Power Input USB Port

This port supplies power to the board but not data. A separate port is used to avoid demanding too much power from the host, which might be a single-board computer without too much power to share with USB devices.

J3
USB4105-GF-A
VBUS
CC1
CC2
D-
D-
D+
D+
SHIELD
GND
SBU1
SBU2
S1
A1
A4
A5
B5
A7
B7
A6
B6
A8
B8
PWR_FLAG
GND

VBUS_IN
5.1k R11
5.1k R12
GND

C12
2.2u
GND

FB3
300@100MHz

C13
2.2u
GND

+5V
PWR_FLAG
C16
100n
GND

3.3V LDO

U9
LD1117S33TR_SOT223
VI
VO
GND
3
2
1
+3.3V
C17
10u
GND

U8
USBLC6-2SC6
5
6
4
1
3
2
GND

VBUS is sent through a TVS diode, then a ferrite bead filter recommended by Phil's Lab on Youtube before going to the rest of the circuit

1.5A total available on +5V, and up to 800mA can be converted to 3.3V

12MHz CMOS Oscillator

+3.3V
C1
100n
EN
Vdd
GND
OUT
4
1
2
3
X1
ASV-12.000MHZ-EJ-T
GND
12MHZ_CLK

Weak connection between MISO and MOSI on the SPI bus. This means that if no device is driving MISO, it will tend to be a "mirror" of MOSI. This way, we can do basic tests of sending out SPI data and seeing if we get the same data back.

SPI.MISO
10k R32
SPI.MOSI

+3.3V
I2C.SCL
1.1k R33
I2C.SDA
1.1k R34

I2C pullups. 1.1k gives 3mA pulldown current, the maximum allowed by the standard. These pullups are compatible with regular I2C but should give decent signal integrity for fast mode plus as well.

I2CEEPROM
I2C{SCL SDA}
File: I2CEEPROM.kicad_sch

FT232H
12MHZ_CLK
USB{D+ D-}
I2C{SCL SDA}
UART{MCU_TX MCU_RX RTS CTS}
SPI{SCLK MOSI MISO SD_CS HW_CS}
BUS_LINES[0..3]
FT232_FUNC_SELECT[0..2]
File: FT232H.kicad_sch

INT_USB1{D+ D-}

ArduinoConnector
I2C{SCL SDA}
UART{MCU_TX MCU_RX RTS CTS}
SPI{SCLK MOSI MISO SD_CS HW_CS}
FT232_FUNC_SELECT[0..2]
GPOUT[0..2]
File: ArduinoConnector.kicad_sch

MicroSDCard
SPI{SCLK MOSI MISO SD_CS HW_CS}
File: MicroSDCard.kicad_sch

USB Hub
12MHZ_CLK
INT_USB1{D+ D-}
INT_USB2{D+ D-}
File: usb_hub.kicad_sch

INT_USB2{D+ D-}

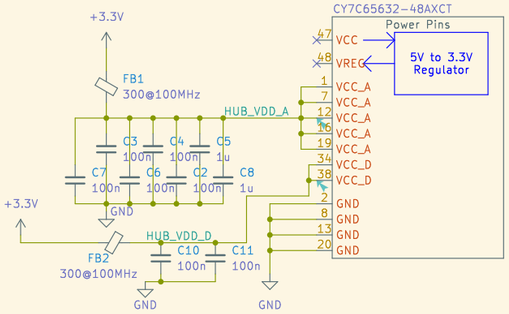
BUS_LINES0
BUS_LINES1
BUS_LINES2
BUS_LINES3
SPI.SD_CS
GPOUT0
GPOUT1
GPOUT2
NT1
NT2
NT3
NT4
NT5
NT6
NT7
NT8
LOGIC_IN0
LOGIC_IN1
LOGIC_IN2
LOGIC_IN3
LOGIC_IN4
LOGIC_IN5
LOGIC_IN6
LOGIC_IN7

Logic Analyzer Input Mapping (depending on FUNC_SELECT settings and specific test):

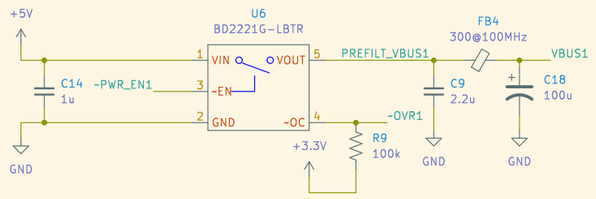
0: UART.MCU_RX/I2C.SCL/SPI.SCLK
1: UART.MCU_TX/I2C.SDA/SPI.MOSI
2: UART.RTS/SPI.MISO
3: UART.CTS/SPI.HW_CS
4: SPI.SD_CS
5: GPOUT0
6: GPOUT1/PWM
7: GPOUT2

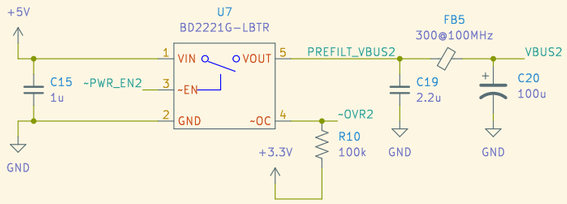
Logic Analyzer
LOGIC_IN[0..7]
USB{D+ D-}
File: logic_analyzer.kicad_sch

The reference design seems to recommend:
– 1x 10uF and 1x 1uF on VREG (which I assume is needed only if the internal regulator is used)
– A 100nF for each VCC_A pin, plus 2x 1uF on VCC_A, plus a ferrite bead
– A 100nF for each VCC_D, plus a ferrite bead



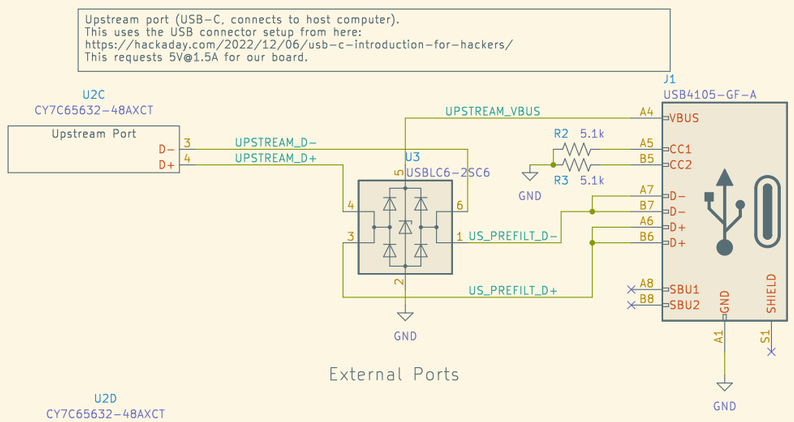
External port power switches



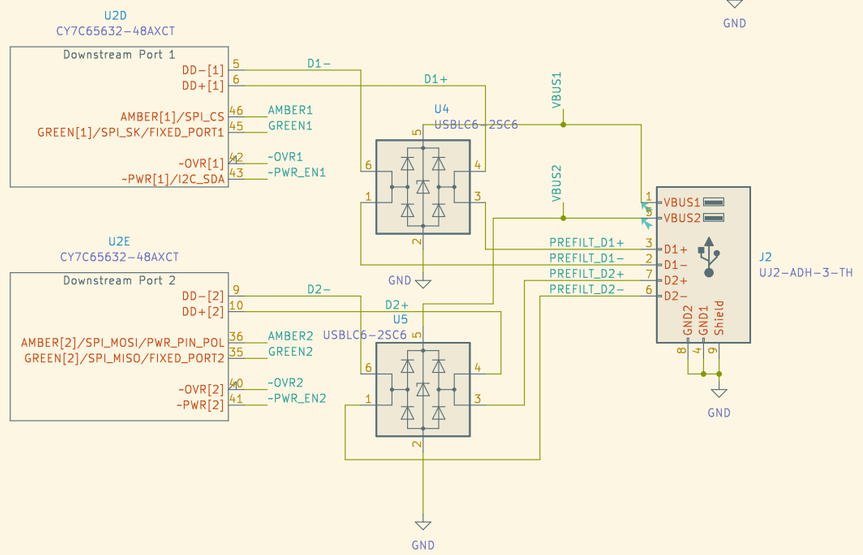


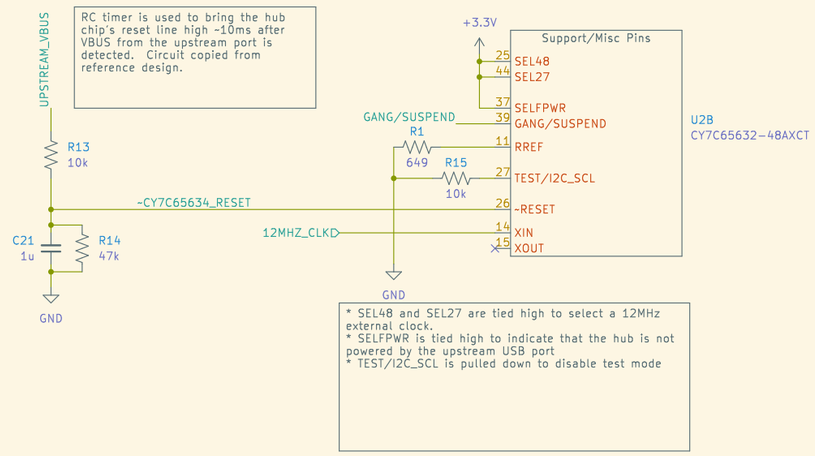
Upstream Port

Upstream port (USB-C, connects to host computer).
This uses the USB connector setup from here:
https://hackaday.com/2022/12/06/usb-c-introduction-for-hackers/
This requests 5V@1.5A for our board.



External Ports





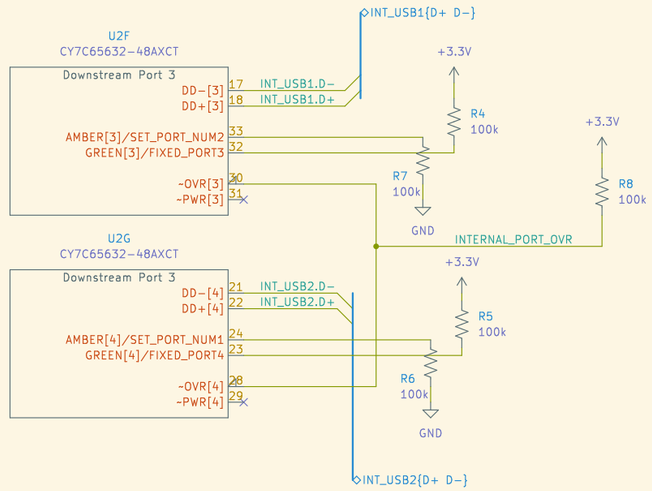
Internal Ports

Internal ports. These ports are used to provide USB to devices inside the CI shield. Due to this, they lack ESD protection or their own power supply.

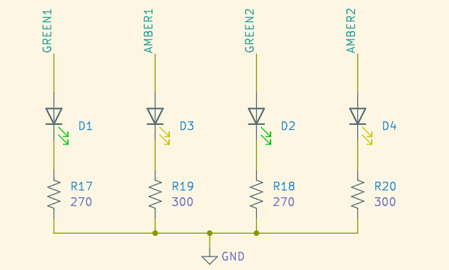
The FIXED_PORT pins are strapped to high to indicate to the hub that these devices are non-removable.

The SET_PORT_NUM[2:1] pins are strapped to 00 to indicate that all 4 ports are in use.

The OVR pins are tied to high to disable overcurrent detection.

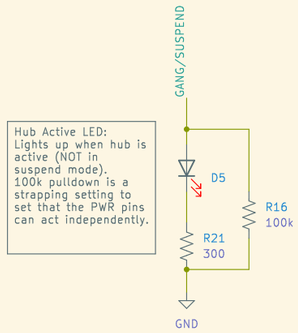


Port Indicators



Note: For the GREEN pins, we rely on the internal pulldown resistor to strap them as low. This configures port 1 and port 2 as removable.

Also, for AMBER2, we allow this to be pulled low internally to strap the ~PWR[] pins to active-low polarity.



The four post-multiplexing bus lines are tapped for the logic analyzer

Analog multiplexer allows the FT232H to be connected to different pins on the MCU depending on what the test suite wants to test.

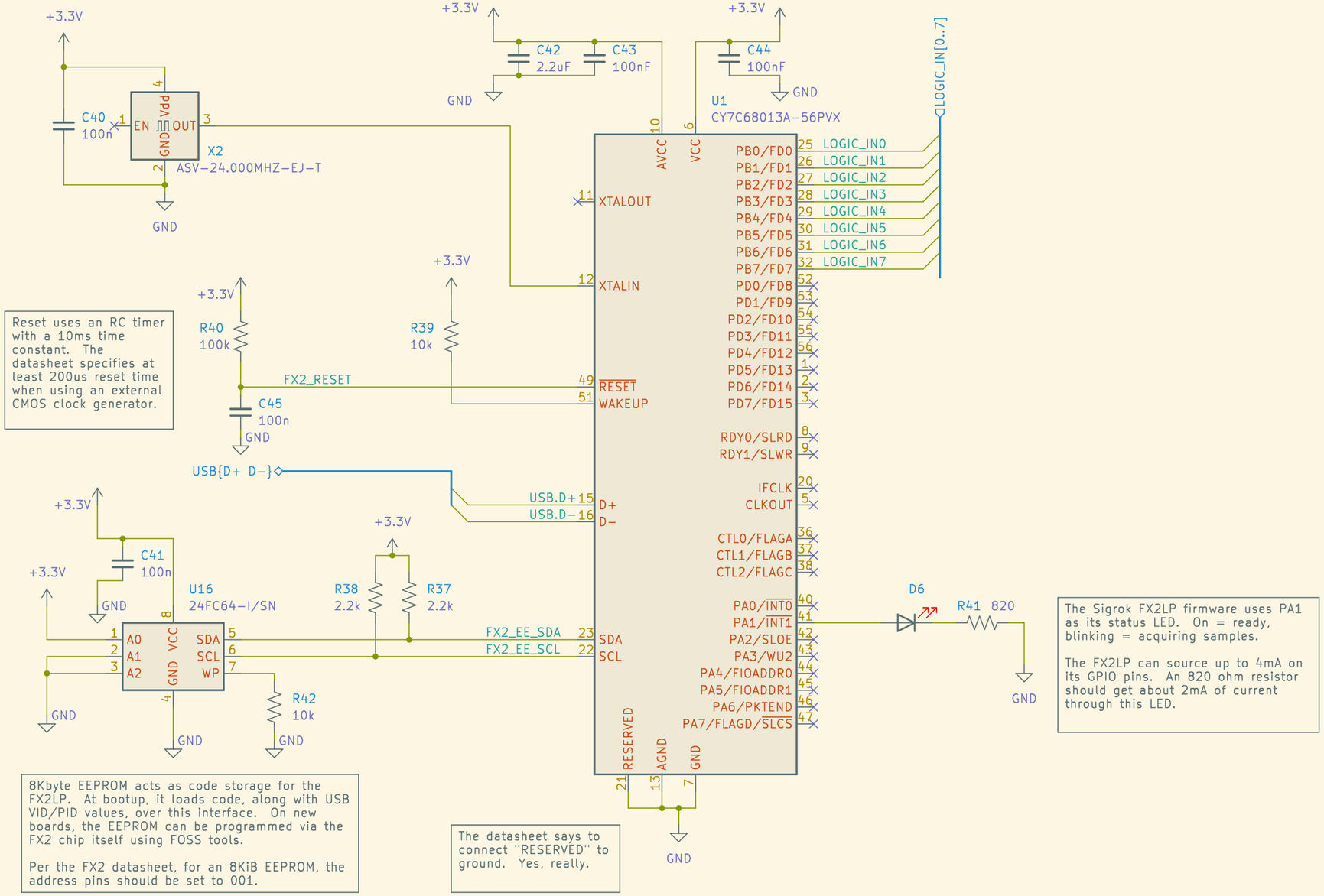
FUNC_SELECT	Function
000	UART
001	I²C
010	SPI
1xx	Hi-Z

Using datasheet recommended reset circuit

Per pyftdi docs, ADBUS7 needs to be connected to SCL in order to support clock stretching. Luckily, ADBUS7 is an input in the default UART mode.

Ensure that the default setting is 100 (Hi-Z)

+3.3V
FB7
300@100MHz
FT232H_VPLL
C30 4.7u
C31 100n
GND
+3.3V
FB6
300@100MHz
FT232H_VPHY
C28 4.7u
C29 100n
GND
+3.3V
C32 100n
C33 100n
C34 100n
GND
BUS_LINES[0..3]
U10
FT232H
VPHY 3
VPLL 6
VCCIO 12
VCCIO 24
VCCIO 46
+3.3V
C24 4.7u
C23 100n
C25 4.7u
C22 100n
C26 100n
C27 100n
GND
USB{D+ D-}
VREGIN 40
VCCD 39
VCCCORE 38
VCCA 37
USB.D- 6 DM
USB.D+ 7 DP
RESET 34
REF 5
EECS 45
EECLK 44
EEDATA 43
XCSI 1
XCSO 2
TEST 42
AGND 4
AGND 9
AGND 41
GND 10
GND 11
GND 22
GND 23
GND 35
GND 36
GND 47
GND 48
ADBUS0 13 BUS_LINES0
ADBUS1 14 BUS_LINES1
ADBUS2 15 BUS_LINES2
ADBUS3 16 BUS_LINES3
ADBUS4 17
ADBUS5 18
ADBUS6 19
ADBUS7 20 I2C.SCL
ACBUS0 21
ACBUS1 25
ACBUS2 26
ACBUS3 27
ACBUS4 28
ACBUS5 29
ACBUS6 30
ACBUS7 31
ACBUS8 32
ACBUS9 33
+3.3V
R25 10k
C35 10n
GND
FT232H_RESET
R26 12k
GND
R29 10k
GND
12MHZ_CLKD
U12
SN74CB3Q3253PWR
U11
SN74CB3Q3253PWR
FT232_FUNC_SELECT0
FT232_FUNC_SELECT1
FT232_FUNC_SELECT2
S0 14
S1 2
1A 7
-OE1 1
2A 9
-OE2 15
VCC 16
GND 8
1B1 6 UART.RTS
1B2 5
1B3 4 SPI.MISO
1B4 3
2B1 10 UART.CTS
2B2 11
2B3 12 SPI.HW_CS
2B4 13
UART{MCU_TX MCU_RX RTS CTS}
I2C{SCL SDA}
SPI{SCLK MOSI MISO SD_CS HW_CS}
+3.3V
R22 10k
FT232_FUNC_SELECT2
FT232_FUNC_SELECT[0..2]
FT232_FUNC_SELECT1
FT232_FUNC_SELECT0
R23 10k
R24 10k
GND

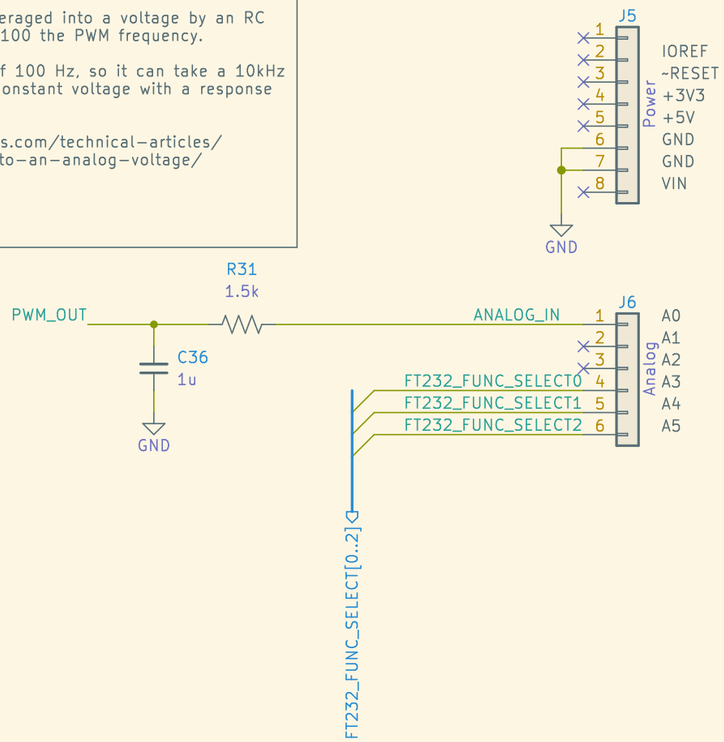

+3.3V
C40
100n
EN
Vdd
GND
OUT
X2
ASV-24.000MHZ-EJ-T
GND
+3.3V
C42
2.2uF
C43
100nF
GND
+3.3V
C44
100nF
GND
U1
CY7C68013A-56PVX
AVCC
VCC
LOGIC_IN[0..7]
PB0/FD0 LOGIC_IN0
PB1/FD1 LOGIC_IN1
PB2/FD2 LOGIC_IN2
PB3/FD3 LOGIC_IN3
PB4/FD4 LOGIC_IN4
PB5/FD5 LOGIC_IN5
PB6/FD6 LOGIC_IN6
PB7/FD7 LOGIC_IN7
XTALOUT
XTALIN
PD0/FD8
PD1/FD9
PD2/FD10
PD3/FD11
PD4/FD12
PD5/FD13
PD6/FD14
PD7/FD15
RDY0/SLRD
RDY1/SLWR
IFCLK
CLKOUT
CTL0/FLAGA
CTL1/FLAGB
CTL2/FLAGC
PA0/INT0
PA1/INT1
PA2/SLOE
PA3/WU2
PA4/FIOADDR0
PA5/FIOADDR1
PA6/PKTEND
PA7/FLAGD/SLCS
RESERVED
AGND
GND
Reset uses an RC timer with a 10ms time constant. The datasheet specifies at least 200us reset time when using an external CMOS clock generator.
+3.3V
R40
100k
R39
10k
FX2_RESET
RESET
WAKEUP
C45
100n
GND
USB{D+ D-}
USB.D+
USB.D-
D+
D-
+3.3V
C41
100n
GND
+3.3V
U16
24FC64-I/SN
R38
2.2k
R37
2.2k
A0
A1
A2
GND
VCC
SDA
SCL
WP
FX2_EE_SDA
FX2_EE_SCL
SDA
SCL
R42
10k
GND
GND
GND
8Kbyte EEPROM acts as code storage for the FX2LP. At bootup, it loads code, along with USB VID/PID values, over this interface. On new boards, the EEPROM can be programmed via the FX2 chip itself using FOSS tools.
Per the FX2 datasheet, for an 8KiB EEPROM, the address pins should be set to 001.
The datasheet says to connect "RESERVED" to ground. Yes, really.
GND
D6
R41 820
GND
The Sigrok FX2LP firmware uses PA1 as its status LED. On = ready, blinking = acquiring samples.
The FX2LP can source up to 4mA on its GPIO pins. An 820 ohm resistor should get about 2mA of current through this LED.

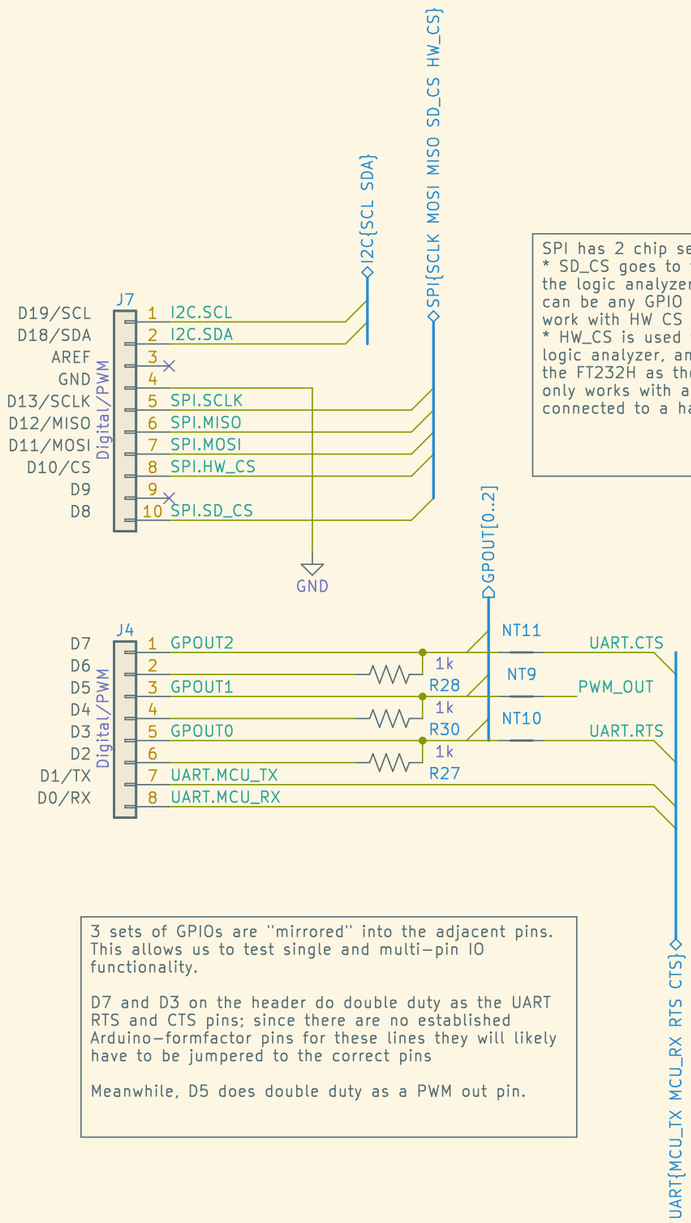
In order to generate analog voltages to test the ADC, we can use a PWM output from the MCU under test itself.

A PWM signal can be effectively averaged into a voltage by an RC filter with a cutoff frequency at 1/100 the PWM frequency.

This filter has a cutoff frequency of 100 Hz, so it can take a 10kHz PWM signal and average it into a constant voltage with a response time of roughly 10ms.

Source: https://www.allaboutcircuits.com/technical-articles/low-pass-filter-a-pwm-signal-into-an-analog-voltage/



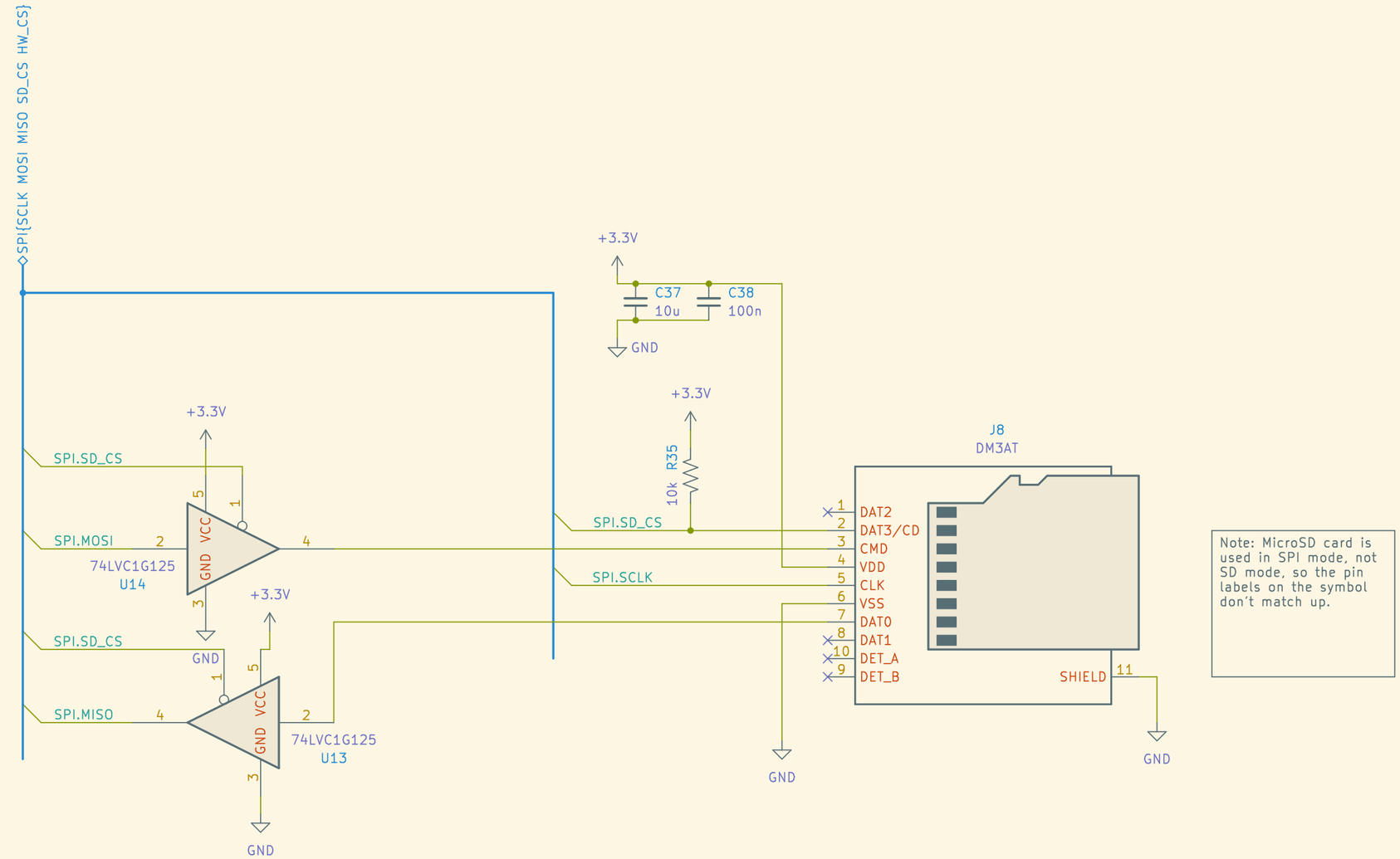


SPI has 2 chip selects:
* SD_CS goes to the SD card socket. It's monitored by the logic analyzer but not hooked to the FT232H. This can be any GPIO pin since the SD card driver does not work with HW CS anyway
* HW_CS is used for master-mode SPI operations to the logic analyzer, and also slave-mode SPI operations with the FT232H as the master. Since slave mode usually only works with a hardware CS pin, HW_CS should be connected to a hardware CS pin if available.

3 sets of GPIOs are "mirrored" into the adjacent pins. This allows us to test single and multi-pin IO functionality.

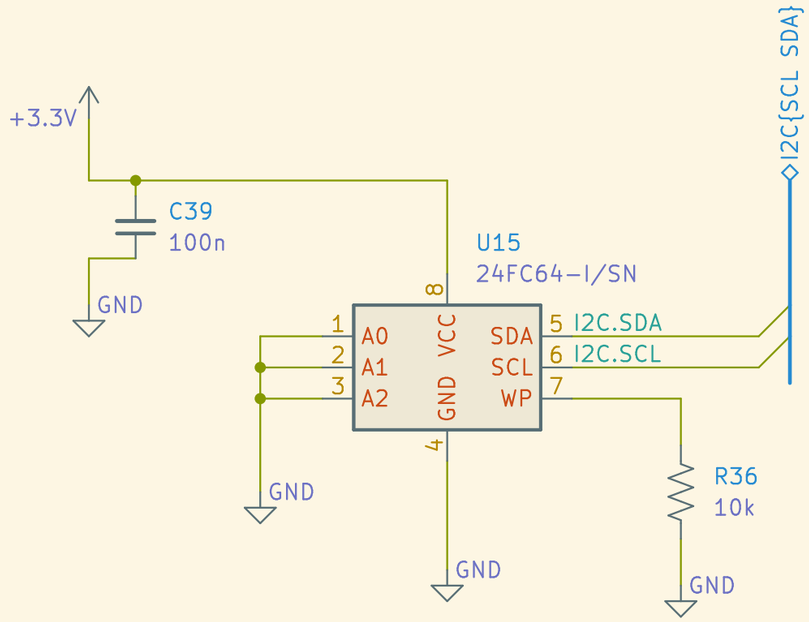
D7 and D3 on the header do double duty as the UART RTS and CTS pins; since there are no established Arduino-formfactor pins for these lines they will likely have to be jumpered to the correct pins

Meanwhile, D5 does double duty as a PWM out pin.



Note: MicroSD card is used in SPI mode, not SD mode, so the pin labels on the symbol don't match up.

When it is in SD mode (instead of SPI mode), the MicroSD card might try to output on its CMD and DAT0 lines in response to the SPI clock going active, which would interfere with other users of the SPI bus. We guard against this by sending the MOSI and MISO lines through a tri-state buffer, so the card is electrically disconnected from the bus when the CS line is high.



This I2C EEPROM provides something to test an MCU's I²C implementation against. There are huge numbers of I2C EEPROMs out there, but I chose this specific one because it supports all three bus speeds in common use: 100kHz, 400kHz, and 1MHz (Fast Mode Plus). Additionally, it's cheap, seems reasonably available, and worked on the previous revision of the board.

If this specific part becomes hard to find, there should be other pin-compatible options, as this form factor of EEPROM is sort of a de-facto standard