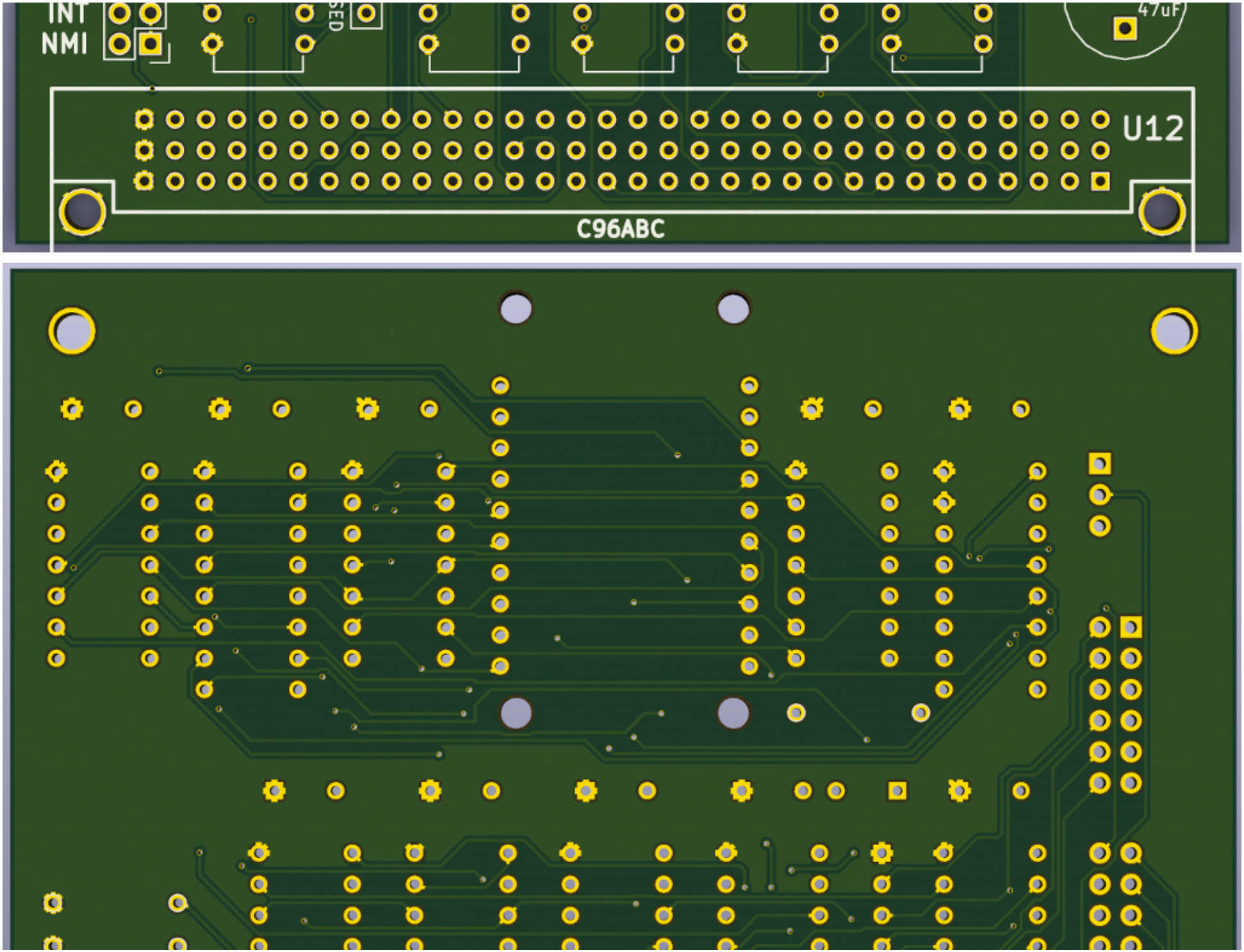
Photographs

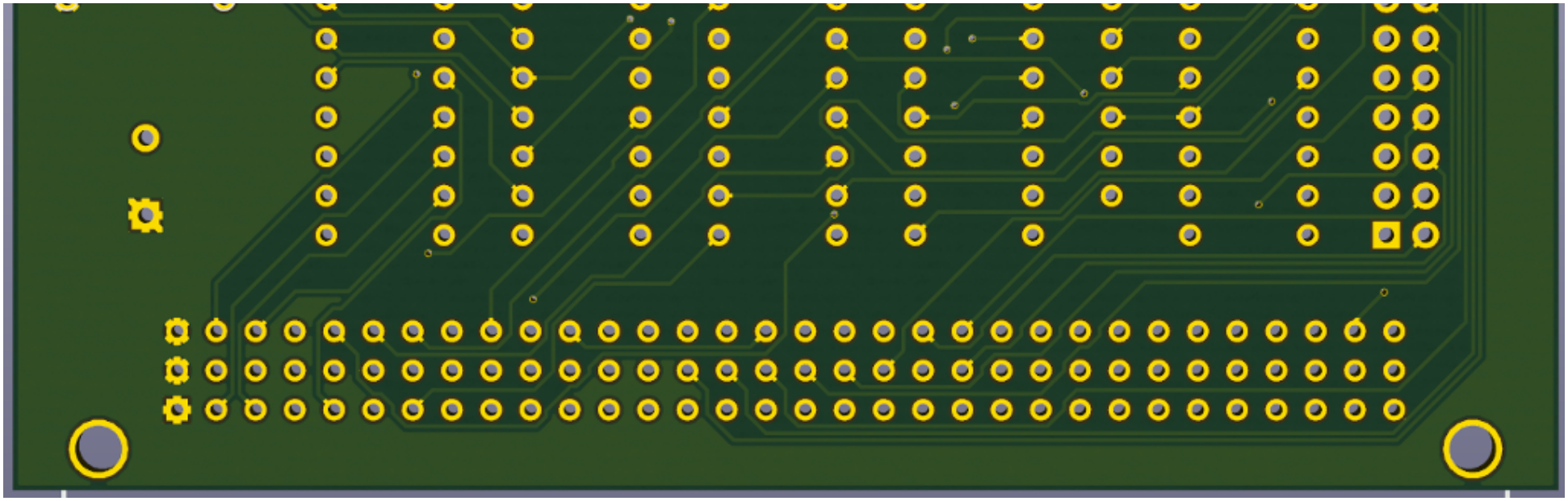



J1
RESET
*KON
J2
A2
A3
A4
A5
A6
A7
JUMPER=0
J3
IR7
IR6
IR5
IR4
IR3
IR2
IR1
IR0
INT
NMI
U6
U5
U11
U8
FT232H
D0
D1
D2
D3
D4
D5
D6
D7
C7
C6
C5
C4
C3
C2
C1
C0
Read notes
R3
C4
C12
0uF
C3
C10
C9
C2
U4
RR1
U3
U1
U9
U2
22K BUSSED
R1
R2
U12




William R Sowerbutts
Retrobrew Computers
ECB USB FIFO (v4)
2017-03-05
http://sowerbutts.com/retro/
Adafruit FT232H Breakout
https://adafruit.com/product/2264
Program FT232H EEPROM for
async 245-style FIFO mode
U10
J1
N8
RESET
*KON
J2
JUMPER=0
A2
A3
A4
A5
A6
A7
04
08
10
20
40
80
J3
IR7
IR6
IR5
IR4
IR3
IR2
IR1
IR0
Read notes
R3
0
RR1
22K BUSS
R1
2200
R2
2200
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
0.1uF
0uF
U1
U2
U3
U4
U5
U6
U7
U8
U9
U11
74LS139
74LS38
74LS32
74LS175
74LS32
74LS688
74LS244
74LS244
74LS244
74LS245


INT
NMI
47uF
U12
C96ABC



Example Code

Note that this code has been optimised on a Z180. On Z80 it may be faster to replace the JR instructions with JP (untested).

```
FIFO_BASE       = 0x0C
FIFO_DATA       = (FIFO_BASE+0)
FIFO_STATUS     = (FIFO_BASE+1)
FIFO_SEND_IMM   = (FIFO_BASE+2)

; Receive a block of length DE bytes from the FIFO to memory starting at address HL
fifo_receive:
        ; Enter with:
        ;   HL = pointer to data buffer
        ;   DE = number of bytes to transmit
        ld b, e                 ; setup loop counter
        dec de
        inc d
        ld c, #FIFO_DATA        ; load port address
waitrx:
        in a, (FIFO_STATUS)
        rla                     ; RX_EMPTY bit into carry flag
        jr c, waitrx            ; loop until data ready
        ini                     ; read from port C, write to HL, HL++, B--
        jr nz, waitrx           ; INI sets Z flag to indicate new value of B
        dec d
        jr nz, waitrx
        ret

; Transmit a block of length DE bytes to the FIFO from memory starting at address HL
fifo_transmit:
```

```
        ; Enter with:
        ;   HL = pointer to data buffer
        ;   DE = number of bytes to transmit
        ld b, e                 ; setup loop counter
        dec de
        inc d
        ld c, #FIFO_DATA        ; load port address
waittx:
        in a, (FIFO_STATUS)
        rra                     ; TX_FULL bit into carry flag
        jr c, waittx            ; loop while tx fifo full
        outi                    ; read from HL, write to port C, HL++, B--
        jr nz, waittx           ; OUTI sets Z flag to indicate new value of B
        dec d
        jr nz, waittx
        ret

; Request transfer of transmitted data ASAP (without this it is delayed up to 17ms)
fifo_flush:
        out (FIFO_SEND_IMM), a  ; we can write any value to the register
        ret
```

I also have a file transfer application for CP/M, FIFOPIPE. Drop me an email (will /at/ sowerbutts.com) if you'd like to try it out.

Software support

RomWBW included driver support for the ecb-usb-fifo board. The driver does not support interrupts. It has been confirmed working on the SBC-V2 and the MKIV. Performance is poor compared to normal serial communication but is dependent on processor speed. The driver can be enabled with the following assembly customization:

```
UFENABLE  .SET  TRUE
```

Inclusion of the driver in RomWBW will result in the board appearing as SIO serial port. The port is virtual in nature and changing serial settings have no affect on it's operation.

```
RetroBrew HBIOS v2.9.2-pre.3, 2019-08-07

SBC Z80 @ 3.992MHz
0 MEM W/S, 1 I/O W/S, INT MODE 2
512KB ROM, 512KB RAM

UART0: IO=0x68 16550A MODE=38400,8,N,1
DSRTC: MODE=STD Thu 2019-08-08 17:42:23 CHARGE=ON
MD: UNITS=2 ROMDISK=384KB RAMDISK=384KB
USB-FIFO: IO=0x0C

Unit        Device      Type              Capacity/Mode
----------  ----------  ----------------  --------------------
Disk 0      MD1:        RAM Disk          384KB,LBA
Disk 1      MD0:        ROM Disk          384KB,LBA
Char 0      UART0:      RS-232            38400,8,N,1
Char 1      SIO0:       RS-232            9600,8,N,1
```

Depending on how many character based ports are in your system, you may be able to use CP/M redirection to access the USB-Fifo port. In order for the the USB-fifo board to be used as the console device it must be set as the first RomWBW CIO device through the FORCECON option.

```
FORCECON .SET 2
```

Serial File transfer using the USB-Fifo can be accomplished using Willl Sowerbutts fifo-pipe program or using XModem.

Windows Driver Configuration

The Windows FTDI drivers allows the capability of the FT232H device to be configured through the device manager.

Recommended changes from the default setting are:

- Serial Enumerator - unchecked
- Disable Modem Ctrl At Startup - checked

Detailed driver setting information can be seen in this application note
[http://www.ftdichip.com/Support/Documents/AppNotes/AN_107_AdvancedDriverOptions_AN_000073.pdf].