1802 Membership Card Loader - Arduino method

Donald T. Meyer
don@sgsw.com
10-Jun–2015

So I built Lee Hart’s slick little 1802 Membership Card and its Front Panel card. Next I wanted to load some programs onto it! There was that nice DB–25 connector all ready to connect to my PC’s parallel port. However… I’ve got a Mac, and haven’t seen a computer parallel port in perhaps a decade.

My solution to this problem was to create my own interface hardware based on an Arduino. I considered using a Spark Core to allow doing things wirelessly via WiFi, but decided that for this purpose the Arduino was something more people were likely to have, and it was just a somewhat simpler solution.

The final design uses an Arduino, a 16-line I2C port expander chip, and some software. The software consists of the Arduino program and a Python program that runs on my Mac. Since it’s Python, it should also run on a Windows system, a Linux box, etc. You can also forego the Python side of things and directly control the Arduino loader from a terminal, or from your own software if you prefer.

Optionally you can connect a 4x20 LCD display to the Arduino to show what’s going on with the loader. This is not required at all, but was very handy during the development effort, so I’ve left that option in the code. *Be aware that it does slow things down by about a factor of 40!.*

Why go to all this trouble? Just a fun little project!

All the software is available free for non-commercial use here: 1802 Membership Card Loader

The hardware is pretty straightforward and documented below. If I get some additional free time and/or there is any demand, I’ve considered making a simple PCB to eliminate most of the wiring.

Parts

I acquired my parts from Adafruit and Jameco.

Required Parts:

- Arduino (I used a Uno).
- MCP23017 Port Expander chip (Adafruit PRODUCT ID: 732)
- Breadboard Wires (Adafruit PRODUCT ID: 153)
- Solderless Breadboard (Adafruit PRODUCT ID: 64)
- ~15ft #22 solid hookup wire (Adafruit PRODUCT ID: 288)
- Male DB–25 solderable connector (Jameco, etc. I usd one from my junkbox)

Optional Parts:

- 4x20 character LCD (I used one with an I2C backpack from Adafruit)
- LEDs (4) and current-limiting resistors. This lets you see the states of the control pins if so desired. Not really needed if you have the LCD display.

Theory

The 16-line port expander is configured as 12 output pins and 4 input pins.

- A0-A3 are inputs that read the output port and multiplexer on the Front Panel card.
- A4-A7 are outputs to the four control signals /BWAIT, /BCLR, /IN, and /WE.
- B0-B7 are outputs that drive the input port on the Front Panel card.

The port expander chip is connected to the Arduino I2C pins. The address is set to 1 since the LCD display I was using is set to 0 be default. (It was easier to set the port expander's address on my breadboard than to solder a jumper onto the I2C LCD backpack!)

The Arduino receives commands via its serial port and manipulates the 1802 control lines and data ports as required.

The one tricky bit is that to read data from the 1802, the high and low nybbles are presented on the DB–25 pins 4-bits at a time. Which nybble is controled by the state of the Input signal, so that line's states and transitions need to be coordinated with the two data reads that are needed from the port expander.

Assembly

A schematic is on my to-do list, but the curcuit is really pretty simple. A lot of port pins to the DB–25 connector, and then power, address, and I2C connections for the rest of the port IC.

My prototype was built using a solderless breadboard. (soldering was required to get appropriate wires between the breadboard and the DB–25 connector.)

Here are the connections between the DB–25 connector and the MCP230127 chip, DB–25 first and the IC pin second:

```
1 – 25
2 – 1
3 – 2
4 – 3
5 – 4
6 – 5
7 – 6
8 – 7
9 – 8
10 – 23
11 – 24
12 – 22
13 – 21
14 – 26
15 – none
16 – 27
17 – 28
18 – none
```

DB–25 pin 19 to GND (ground) of the Arduino etc.

Additional connection for other pins of the MCP230127 chip, chip pins first:

```
9 – +5V (Vdd)
10 – GND (Vss)
12 – Arduino A5 (SCL)
13 – Arduino A4 (SDA)
15 – +5V (A0)
16 – GND (A1)
17 – GND (A2)
18 – +5V (/Reset)
```

Connection & Use

- Set the Wait and Clear switches in the “up” position.
- Set the Read/Write switch to “Read”.
- Set all of the data input switches to the “up” position.

Python software

The loader control utility is written in Python that reads Motorola S-Records, Intel Hex, and raw Hex files for download to the 1802.

Writes those same formats for uploading and writing dumps of the 1802's memory.

There are two 3rd-party modules required: `pyserial` and `bincopy`. Pyserial provides serial port access on pretty much any common OS. Bincopy adds support for reading and writing S-=Records and Intel Hex.

Installation

The loader host tool can be run from the command line by treating it as an executable script.

Installation of the 3rd=party moduules on OSX:

```
> sudo easy_install -U bincopy
> sudo easy_install -U pyserial
```

To use the simple assembler I have written, download my Cosmac 1802 Assembler

Place the assembler Python file `cosmacasm.py` in the same directory as `mcard.py`.

Or, on a Mac or Linux system you can create a symlink to the assembler. (Not sure what the options are for a Windows system)

```
cd <directory containing "mcard.py">
ln -s <path-to-the-assmebler-repo>/cosmacasm.py cosmacasm.py
```

Usage

To see the commands and options, you can get the help info by running the program like this:

```
  >./mcard.py --help
```

Mcard.py will attempt to automatically choose the serial port your Arduino is connected to. If there are multiple ports that could be the Arduino, you must specify the correct port on the command line.

To list the possible ports, use the `-l` option. To set the port, use `-p`.

Download

To download a file, use the `-d` option and the path to the file containing the data (e.g. an S-Record file).

The default download address is 0x0000 but that can be changed with the `-a` option.

Upload

You can upload the contents of the 1802 memory and have it written to a data file.

Set the upload size with the `-s` option.

The default upload address is 0x0000 but that can be changed with the `-a` option.

Set the file format with the `--format` option.

Run

The run command `-r` can be used to run an existing program already loaded on the 1802, or it can be given with the download command to cause the program to run after the download is complete.

Terminal Mode

The loader host supports a simple terminal mode that makes it easy to send commands to the Arduino Loader. It is entered by launching with the `-t` option.

Automatic Assembly

If the associated 1802 assembler (also written in Python) is present, you can have an assembler source file assembled and downloaded in one step. This makes it quite easy to modify a program and have it assembled, downloaded, and executed with one command.

```
> mcard.py --baud 19200 -r -d slowq.src
```

Performance

At 19,200 baud, an 8k download takes a little under 40 seconds.

Examples

Set the serial port speed to use, download a hex file, and run it when the download completes:

```
mcard.py --baud 19200 -r -d slowq.hex
```

Upload 64 bytes, display them on the terminal, and write them to file `foo.hex` in hex format.

```
mcard.py -s 64 --baud 19200 --dump -u --format hex foo.hex
```

Arduino Software

Overview

The Arduino software responds to commands over the Arduino's USB serial port to control Lee Hart's CDP1802-based Membership Card.

The command set is designed to make it fairly easy to talk to the Arduino loader directly from a terminal.

You can directly enter simple programs in hex, read back memory locations, etc. Obviously to do anything complicated you'll want to use some host software such at the host utility I wrote in Python.

Each command is terminated with a newline character.

Most commands have their completion acknowledged with a "!" character. The only exception is the "read bytes" command where the bytes themselves are an implicit acknowledgement.

All commands that take letters may use either the uppercase or lowercase letter. As in, the download command may be either `*D` or `*d`, and the clear pin set high command could be either `+C` or `+c`.

High-level Commands

Command	Description
`*D[addr]`	Download Mode
`*U[addr]`	Upload Mode
`*R`	Run Mode
`*P`	Pause Mode
`*C`	Reset Mode
`^<addr>`	Advance to address
`<n`	Read n hex bytes
`+p`	Set signal pin high
`-p`	Set signal pin low
`>hh`	Set output port

Detailed Descriptions

`*D[addr]` - Download Mode

The 1802 is placed in Load mode with memory writes enabled. In this mode the loader expects pairs of hex digits, which are loaded sequentially into memory. These must be pairs (e.g. "0A"), and a maximum of 16 pairs may be sent per line (before a newline). The only other valid commands in this mode are the "mode" commands and the "advance to adress command". Address is in hex, and can be 1 to 4 digits. (e.g "E", "17", "000E", "AA00"). The address is optional, and if no address given 0x0000 is used.

`*U[addr]` - Upload Mode

In this mode, the primary command is the read-bytes command. The only other valid commands in this mode are the "mode" commands and the "advance to adress command". Address is in hex, and can be 1 to 4 digits. (e.g "E", "17", "000E", "AA00"). The address is optional, and if no address given, 0x0000 is used.

`*R` - Run Mode

Runs from current address, so to start at address zero, you must first isssue a reset command.

`*P` - Pause Mode

The 1802 is placed in Pause mode.

`*C` - Reset Mode

The 1802 is placed in Reset mode.

`^<addr>` - Advance to the given address This command can be used any time in the Upload or Download mode. Address is in hex, and can be 1 to 4 digits. (e.g "E", "17", "000E", "AA00"). *Note: If the address is less than the current address, a reset cycle will occur.*

`<n` - Read n hex bytes

This cycles the Input line to High and back to Low. Expects input line to already be in the Low state! (as it is configured by the Upload Mode command.) The 'n' is optional, and one byte is read if the 'n' is not present. N can range from 1–16, and is in decimal. *Note: This command is NOT acked with a `!` character.*

Low-level Commands

`+p` - Set signal pin high

Values for "p":

- `M` - /MWrite
- `C` - /Clear
- `W` - /Wait
- `I` - Input

`-p` - Set signal pin low

Values for "p":

- `M` - /MWrite
- `C` - /Clear
- `W` - /Wait
- `I` - Input

`>hh` - Place hex byte on output port

This places the given hex byte on the loader's data output port lines. This can then be loaded into memory via a "manual" Load cycle, or read by a program running on the Membershop Card.

Examples

Download Examples

Starting load of a simple Q output blinker program at address 0x000, and then running the program.

```
*D
7B7A
3000
*C
*R
```

Starting load at address 0x000, and then skipping locations to load at a higher address

```
*D
7B7A
3050
^50
3000
```

Upload Examples

Upload 32 bytes from address 0x0000

```
*U
<16
<16
```

Upload 1 byte from address 0x0000, and 8 bytes from address 0x44

```
*U
<
^44
<8
```

Misc. Examples

Set the memory write line low.

```
-M
```

Set the Input line high.

```
+I
```