

# Altair-Duino Assembly & Operations



Copyright © 2017-2020 Chris Davis

Up-to-date instructions are always available at [www.altairduino.com/instructions](http://www.altairduino.com/instructions). **Be sure to check this page before starting construction for addendums.**

I would strongly suggest comparing the parts you received with the list below. Let me know if you are missing anything and I will send a replacement. (It will not be unusual to have a few extra minor parts, like resistors/LEDs/transistors.)

## PARTS LIST

- 1 x PC board
- 36 x 5mm red LED
- 36 x 12mm LED standoff
- 36 x NPN transistor
- 36 x 1k $\Omega$  resistor
- 36 x 10k $\Omega$  resistor
- 17 x Mini toggle on-off
- 8 x Mini toggle (on)-off-(on)
- 1 x 470 $\Omega$  resistor
- 1 x 47 $\mu$ F capacitor
- 2 x 1k $\Omega$  resistor
- 2 x 0.1 $\mu$ F capacitor
- 1 x Dual pin header
- 1 x Expansion PC Board
- 1 x 16 pin DIP socket
- 1 x 28 pin DIP socket
- 1 x VGA connector
- 1 x DB9 connector
- 1 x 3.5mm audio jack
- 1 x Micro SD module
- 1 x DC power jack
- 1 x USB-A connector
- 2 x 4.7k $\Omega$  resistors
- 1 x 220 $\Omega$  resistor
- 2 x Single pin header
- 1 x 6 Pin female header
- 1 x USB cable
- 4 x 10mm nylon bolts
- 1 x USB panel jack/extension
- 2 x 8mm steel screws
- 1 x 9v power supply
- 1 x Pre-programmed Arduino Due
- 1 x Front panel
- 1 x "Altair 8800" metallic sticker
- 1 x Micro SD card
- 2 x 150 $\Omega$  resistor
- 1 x 100k $\Omega$  resistor
- 1 x 470 $\Omega$  resistor
- 2 x 27pF capacitors
- 7 x 100nF capacitors
- 3 x 10 $\mu$ F capacitors
- 1 x 5mm LED
- 1 x 8Mhz crystal
- 1 x MCP-1700 regulator
- 1 x MAX3232 IC
- 1 x PIC32 IC

- 1 x Dual pin header
- 1 x Single pin header
- 8 x Jumpers
- 4 x Rubber feet
- 4 x 15mm M-F standoffs
- 8 x 8mm M-F standoffs
- 8 x Nylon nuts
- 4 x 6mm nylon bolts
- 14 pin (2x7) ribbon cable
- 1 x Laser-cut rear panel
- 11 x 14mm steel screws
- 11 x square steel nuts
- 19 x 8mm steel screws
- 19 x Hex steel nuts
- 12 x acrylic pieces (blue & gray)

# OTHER PARTS YOU MAY NEED

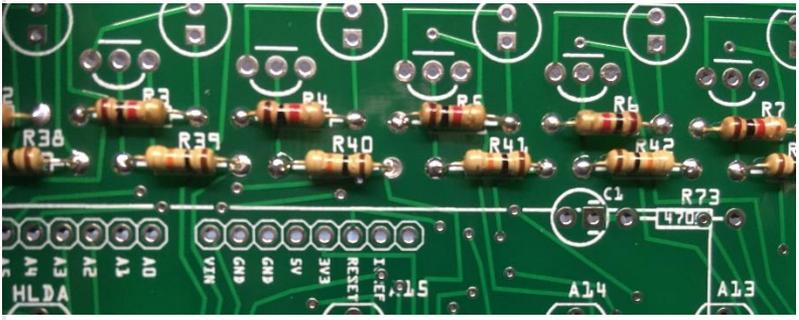
- Soldering Iron with a nice fine tip
- Good Solder (I recommend Alpha Fry Rosin Core 0.032")
- De-soldering Iron (optional)
- Phillips Screwdriver
- Needle-nose Pliers
- Side Cutters (Nippers)
- Computer

A word about soldering: Do not underestimate the need for good solder and a good soldering iron. Most problems I've seen people have with this kit are caused by cold joints or insufficient wetting. That does not necessarily mean you have to spend a lot of money. I have had good luck with \$8 soldering kits from eBay (however, I do throw away the solder that comes with those...) Just make sure it has an adjustable temperature and comes with an assortment of tips. Right now, I'm using a \$55 soldering station and it works great. I strongly advise you to get quality 60/40 Rosin core .032" diameter solder (I use Alpha Fry or Kester). The spools I buy are only \$10 and well worth it. I set my iron to 400 degrees and use the fine point tip.

A note about assembling acrylic pieces: Some people may want to glue the case together. This is fine, and will add some strength to the project, but I would suggest not using "super glue" or similar plastic cement. The fumes can adhere to the acrylic causing undesirable marks. I have had luck with "Loctite Plastics Bonding System" or other types of adhesives with activators. Assemble first to be sure everything fits, then disassemble and use adhesive.



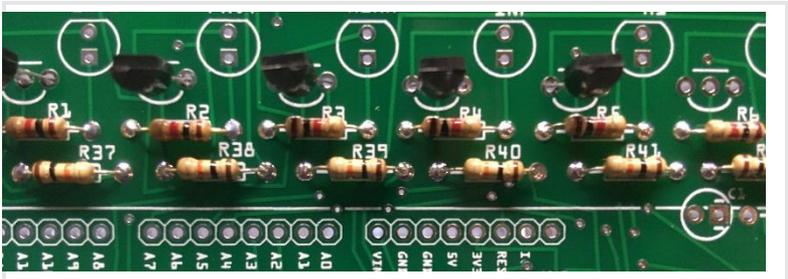
Add the 36-10k $\Omega$  resistors to the second row in R37-R72.



Next is the bag of 36 transistors.



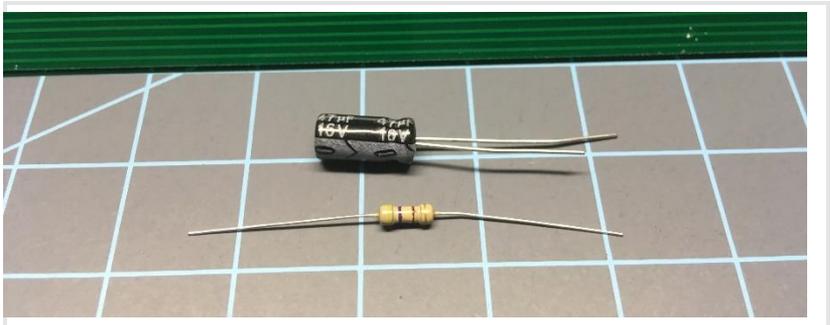
The orientation of the transistor is crucial, but relatively simple. Just make sure the flat end of the transistor is facing up, just like the image printed on the circuit board.



Grab the ziplock bag of assorted parts.



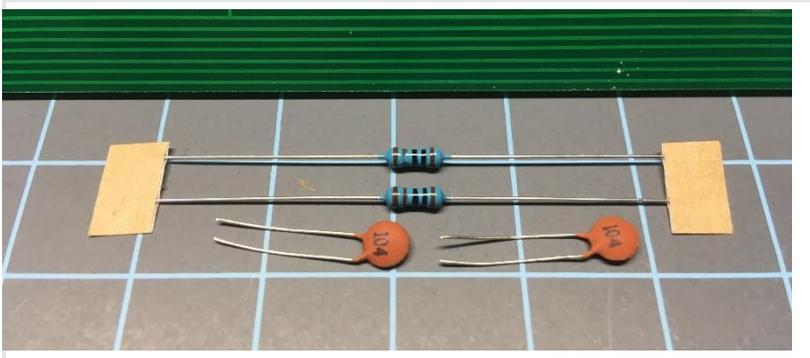
In that bag you will find one electrolytic capacitor and one  $470\Omega$  resistor (Yellow, Violet, Brown, Gold color code). If you don't want to decipher the color code, just know you will find three resistors in the bag, two that match, and one that does not. This is the one that does not match another.



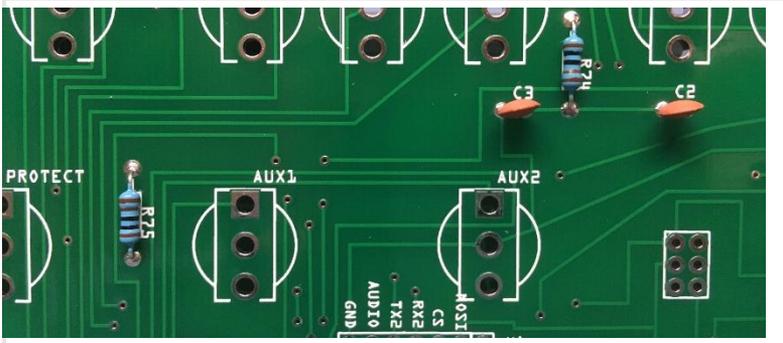
These two components mount on the PC board in R73 & C1. The orientation of the resistor does not matter, but make sure the short lead of the capacitor (marked with a "-" on the side) goes in the hole also marked with a "-".



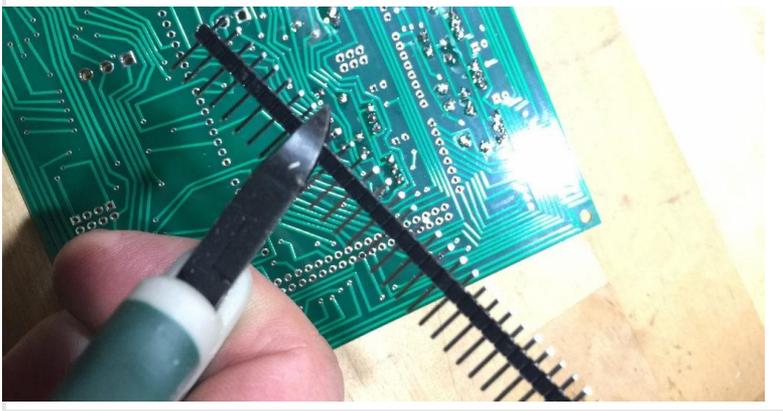
In the same bag, you will find two  $0.1\ \mu\text{F}$  capacitors (marked with "104") and two  $1\text{k}\Omega$  resistors (Brown, Black, Red, Gold).



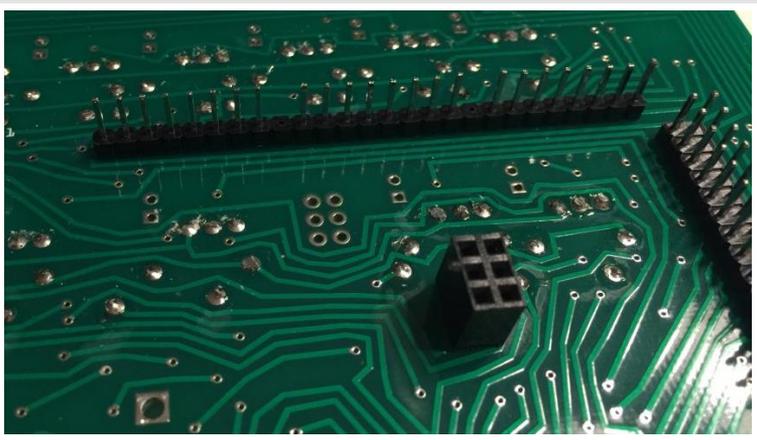
Those are mounted on the front of the PC board in R74, R75, C2, and C3 on the lower right side of the board. The orientation of the components does not matter.



Use your side cutters to clip the single pin headers into 5 segments of 8 pins, and 1 segment of 10 pins to mount the Arduino board. Also clip the double header to a 36-pin (2×18 pin) header.



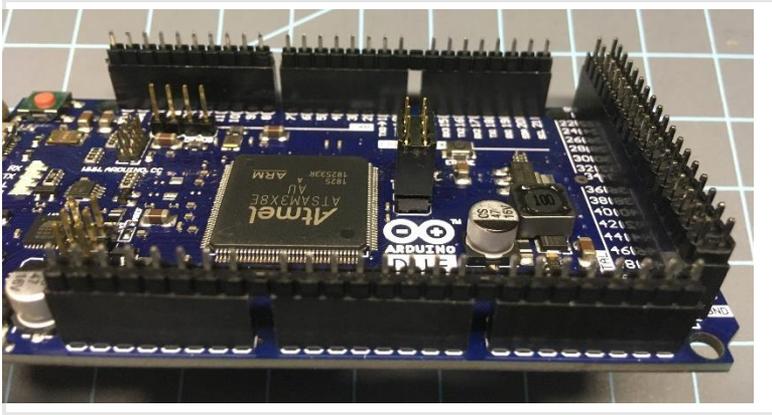
Add the male headers to the underside of the PC board. You can also add the six-pin female header for the SPI connector on the Arduino. Make sure you add these to the correct side of the board because desoldering 92 connections would not be fun!



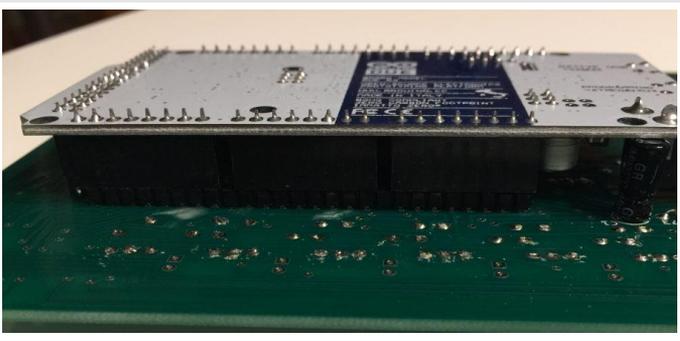
Solder the headers carefully. Make sure they are as close to vertical as possible, and make sure the solder flows

completely over the connection. **Most problems happen here with cold solder joints, or solder bridges.**

It may be helpful to install the headers in your Arduino Due, then insert the headers in the circuit board to hold them in the correct position while soldering.



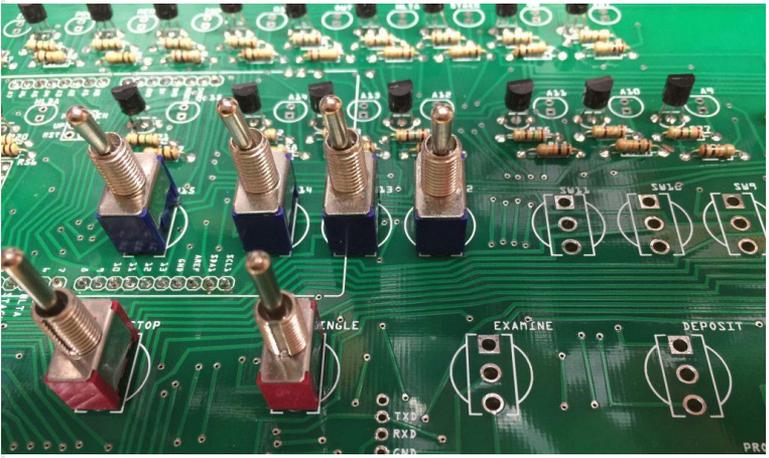
After you are done soldering the headers, try putting the Arduino in place and check for a secure fit. Remove it when you are done.



Get the bag containing 35 toggle switches. If your switches have nuts and washers, you may remove and discard them. Flip each switch back-and-forth a few times making sure it switches freely.



Put all the switches in place *without soldering them*. 17 two position toggles on the top row and the power switch location, 8 three position momentary toggles on the bottom row.

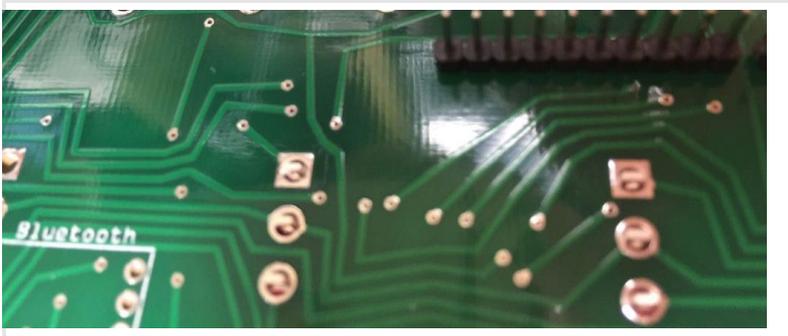


Put the front panel in place to hold the switches in the correct position (having all two-way switches in the down position makes this easier.)



Turn the board over and solder the switches.

HINT: Before you solder, make sure all three legs are protruding through the holes. Two isolated incidents have been reported where the leg was pushed up into the switch, causing a short which was very difficult to diagnose!

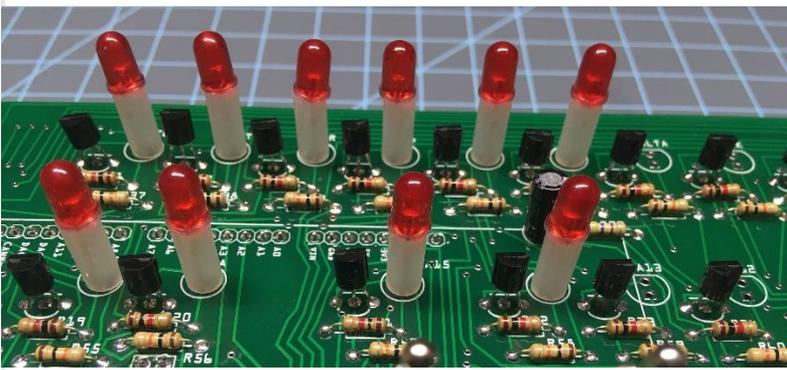


Next is the bag of LEDs and 12mm spacers.

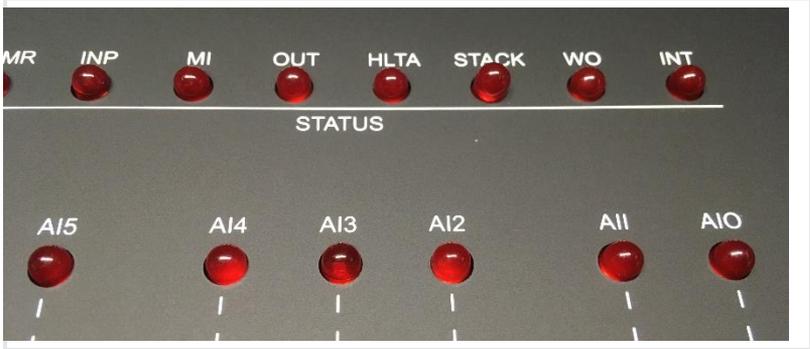


Just like the switches, put the LEDs and spacers in place and do not solder them. You do not have to do them all at once, it may be easier to do them in two or three groups.

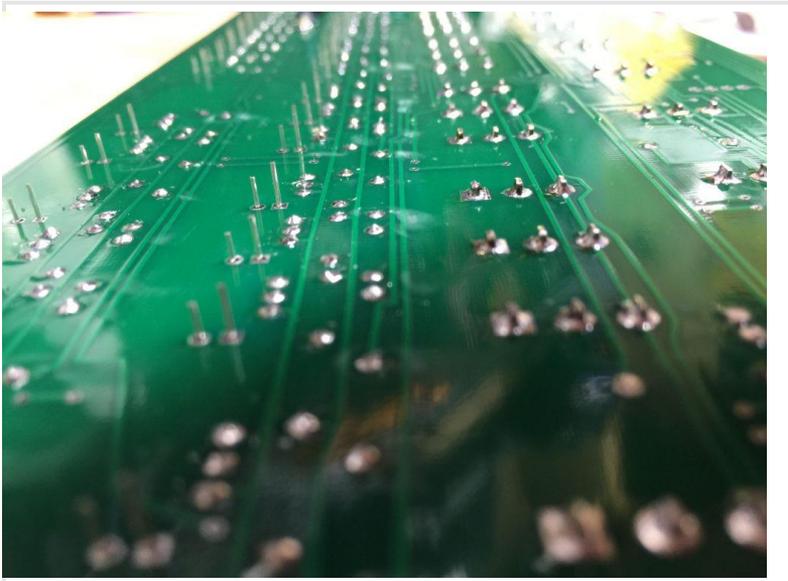
The orientation of the LEDs is crucial. Make sure the long lead of the LED is toward the bottom of the PCB and the flat side of the LED is toward the top.



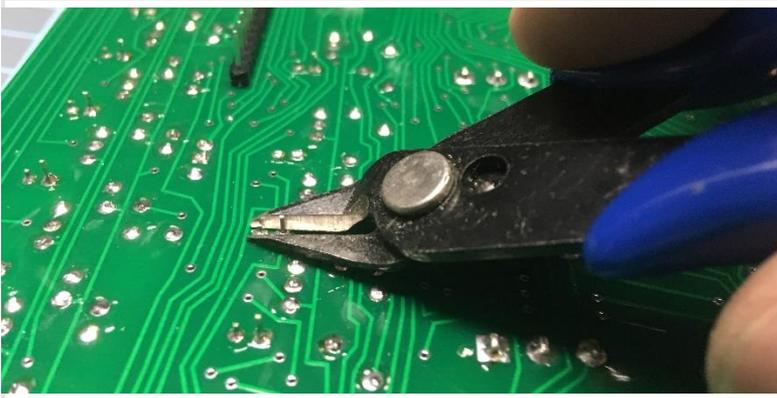
Again, like the switches, put the front panel in place to hold the LEDs while you solder them.



While you are soldering the LEDs, you can verify that the long lead is toward the bottom of the board.



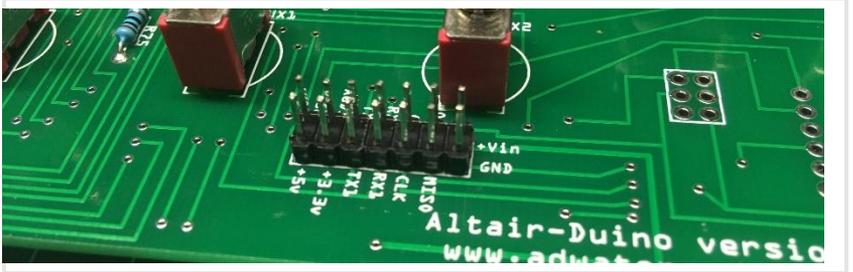
Make sure you trim the leads near the Arduino position after you solder the components. The protruding leads can interfere with the mounting of the Arduino.



Put the Arduino in place.



Get the main circuit board and solder a fourteen pin dual header on the lower right.



This is a good time to test your kit so far. Take the USB cable and plug it into the Programming port on the Arduino. Turn the board over (so you are looking at the face) and plug the other end of the USB cable into a computer or USB power supply. When the kit is powered, all LEDs will briefly flash, then go dark, and a second or two later, a random pattern of LEDs will light. My favorite quick-and-easy test is to set SW1 on and lower the AUX1 switch. This will run "Kill The Bit" and you will see LED15 to LED8 light in sequence.

Here's a good second test: set all address switches (two-way toggles) to ON and raise the EXAMINE toggle. All address LEDs (0-15) should light. If not, there is either a problem with that addresses LED or toggle switch.

In your kit, you will find a bag with smaller circuit board and bags of miscellaneous parts.



There are seven resistors to install with six different values. You will need to identify your resistors with a multimeter, or with the color codes (see [resistor-calculator.com](http://resistor-calculator.com) for a handy tool.) The locations for the resistors are clearly marked on the circuit board.

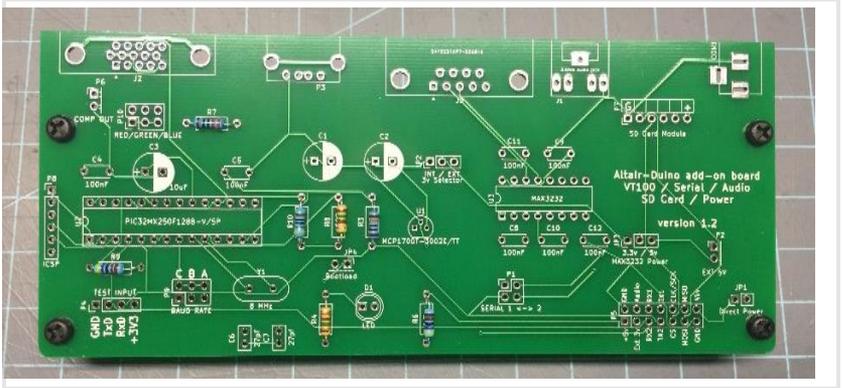
2 x 4.7k $\Omega$       (Yellow – Violet – Black – Brown)

1 x 470 $\Omega$       (Yellow – Violet – Black – Black)

2 x 150 $\Omega$       (Brown – Green – Black – Black)

1 x 220 $\Omega$       (Red – Red – Black – Black)

1 x 100k $\Omega$       (Brown – Black – Black – Orange)



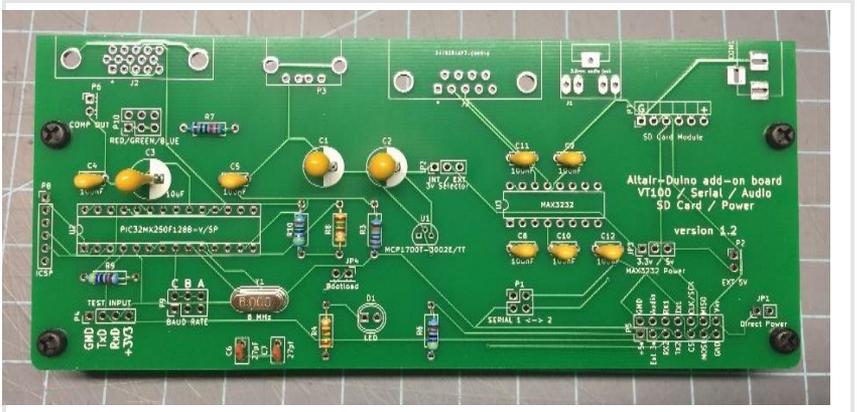
(If your board has a location for an 82Ω resistor, put a 150Ω resistor in its place.)

Add the capacitors and 8MHz crystal where marked.

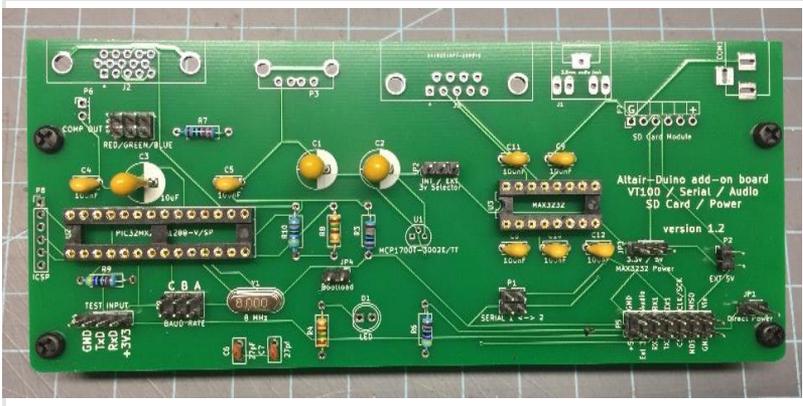
7 x 100nF (marked "104")

2 x 27pF (marked "27")

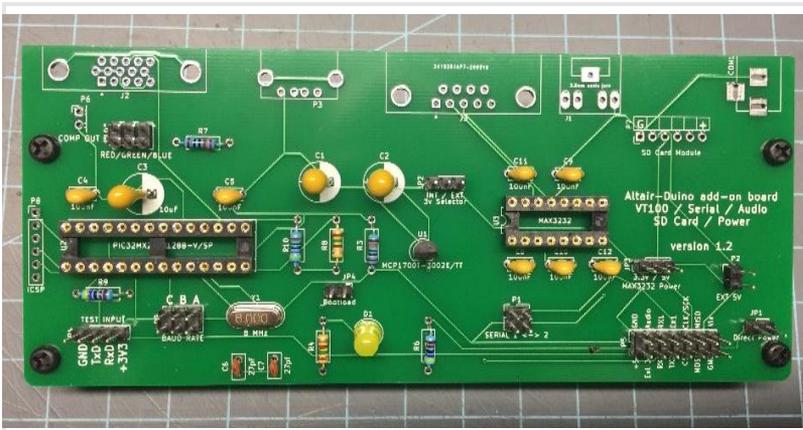
3 x 10uF (Important: insert long lead in + hole)



Next add the IC sockets (with the notch oriented to the left) and the single pin and double pin headers.



Add the LED \*with long lead to the right and flat side to the left) and the MCP1700 voltage regulator with the flat side down.



Add the jacks and connectors.



Finally, insert the MAX3232 IC and PIC32 IC with the notch facing left.



Add the required jumpers to the I/O expansion. The following are the "default" jumpers. Later you can read more about the jumper options on the website.

From upper left to lower right:

**Red/Green/Blue:** Your choice of text color on VGA monitor, you must jumper at least one.

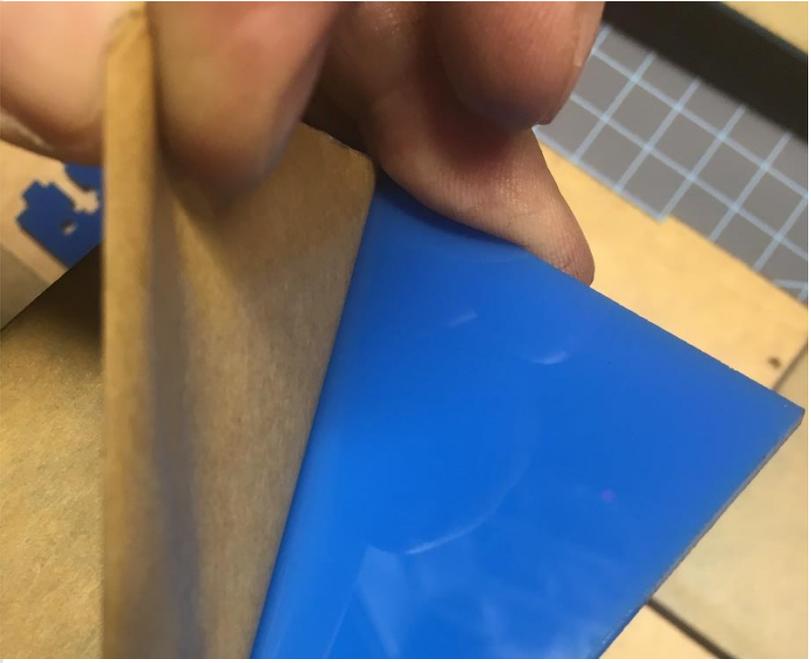
**3v Selector:** Set to EXT.

**Baud Rate:** Jumper C and B for 9600 baud.

**Bootload:** No jumper.

**MAX3232 Power:** Set to 3.3v.

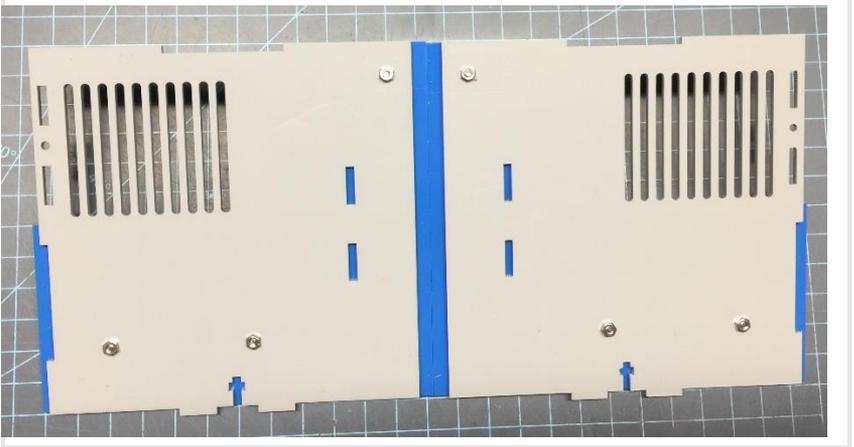
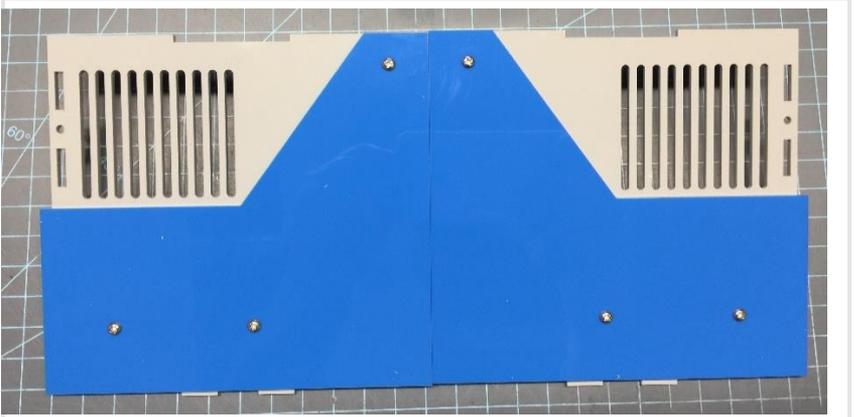
Get the acrylic pieces and start removing the masking paper.



Take the two smaller acrylic pieces with the word "Top" engraved. Add four 8mm nylon standoffs to the side that says "Top" and secure each with a nylon nut. (These pieces may be blue or gray.)



Next take the two gray side pieces (with "ventilation holes" and attach the blue side pieces with three 8mm bolts each.



Attach the blue top piece to the gray top piece as shown with 8mm bolts. Make sure the blue piece extends 6mm over each side (turn it over if it doesn't.)



Attach the two back pieces together with 8mm bolts. Note the position of the notch in the lower right.



Attach the I/O circuit board to the blue bottom piece with four 8mm standoffs, nylon nuts and nylon bolts.



Attach the gray piece (as shown) to the bottom with 8mm bolts.

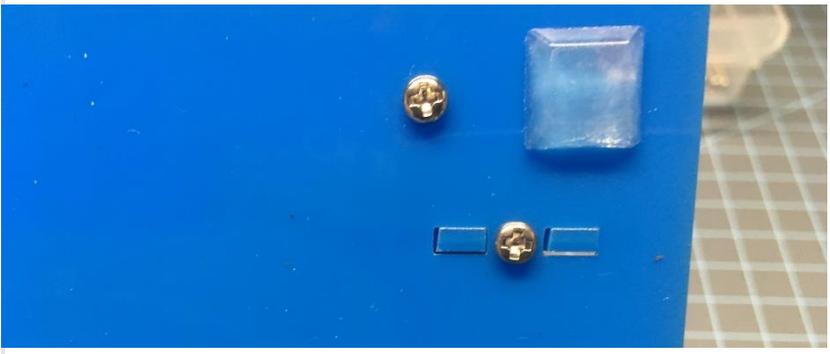


This would also be a good time to apply the rubber feet to the bottom piece.



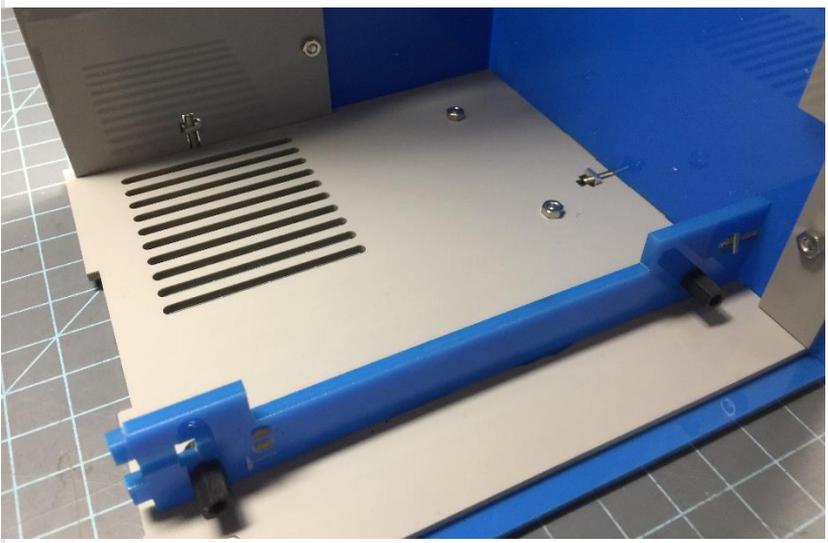
Next we're going to start joining acrylic pieces with a t-slot joint. Start with the uprights where we will eventually mount the main circuit board. Attach them to the bottom piece with 14mm bolts and square nuts. Tighten firmly, but not too tight – acrylic can crack.



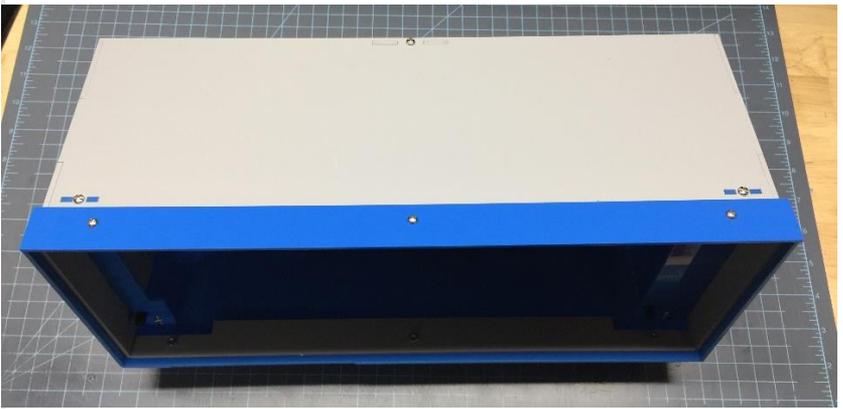


Add the back piece and side pieces and bolt in place with four 14mm bolts.





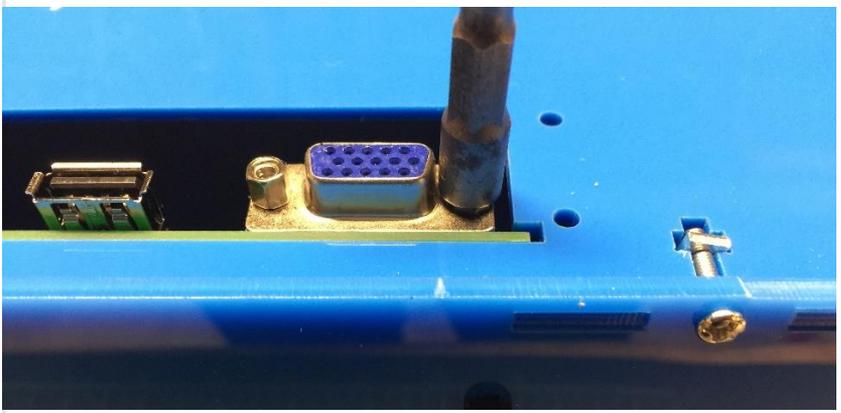
Add the top piece as shown and secure with three 14mm bolts.



Get the small rear panel (for the ports) and add the USB extension with two 8mm bolts.



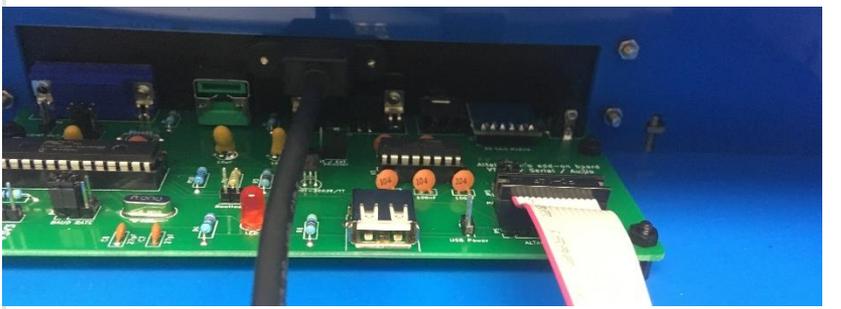
Remove the jack screws from the VGA and Serial connectors with a needle-nose pliers, or a 3/16" or 5mm socket.



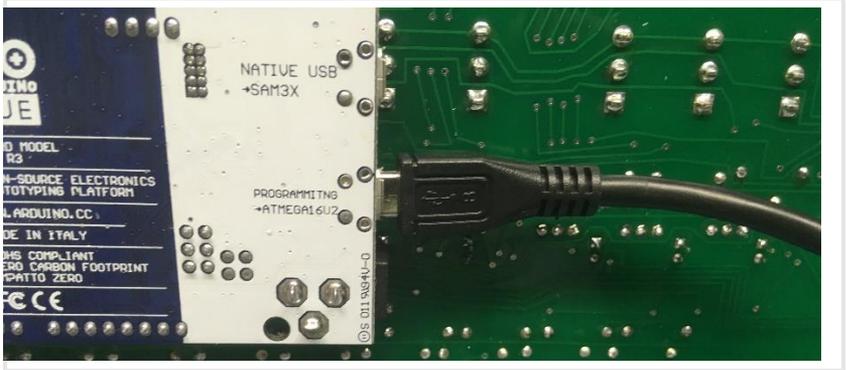
Place the rear panel in place and secure it with the jack screws and four 8mm bolts.



Connect the fourteen pin IDC cable to the I/O board, with the red stripe to the left.



Connect the USB connection cable to the Programming port on the Arduino Due.



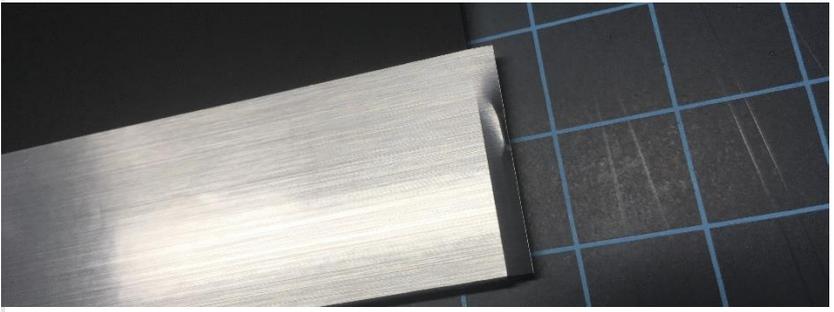
Attach the main circuit board to the case with 15mm M-F nylon standoffs.



Apply the Altair 8800 sticker to the front panel. The adhesive is forgiving, so if you place it wrong, you can pull it up and put it in place again. Use a small Phillips screwdriver or awl to poke holes where the bolts will go.

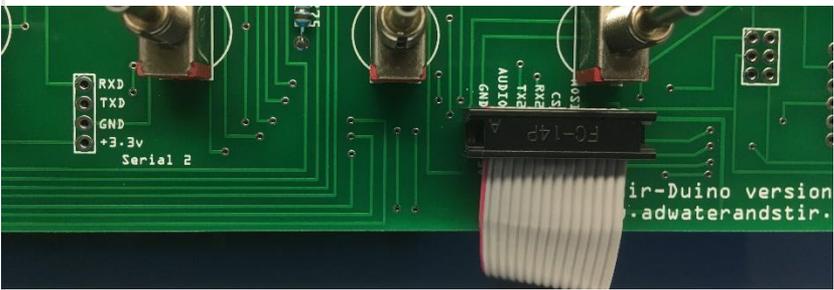


Your label will be a little bit longer than the front panel.



You may wrap the excess around the edge of the panel or trim it with a sharp razor.

Plug the fourteen pin IDC cable onto the front of the circuit board with the red stripe on the left.



Add the front panel to the main circuit board, push down around the toggle switches, and secure it with four nylon bolts.



# CONGRATULATIONS! YOUR ALTAIR 8800 IS COMPLETE!

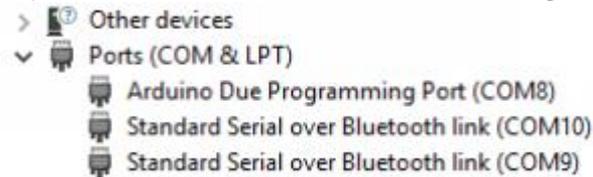
See the web page [www.adwaterandstir.com/operation](http://www.adwaterandstir.com/operation) for full documentation and easy step-by-step things to do.

Here are a few easy things to try:

**By default, your Altair-Duino is set up to communicate through the USB port.**

1. Plug USB cable into computer and the other end to your Altair-Duino.
2. Windows 10 should automatically recognize a new serial port. To check, launch "Device Manager".

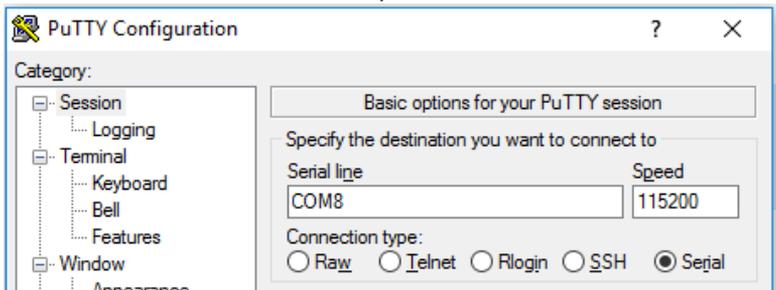
3. Expand "Ports (COM & LPT)" in Device Manager



4. Your port should be identified as "Arduino Due Programming Port".

5. Launch PuTTY (or another terminal program if you choose.)

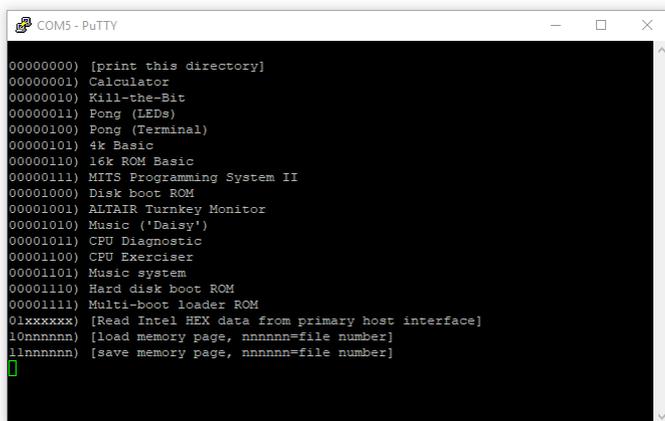
6. Connect to the indicated COM port at baud rate 115200.



7. The front panel lights will flash briefly while it connects.

8. With all switches down, press AUX1 down.

9. On the terminal, you should see a directory of options for front panel switches.



### **If you have a serial device (such as a dumb terminal):**

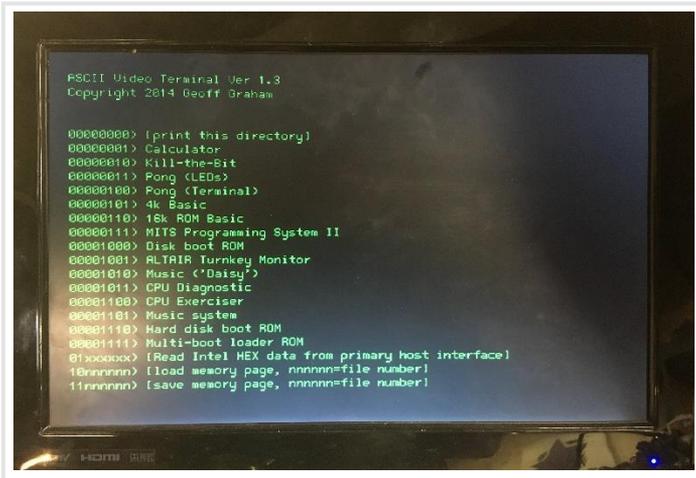
1. Plug a serial cable from the Altair-Duino to your serial device.
2. Connect a power supply to the Altair-Duino.
3. Set front panel data switches to "2" (switch 1 up, all other switches down).
4. Raise (and hold) DEPOSIT up.
5. Turn on Altair-Duino.

This will cause the Altair-Duino to load configuration 2 on power up. This configuration has been saved to communicate on serial port 2 at 9600 baud.

### **If you have Geoff Graham's VT-100 emulator on serial port 1 (using the I/O expansion board):**

1. Plug a VGA monitor and PS2 keyboard into the Altair-Duino. Power on the monitor.
2. Connect a power supply to the Altair-Duino.
3. **Make sure the SD card is inserted.**
4. Set front panel data switches to "1" (switch 0 up, all other switches down).
5. Raise (and hold) DEPOSIT up.
6. Turn on Altair-Duino.

You should see "ASCII Video Terminal Version 1.3 Copyright 2013 Geoff Graham" on the VGA monitor. With all switches down, press AUX1 down. On the monitor, you should see a directory of options for front panel switches.



Please see the website ([adwaterandstir.com](http://adwaterandstir.com)) for many other examples and walk-throughs for common functions. Also visit the online forum to discuss the Altair-Duino with other enthusiasts, or to ask questions ([adwaterandstir.com/forum](http://adwaterandstir.com/forum)).

### **Altair 8800 Simulator - Copyright (C) 2017 David Hansel**

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

**VT100 Terminal – Copyright (C) 2014 Geoff Graham**  
**All rights reserved.**

# Altair 8800 Simulator

© 2017-2019 David Hansel

<https://www.hackster.io/david-hansel/arduino-altair-8800-simulator-3594a6>

<https://github.com/dhansel/Altair8800>

## Table of Contents

Acknowledgements.....	- 2 -
Highlights .....	- 3 -
Front Panel Elements.....	- 4 -
Auxiliary Switch Functions .....	- 5 -
Serial ports .....	- 12 -
Interacting with software via a terminal.....	- 15 -
Printer Emulation .....	- 18 -
MTS Disk Controller Support.....	- 19 -
Tarbell Disk Controller Support.....	- 21 -
Hard Disk Support .....	- 23 -
Cromemco Dazzler Support .....	- 25 -

Processor Technology VDM-1 Support ..... - 26 -

Configuration Menu ..... - 28 -

File System Manager ..... - 38 -

Debugging Capabilities ..... - 38 -

Example program: Kill-the-bit game ..... - 39 -

Loading 4k BASIC the old-school way ..... - 41 -

Altair Time Sharing BASIC ..... - 43 -

Music System ..... - 46 -

MIT'S Programming System II ..... - 49 -

Interfacing external hardware via data/address buses ..... - 57 -

**Acknowledgements**

I must give many credits to Mike Douglas of altairclone.com who has spent countless hours collecting information about the Altair and made it all available on his web site for the community to use. Additionally, most of the software included in this simulator was collected, put into working condition and in many cases amended with additional functionality by Mike. Without his work, this project would have been significantly more complicated to put together, way less fun and would probably not happened at all.

Another big thanks to Martin Eberhard who has written many tools for the Altair community, for example the combo disk boot loader and hard disk loader which are used by the simulator. His ADEXER tool was invaluable for debugging the hard disk support. Thanks also for helping me sort through various issues with the hard disk emulation.

## Highlights

- Runs at about the same speed as the original
- Emulated memory size (RAM) is 64KB
- Can emulate I8080 and Z80 processor (Z80 emulation runs at reduced clock rate of about 2.6MHz)
- ALTAIR extended BASIC ROM (16k) can be mapped to addresses 0xC000-0xFFFF
- Emulates one ALTAIR 88-SIO, 88-ACR and one or two 88-2SIO board(s). Each device can be mapped to the Arduino's serial port(s). Data sent to each device can be captured and replayed.
- The ACR device also supports using the CSAVE/CLOAD commands in extended BASIC. When invoking CSAVE, the program will automatically be saved to a file specified by the file name in the CSAVE command. For CLOAD, all programs saved with CSAVE will automatically be played back until the program specified by the file name is found (or not).
- By default, both the 88-SIO and 88-2SIO boards are mapped to the Arduino's main serial interface which is configured to 115200 baud 8n1. That interface be accessed via pins 0/1 or the Arduino's USB cable (on the Due make sure to connect the USB cable to the programming USB port, not the native port).
- Processes input/output on serial devices to deal with 7/8 bit output, upper-case input and backspace.
- Emulates a MITS 88-DCDD disk controller with 4 (can be configured to up to 16) disk drives attached.
- Emulates a Tarbell SD disk controller with 4 disk drives attached.
- Emulates a MITS 88-HDSK hard disks drive with one unit (can be configured up to 4) and 4 platters.
- Emulates the 88-RTC-VI board which makes it possible to run ALTAIR Time Sharing BASIC.
- All settings can be modified via a configuration editor (invoke by holding STOP and raising AUX1)
- Above specs apply when running on an Arduino Due. The simulator can also work when running on an Arduino Mega 2560. In that case, it runs at about 25% original speed and has

6k emulated RAM. Disk drive emulation is not supported on the Mega.

## Front Panel Elements

### Lights:

A15-A0	Shows the current address bus state (i.e. PC during normal operation)
D7-D0	Shows the current data bus state

### Switches:

SW15-SW0	Address/Data entry switches
RUN	Execute instructions starting at current PC location
STOP	Stop program execution
SINGLE STEP	Execute single instruction at current PC location and increment PC
EXAMINE	Set PC to address from SW15-0 and show content of that address on D7-0
EXAMINE NEXT	Increment PC and show content of that address on D7-0
DEPOSIT	Store value of D7-0 at current PC address (If held down during power-up the simulator will load the configuration selected with the SW0-7 switches instead of configuration 0)
DEPOSIT NEXT	Increase PC by one and store value of D7-0 at that address
CLR	Stop serial capture/replay activated by AUX2 (see below) Holding CLR down during power-up will initialize memory and CPU registers to 0

RESET	Reset processor (set PC to 0) (If held up during power-up the simulator will start up with default settings)
PROTECT	Mark current memory page as write-protected (cannot be written to)
UNPROTECT	Remove write-protect status of current page
AUX1/AUX2	See detailed information below

## Auxiliary Switch Functions

### AUX1 down:

Function depends on current setting of SW7-0:

0...0000: Print this list to serial interface

0...0001: Calculator (David Hansel, 2015)

- Flipping SW15 (on->off or off->on) executes operation selected by SW14-13 between operand currently showing on A15-8 lights and operand selected by SW12-8 switches. Result is shown on A15-8 lights.
- Operations (SW14-13): 00=Add, 01=Subtract, 10=Multiply, 11=Divide

0...0010: Kill-the-bit (Dean McDaniel, 1975)

- Kill the rotating bit. If you miss the lit bit, another bit turns on leaving two bits to destroy.
- Quickly toggle the switch, don't leave the switch in the up position.
- Before starting, make sure all the switches are in the down position.

0...0011: Pong game using front panel (Mike Douglas of altairclone.com)

- Left player quickly toggles SW15 to hit the "ball."
- Right player toggles SW8. Score is kept in memory locations 0x80 and 0x81 (left and right).
- Score is missed balls, so the lower number wins.

0...0100: Pong game using serial terminal (David Hansel, 2015)

- Needs terminal connected to serial interface. Terminal must understand escape sequences for cursor movement (“ESC-[“)
- Left player uses A/Z keys (paddle up/down), right player uses K/M keys (paddle up/down)

#### 0...0101: 4k Basic (Bill Gates, Paul Allen, Monte Davidoff 1975)

- SW11 down: I/O to SIO device, SW11 up: I/O to 2SIO device (either setting will work in default simulator configuration)
- Answering N to questions about SIN/RND/SQR slightly increases available BASIC memory

#### 0...0110: MITS 16k ROM Basic

- On Arduino Due, maps ROM extended BASIC to addresses 0xC000-0xFFFF until next RESET. (On Arduino Mega, ROM extended BASIC is always mapped to that range).
- Sets PC to 0xC000 and starts execution (i.e. starts BASIC)
- Before activating, set SW15-12 switches to  
0000: I/O goes to 2SIO device  
0010: I/O goes to SIO device  
(with setting will work in default simulator configuration)
- When asked MEMORY SIZE, just press enter.
- When asked LINEPRINTER, enter capital O and press enter
- Available memory for BASIC programs is 48101 bytes (~3000 bytes on Arduino Mega)
- CSAVE and CLOAD commands can be used to save/load programs to/from internal storage

#### 0...0111: MITS Programming System II (Due only)

- SW11 down: I/O to SIO device, SW11 up: I/O to 2SIO device (either setting will work in default simulator configuration)
- Editor, Assembler and Debugger
- See intro at the end of this document, google for full documentation

0...1000: Combo Disk Boot Loader ROM V2.05 (Martin Eberhard, Mike Douglas, 2016) (Due only)

- See MITS Disk Controller Support section below

0...1001: ALTAIR Turnkey Monitor (Due only)

- Uses 2SIO device for I/O

0...1010: Music “Daisy, Daisy...” (Steve Dompier, 1975) (Due only)

- In configuration menu, set throttle delay to 5 before running (automatic throttle introduces noise due to the throttle-adjustment code that runs 40 times per second)
- An AM radio held on top of Simulator, close to D0-D7 lights will pick up the song
- See: <http://www.digibarn.com/collections/weirdstuff/altair-sheetmusic/>

0...1011: 8080 CPU Diagnostic (Microcosm Associates, 1980)

- Basic test, takes about a second to finish
- Outputs to serial interface (“CPU IS OPERATIONAL”)

0...1100: 8080 CPU Exerciser (Frank D. Cringle 1994, Ian Bartholomew 2009)

- Very thorough test that generates a CRC code for each group of tests.
- Outputs to serial interface.
- Full test takes about 4 hours. The "aluop <b,c,d,e,h,l,m,a>" section takes especially long

0...1101: Music System (Processor Technology, 1977) (Due only)

- Processor Technology’s Music System for the Altair
- Loads the music system (at 0x0) and ACUTER monitor (at 0xF000) into Altair RAM and starts ACUTER monitor.
- See “Music System” section below for more info

0...1110: Hard-Disk boot ROM V2.0 (Martin Eberhard, 2014) (Due only)

- See Hard Disk Support section below

0...1111: Enhanced Multi boot loader V3.0 (Martin Eberhard, Mike Douglas, 2016) (Due only)

- Set SW10-8 to select boot device:  
000=2SIO port 1, 010=SIO, 011=ACR, 110=2SIO port 2

0...10000: Tarbell disk boot loader (Due only)

- Runs at FF00h (unlike the original, which ran at 0h)

0100000: Read Intel HEX data from serial input (primary host interface)

11xxxxxx: Save the 256-byte memory page currently selected on the SW15-8 switches to file #xxxxxx

10xxxxxx: Load the 256-byte memory page currently selected on the SW15-8 switches from file #xxxxxx

### **AUX1 up:**

If STOP is held up while AUX1 is raised, then invoke the configuration editor (see Configuration menu below).

Otherwise, run the program configured via the corresponding setting in the configuration menu.

### **AUX2 down:**

If SW14-12 are set to 001 then mount disk in MITS disk drive (see *MITS Disk Controller Support* section below).

If SW14-12 are set to 101 then mount disk in Tarbell disk drive (see *Tarbell Disk Controller* section below).

If SW14-12 are set to 011 then mount image in hard disk (see *Hard Disk Support* section below).

Otherwise, play back captured data or example programs. SW15-13 select the device

to which data is played back and SW8-0 select what data is being played back:

- SW15 down: Use the serial device which is mapped to the host serial output (primary) and has last seen input/output activity. In most cases this will automatically select the intended device.
- SW15 up: Use serial device selected on SW14-13:
  - 00: 88-SIO (port 0x00/0x01)
  - 01: 88-ACR (port 0x06/0x07) (audio cassette interface)
  - 10: 88-2SIO, serial 1 (port 0x10/0x11)
  - 11: 88-2SIO, serial 2 (port 0x12/0x13)

- SW8 down: Play back basic/assembly example # selected on switches SW7-0 (see below)
- SW8 up: Play back captured data in file # selected on switches SW7-0 (see below)

Loading BASIC/assembly examples (assembly examples **not** included if Z80 support is enabled):

- SW7 down: BASIC example, SW7 up assembly example
- If SW6-0 are all 0 then a list of available examples will be transmitted
- Set SW6-0 to the example number in the list and push AUX2 DOWN to transmit example
- Playback can be stopped by pushing CLR or by pushing AUX2 DOWN again

Playing back captured data:

- Play back data previously captured via AUX2 up
- The file number to be played back must be selected via SW7-0
- Playback can be stopped by pushing CLR or by pushing AUX2 DOWN again

## AUX2 up:

If SW14-12 are set to 001 then unmount disk from disk drive (see *MTS Disk Controller Support* section below).

If SW14-12 are set to 101 then unmount disk from Tarbell disk drive (see *Tarbell Disk Controller* section below).

If SW14-12 are set to 011 then unmount image from hard disk (see *Hard Disk Support* section below).

Otherwise, capture data. SW15-13 select the device from which data is captured (same as SW15-13 settings in *AUX2 down* section above).

SW7-0 specify the file number under which the captured data will be saved.

Capturing continues until AUX2 is again pressed UP.

Capturing serial data can be used to save a BASIC program in BASIC (note that in extended BASIC the CSAVE/CLOAD commands provide an easier way of doing this):

1. Set SW7-0 to the desired storage file number
2. Set SW15-13 to 000 (to automatically select capture device)
3. type "list" (but **not** ENTER)
4. activate AUX2 up to start capture
5. press ENTER
6. wait until listing is finished
7. activate AUX2 up again to finish capture

To later restore a program:

1. Set SW7-0 to a storage file number under which serial data has been captured before
2. Set SW15-13 to 000 (to automatically select replay device)
3. type "new" to clear the current program
4. activate "AUX2 down" to start replaying the captured data
5. Ignore the "SYNTAX ERROR" at the end (reported because the final "ok" of the "list" function was also captured)

Capturing cassette data can be used to save BASIC variable contents in Extended BASIC or saving programs from MITS Programming System II

1. Set SW7-0 to the desired storage file number
2. Set SW15-13 to 101 (to capture from ACR device on I/O address 6/7)
3. activate AUX2 up to start capture
4. enter CSAVE\*v [where v is the variable name that is supposed to be saved]
5. Repeat the previous step if more variables need to be saved
6. wait until listing is finished
7. activate AUX2 up again to finish capture

To later load the data:

1. Set SW7-0 to a storage file number under which cassette data has been captured before
2. Set SW15-13 to 101 (to replay to ACR device on I/O address 6/7)
3. Set SW8 to 1 (to select file replay, not BASIC example)
4. activate AUX2 down to start replaying the captured cassette data
5. enter CLOAD\*v [where v is the variable name that is to be loaded]
6. Repeat the previous step if more variables need to be loaded
7. Activate AUX2 down to stop replay

## Serial ports

The Emulator simulates an Altair 8800 with one 88-SIO, one 88-ACR and one or two 88-2SIO card(s) installed. These cards are visible to the software running within the emulator (for example ALTAIR BASIC). All together there are up to seven emulated serial ports available to the software.

The Arduino, which runs the emulator, has its own serial ports that actual terminals can be connected to. The configuration editor allows the user to map emulated serial ports to real serial ports.

While the Arduino Mega has 4 serial ports, only one of them is available to use with the emulator because the pins that the others would use are connected to front panel elements such as LEDs or switches. The Mega's single serial port can be accessed either by plugging in the USB cable or by connecting a serial device to pins 0 (RX) and 1 (TX). Note that these pins use 5V levels, not the +12V/-12V used by real serial hardware. However, the levels can be boosted to proper serial levels using an adapter such as a MAX232.

When using the Arduino Due, there are up to five serial ports available for the emulator:

1. *Programming port USB connection.* After connecting a USB cable to the Programming Port of the Arduino it will show up on a PC as a serial interface. This is the default interface that the emulator uses (at 115200 baud, 8N1) when first activated or reset to its defaults.

This connection can also be accessed on pins 0 (RX) and 1 (TX) of the Arduino Due **BUT** there appears to exist a problem with (some versions of) the Arduino Due where pin 0 (RX) does not work. For more information about that issue see here:

<https://forum.arduino.cc/index.php?topic=474483.0>

The pins are 3.3V and are **not** 5V tolerant. This port can be configured from 600 to 1050000 baud, except 1200 baud since opening a USB serial connection at 1200 baud causes the Due to erase its flash memory and go into programming mode. Baud rates of 300 and below are not available because the USB-to-serial chip on the Due does not support them:

<https://github.com/arduino/Arduino/issues/4714>

2. *Native port USB connection.* The second USB port on the Arduino Due can also be used as a serial connection by plugging in a USB cable. This port does **not** have corresponding I/O pins and can only be accessed via USB. Since it is a pure USB connection the baud rate is irrelevant and is fixed at 115200.
3. *Serial port on pins 18 (TX) and 19 (RX).* The pins are 3.3V and are **not** 5V tolerant. This port can be configured from 110 to 1050000 baud.
4. *Serial port on pins A6 (RX) and A7 (TX).* This port is disabled in the default configuration since the same pins (A6/A7) are used for the PROTECT/UNPROTECT switch on the front panel. However, the Altair's PROTECT functionality is rarely (if ever) used and therefore the pins can be re-purposed to act as a serial port by changing two `#define` statements in the code (see below).

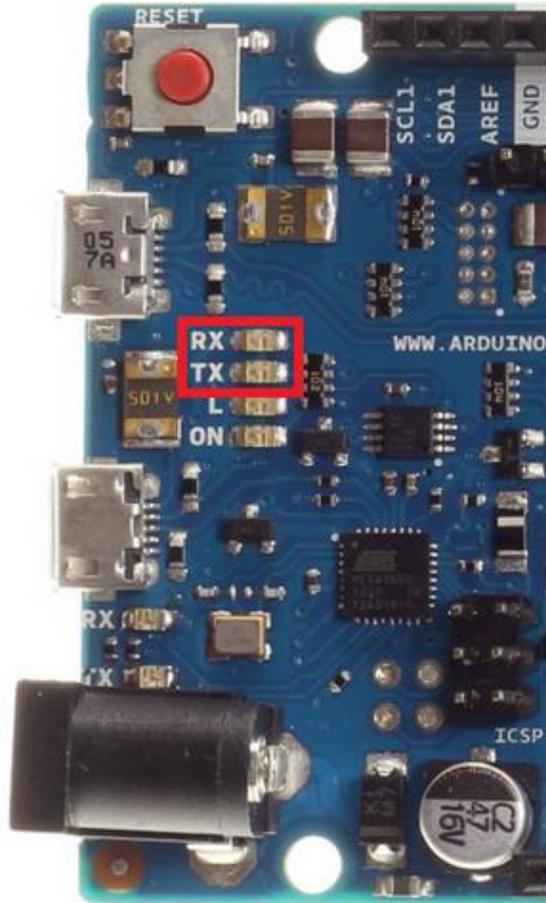
**IMPORTANT:** If pins A6/A7 of your Arduino are already wired to the PROTECT/UNPROTECT switch I highly recommend to physically disable the switch (for example by removing the connection of the middle pin ground) before enabling this port in the software. If the serial port is enabled in the software and the switch is connected, pressing either PROTECT or UNPROTECT can destroy pin A7 on the Arduino or possibly a connected serial device. This is because serial lines idle high (3.3V) and the switch, when pressed, connects the pin directly to ground. The best solution is to connect the middle pin of the PROTECT/UNPROTECT switch to ground via a 1k resistor (instead of directly). That way the switch can be used if enabled in software but does not cause damage when pressed while the serial port is enabled.

To enable this port in the emulator, change the following in `host_due.h`:

```
"#define USE_SERIAL_ON_A6A7 0" to "#define  
USE_SERIAL_ON_A6A7 1"
```

The pins are 3.3V and are **not** 5V tolerant. This port uses a software UART implementation which can be configured from 110 to 38400 baud.

5. *Serial port on pins RXL / TXL.* This port is disabled by default since it uses two I/O pins that are not connected to headers on the Arduino Due: the pins controlling the RX and TX LEDs next to the Native USB port (framed in red in the image on the right). These pins are accessible as digital pins 72 (RX) and 73 (TX) on the Arduino Due. Despite their location next to the Native USB port they have no connection to the port and can be freely controlled by software. To add a serial interface using these pins do the following:



- (Optional but recommended): remove the RX/TX LEDs. I did try leaving the LEDs in place and just soldered onto the LEDs themselves and serial communication did work. However, the LEDs go to +3.3V through a 1k resistor which could possibly interfere with the serial signals. Your mileage may vary.
- Solder wires to the pads on the left side of the LEDs (the side closer to the “RX” and “TX” labels). These will be the RX and TX wires for the serial connection. Take the required GND wire for the connection from any GND connection on the Arduino.
- In file `host_due.h`, change `#define USE_SERIAL_ON_RXLTXL 0` to

```
#define USE_SERIAL_ON_RXLTXL 1
```

and upload the sketch to the Arduino.

The pins are 3.3V and are **not** 5V tolerant. This port uses a software UART implementation which can be configured from 110 to 38400 baud.

## Interacting with software via a terminal

The main way to interact with programs on the Altair is via a terminal connected to a serial interface. The two most common serial interface cards used with the Altair were the 88-SIO and 88-2SIO. The 88-SIO offered one serial port which (although configurable via jumpers) was most often set to use I/O addresses 0 and 1. The 88-2SIO offered two serial ports (again configurable via jumper) that usually used addresses 16/17 (first port) and 18/19 (second port).

Most systems had at least one of these installed and most programs would expect to interact via either an 88-SIO at addresses 0/1 or the first port of an 88-2SIO at addresses 16/17. Some programs used the SW15-8 sense switches at startup to determine which one to use. For example:

*4k BASIC and Programming System II:* If SW11 is up then use an 88-2SIO at 16/17 otherwise use 88-SIO at 0/1.

*16k ROM BASIC:* If SW13 is up then use 88-SIO at 0/1 otherwise use 88-2SIO at 16/17.

The simulator simulates a system with both an 88-SIO and 88-2SIO installed at their default addresses. In the default configuration, **both** the 88-SIO and the first port of the 88-2SIO are mapped to the Arduino's main serial port, i.e. any output that goes to either one will show up on a terminal connected to the Arduino and inputs coming from the terminal will be sent to both cards. This was not possible in reality (it would mean to hook up one terminal to two serial ports) but in the simulated environment it works just fine. It allows users to not have to worry about properly setting sense switches before starting BASIC or other programs.

The one drawback is that when changing device settings in the configuration menu, you must first know which device is being used. The easiest way to determine that is to just un-map one of them in the configuration and see if the serial I/O still works.

**7-bit vs 8-bit characters.** Early terminals used only 7-bit characters and a number of Altair programs (such as 4k BASIC) use the 8<sup>th</sup> bit of a character to define end-of-string which would work fine since the terminal would ignore the 8<sup>th</sup> bit anyways. Modern terminals use the 8<sup>th</sup> bit and so display some strange graphics character for characters where the 8<sup>th</sup> bit is set. The simulator (in the serial device settings) offers a way to filter out the 8<sup>th</sup> bit by enabling the “Use 7 bits” setting.

**Uppercase input.** Early terminals only had upper case characters and so early Altair programs (4k BASIC again) cannot handle lower-case characters. The simulator offers a serial device setting (“Serial input uppercase”) that will automatically convert incoming lower-case characters to upper case.

**Backspace handling.** Some early Altair software does not use the backspace (0x08) character and instead has its own way of undoing inputs, e.g. in 4k BASIC receiving an underscore (“\_”) will print the underscore but internally delete the last character from the input buffer, i.e. have the functionality of a backspace. Other programs expect a “delete” (0x7f) character instead of “backspace” (0x08). The simulator offers some help by offering an option to translate between backspace and underscore or backspace and delete.

When converting back from underscore to backspace, the simulator actually sends a backspace-space-backspace sequence to delete the character left of the cursor even if the connected terminal does not do a destructive backspace.

**Pasting text into the terminal.** When pasting large amounts of text into the terminal, the terminal program usually sends that text at the given baud rate. If the connection is using 9600 baud, then a new character will arrive at the simulator about every millisecond. With many simulated programs (and especially when running the simulator on the Arduino Mega) this can cause characters to get lost because the

simulated program cannot keep up processing the characters at the rate that they arrive.

There are two ways to deal with this:

1. Lower the baud rate of the host serial interface and your terminal program, which forces the terminal to send characters at a lower speed. This works to some degree but (a) you may not be able to reduce the baud rate enough to guarantee that no characters are lost (especially when using the Mega) and (b) reducing the baud rate will also lower the transmission speed from the simulator to your terminal, which can become annoying when outputting large amounts of text.
2. The better option is to tell your terminal to add a delay between sending characters. TeraTerm allows this by changing the “Transmit delay” settings (In the Setup->Serial port menu). The terminal will still send at the given (fast) baud rate (and more importantly *receive* at the fast baud rate) but will wait for a specified amount of time before sending the next character. Moreover, TeraTerm allows to specify a delay after a carriage return/line feed. This is important because some programs (such as BASIC) need additional time after seeing a carriage return to process the input.

Some testing has shown that for the Arduino Mega, setting the Transmit delay to 10msec/char and 200msec/line works well even with the host serial rate set at 115200 baud. On the Due, 3msec/char and 25msec/line should be sufficient. These are just some basic benchmarks. The optimum setting (not too much delay but also not too little) will also depend on the baud rate.

Unfortunately it appears that Putty does not support a transmit delay.

## Printer Emulation

One printer can be emulated by enabling the corresponding options in the configuration menu (see Configuratoin Menu section below). The printer emulation can be configured for an “Okidata” (88-LP) printer or “Centronics C700”. Most of the included software (e.g. BASIC) support both (enter “O” or “C” at the LINEPRINTER prompt).

In addition to the Okidata and Centronics printers, a “generic” printer option is available. The generic printer does not perform any processing of input data for formatting or linefeeds/carriage returns. It forwards all data directly to the configured serial port. The generic printer option also allows to specify values for the printer control port (port address 02h) if the printer (i.e. serial port) is busy and not busy. The default values support using a printer in CP/M.

Note that in prior versions of the firmware, the “Okidata” printer emulation included some workarounds to allow that printer to be used in CP/M (although flow control via the status register did not work properly). With the addition of the “generic” printer which works perfectly in CP/M those have been removed, i.e. the Okidata printer no longer works with the CP/M version included on the disks. Use the “generic” printer instead.

Printer emulation does support interrupts. If interrupts are enabled by the software then the printer emulation automatically switches to real-time mode, i.e. it roughly simulates the time it would take the printer to print. This is necessary because otherwise (in an interrupt driven software system such as Time-Sharing BASIC) the printer would rapidly produce interrupts, disrupting access to other devices. Real-time operation can be forced even if interrupts are not enabled by enabling the “Force real-time mode” option in the configuration menu.

The printer output can be directed to either one of the host’s serial ports.

## MITS Disk Controller Support

Disk drive support is optional and requires a SD card attached to the Arduino Due's SPI header (the 2-row, 6-pin header marked "SPI"). See the *Wiring SD card to Arduino DUE* section at the end of this document about how to physically hook up the card.

Disk drive support is **not** available in the Arduino Mega build, mainly for two reasons: 1. The SPI pins on the Mega are directly connected to some general I/O pins which are already used for the front panel and 2. The Mega only provides 6k of emulated RAM. Most disk based programs require more than that.

Disk drive support emulates a MITS 88-DCDD disk drive controller with up to 16 drives attached.

The following files are expected to be found in the root directory of the (FAT format) card (the `disks` subdirectory in the source archive contains several disks including CP/M and Altair DOS):

- **DISKxx.DSK** (where xx is a 2-digit hexadecimal number): Disk images that the simulator can mount.
- **DISKDIR.TXT**: A text file whose contents will be sent to the serial connection (i.e. shown to the user) if front panel switches are set to 0001xxxx00000000 and the AUX2 switch is pressed down. This should contain information about each of the DISKxx.DSK files

To mount disks in the drive, use the AUX2 down switch:

- Set SW15-0 to: 0001nnnnDDDDDDDD where nnnn is a 4-bit number selecting the drive (i.e. drive 0-15) and DDDDDDDD is an 8-bit number selecting the disk number and press AUX2 down.
- The 8-bit disk number corresponds to the xx in the DISKxx.DSK files on the SD card.
- For example, setting SW15-0 to 0001 0010 0000101 and pressing AUX2 down will mount disk number 5 in drive 2.

- Selecting disk number 0 is a special case. If disk 0 is selected for mounting, it will not be mounted but instead the contents of the DISKDIR.TXT file will be sent to the serial output (i.e. shown to the user).  
Note that that means a file named DISK00.DSK can not be mounted.
- If a disk is already mounted in the drive the mounted disk will be unmounted before mounting the new disk
- If the disk file does not exist, it is like inserting an empty disk in the drive. If the operating system writes to the disk, the selected disk file will be created. So inserting a non-existent disk and then formatting that disk via the operating system (e.g. CP/M) will create a new empty disk.
- It is possible to mount the same disk in multiple drives. The simulator has no problem with that but the running operating system may get confused.

To unmount a disk from a drive, use the AUX2 up switch:

- Set SW15-0 to: 0001nnnnxxxxxxxx where nnnn is a 4-bit number selecting the drive (i.e. drive 0-15) and press AUX2 up.
- It is not necessary to unmount disks before turning off the computer. Each write operation to a disk gets flushed to the SD card immediately so turning the computer off with disks mounted will not lose data.

To run a bootable disk image, first mount the disk and then start the Disk Boot ROM:

- Set SW0-7 to 00001000 (to select Disk Boot ROM)
- Press AUX1 down
- This will install the Disk Boot ROM at 0xFF00 and immediately start it. If a bootable disk has been mounted it should automatically start now

## Tarbell Disk Controller Support

In addition to the MITS 88-DCDD disk controller, the simulator can also emulate the Tarbell single density FDC with up to 4 drives attached to the controller.

Tarbell disk controller support is optional and requires a SD card attached to the Arduino Due's SPI header (the 2-row, 6-pin header marked "SPI"). See the *Wiring SD card to Arduino DUE* section at the end of this document about how to physically hook up the card.

To enable Tarbell disk controller support in the firmware, set the "#define NUM\_TDRIVES n" setting in config.h to a value of n between 1 and 4.

The following files are expected to be found in the root directory of the (FAT format) card (the `disks` subdirectory in the source archive contains two different versions of CP/M):

- **TDISKxx.DSK** (where xx is a 2-digit hexadecimal number): Disk images that the simulator can mount.
- **TDISKDIR.TXT**: A text file whose contents will be sent to the serial connection (i.e. shown to the user) if front panel switches are set to 0101xxxx00000000 and the AUX2 switch is pressed down. This should contain information about each of the DISKxx.DSK files

To mount disks in the drive, use the AUX2 down switch:

- Set SW15-0 to: 0101xxnnDDDDDDDD where nn is a 2-bit number selecting the drive (i.e. drive 0-3) and DDDDDDDD is an 8-bit number selecting the disk number and press AUX2 down.
- The 8-bit disk number corresponds to the xx in the TDISKxx.DSK files on the SD card.
- For example, setting SW15-0 to 0101 0010 0000001 and pressing AUX2 down will mount Tarbell disk number 1 in drive 2.
- Selecting disk number 0 is a special case. If disk 0 is selected for mounting, it will not be mounted but instead the contents of

the TDISKDIR.TXT file will be sent to the serial output (i.e. shown to the user).

Note that that means a file named TDISK00.DSK cannot be mounted.

- If a disk is already mounted in the drive the mounted disk will be unmounted before mounting the new disk
- If the disk file does not exist, it is like inserting an empty disk in the drive. If the operating system writes to the disk, the selected disk file will be created. So inserting a non-existent disk and then formatting that disk via the operating system (e.g. CP/M) will create a new empty disk.
- It is possible to mount the same disk in multiple drives. The simulator has no problem with that but the running operating system may get confused.

To unmount a disk from a Tarbell drive, use the AUX2 up switch:

- Set SW15-0 to: 0101nnnnxxxxxxxx where nnnn is a 4-bit number selecting the drive (i.e. drive 0-15) and press AUX2 up.
- It is not necessary to unmount disks before turning off the computer. Each write operation to a disk gets flushed to the SD card immediately so turning the computer off with disks mounted will not lose data.

To run a bootable disk image, first mount the disk and then start the Tarbell Disk Boot ROM:

- Set SW0-7 to 00010000 (to select Tarbell Disk Boot ROM)
- Press AUX1 down
- This will install the Tarbell Disk Boot ROM at 0xFF00 and immediately start it. If a bootable disk has been mounted it should automatically start now

## Hard Disk Support

Just like disk drive support, hard disk (88-HDSK) support requires a SD card and is not available on the Arduino Mega build. See the first two paragraphs of the “MITS Disk Controller Support” section above for more details.

The 88-HDSK controller could handle up to 4 units, each holding up to 4 platters. In its default configuration, only one unit is enabled, which should be sufficient for most cases. Change the NUM\_HDSK\_UNITS setting in the config.h file to enable more (at most 4) units.

The following files are expected to be found in the root directory of the (FAT format) card (the `disks` subdirectory in the source archive contains some hard disk images including the Altair Accounting System):

- **HDSKxx.DSK** (where `xx` is a 2-digit hexadecimal number): Disk images that the simulator can mount.
- **HDSKDIR.TXT**: A text file whose contents will be sent to the serial connection (i.e. shown to the user) if front panel switches are set to `0011xxxx00000000` and the AUX2 switch is pressed down. This should contain information about each of the HDSKxx.DSK files

To mount hard disk images on a unit/platter, use the AUX2 down switch:

- Set SW15-0 to: `0011uuppDDDDDDDD` where `uu` is the hard disk unit (2-bit, 0-3) and `pp` is the platter number within that unit (2-bit, 0-3). `DDDDDDDD` is an 8-bit number selecting the hard disk image number. Then press AUX2 down.
- The 8-bit disk image number corresponds to the `xx` in the HDSKxx.DSK files on the SD card.
- For example, setting SW15-0 to `0011 0001 0001100` and pressing AUX2 down will mount HDSK0C.DSK on platter 1 of unit 0.
- Selecting hard disk image number 0 is a special case. If disk image 0 is selected for mounting, it will not be mounted but instead the contents of the HDSKDIR.TXT file will be sent to the

serial output (i.e. shown to the user).

Note that that means a file named HDSK00.DSK can not be mounted.

- If a hard disk image is already mounted in the drive the mounted disk will be unmounted before mounting the new disk
- If the hard disk image file does not exist, it is like inserting a new platter in the hard disk. If the operating system writes to the hard disk, the selected image file will be created. Inserting a non-existent image and then formatting via the operating system (e.g. using ADEXER in CP/M) will create a new empty image.
- It is possible to mount the same image in multiple drives. The simulator has no problem with that but the running operating system may get confused.

To unmount an image from a unit/platter, use the AUX2 up switch:

- Set SW15-0 to: 0011uuppxxxxxxx where *uu* is the hard disk unit (2-bit, 0-3) and *pp* is the platter number within that unit (2-bit, 0-3). Then press AUX2 up.
- It is not necessary to unmount images before turning off the computer. Each write operation to an image gets flushed to the SD card immediately so turning the computer off with disks mounted will not lose data.

To run a bootable hard disk image, first mount the image and then start the Disk Boot ROM:

- Set SW0-7 to 00001110 (to select Hard Disk Boot ROM)
- Press AUX1 down
- This will install the Hard Disk Boot ROM at 0xFC00 and immediately start it. If a bootable image has been mounted it should automatically start now. The Hard Disk Boot ROM always boots from unit 0, platter 0.

## Cromemco Dazzler Support

In 1976 the Cromemco Dazzler graphics card for the Altair was released. The card accesses the Altair's memory via DMA to read pixel data and produces a picture of 32x32, 64x64 or 128x128 pixels, depending on memory configuration and color mode. While these resolutions seem almost laughable today, back in 1976 this was a major accomplishment.

The Simulator (when running on the Arduino Due) can emulate the Dazzler when paired with a client to display the picture. For more information on the Dazzler client, see my project at:

<https://www.hackster.io/david-hansel/dazzler-display-for-altair-simulator-3feb6c>

To enable support for the Dazzler in the Simulator, do the following:

- Download the latest release of the Simulator firmware
- Change the USE\_DAZZLER setting in config.h from 0 to 1
- Re-upload the modified firmware to the Arduino
- In the configuration menu, there should now be a "Configure Dazzler" setting.  
Change that setting to "USB Native Port".

There are a number of Dazzler example programs provided on DISK10. To use them, mount DISK10 in drive 0 and start the disk boot loader. This will boot CP/M. You can now start the different programs on the disk from within CP/M. Note that none of these programs has any exit mechanism, so to run a different program you will have to stop and reset the Altair and boot into CP/M again.

- Kaleidoscope (also available via AUX1 down)
- DazzleDoodle (*very* basic drawing program operated by joystick)
- DazzleMation (basic animation program, needs Cromemco serial card, see below)
- DazzleWriter (allows to type text, needs Cromemco serial card, see below)
- Life (Conway's game of life, expects input on first port of 2-SIO)
- Labyrinth (needs joystick)

- Chase (needs two joysticks)
- Space War! (needs two joysticks)
- Tank War! (needs z80 processor and two joysticks)
- Ambush (needs Z80 processor and one joystick)
- Dogfight (needs Z80 processor and two joysticks)
- Gotcha! (needs two joysticks)
- 4-d Tic-Tac-Toe
- Chess (Microchess using the Dazzler as a graphics display)

For programs requiring a joystick, see the joystick instructions in the Dazzler project. Note that the Windows client recognizes joysticks plugged into the computer running the client.

For programs that need a Cromemco serial card, make sure the “SIO revision” setting in the SIO card configuration is set to “Cromemco”.

For programs requiring a Z80 processor, make sure the simulator is properly configured to emulate a Z80 processor.

## **Processor Technology VDM-1 Support**

The VDM-1 card, created in 1975 by Processor Technology, was the first video card for the Altair. It could be directly connected to a monitor (or a modified TV) and produced a display of 16 lines with 64 characters each.

The screen contents could be directly modified by writing to a 1k region of main memory which allowed for much faster screen compared to a serial terminal.

To emulate the VDM-1, the simulator must be paired with a client to display the picture. The client can either be a Windows computer or one of Geoff Graham’s ASCII Video Terminals updated with a modified firmware. For more information see: <https://github.com/dhansel/VDM1>

To enable support for the VDM-1 in the Simulator, do the following:

- Download the latest release of the Simulator firmware
- Change the USE\_VDM1 setting in config.h from 0 to 1
- Re-upload the modified firmware to the Arduino

- In the configuration menu, there should now be a “Configure VDM-1” sub-menu.
- Within the VDM-1 sub-menu, change the “Map to interface” setting to the interface that the VDM-1 client is connected to. For more information on how to connect a VDM-1 client, see <https://github.com/dhansel/VDM1/blob/master/README.md>

The VDM-1 card had several jumpers that could be used to change certain settings. Within the VDM-1 sub-menu of the simulator’s configuration menu you will find options that mirror those jumpers:

- *Memory address*  
The base address of the RAM that the VDM-1 uses as its screen memory can be configured in 1k steps. Note that most programs assume the screen memory is at CC00 (the default).
- *DIP switch 1+2*  
Switches between regular display, inverted display and display off.
- *DIP switch 3+4*  
Changes display behavior of cursor characters (characters with ASCII code >127).
- *DIP switch 4+5*  
Controls blanking behavior for CR (0x0d) and VT (0x0B) characters.

For more information on the specifics of the DIP switch settings please consult the original VDM-1 manual:

<https://github.com/dhansel/VDM1/raw/master/doc/vdm1.pdf> (mostly section 3 - theory of operation).

The VDM-1 GIT repository contains some programs that use the VDM-1: <https://github.com/dhansel/VDM1/tree/master/programs>

Alternatively, the included DISK11 contains those programs (and some more). DISK11 contains a specialized CP/M that outputs to the VDM-1. To use this disk, do the following:

- In the VDM-1 configuration menu, set “Map keyboard to” to 2SIO port 1.

- Run the “Cuter for VDM-1” program by setting switches SW7-0 to 00010000 and pressing AUX1 down
- Press STOP (not actually using CUTER, it just needs to be in RAM for CP/M to use its output routines)
- Mount DISK11 by setting SW12, SW4, SW1 up (all others down) and pressing AUX2 down
- Run the disk boot ROM by setting SW7-0 to 00001000 and pressing AUX1 down

This should show a CP/M boot prompt on the VDM-1 and accept input from the keyboard connected to the VDM-1.

## Configuration Menu

The simulator configuration menu can be entered by holding STOP up and raising AUX1. Settings are:

- *Enable profiling*  
If enabled and the simulator is running (i.e. not in STOP mode) a message showing the current performance of the simulator is written to the (primary) serial output.
- *Set throttle delay (Arduino Due only)*  
Allows to throttle simulator performance. Throttling is done by busy-waiting, i.e. an empty loop counting down from the value selected here to zero. If set to “auto adjust”, the simulator evaluates performance about 40x a second and automatically adjusts the throttle delay to get as close to 100% of the original (i.e. 2MHz clock frequency) as possible. If profiling is enabled too then the effect of changes in the throttle delay can easily be observed.
- *Enable serial panel*  
Shows a simple representation of the front panel lights and switches on the (primary) serial output. Mostly useful when STANDALONE mode is enabled.

- *Enable serial input*  
When stopped, the simulator processes the inputs described in the “Debugging capabilities” section below.
- *Enable serial debug*  
Print processor status and disassembled opcode during single stepping
- *Processor* (only if processor switching is enabled in config.h)  
Switch between Intel 8080 and Zilog Z80 processor
- *Configure memory*  
See “Memory configuration” section below.
- *Aux1 shortcut program*  
Sets the program to be run when the AUX1 switch is raised. This can be any of the built-in programs available via AUX1 down or a disk. If a disk is selected, pushing up AUX1 will mount the disk and then install and run the disk boot loader.
- *Configure host serial*  
See “Host serial interface configuration” section below.
- *Configure serial cards*  
See “Serial device configuration” section below.
- *Configure printer*  
See “Printer configuration” section below.
- *Configure Disk Drives* (only if disk drive emulation is enabled in config.h)  
See “Disk Drive configuration” section below.
- *Configure Hard Disks* (only if hard disk emulation is enabled in config.h)  
See “Hard Disk configuration” section below.
- *Configure Dazzler* (only if Dazzler emulation is enabled in config.h)  
See “Cromemco Dazzler support” section.
- *Configure VDM-1* (only if VDM-1 emulation is enabled in config.h)  
See “VDM-1 support” section.
- *Configure Interrupts*  
See “Interrupts configuration” section below.

- *Manage Filesystem*  
Starts file system manager (see File System Manager section below).
- *File System manager for SD card* (only if SD card is attached)  
Starts a file system manager to modify the contents of the SD card. The file manager also allows transferring files from and to the card via XMODEM protocol.
- *Save configuration*  
Saves the current configuration. Up to 256 different configurations can be saved.  
Configuration #0 is automatically loaded when the simulator starts.
- *Load configuration*  
Loads a saved configuration
- *Reset to defaults*  
Resets all settings to their default values.

### **Memory configuration**

- *RAM size*  
Configures the size of the emulated RAM in steps of 1k. The emulated RAM starts at address 0 and is contiguous from there.
- *Clear memory on powerup*  
If enabled, the emulated RAM memory will be cleared (set to 0) when the simulator starts up.  
Otherwise, memory content is random (as with the original).
- *Clear memory now*  
Clear the emulated RAM memory (set to all 0)
- *Add ROM*  
Adds a new ROM into the emulator. The ROM content must be sent in Intel HEX format. A name can be specified to make it easier to identify the ROM later.
- *Remove ROM*  
Remove a previously added ROM
- *Auto-start ROM*  
Choose a ROM to auto-start (or enter 0 to disable auto-start).

When the Altair is turned on or the RESET button is pressed, the program counter (PC) will be set to the first address of the selected ROM. Additionally, if auto-start is enabled then the Altair will automatically enter RUN mode on startup.

### **Host serial interface configuration**

- *Host Serial ... settings*  
For serial ports that support configuration settings other than baud rate, this opens a sub-menu to configure baud rate, number of bits, parity and number of stop bits. Otherwise it toggles the baud rate used for the port (those ports always use 8N1 configuration). Note that different ports may have different ranges of possible baud rates. See “Serial ports” section above.
- *Primary host serial* (Arduino Due only)  
Selects which serial interface is used as the primary interface. All simulator related output (such as the configuration menu) is sent to the primary serial interface. Also, when auto-detecting the serial device for capturing/replaying data, only devices mapped to the primary serial interface are considered.
- *Apply host serial settings*  
When making changes to the host serial settings (baud rate, primary interface), those are not applied immediately. Select this option to apply the modified settings.

### **Interrupt configuration**

The simulator can emulate a 88-RTC-VI board which provides a real-time clock and vector interrupt capability (both are necessary to run Altair Timesharing Basic). The following settings can be configured:

- *Real Time Clock*  
Can be either disabled or set to produce an interrupt at one of

the following frequencies:

0.06, 0.6, 6, 10, 60, 100, 1000 or 10000Hz

Note that (despite the name) the frequency is based on simulated time, not real time.

- *Vector Interrupt Board*

If set to “Interrupts connected directly to CPU” then the Vector Interrupt functionality is disabled.

In that case, the device interrupt settings below can only be changed between “connected” or “not connected”. Note that the simulator (unlike the original) allows to connect multiple devices to the CPU’s interrupt line.

If this is set to “Use Vector Interrupt Board” then each device interrupt can be assigned a level/priority according to the 88-RTC-VI board’s functionality. Consult the 88-RTC-VI board’s user manual for more information about the interrupt levels.

- *[Device] interrupt*

This configures the connection of the interrupt line for each of the listed devices. If the VI board is disabled, then an interrupt line can either be connected or not connected to the CPU. If the VI board is enabled, then the interrupt can be connected to a specific level on the VI board.

### **Disk drive configuration**

If MITS disk controller support is enabled, this menu allows to modify drive related settings:

- *Force real-time mode*

If a running program that interacts with the disk drive does not enable interrupts for the drive then the drive emulation works in a rapid mode in which new data is presented to the program every time it checks if new data is available. This makes for very fast disk emulation. If interrupts are enabled for the drive then it operates in real-time mode, only producing interrupts when new data would be available on a real drive.

Enabling the “Force real-time mode” option will always operate the drive in the slower real-time mode, making for a more realistic LED blinking pattern while interacting with the disk drive.

- *Drive n mounted disk image*  
Shows which disk image (DISKxx.DSK) is currently mounted in which drive and cycles through the available images.

### **Tarbell disk drive configuration**

If Tarbell disk controller support is enabled, this menu allows to modify drive related settings:

- *Drive n mounted disk image*  
Shows which disk image (TDISKxx.DSK) is currently mounted in which drive and cycles through the available images.

Note that Tarbell disk emulation currently does not support interrupts and does not have a real-time mode.

### **Hard disk configuration**

If hard disk support is enabled, this menu allows to modify drive related settings:

- *Force real-time mode*  
If a running program that interacts with the hard disk does not enable interrupts for the drive then the drive emulation works in a rapid mode in which new data is presented to the program every time it checks if new data is available. This makes for very fast hard disk emulation. If interrupts are enabled for the hard disk then it operates in real-time mode, only producing interrupts when new data would be available on a real drive. Enabling the “Force real-time mode” option will always operate

the drive in the slower real-time mode, making for a more realistic LED blinking pattern while interacting with the hard disk.

- *Hard disk [unit n] platter m image*  
Shows which image (HDSKxx.DSK) is currently mounted in which unit/platter and cycles through the available images. The [unit n] is only shown if the simulator is set up to emulate more than one unit  
(NUM\_HDSK\_UNITS setting in config.h).
- *Reset hard disk controller*  
Resets the controller. Mainly this sets the CRDY flag. Do this if emulated software instructs you to turn the hard disk unit off and back on.

### **Printer configuration**

To configure printer options, select “(5) Configure printer” in the main configuration menu. The following configuration options are available for printer emulation:

- *Printer type*  
Select which printer should be emulated or turn off printer emulation:  
*Okidata* – emulates an Okidata (88-LPC) printer at I/O ports 2/3  
*C700* – emulates a Centronics C700 printer at I/O ports 2/3  
*Generic* – generic printer at I/O ports 2/3. This forwards all data sent to the printer directly to the configured serial port.
- *Map printer to interface*  
Selects the host’s serial interface to which the emulated printer’s output should be directed
- *Force real-time mode*  
If enabled, the printer emulation will enforce timing similar to a real printer (although not necessarily matching the actual emulated printer model). Provides a printer-like effect when watching the output.
- *Status register busy value* (only for Generic printer type)  
Value that the printer status register (I/O port 2) should show

when the printer (i.e. the serial port) is busy and cannot accept more data.

- *Status register ready value* (only for Generic printer type)  
Value that the printer status register (I/O port 2) should show when the printer (i.e. the serial port) is ready to accept more data.
- **Serial device configuration**

The simulator emulates up to six serial devices connected to the Altair:

- 88-SIO card at port 0x00/0x01
- 88-ACR audio cassette interface at port 0x06/0x07
- 88-2SIO, card with serial 1 at port 0x10/0x11 and serial 2 at port 0x12/0x13
- A second emulated 88-2SIO card can be enabled by changing the following setting before uploading the sketch to the Arduino. In file `config.h` change:  

```
"#define USE_SECOND_2SIO 0" to "#define USE_SECOND_2SIO 1"
```

  
The second 88-2SIO has serial 1 at port 0x14/0x15 and serial 2 at port 0x16/0x17

For each of these devices the following settings can be configured in the configuration menu:

- *Map to host interface*  
Select to which host interface input/output of device gets directed.  
When using Arduino Mega, there is only one host interface (Serial). When using the Due, there is the primary and secondary interface. The physical interface (Serial/Serial1) that the primary interface maps to can be picked on the main setup screen. The other one becomes the secondary interface.
- *Simulated baud rate*  
If receive interrupts are disabled for a serial device and "Force baud rate" is off, the simulator just makes a new byte available for playback whenever the running program requests one. That way, playback runs as maximum speed without the program

missing any characters. If receive interrupts are enabled however, the characters must be sent at a rate that gives the running program a chance to keep up. Similarly, if transmit interrupts are enabled, the program expects some time to pass between sending characters.

This setting selects the baud rate at which input/output is processed. Note that the timing is based on simulated time, not real time. That means that on the Mega (since it is running at 25% original speed) a rate of 110 baud may actually look more like 25 baud in real time.

- *Force baud rate*

If this option is on then the simulator will always use the given baud rate for receive and transmit, even if interrupts are disabled. This can be used to get more realistic timing for code that is not using interrupts.

- *Example playback NULs*

When playing back data to the running program (e.g. the AM2 assembler), the program may need some extra processing time after a carriage return to process the previous line. This is done (as it would have been on the original) by sending a number of NUL (0) bytes after a carriage return. Note that this setting only affects the case when playing back examples stored in the simulator, not for captured data being played back. For captured data, make sure to set the program from which you are capturing to produce the proper number of NULs.

- *Use 7 bits*

Some Altair programs (e.g. 4k BASIC) use the 8<sup>th</sup> bit of a character to signal end-of-string, assuming that the output device only uses 7 bits. If this is the case, then the last character of any string will appear mangled in the output. If this option is enabled, the simulator will always clear the 8<sup>th</sup> bit before sending it to the serial output.

If the option is set to “autodetect”, the s will detect (for some known programs) based on the memory location of the “OUT” instruction whether the bit needs to be cleared or not.

- *Serial input uppercase*

Some Altair programs (e.g. 4k BASIC) only display uppercase

characters and expect all input to be uppercase characters. If this option is enabled, the simulator will translate any incoming lower-case character to upper-case.

If the option is set to “autodetect”, the simulator will detect (for some known programs) based on the memory location of the “IN” instruction whether the translation is required.

- *Translate backspace to*

Allows to map the backspace character to other characters expected by different ALTAIR programs:

*off* – backspace is sent as backspace (ASCII 8)

*underscore* – backspace is sent to the simulated program as an underscore ‘\_’ (ASCII 95) and an underscore sent by the simulated program is interpreted as backspace. This is useful for 4k BASIC and ROM BASIC.

*delete* – backspace sent to the simulated program as a delete (ASCII 127) character. Time-sharing BASIC expects this

*autodetect* – the simulator attempts to automatically determine which conversion is necessary by the memory location of IN/OUT opcode.

- *Enable CLOAD/CSAVE traps (for ACR device only)*

If this option is enabled then the simulator will catch BASIC CLOAD/CSAVE calls and automatically save/load to the internal simulator storage, avoiding the need to manually start cassette capture/replay.

- *SIO board revision (for SIO device only)*

Very early Altair SIO board had slightly different arrangement of bits in the status register. This design (revision 0) was changed very quickly. Very few programs exist that require the rev0 board, for example the original prototype Microsoft BASIC. This setting allows switching between the revisions.

The setting also allows to switch the SIO card into “Cromemco” mode which emulates a Cromemco serial card used by some of the Dazzler programs. See “Cromemco Dazzler support” below.

## File System Manager

The emulator includes a very simple mini file system to store the different types of data that can be saved/captured. The file system manager can be invoked by selecting (M) in the configuration menu.

The file system manager supports the following commands:

- **F**: Format file system (erases all files)
- **d**: Delete a file from the file system
- **r**: Read a file and show contents on screen

## Debugging Capabilities

When simulation is stopped (i.e. the WAIT LED is on) and the “Serial debug” option is enabled in the configuration menu, the following keys have a function (those marked as STANDALONE only work if stand-alone mode is enabled in setup.h):

0-9,a-f	STANDALONE only: Toggle SW0-15
/	STANDALONE only: Prompt for value to set SW0-15
r	Run
o	Stop
t	Step
R	Reset
!	Hard reset (STOP+RESET)
X/x	Examine/examine next
P/p	Deposit/deposit next
U	AUX1 up
u	AUX1 down
s	STANDALONE only: Capture serial data (AUX2 up)
l	STANDALONE only: Play back captured serial data or BASIC example (AUX2 down)
m	STANDALONE only: Mount (hard) disk image (AUX2 down)

Q            Protect  
q            Unprotect  
>            Run from address  
B            Add breakpoint (only if breakpoints enabled in  
setup.h)  
V            Delete last breakpoint  
D            Disassemble (will prompt for start address, space  
bar continues, any other key exits)  
M            Dump memory (will prompt for start address, space  
bar continues, any other key exits)  
n            change number system  
(hexadecimal/octal/decimal)  
C            Enter configuration menu  
L            Load a program or data through serial input into  
simulated memory  
              First value is start address, second value is length,  
followed by data bytes  
              (all values separated by spaces). Easier to enter data  
this way than using the switches.  
H            Load a program in Intel HEX format through serial  
input.  
              [https://en.wikipedia.org/wiki/Intel\\_HEX](https://en.wikipedia.org/wiki/Intel_HEX)  
              Useful to deposit programs and/or data directly into  
the simulated memory.  
h            Dump memory in Intel HEX format through serial  
output.

## **Example program: Kill-the-bit game**

To enter the kill-the-bit game below into the ALTAIR:

1. Activate RESET
2. Set SW7-0 to first byte of program (041 octal, 00 100 001 binary)
3. Activate DEPOSIT
4. Set SW7-0 to next byte of program
5. Activate DEPOSIT NEXT
6. Repeat steps 4-5 until all bytes have been entered

7. If running the simulator on an Arduino MEGA, the game runs slow.  
To compensate, change the content of memory location 006 from 016 to 056 (all octal).

To run the game:

1. Activate RESET
2. Set SW15-8 switches to 0
3. Activate RUN

To save the game to Arduino storage:

1. Set SW15-8 to 0 (selects memory page 0)
2. Set SW7 to 1 (selects memory page operation)
3. Set SW6 to 1 (selects memory page save)
4. Set SW5-0 to 0 (selects file number 0)
5. Activate AUX1 down

To load the game from Arduino storage:

1. Set SW15-0 as above, except SW6 to 0 (selects memory page load)
2. Activate AUX1 down

```

                                ; Kill the Bit game by Dean McDaniel,
May 15, 1975
                                ;
                                ; Object: Kill the rotating bit. If you
miss the lit bit, another
                                ; bit turns on leaving two bits
to destroy. Quickly
                                ; toggle the switch, don't
leave the switch in the up
                                ; position. Before starting,
make sure all the switches
                                ; are in the down position.
                                ;
0000                                org    0
0000 210000                          lxi    h,0    ; initialize
counter
0003 1680                            mvi    d,080h ; set up
initial display bit
0005 010E00                          lxi    b,0eh ; higher
value = faster

```

```

0008 1A      beg:      ldax  d      ; display bit
pattern on
0009 1A      ldax  d      ; ...upper 8
address lights
000A 1A      ldax  d
000B 1A      ldax  d
000C 09      dad   b      ; increment
display counter
000D D20800  jnc   beg
0010 DBFF    in    0ffh  ; input data
from sense switches
0012 AA      xra   d      ; exclusive
or with A
0013 0F      rrc           ; rotate
display right one bit
0014 57      mov   d,a    ; move data
to display reg
0015 C30800  jmp   beg    ; repeat
sequence
0018      end

```

Here is the program in octal for easier entry into the Altair:

```

000: 041 000 000 026 200 001 016 000
010: 032 032 032 032 011 322 010 000
020: 333 377 252 017 127 303 010 000

```

## Loading 4k BASIC the old-school way

The simulator provides a quick and easy way to load 4k BASIC by setting SW0-7 to 00000101 and pressing AUX1 down (see “Aux1 down” section above).

However, it is also possible to load 4k BASIC the original (slow) way via a boot loader.

Here’s how to load BASIC as if it came from a paper tape reader:

- Start the configuration editor and select (1) to configure the SIO device
- Make sure the device is mapped to the host serial port to which your terminal is connected
- Set the “Use 7 bits”, “Serial input uppercase” and “Translate Backspace to” settings to “autodetect”

- Set the simulated baud rate to 110 baud. Obviously you can choose other baud rates here but 110 baud is the speed at which a typical TeleType tape reader operated. **Note:** the baud rate is based on simulated time (not real time), so when running on the Arduino Mega, 110 baud will be closer to 25 baud. All time estimates given below will be about 4x longer. I suggest using 600 baud in that case.
- Enable the “Force baud rate” option
- Exit the configuration editor
- Key in the SIO bootloader (same procedure as entering kill-the-bit above):
 

```
000: 041 256 017 061 022 000 333 000
010: 017 330 333 001 275 310 055 167
020: 300 351 003 000
```
- You may want to save the bootloader to a file so you can reuse it later (same procedure as for the kill-the-bit example above)
- Activate RESET (to reset program counter to 0, the start of the boot loader)
- Activate RUN. The boot loader is now running. You should see LEDs A0-A4 on, A5-A15 should be off.
- Set all switches to 0, then raise SW15, SW7 and SW6. This selects example 0xC0 (SW7=0=0xC0) which is the 4k BASIC tape image to be played back to the SIO device (SW15-13=100)
- Activate AUX2 down. The HLDA LED will come on indicating that data replay has started.
- Immediately switch A15 down (otherwise the second stage boot loader will attempt to load from ACR instead of SIO).
- For the first 17 seconds (while the first-stage boot loader is running) the pattern on the A15-A0 LEDs should not change.
- After about 17 seconds the second stage (checksum) boot loader should start running.  
At that point, LEDs 15-12 should be off, A11-A7, A5, A2, A1, A0 should be on and A6, A4 and A3 should be very faintly flashing (about 10x per second, one flash per byte received).
- The second stage boot loader takes about 6 ½ minutes. Once it is done, the LED pattern should change to A9-A7

and A3-A0 on, all others off and your terminal should show the “MEMORY SIZE?” prompt.

- After you answer the usual startup questions, BASIC should be up and running.
- To load a BASIC example program, set SW0-7 to the program number (e.g. 00011011 for a simple prime number computation) and activate AUX2 down.

## **Altair Time Sharing BASIC**

In its default configuration the simulator allows to run Altair Time Sharing BASIC with two users, one on a terminal connected to the Arduino Due’s main serial port (either use pins 0 and 1 or the USB connection), the other using the Due’s Serial1 port at pins 18 and 19. See the next page for information on how to enable 4 users.

To set up the simulator, enter the Configuration Editor and make sure both the Serial and Serial1 baud rates are set properly for your two terminals. The primary host serial port can be either one, whichever is more convenient.

Next configure the emulated devices. Time Sharing BASIC only supports 2SIO cards, so set the SIO and ACR cards to “Not mapped”. Configure the the two ports of the 2SIO card as follows:

- One mapped to primary, the other mapped to the secondary host interface
- Simulated baud rate: 2400 (anything higher can cause problems when trying to play back examples)
- Example playback NULs: 0
- Use 7 bits: on
- Serial input uppercase: on
- Translate backspace to: delete

Time Sharing BASIC relies on interrupts to give each user their proper time slice and uses the 88-RTC-VI (Real-Time Clock and Vector Interrupt board) to do so. Therefore, both the Real-Time Clock and Vector Interrupts must be set up properly in the simulator. Select “Configure Interrupts” sub-menu:

- Enable the Real-Time Clock and set it to 60Hz or 100Hz
- Enable the Vector Interrupt board (Use Vector Interrupt Board)
- Set the Disk drive interrupt to VIO
- Set the Real-Time Clock interrupt to VI1
- Set the 88-2SIO port 1 interrupt to VI2
- Set the 88-2SIO port 2 interrupt to VI2
- Set the 88-SIO and 88-ACR interrupts to “Not Connected”

For convenience, you may want to configure the “Aux1 shortcut program” setting to the Time Sharing

Basic v1.1 disk. After everything is set, you may want to save the configuration so you can later just load it.

Ready to run Time-Sharing BASIC!

First, connect your two terminals to the Serial and Serial1 connection on the Arduino. Make sure the terminal settings match the **host** interface baud rates set above (NOT the simulated baud rates).

Next, either press AUX1 UP (if you have configured it) or mount the Time-Sharing BASIC disk and run the disk boot loader (see the MITS Disk Controller Support section for more information). After a few seconds, the ALTAIR T/S DISK BASIC V1.1 prompt should appear. Answer the configuration questions as follows:

- RECONFIGURE? N (you may say L and verify that LEVEL1 is set to TIMER and LEVEL2 is set to 2 – 16, 18)
- MEMORY SIZE? [just press ENTER]
- NUMBER OF USERS? 2
- TERMINAL ADDRESS? 16
- REGION SIZE? 20000
- TERMINAL ADDRESS? 18
- REGION SIZE? 20000
- MOUNT PASSWORD? [pick a password and press ENTER]

After the last prompt, a BASIC startup message (ALTAIR T/S DISK BASIC V1.1) should show up on both terminals. Both terminals can be used independently to interact with BASIC and load and run programs. Refer to the ALTAIR Time Sharing BASIC manual (can be found online) on how mount disks and load programs.

To allow more than two users, the simulator software must emulate a second 88-2SIO card. To see if it is configured for two cards, start the configuration editor. If it shows options to change settings for “2SIO-2 port 1” and “2SIO-2 port 2” (in addition to “2SIO port 1” and “2SIO port 2”) then it is already set. Otherwise, you need to update the software:

1. Load `config.h` in an editor
2. Change “`#define USE_SECOND_2SIO 0`” to “`#define USE_SECOND_2SIO 1`”
3. Upload the simulator software to your Arduino

After that is done, start the configuration editor and map “2SIO-2 port 1” to the Arduino’s Native USB port. If you want an additional user, follow the instructions in the “Serial Ports” section to enable the serial port on the A6/A7 pins of the Arduino.

Configure the “2SIO-2 port 1” (and “2SIO-2 port 2”) ports following the instructions for the other ports above, mapping them to Native USB port (and Serial A6/A7). Likewise, configure the interrupts for the new ports:

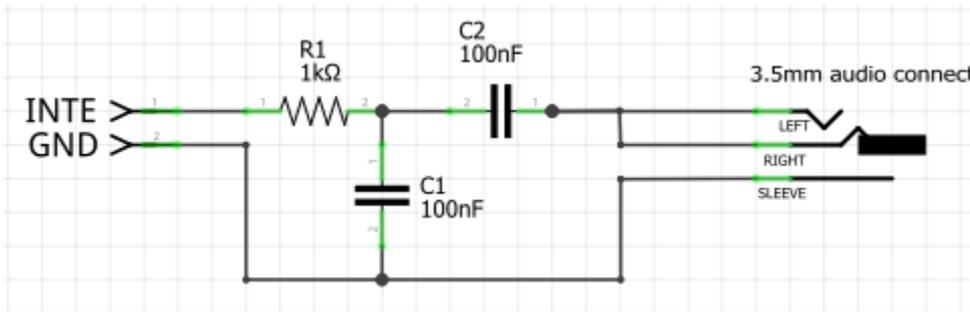
- Set the 88-2SIO port 1 interrupt to VI3
- Set the 88-2SIO port 2 interrupt to VI3

Now you can start Time-sharing basic either for three users (terminal addresses 16, 18, 20, region sizes of 14000) or four users (terminal addresses 16, 18, 20, 22, region sizes of 10000).

## Music System

The Music System from Processor Technology was an affordable Music System for the Altair that required only a minimal hardware addition. To learn more about the Music System and how it ran on the original Altair, head over to [http://altairclone.com/music\\_system.htm](http://altairclone.com/music_system.htm) where Mike Douglas has put together a great description.

You can run the music system on the simulator without any changes but in order to get any actual sound you will need to add just a minimal number of components: two capacitors and one resistor. Here's the schematic (taken from page 7 of the Music System [documentation](#)):



Connect the INTE input of the above schematic to the Arduino pin that drives the INTE light (pin D12 on the Due, pin D38 on the Mega) and GND to ground. The output is a line level mono signal that can be connected to the input of any amplified speaker system. It is not strong enough to drive headphones although but some mini earphones do produce a (not very loud) sound when plugged in.

The music system consists of two parts:

- The music system itself, which allows to enter, compile and play music (see [User's Guide](#))
- The ACUTER monitor (a version of the CUTER monitor ported to the Altair and enhanced by Mike Douglas) which is necessary to load and save music as well as handling input/output (see [User's Guide](#)). The music system can **not** run on its own without the monitor.

To run the music system on the simulator, do the following:

- Enter the configuration monitor and set “throttle delay” to 12 (leaving it at automatic delay adds a weird vibrato effect when playing music)
- Make sure that 2SIO port 1 is mapped to your primary host interface
- Enable the “Serial input uppercase” option for the 2SIO port1
- Use a baud rate of no more than 9600 for your primary host interface (otherwise importing the HEX data for music examples will not work properly)
- Exit the configuration monitor
- Turn on switches SW0, SW2 and SW3, all others off
- Activate AUX1 down
- You should now see a “>” prompt in your terminal. This is the command prompt of the ACUTER monitor.
- Enter “EX 0” to initialize the music system itself.
- You should see “THE MUSIC SYSTEM (C) 1977 SOFTWARE TECHNOLOGY CORP”
- Enter “RET” to return to ACUTER (should see the “>” prompt)

Now we need to get some music into the system. Mike Douglas has collected a number of music examples for the system. Follow this [link](#) to get them (look in the subdirectories). To get an example into the system, we use the HGET command that Mike has added to ACUTER:

- Enter “HGET” in ACUTER
- You should see “Send/Rcv on port 0”
- Now click on one of the HEX files of music examples from the web site (e.g. SALLY.HEX) so it opens in your web browser. It should show many lines looking like  
:1008D3003430303130202F41495220574954482079
- Select all (likely CTRL-A) in your browser and copy it to the clipboard (CTRL-C)
- Now paste it into your terminal.
- The terminal should show many lines listing addresses and return to the “>” prompt when done.

- Now enter “EX 0” to go back to the music system (there will be **no** prompt after entering the system)
- Enter “FILE”, which causes the music system to scan memory for the imported music file
- Next, enter “SCORE”, which compiles the music
- Finally, enter “PLAY” to play the compiled music. You should hear the music in your speakers.

For a detailed description of the ACUTER and music system commands refer to their User manuals: [ACUTER Music System](#). Here’s an example session (user input in **bold italics**, system output normal):

```
>EX 0
THE MUSIC SYSTEM (C) 1977 SOFTWARE TECHNOLOGY CORP.
<77-05-19>
08D3 08D3
RET
>HGET
[paste contents of SALLY.HEX into terminal]
08D3
08E3
08F3
[...]
10E3
>EX 0
FILE
08D3 10F0
SCORE
10F1 17D1
PLAY
```

If desired, a piece of music that was imported via HGET can be saved within the simulator to a virtual tape by using the capture/replay functionality. To save an example to a file from within ACUTER:

```
>HGET
[paste contents of SALLY.HEX into terminal]
08D3
08E3
08F3
[...]
10E3
[SW15, SW13, SW8, SW0 up, all others down => prepare to capture data from ACR to file #1]
```

```
[AUX2 up => start capture]  
>SAVE SALLY 08D3 10F2  
>  
[AUX2 up => end capture]
```

To load an example from a file within ACUTER:

```
>GET  
[SW15, SW13, SW8, SW0 up, all others down => prepare to play back file #1 to ACR]  
[AUX2 down => start playback]  
SALLY 08D3 0820  
>
```

## **MIT S Programming System II**

[adapted from Mike Douglas' instructions at [altairclone.com](http://altairclone.com)]

The MIT S Programming System II allows development of 8080 assembly language applications

on the Altair 8800 using just paper tape or cassette for mass storage.

The package consists of an

editor, assembler, debugger and a monitor that allows execution of these programs as well as the programs you may develop.

To use the programming package, the monitor program is loaded from paper tape or cassette in the

same manner as loading BASIC. Once loaded, the monitor is then used to load the editor,

assembler, debugger or user programs.

The monitor loads from paper tape or cassette using the same bootstrap loader as would be used

for BASIC version 3.2. Sense switch settings are the same with the addition of A9 functionality:

A9 up – the monitor does NOT use serial input interrupts

A9 down – the monitor uses serial input interrupts

## **Loading the Monitor, Editor and Assembler**

- 1) Set A15-A3 down and A2-A0 up.
- 2) Raise A9 if you want to disable interrupts. If interrupts are enabled, serial replay is slower (because it must run at the given baud rate). On the other hand, enabling interrupts enables the use of Ctrl-C.
- 3) Push AUX1 down to load the monitor.
- 4) The monitor prompt is two spaces and "?"
- 5) STOP! Do not type any commands to see "what happens." If what is typed is not a command, the monitor tries loading a program of that name from the cassette and hangs there until you provide that program. If the ABS device has been set to audio-cassette (AC), then Ctrl-C will return to the monitor prompt if interrupts were enabled during the boot process. Otherwise, follow the instructions below to restart the monitor from the front panel.
- 6) Assign the program load device to the cassette, type: "OPN ABS,AC<cr>"
- 7) Load the editor, type: "EDT<cr>". When loaded, the editor's prompt "\*" is displayed. Type "E<cr>" to return the monitor.
- 8) Move the editor's buffer location into high memory so the assembler can reside in memory at the same time as the editor. For larger programs, more than the default 2K of buffer space will probably be needed as well. Here are settings for an 8K buffer:
  - Type "DEP 5124<cr>" and enter "0<cr>" then "100<cr>" then ctrl-z.  
This specifies the 16-bit octal address 40000 (0x4000) for buffer start.
  - Type "DEP 5530<cr>" and enter "0<cr>" then "140<cr>" then ctrl-z.  
This specifies the 16-bit octal address 60000 (0x6000) for buffer end.
- 9) Load the assembler, type: "AM2<cr>". When loaded, the assembler prompt is "\*ASM\*<cr>". Type "EOA<cr>" (end of assembly) to return to the monitor.
- 10) DBG, EDT and ASM share the same memory space (see Memory Allocation section below). If two of them are loaded sequentially, the system will just start the second one loaded, even if invoking the first. For example, if EDT is loaded first and then ASM, the monitor will invoke ASM if EDT is typed as a

command. To force reloading of a component, use CLR to remove it from the program table, i.e. in the situation mentioned before, typing CLR EDT and then EDT will invoke the editor.

The computer is now ready to iteratively edit, assemble and run/test a program.

## Using the Editor and Assembler

1. To start the editor, type: "EDT<cr>" This starts the editor and clears the edit buffer. To subsequently re-edit a program, type "EDT(R)<cr>" This leaves the existing source code in memory. If the "(R)" is left off, the program source will be erased.
2. At the editor prompt type "I" for insert. Ctrl-Z exits the insert mode. "P" prints (displays) the file with line numbers. "W" displays the file without line numbers. "D line[,line]" deletes line(s). "R line" replaces a line. "I line" inserts after the specified line. "I" by itself inserts before the first line. "E" exits the editor and returns control to the monitor.
3. End programs with:
  - BEG start label
  - END program name
  - EOAWhere "start label" is the program entry point, "program name" is a three character program name. "EOA" means end of assembly.
4. Assign source file input to come from the edit buffer instead of a device and declare file type of ASCII (text file): Type, "OPN FIL,EB,A<cr>" This only has to be done once during a session (or if the FIL device was subsequently assigned to a different device during the session).
5. Run the assembler: Type "AM2<cr>"
6. Tell the assembler to take input from a file: Type "FILE<cr>" The assembler will run and show errors and undefined symbols. Note: The "Undefined Symbols" heading is displayed even when there are no undefined symbols. Control is returned to the monitor.

7. Run the program by jumping to the starting address: Type "JMP xxxxxx" where xxxxxx is the program starting address in octal.
8. You can enter the program into the monitor's program table as follows: After a successful assembly, re-enter the assembler and preserve symbols: Type "AM2(P)<cr>"
9. Type "RUN name<cr>" where name is the three character name for the program. The program will run. From here on, the program can be run by typing the "name" specified at the monitor prompt. This step does not have to be repeated after subsequent assemblies if the entry address has not changed.

## Memory Allocation

- The monitor is about 2.5K in length and uses RAM up through 0x0A3F
- The editor is about 2K in length and resides just above the monitor at 0xA40 – 0x11B1, followed by the default 2K edit buffer space from 0x11B2 – 0x19B1.
- Two versions of the assembler are available. Each are about 3K in length. ASM (assembler version 1) loads at the same address as the editor (0xA40). This is inconvenient for the iterative cycle of edit, assemble and test. As an alternative, AM2 (assembler version 2) loads just above the editor at 0x1350 – 0x1D78. The assembler's symbol table grows up from 0x1D78. Note that the default location of the edit buffer conflicts with the load address of AM2 as the edit buffer grows. Before using the editor for longer programs, the location of the edit buffer should be moved to a free area in memory by using the monitor DEP command to patch the buffer start address into octal locations 5124-5125 and the buffer end address+1 into octal locations 5530-5531.
- When using the AM2 assembler, a program loaded at 0x2000 leaves about 650 bytes of symbol table space (0x1D78 – 0x1FFF). This is enough for small demo programs of 100 lines or less. Otherwise, a higher starting address should be used for the target program
- The debugger is about 2K in length and overlays the editor at 0xA40 – 0x133F. The debugger and AM2 assembler can both

reside in memory at the same time. This makes it easy to patch programs with the assembler while debugging the program.

## Loading and Saving Files

1. Assign the FIL device to the audio cassette and specify ASCII files:  
Type "OPN FIL,AC,A<cr>"
2. In the editor, type "L<cr>" to load a source file. Type "S<cr>" to save a source file.
3. After loading a source file into the editor, you can re-assign FIL to the edit buffer for use as the assembler input: Type "OPN FIL,EB,A<cr>"
4. When you want to save an edited file after the edit and assemble process, be sure to set the FIL device back to AC before using the "S" command from within the editor.
5. Optionally, you can take assembler source file input from the audio cassette. After assigning FIL to the AC (see step 2), in the assembler, type "FILE name<cr>" where "name" is the three character source file name for the assembler to read. The source file is then read directly from the audio cassette instead of the edit buffer.

## Additional Notes

- For space reasons the assembly examples (see below) are NOT included if Z80 support is enabled.
- When using the edit buffer as the source for the assembler, the source file must be "rewound" by editing the source file in between successive assemblies.
- To restart the monitor, stop the machine, set all switches off except A6 (i.e., set address of octal 100). Raise examine, set front panel switches back the way they were at load time, then depress run.
- When sending large amounts of text (e.g. pasting source code into the serial terminal), some characters may get lost. Refer the section about pasting text into the terminal in the "Interacting with running programs" about possible solutions.

## Example usage #1: assemble directly from input

- 1) Make sure simulator is configured to send SIO output and SIO2 port 1 output to your terminal
- 2) *[STOP]+[RESET]* reset Altair
- 3) *[A15-A3 down, A2, A1, A0 up]* select program #7 (PS2 monitor)  
using SIO card with interrupts
- 4) *[AUX1 down]* load and run PS2 monitor and  
mount PS2 cassette tape
- 5) OPN ABS,AC assign ACR to ABS device (to load  
editor/assembler)
- 6) AM2 start assembler (to load it into  
memory)
- 7) *[CLR]* un-mount the PS2 tape
- 8) *[A7 up, A2 down, A1 up]* select ASM example #2 (“dump”  
example from PS2 documentation)
- 9) *[AUX2 down]* start ASM source code example  
replay
- 10) RUN DUMP runs DUMP example directly  
from assembler

## Example usage #2: insert into edit buffer, then assemble

- 1) Make sure simulator is configured to send SIO output and SIO2 port 1 output to your terminal
- 2) *[STOP]+[RESET]* reset Altair
- 3) *[A15-A3 down, A2, A1, A0 up]* select program #7 (PS2 monitor)  
using SIO card with interrupts
- 4) *[AUX1 down]* load and run PS2 monitor and  
mount PS2 cassette tape
- 5) OPN ABS,AC assign ACR to ABS device (to load  
editor/assembler)
- 6) EDT start editor (to load it into  
memory)
- 7) E end editor
- 8) AM2 start assembler (to load it into  
memory)
- 9) EOA exit assembler
- 10) DEP 5124 move edit buffer to 0x4000-  
0x6000 = 8K length

11) 0	start address 100-000 = 40000
octal = 0x4000	
12) 100	
13) [CTRL-Z]	end of input
14) DEP 5530	
15) 0	end address 140-000 = 60000
octal = 0x6000	
16) 140	
17) [CTRL-Z]	end of input
18) EDT	start editor again (with new
buffer)	
19) I	enter "insert" mode
20) [CLR]	un-mount the PS2 tape
21) [A7 up, A2 down, A1 down, A0 up]	select ASM example #1 (PONG)
or [A7 up, A2 down, A1 up]	select ASM example #2 (DUMP)
22) [AUX2 down]	start ASM source code example
replay	
23) [CTRL-Z]	exit "insert" mode after example
is loaded	
24) E	exit editor
25) OPN FILE,EB,A	select edit buffer as input device
26) AM2(S)	go into assembler (S parameter
to print symbol table at end)	
27) FILE	load input file (from edit buffer)
28) EOA	return to monitor
29) JMP 20000	run program

### Example usage #3: insert into edit buffer, assemble, write to file, run file

1) Make sure simulator is configured to send SIO output and SIO2 port 1 output to your terminal	
2) [STOP]+[RESET]	reset Altair
3) [A15-A3 down, A2, A1, A0 up]	select program #7 (PS2 monitor)
using SIO card with interrupts	
4) [AUX1 down]	load and run PS2 monitor and
mount PS2 cassette tape	
5) OPN ABS,AC	assign ACR to ABS device (to load
editor/assembler)	
6) EDT	start editor (to load it into
memory)	
7) E	end editor

- |  |                                    |
|--|------------------------------------|
| 8) AM2<br>memory)  | start assembler (to load it into   |
| 9) EOA   | exit assembler                     |
| 10) DEP 5124<br>0x6000 = 8K length                               | move edit buffer to 0x4000-        |
| 11) 0<br>octal = 0x4000  | start address 100-000 = 40000      |
| 12) 100  |                                    |
| 13) [CTRL-Z]   | end of input                       |
| 14) DEP 5530   |                                    |
| 15) 0<br>octal = 0x6000  | end address 140-000 = 60000        |
| 16) 140  |                                    |
| 17) [CTRL-Z]   | end of input                       |
| 18) EDT<br>buffer)   | start editor again (with new       |
| 19) I  | enter "insert" mode                |
| 20) [CLR]<br>ACR   | un-mount the PS2 tape from         |
| 21) [A7 up, A2 down, A1 down, A0 up]                             | select ASM example #1 (PONG)       |
| 22) [AUX2 down]<br>replay  | start ASM source code example      |
| 23) [CTRL-Z]<br>is loaded  | exit "insert" mode after example   |
| 24) E  | exit editor                        |
| 25) OPN FIL,EB,A   | select edit buffer as input device |
| 26) AM2(S,A)<br>to print symbol table at end,                    | go into assembler (S parameter     |
|  | A to dump output binary to file)   |
| 27) [A15, A13, A8 up, A7 down]<br>ACR device                     | Prepare to capture file #1 from    |
| 28) FILE<br>[wait for "SENSE SW 15 FOR DUMP" message]            | load input file (from edit buffer) |
| 29) [AUX2 up]  | start capturing                    |
| 30) [A15 down, A15 up]<br>[wait for "?" prompt from PS2 monitor] | tells AM2 to start writing         |
| 31) [AUX2 up]  | finish capturing                   |
| 32) [AUX2 down]  | start replay                       |
| 33) PONG   | load and run PONG                  |

# Interfacing external hardware via data/address buses

While the Simulator does not (and cannot) provide a full S-100 bus, it is possible to use the connections to address LEDs (A15-0) and data LEDs (D7-0) together with the as the INP/OUT/WO status LEDs to interface external hardware.

Although this is not enough to connect vintage S-100 cards to the Simulator, it is possible to create new hardware that interfaces via those signals. Note that all signals can be picked up directly at the Arduino's pins, making it possible to place such external hardware on an Arduino "shield".

The Simulator software changes the state of the status LEDs **after** setting the address/data LEDs to their respective values, so external hardware can intercept

- OUT operations by reacting to a low->high edge of the OUT signal
- IN operations by reacting to a low->high edge of the INP signal
- Memory WRITE operations by reacting to a high->low edge (negative logic) of the WO signal.

Read below for details on how to handle each operation.

Note that it is currently not possible to intercept Memory READ operations as the Simulator software has no way of knowing whether to read from the emulated RAM or from data bus.

For all operations, address and data signals are updated just before the signal LED changes state and the signal should be stable at that point.

Note that the Simulator timings are not precise. The actual length of an OUT/INP/WO operation can vary greatly, depending on many conditions such as the Simulator configuration, compiler optimization and others. Not all cycles will have a consistent length, therefore only minimum lengths for the operations are given below.

## Intercepting OUT operations

The D7-0 LEDs in the Altair 8800 were connected to the S-100 bus' DI7-0 lines, meaning they would show the state of the data bus going *into* the CPU. During OUT and memory WRITE operations, data was put on the DO7-0 lines and the DI7-0 lines were left floating, so the D7-0 LEDs would all be on no matter what value was output.

In its default configuration, the Simulator emulates this behavior, so external hardware monitoring the D7-0 LEDs would not be able to see the actual data values. A setting in config.h provides a means to change the behavior such that the D7-0 LEDs show the proper values for OUT and memory WRITE operations:

In config.h change `"#define SHOW_BUS_OUTPUT 0"` to `"#define SHOW_BUS_OUTPUT 1"`

The "OUT" pulse lasts 500 nanoseconds or longer.

### **Intercepting IN operations**

The Simulator software emulates a number of hardware devices (disk drive, hard disk, serial cards). If an IN instruction is executed that requests data from a port that is not emulated, the input value will be 0xFF (as it would be in the original). However, there is a setting in config.h that changes the behavior when inputting data from a port that is not emulated. Setting `"#define READ_UNUSED_PORTS_EXT 1"` in config.h will make the Simulator read the status of the SW15-8 input lines in that case (instead of just returning 0xFF). The input value will be read 700 nanoseconds (or later) after the low-high edge on the INP signal. The whole "INP" pulse lasts 1500 nanoseconds or longer.

### **Intercepting memory WRITE operations**

As during the OUT operation, memory WRITE operations would also not show the output value on the D7-0 LEDs. The same config.h switch as in the "OUT" section above will cause the Simulator software to show the output values so they can be visible to external hardware. The "WO" pulses last 200 nanoseconds or longer.