retro 1 design/learning log

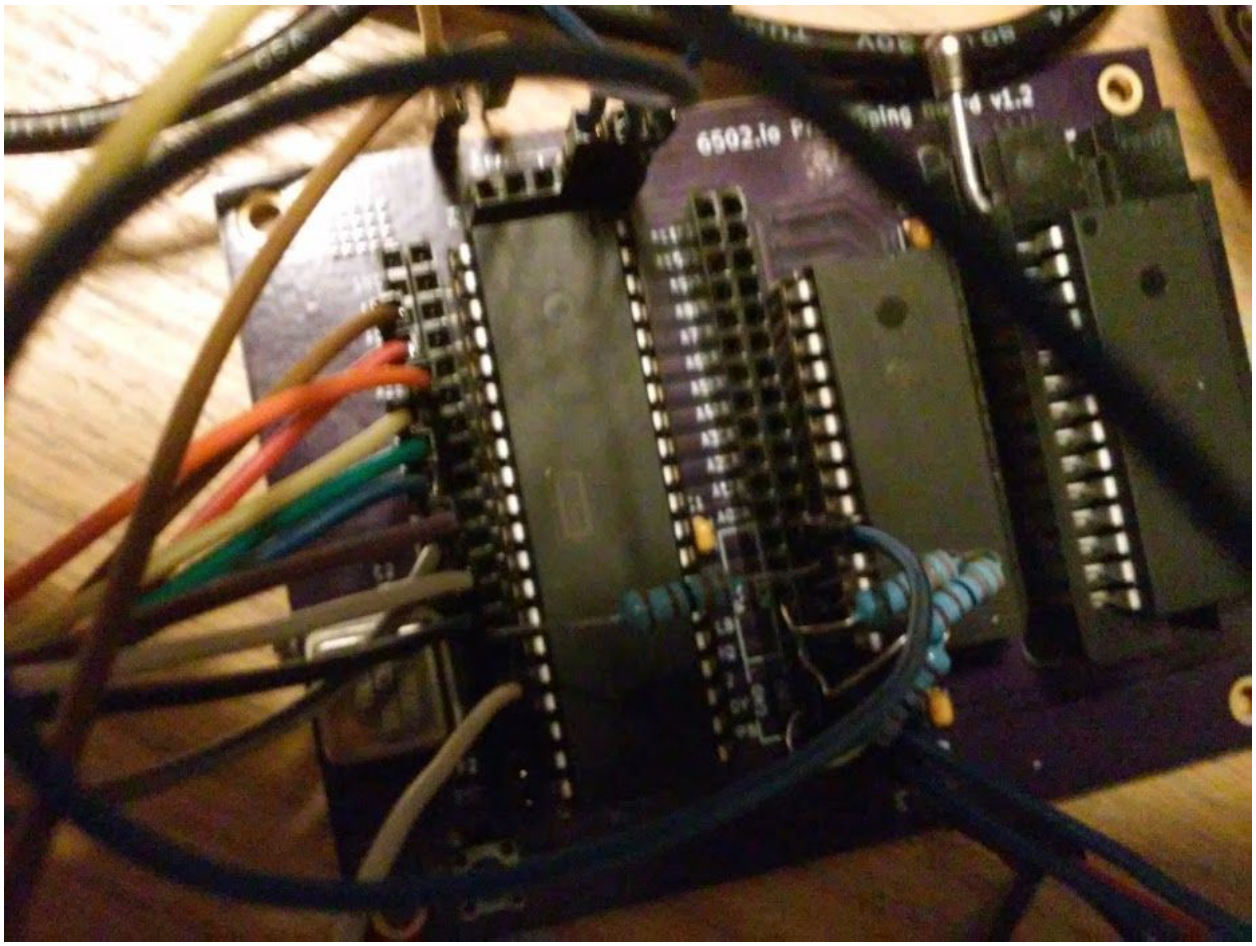
1. Setup Circuit like here (except don't use SO pin): http://www.grappendorf.net/projects/6502-home-computer/eeprom-and-a-first-program
2. Add a 3.3k resistor between BE pin and vcc (this is needed for 65c02)
3. hook up logic analyzer to data pins and check that data is correct ie: A9,12,4C,00,80 etc…

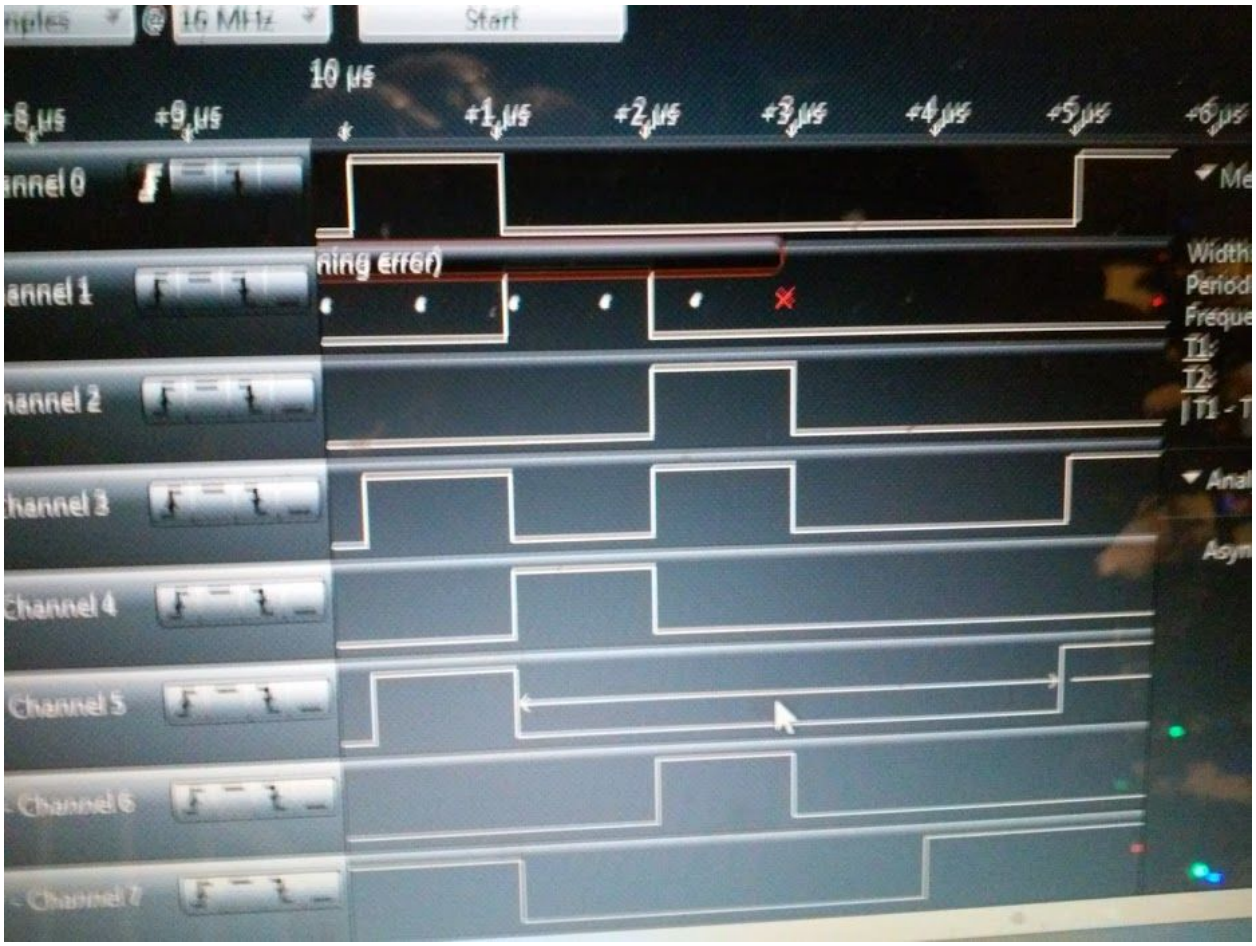
Code (note I changed fff0 address to 1ff0):

```
00008000  a9 12 4c 00 80 00 00 00  00 00 00 00 00 00 00 00
00008010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
00001ff0  00 00 00 00 00 00 00 00  00 00 00 80 00 80 00 80
00010000
```

What's happening?

When the 65C02 starts up it first will execute whatever code is at the reset vector. Therefore it'll put 1ffc and 1ffd (the reset vector location. Actually the real location seems to be fffc and fffd) out on its address bus. It's up to the glue logic to make sure the rom is setup to respond to this address. The above reset vector points to address 8000 (remember that this is written in little endian so the 80 will be before 00). In this particular circuit, there is no glue logic. We wire up rom oe/ce to ground and we to vcc. This essentially just makes the rom spit out whatever data is on the address bus. Next we will try to add some glue logic.


6502.io



Basic Glue Logic

Keep 65c02 setup as above. Setup via/decoder as explained here :http://www.reocities.com/SiliconValley/2072/6502prj1.htm.

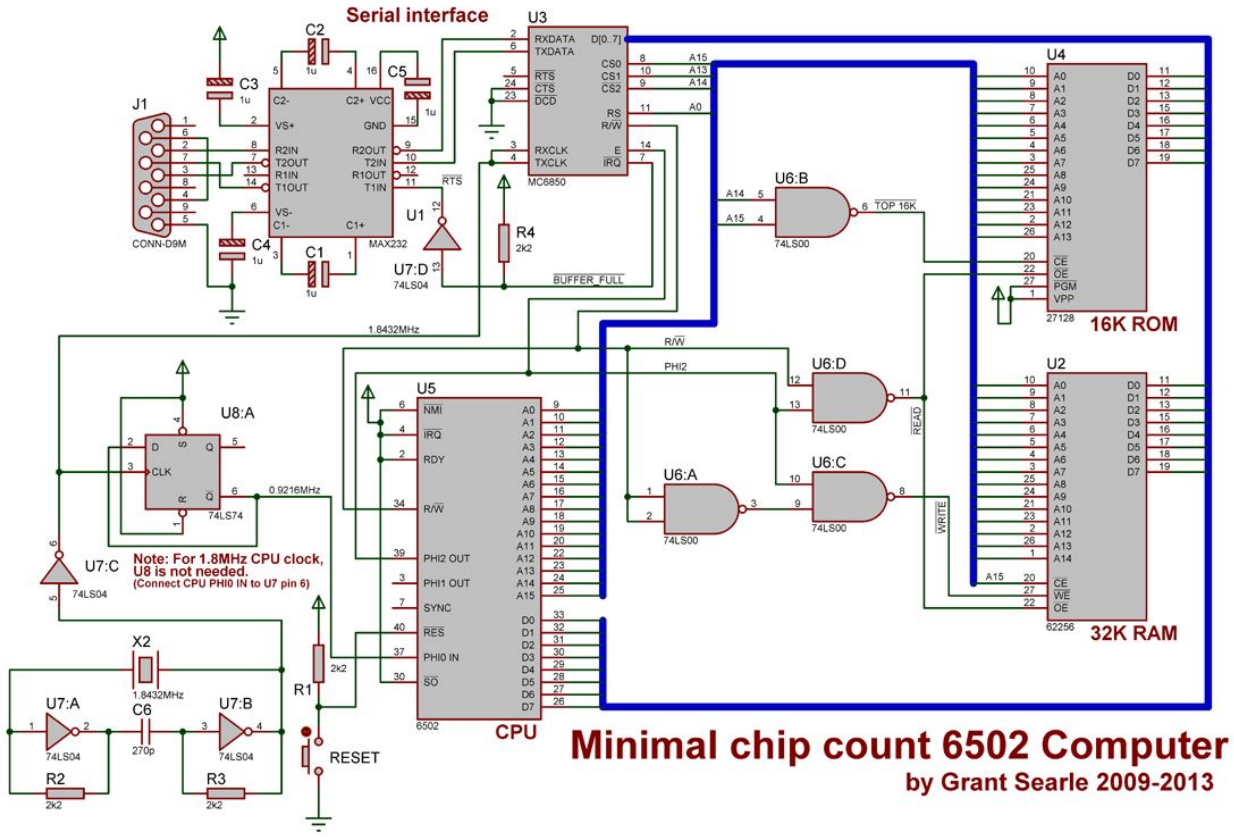
1. For program, since we're using the AT28C256 ROM, our reset vector is at 7FFC/7FFD. And our code will have to start at address 6000 to work without modifying anything else.

a. I figured out reset vector since the cpu will look for the vector at FFFC/FFFD - but since the AT28C256 only has 15 address lines, it won't see the last binary 1. So instead of 1111111111111100(FFFC) the ROM will see 11111111111110(7FFC).

b. if our reset vector contains E000, that will translate to address 6000 by the ROM since it will only see the lower 15 address lines (ie instead of 1110000000000000 it sees 110000000000000)

Better glue logic

keep BE set high as well as RDY, IRQ, and NMI to vcc. Setup 74ls00 for address decoding as shown here:
http://zx80.netai.net/grant/6502/Simple6502.html
https://web.archive.org/web/20130816003149/http://searle.hostei.com/grant/6502/Simple6502.html



This creates the following memory map:
0000-7FFF 32K RAM
8000-9FFF FREE SPACE (8K)
A000-BFFF SERIAL INTERFACE (minimally decoded)
C000-FFFF 16K ROM (BASIC from C000 TO DED3, serial routines FF00 to FFFF, so a large amount of free space suitable for a monitor etc)

As you can see this memory map has the RAM at the beginning of the map, the IO in the middle, and the ROM placed at the end. This is fairly typical of computers built around the 80s. We typically place the ROM at the end of the map because that's the address that the reset

vector will try to access. Having the RAM at the beginning gives us options of having the zero page stored in RAM.

This address decoding is superior to the one presented with the 738 decoder because it's actually using the rw and phase 2 lines to correctly assign the read and write cycles for both the ROM and RAM. Doing the address decoding this way should make the system more stable as well as provide a better framework for future IO expansion. This is explained a little better here:
http://forum.6502.org/viewtopic.php?f=12&t=3214
http://forum.6502.org/viewtopic.php?t=168
http://forum.6502.org/viewtopic.php?t=511

For another address decoding example you can check out:
http://www.6502.org/users/garth/projects.php?project=1&schematic=2
http://wilsonminesco.com/6502primer/addr_decoding.html

In future revisions I may consider swapping out the 74ls00 with something that can accommodate higher speeds, like maybe something from the 74hc or even better 74ac family(http://www.digikey.com/product-search/en?mpart=CD74AC00E&vendor=296).

With this setup our reset vector is at 7FFC/7FFD and contains address C000. Since our ROM only sees the lower 15 lines, this address gets translated to 4000 on the ROM, so that's where we place our code. You can use the same program as above to test the ROM again, only replace the jump to 8000 with C000. ie

```
00004000  a9 12 4c 00 c0 00 00 00  00 00 00 00 00 00 00 00
```

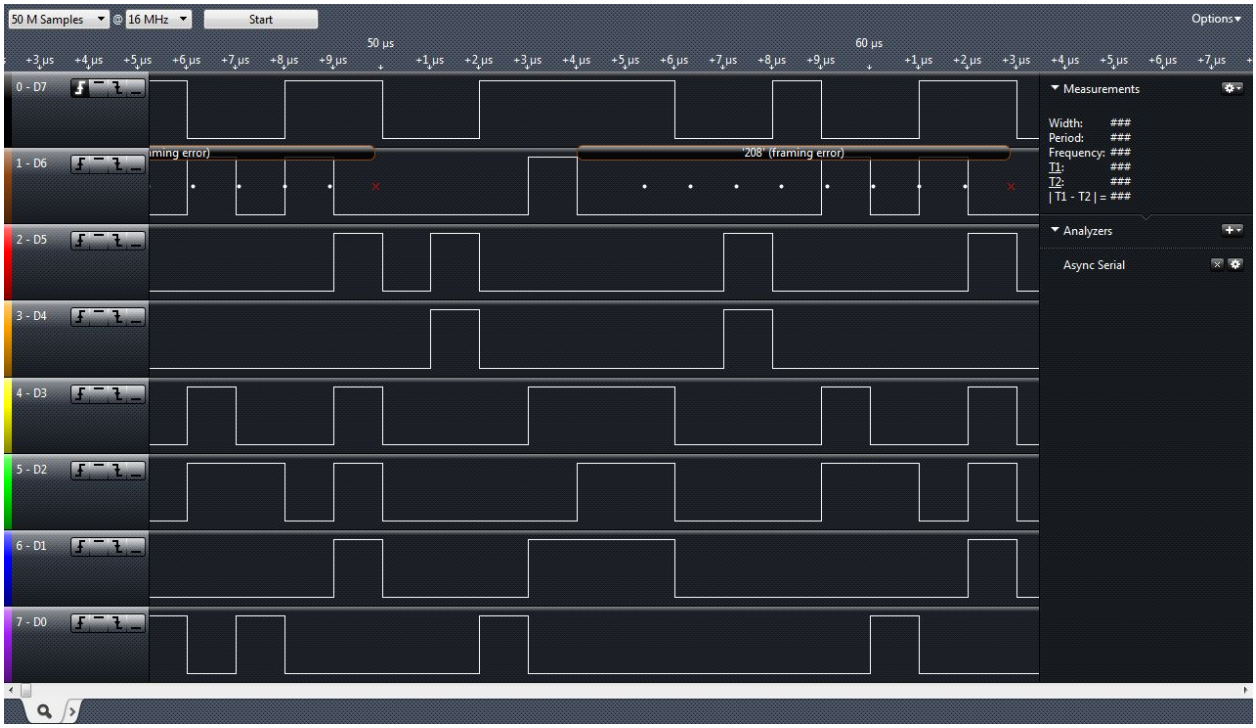
I made this little program to test the RAM:

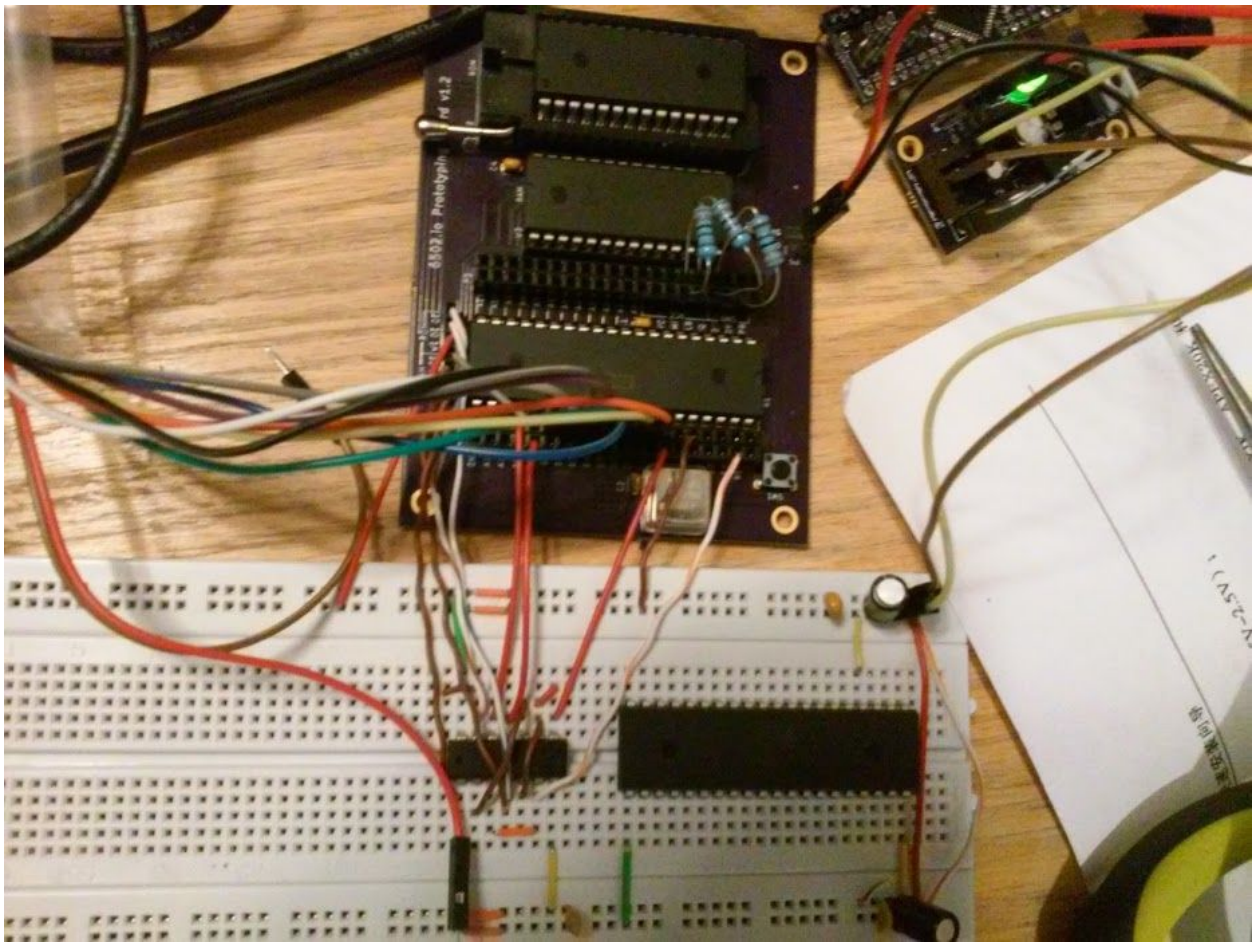
```
LDX #$FF
STX $3000     ;$3000 = #$FF

main: LDX $3000
DEX
STX $3000
JMP main
```

essentially this stores FF at RAM location 3000 and then decrements it and stores the value in back in the x register. After assembling this code I made sure the address for main after jmp was pointing to C005 which is the address where the instruction LDX $3000 begins.

```
00004000  a2 ff 8e 00 30 ae 00 30  ca 8e 00 30 4c 05 c0 00
```

then all that was left to do was hook up the logic analyzer to the data bus and see if the decrement was happening, which it was! yay!



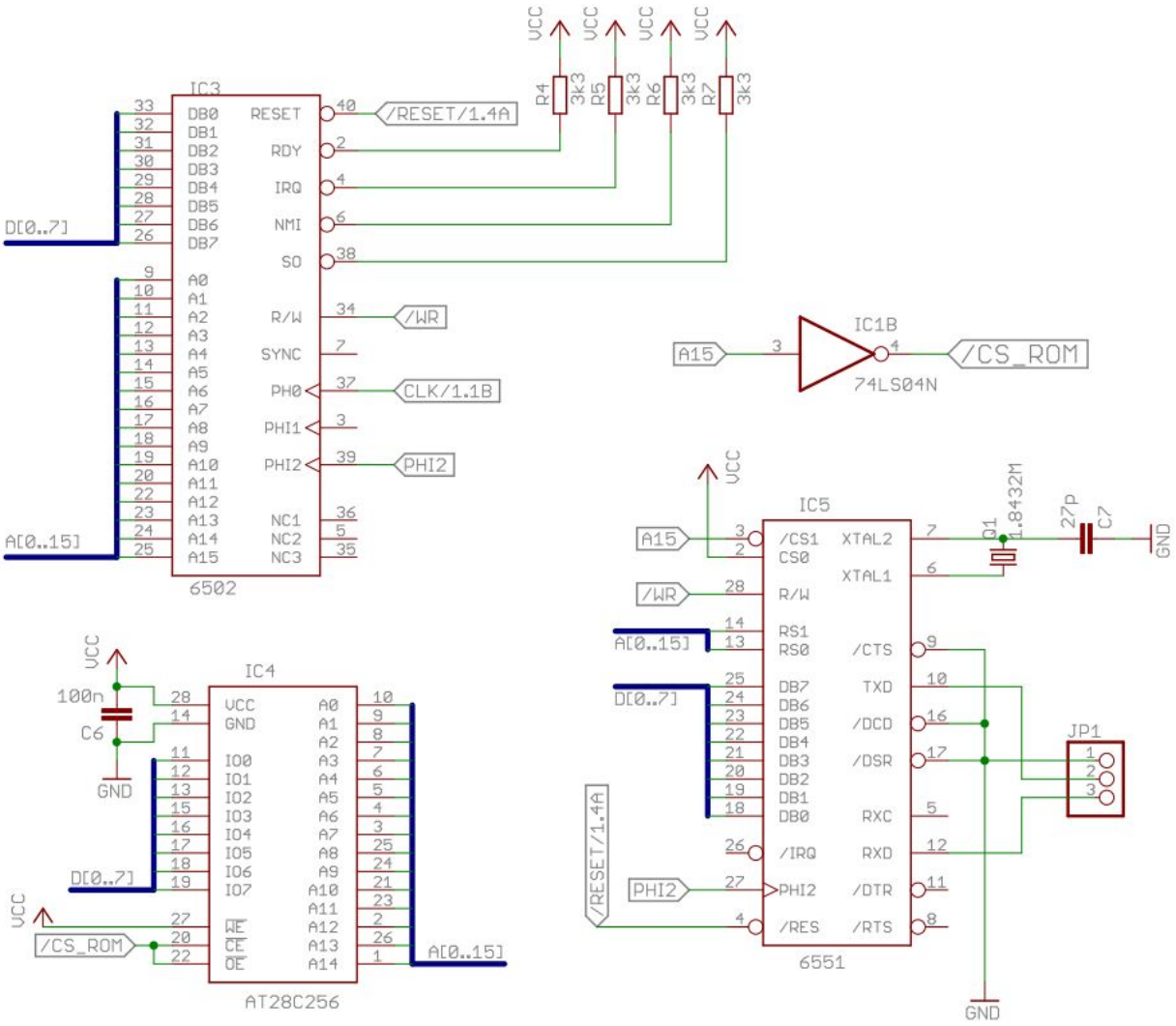


Connecting an ACIA (MC6850)

A good rundown of how the 6850 is setup is found here: http://alanclements.org/serialio.html
datasheet for MC6850: http://dev-docs.atariforge.org/files/MC6850.pdf
https://books.google.com/books?id=YMyjBQAAQBAJ&pg=PA157&lpg=PA157&dq=acia+6850+example&source=bl&ots=LQs-rqLlVx&sig=wY1piDFI7EmCYBK8JsvLpax8dBI&hl=en&sa=X&ved=0ahUKEwjX-Mi50vnJAhUK7mMKHWdACEI4ChDoAQgoMAI#v=onepage&q=acia%206850%20example&f=false (some example 6502 code to interface with the 6850)

Connecting an ACIA (65c51)

http://forum.6502.org/viewtopic.php?f=4&t=2543&start=30 (shows workaround for errata wdc version)
connect 65c51 as described here (ignore address decoding logic though):
http://www.grappendorf.net/projects/6502-home-computer/acia-serial-interface-hello-world



Instead of the 1.8432 crystal I used a can oscillator attached to xtl1 only. A0 is hooked to rs0, A1 is hooked to rs1. cs1 is hooked to A14, and A15 is hooked to CS0. This way when the cpu accesses A000 the binary looks like 1010 for A, A15 is high and A14 is low, thereby selecting the chip.

NOTE: I discovered some problems with the above address decoding. The problem is that the address space for the acia was conflicting with the rest of the IO address space. So for instance, if I had the acia initialized, and tried to set a register on one of the VIAs, the acia would lose connection. To solve this issue I had to connect the acia cs to the 74ls138 demultiplexer.

In programming the 65c51 put out by wdc, I discovered that there's a flaw in the transmit function of the chip. This is explained better here, as well as a workaround: http://forum.6502.org/viewtopic.php?f=4&t=2543&start=30

Here’s the finished code which writes Hello World out on the serial line @ 19200 baud.

```
                                .setcpu "65C02"

                                ACIA_DATA = $A000
                                ACIA_STATUS = $A001
                                ACIA_COMMAND = $A002
                                ACIA_CONTROL = $A003

                                ;.segment "VECTORS"

                                ;.word   nmi
                                ;.word   reset
                                ;.word   irq

                                .code

;reset:                 jmp main

;nmi:                           rti

;irq:                           rti

main:
init_acia:              lda #%00001011                  ;No parity, no echo,
no interrupt
                                sta ACIA_COMMAND
                                lda #%00011111                  ;1 stop bit, 8
data bits, 19200 baud
                                sta ACIA_CONTROL

write:                  ldx #0
next_char:
wait_txd_empty:         lda ACIA_STATUS
                                ;and #$10
                                ;beq wait_txd_empty
                                lda text,x
                                beq read
                                sta ACIA_DATA
                                inx
                        jsr DELAY_6551
                                jmp next_char
```

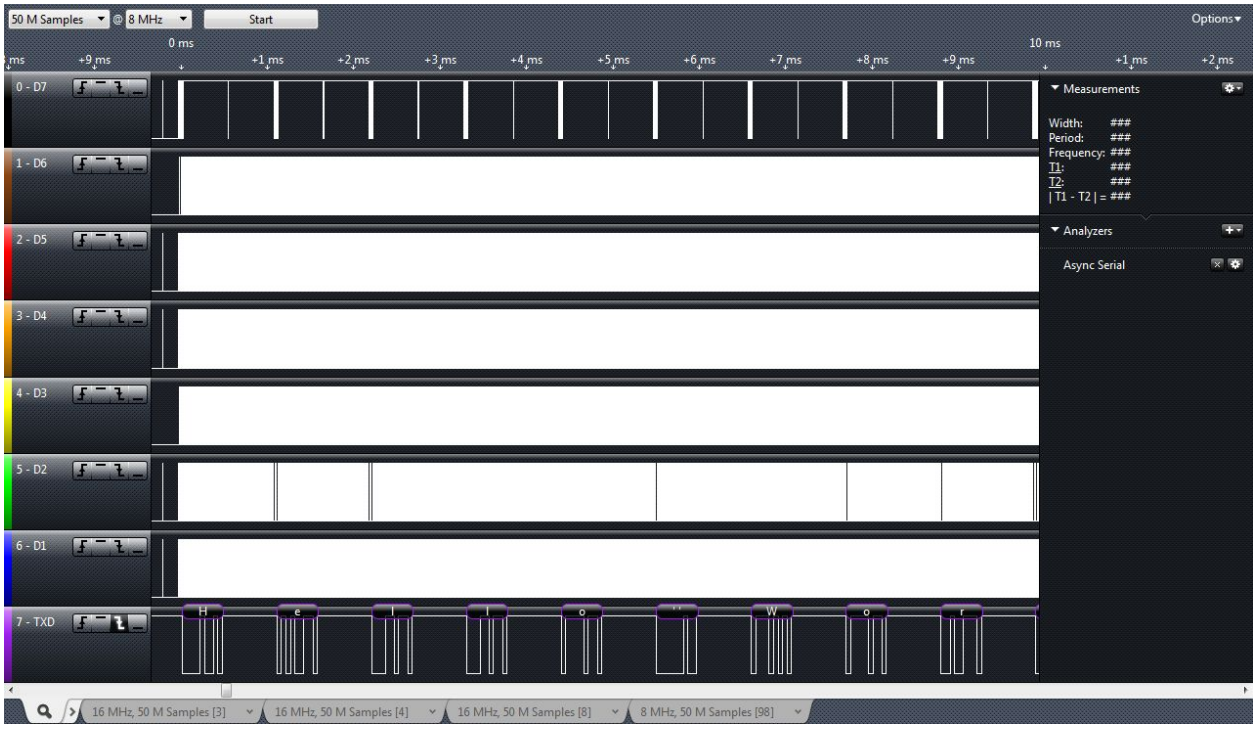
```
read:
wait_rxd_full:          lda ACIA_STATUS
                        and #$08
                        beq wait_rxd_full
                        lda ACIA_DATA
                        jmp write

text:                   .byte "Hello World!", $0d, $0a, $00

DELAY_6551:   PHY      ;Save Y Reg
        PHX     ;Save X Reg
DELAY_LOOP:   LDY   #2    ;Get delay value (clock rate in MHz 2 clock cycles)
;
MINIDLY:   LDX   #$68     ;Seed X reg
DELAY_1:     DEX        ;Decrement low index
        BNE   DELAY_1   ;Loop back until done
;
        DEY        ;Decrease by one
        BNE   MINIDLY   ;Loop until done
        PLX        ;Restore X Reg
        PLY        ;Restore Y Reg
DELAY_DONE:   RTS
```

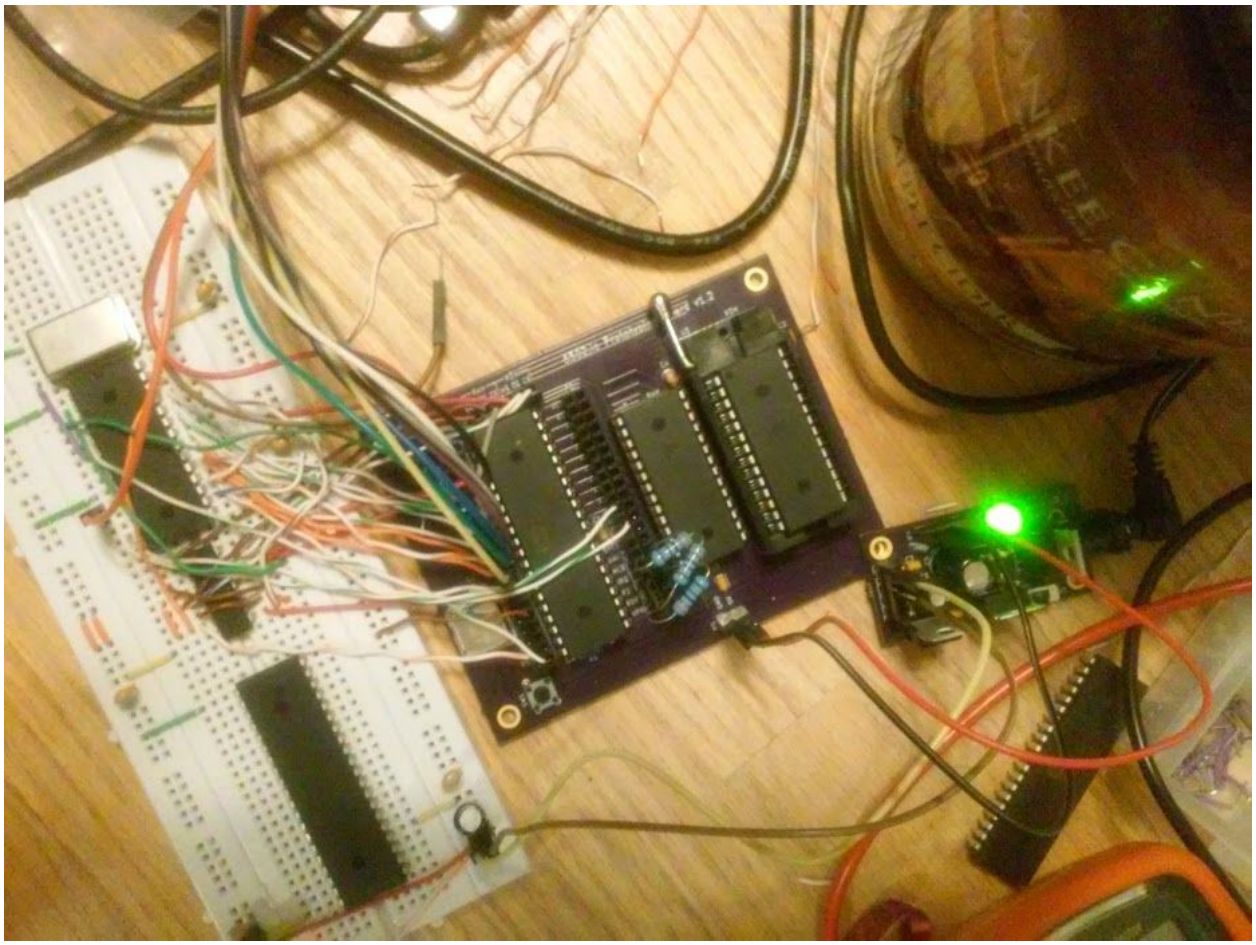
And here's the compiled binary

```
004000:  A9 0B 8D 02 A0 A9 1F 8D 03 A0 A2 00 AD 01 A0 BD  ................
004010:  2B C0 F0 0A 8D 00 A0 E8 20 3A C0 4C 0C C0 AD 01  +....... :.L....
004020:  A0 29 08 F0 F9 AD 00 A0 4C 0A C0 48 65 6C 6C 6F  .)......L..Hello
004030:  20 57 6F 72 6C 64 21 0D 0A 00 5A DA A0 02 A2 68   World!...Z....h
004040:  CA D0 FD 88 D0 F8 FA 7A 60 00 00 00 00 00 00 00  .......z`.......
```

And the reset vector:

```
007FD0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
007FE0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
007FF0:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 C0 00  ................
```


50 M Samples
8 MHz
Start
Options
0 ms
10 ms
+9 ms
+1 ms
+2 ms
+3 ms
+4 ms
+5 ms
+6 ms
+7 ms
+8 ms
+9 ms
+1 ms
+2 ms
0 - D7
1 - D6
2 - D5
3 - D4
4 - D3
5 - D2
6 - D1
7 - TXD
H
e
l
l
o
W
o
r
Measurements
Width: ###
Period: ###
Frequency: ###
T1: ###
T2: ###
| T1 - T2 | = ###
Analyzers
Async Serial
16 MHz, 50 M Samples [3]
16 MHz, 50 M Samples [4]
16 MHz, 50 M Samples [8]
8 MHz, 50 M Samples [98]



Todo: should I wire acias rtsb to its own ctsb? Also need to probably have parity checking etc.

IO Expansion

might use something like these 3-8 decoders:
http://www.digikey.com/product-search/en?mpart=74AC11138N&vendor=296 (faster but more expensive.)
http://www.digikey.com/product-search/en?mpart=CD74AC138E&vendor=296 (slower but cheaper.)

I'm thinking of shifting around the memory map a bit possibly to expand my rom a bit and still provide plenty of IO, see post here: http://forum.6502.org/viewtopic.php?f=4&t=3211&start=0

I may try to get a faster eeprom:
http://www.digikey.com/product-search/en?pv143=17&FV=fff40027%2Cfff80434%2C23c0005%2C23c0011%2Cf040002&mnonly=0&newproducts=0&ColumnSort=0&page=1&stock=1&quantity=0&ptm=0&fid=0&pageSize=25
http://www.digikey.com/product-detail/en/AT28HC64B-70JU/AT28HC64B-70JU-ND/1914233

To prototype this section of the computer I'm sticking with the sn74ls138 3 to 8 demultiplexer, which is a pin compatible although slower version of the CD74AC138E chip. My current memory map includes approximately 8k of free space between $8000-$9FFF, which will encompass my IO ports. Using the sn74ls138 allows me to add up to 8 IO devices with just a single chip. To get a better understanding of where the sn74ls138 will go and how it will divide up the memory for our IO devices, let's take a look at our current memory map:

```
RAM START ($0000)        = 0000 0000 0000 0000
RAM END ($7FFF)          = 0111 1111 1111 1111
FREE SPACE START ($8000) = 1000 0000 0000 0000
FREE SPACE END ($9FFF)   = 1001 1111 1111 1111
SERIAL START ($A000)     = 1010 0000 0000 0000
SERIAL END ($BFFF)       = 1011 1111 1111 1111
ROM START ($C000)        = 1100 0000 0000 0000
ROM END ($FFFF)          = 1111 1111 1111 1111
```

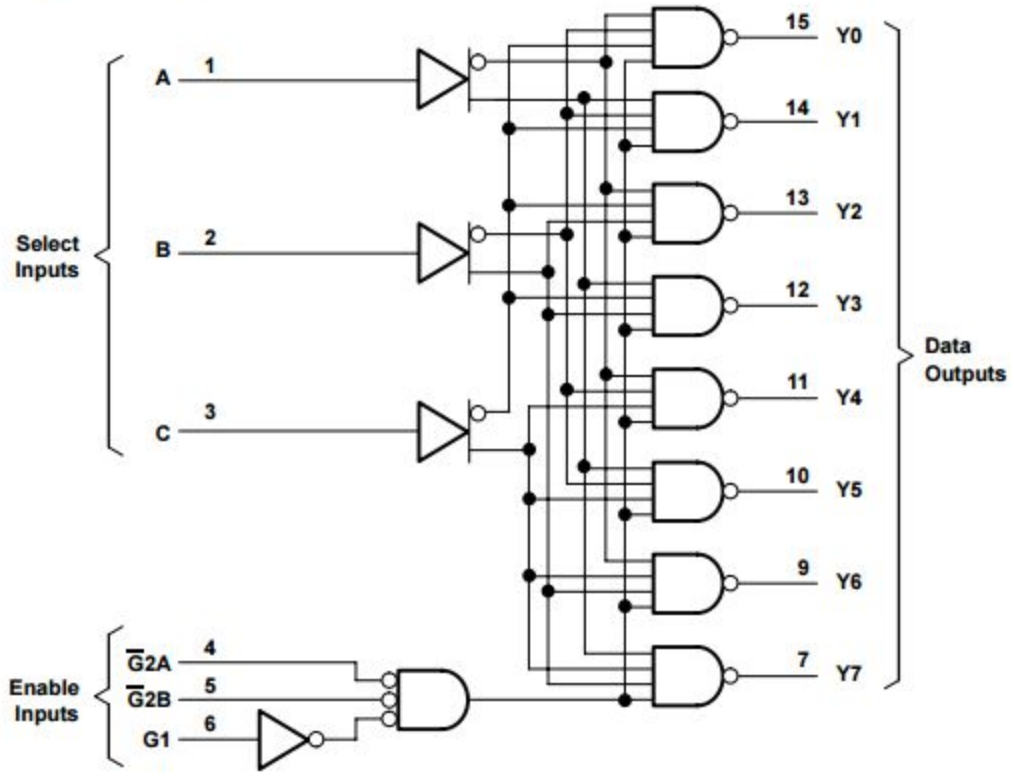
To begin thinking about how we're going to provide address decoding to our free space ($8000-$9FFF) we need to take a look at how the address lines that make up that space are unique. Typically you want to hook up your address decoding to the last 3 or 4 address lines on the bus, so A15-A12, so let's look specifically at those values. To do this we'll look at the address space

immediately preceding the free space, as well as the one immediately following it.

	A15	A14	A13	A12
RAM END ($7FFF)	0	1	1	1
FREE START ($8000)	1	0	0	0
FREE END ($9FFF)	1	0	0	1
SERIAL START ($A000)	1	0	1	0

Now, looking at these address lines, what can we see that would differentiate our free space that we want to decode versus the space surrounding it? First off we notice that A15 in our free space always is a 1, whereas our RAM section holds a 0 at A15. This isn't enough to completely differentiate free space from our serial start address though, since serial also has a 1 in A15. Next we can see that A13 for free space is always 0, whereas the other 2 sections are 1. In addition, A14 in free space is always 0. Using this information we can see that in order to decode our free space memory, we need our address decoder to check that A15 is 1, as well as A14 and A13 are 0. So, how do we accomplish this? Checking the datasheet for the SN74LS138, we can see that there are 3 enable pins which can create the glue logic we need.

logic diagram (positive logic)



FUNCTION TABLE

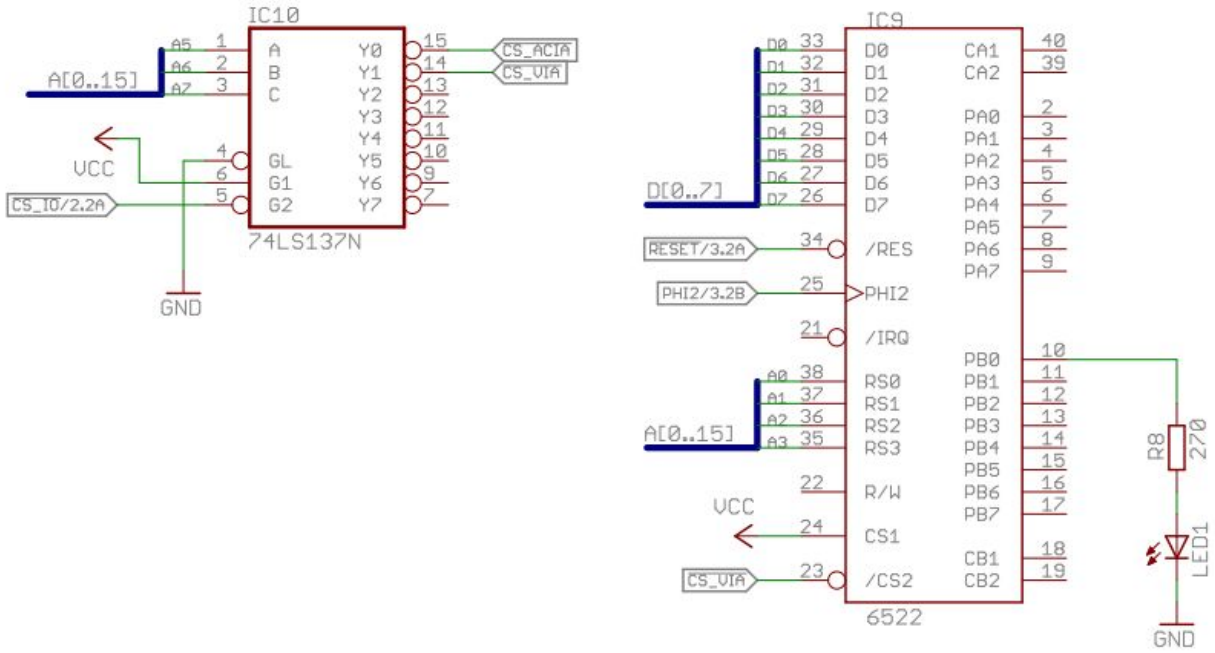
ENABLE INPUTS			SELECT INPUTS			OUTPUTS							
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L

As you can see, G1, G2A, and G2B are used to enable the output of the SN74LS138. We can use these pins to come up with the logic needed to decode our free space. First off the outputs are only enabled when G1 is high (logic 1), so we can tie A15 to G1 since we know that needs to always be a 1 for our free space section. Now we just need to ensure that A14 and A13 are both 0. We can see from the function table above that G2A and G2B need to both be low in order for the outputs to be enabled. This works out perfectly for us if we tie A14 to G2A and A13 to G2B, since both those address lines will be low in our address map. Using this address decoding method, we now have everything we need to decode our free space.

Now that we have our enable pins figured out on our SN74LS138, we need to figure out where to hook up the A,B,C inputs in order to enable the specific Y0-Y7 chip select pin that we need. From the function table above we can see that the A,B, and C pins determine which one of the 8 Y output pins will be set low. By connecting A8 to A, A9 to B, and A10 to C, we can come up with the following IO Decoding table:

		C	B	A
		A10	A9	A8
VIA1	IO1($8000)	0	0	0
VIA2	IO2($8100)	0	0	1
SID	IO3($8200)	0	1	0
	IO4($8300)	0	1	1
	IO5($8400)	1	0	0
	IO6($8500)	1	0	1
	IO7($8600)	1	1	0
	IO8($8700)	1	1	1

With this knowledge we are ready to hook up our vias. For both vias, connect them as shown here:



For CS2, run them to the appropriate Y connector of the SN74LS138 (Y0 to Via1 and Y1 to Via2).
On the final board I want to add jumpers from the irq pin of the 6522 to choose to connect the pin to either the nmi or irq pin of the 6502. Note, that I should be able to chain these irq lines together, as long as I choose the right kind of 65c52 (I think it's 65c52s …?).

Next I wrote this little program to test the 2 vias. It should cause PB0(pin10) and PB1(pin11) to blink.

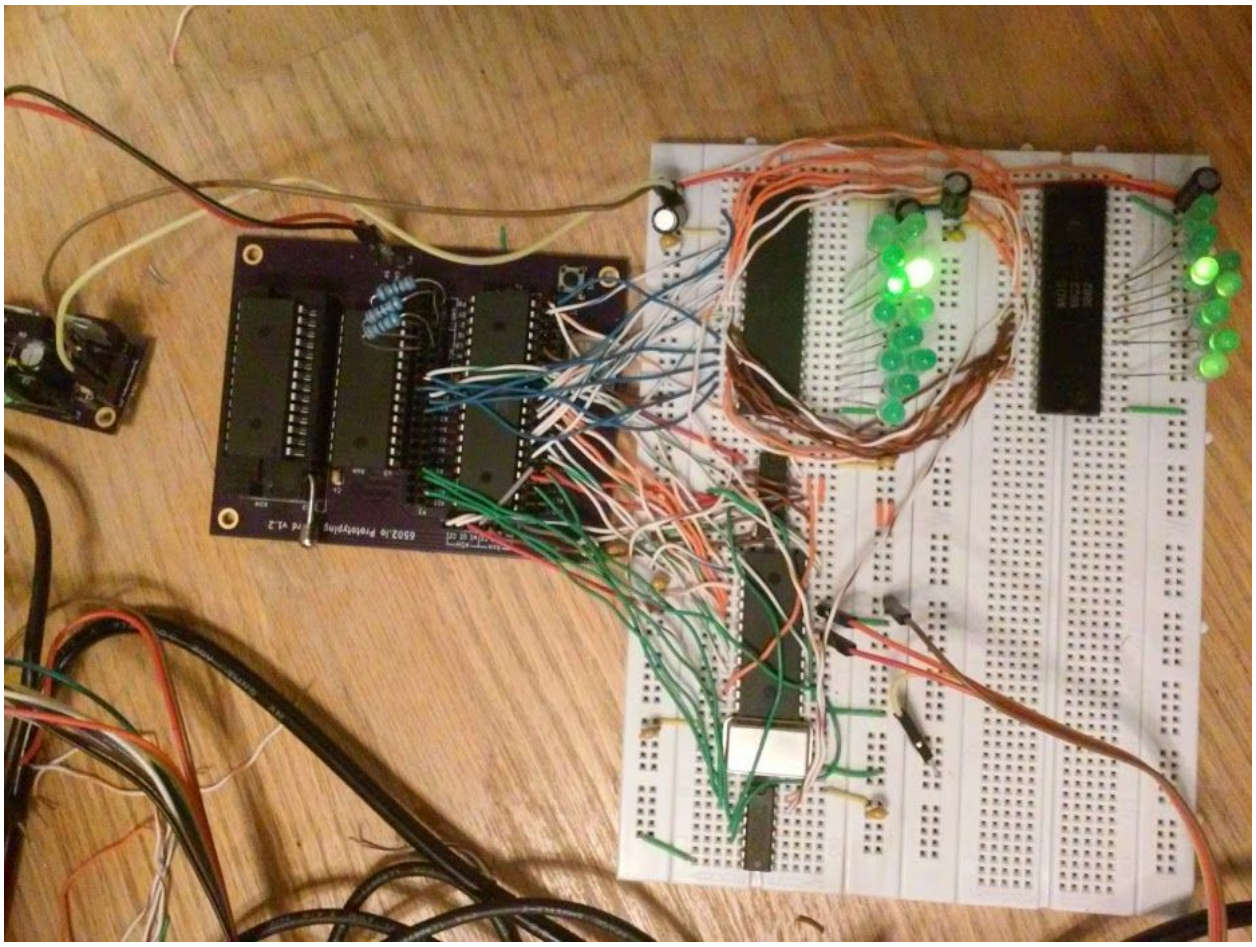
```
.setcpu "65C02"

VIA1_DDRB = $8022
VIA1_ORB  = $8020
VIA1_DDRA = $8023
VIA1_ORA  = $8021
VIA2_DDRB = $8122
VIA2_ORB  = $8120
VIA2_DDRA = $8123
VIA2_ORA  = $8121


.segment "VECTORS"

.word   nmi
.word   reset
```

```
        .word   irq

        .code

reset:    jmp main

nmi:      rti

irq:      rti

main:     lda #$ff
        sta VIA1_DDRB
        sta VIA2_DDRB

loop:     lda #$01
        sta VIA1_ORB
        sta VIA2_ORB
        jsr delay
        lda #$02
        sta VIA1_ORB
        sta VIA2_ORB
        jsr delay
        jmp loop

delay:    ldx #200
@delay2:  ldy #0
@delay1:  dey
        bne @delay1
        dex
        bne @delay2
        rts
```

And here's the config file:

```
#check out http://www.cc65.org/doc/ld65-5.html for info on this config

MEMORY
{
  ROM: start=$8000, size=$8000, type=ro, define=yes, fill=yes, file=%O;
}

SEGMENTS
```

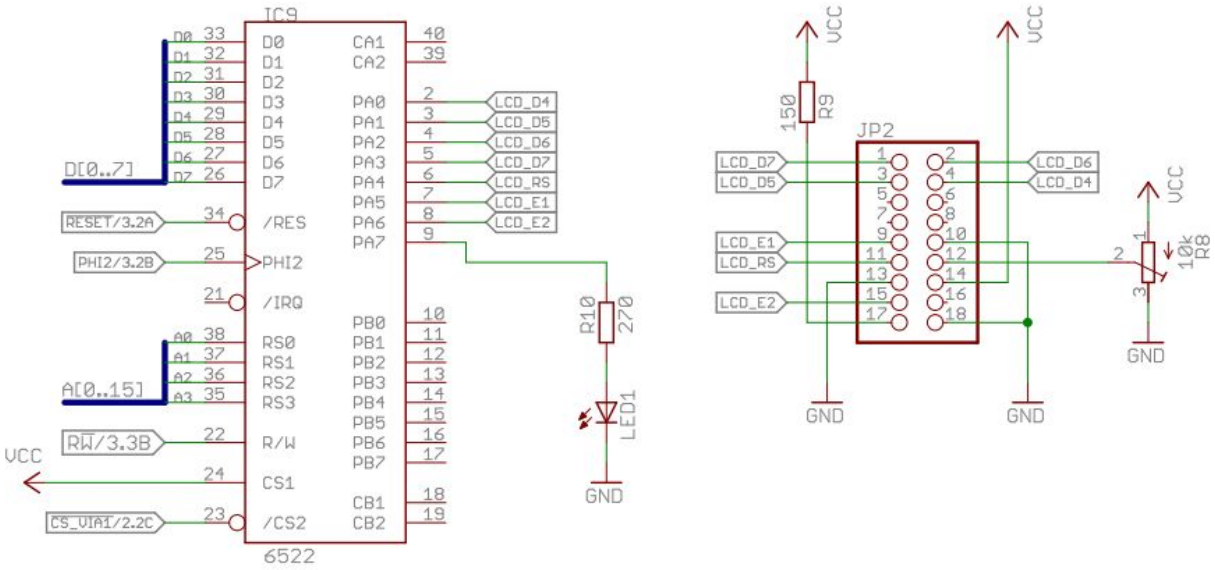
```
{
  CODE: load=ROM, type=ro, offset=$4000;
  VECTORS: load=ROM, type=ro, offset=$7FFA;
}
```



Adding an LCD

Now that we have the IO figured out and the 2 VIAs connected, we can move on to adding an lcd!
I chose the HD44780 lcd chipset, since I already had a couple lying around. The one I hooked up was from an old 90s printer I salvaged! These lcds are also fairly common today. I included links to datasheets and other sites that helped me interface with it in the resources section.

To start out with I connected VIA2 to the lcds data, register select, and enable lines as shown here:



Note that in my case I didn't need to connect an E2(Enable 2) pin, since my lcd only has 2 lines, and therefore only had one enable pin. So just ignore the connection going from PA6(pin 8) of the via.

Here's a little program I wrote to show how to initialize the lcd and write out "Hello World" on the screen:

```
                .setcpu "65c02"
                .include "macros.inc65"

                .include "io.inc65"


                LCD_D4  = VIA_PA0
                LCD_D5  = VIA_PA1
                LCD_D6  = VIA_PA2
                LCD_D7  = VIA_PA3
                LCD_RS  = VIA_PA4
                LCD_EN1 = VIA_PA5
                LCD_EN2 = VIA_PA6




enable_bits:    .byte 0

                .segment "VECTORS"
```

```
          .word   nmi
          .word   reset
          .word   irq

          .code

reset:                    jmp main

nmi:                            rti

irq:                            rti

main:

                     jmp lcd_init2


loop:

                jmp loop
lcd_init2:
                lda #$ff
                sta VIA2_ORA
                sta VIA2_DDRA
                ldx #50
                jsr delay_ms

                ;set register select bit to 0 for command/control

                ;start init sequence
                lda #$03
                jsr strobe_enable
                ldx #5
                jsr delay_ms

                lda #$03
                jsr strobe_enable
                ldx #5
                jsr delay_ms

                lda #$03
                jsr strobe_enable
                ldx #5
                jsr delay_ms

                lda #$02
                jsr strobe_enable
```

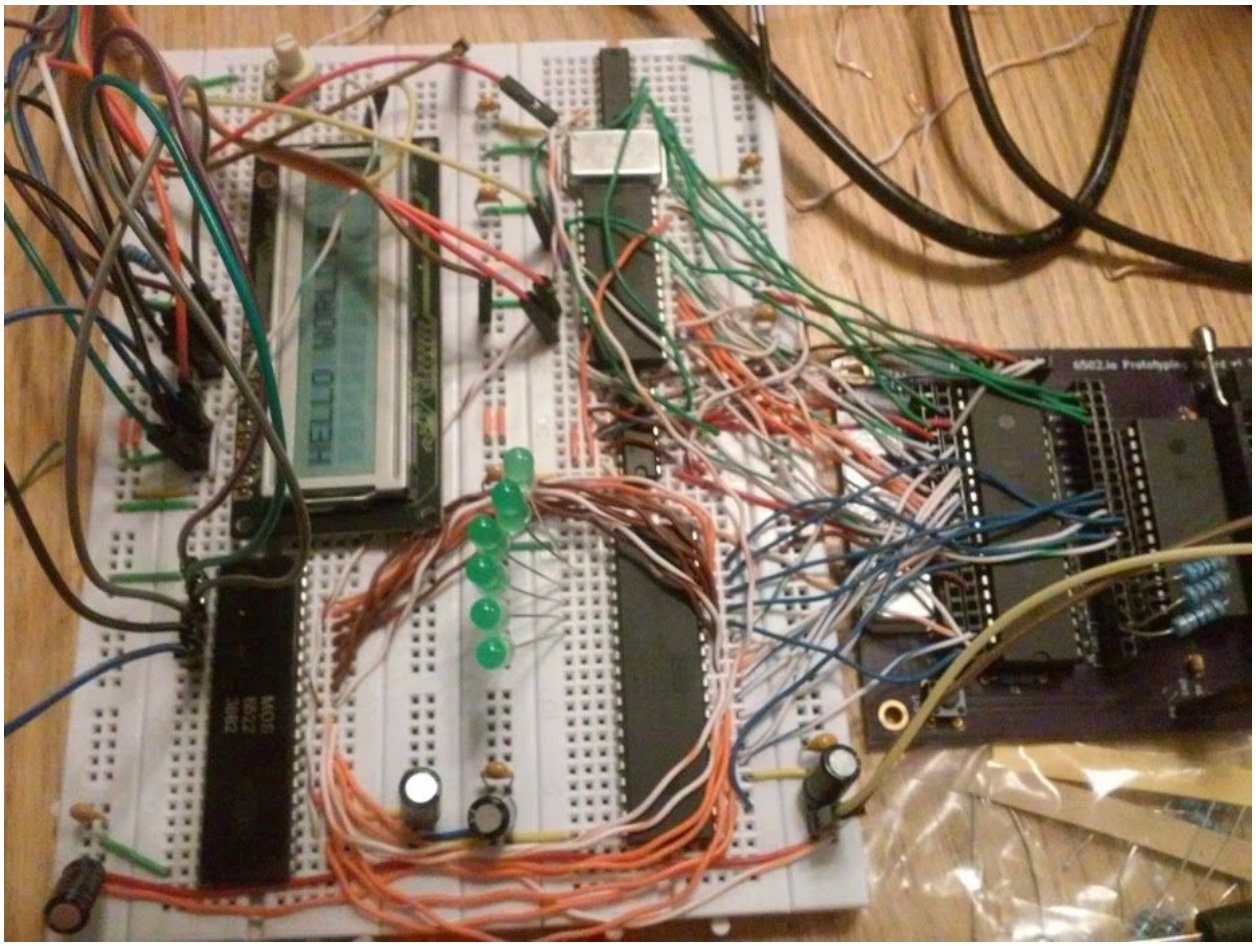
```
ldx #3
jsr delay_ms

;now in 4 bit mode

;function set
lda #$02 ;set data length to 4 bits
jsr strobe_enable


;no delay needed here


lda #$08 ;2 lines 5x8 pixels
jsr strobe_enable
ldx #3
jsr delay_ms

lda #$00 ;turn display off
jsr strobe_enable
lda #$08
jsr strobe_enable
ldx #3
jsr delay_ms

lda #$00 ;clear display
jsr strobe_enable
lda #$01
jsr strobe_enable
ldx #3
jsr delay_ms

lda #$00 ;entry mode set
jsr strobe_enable
lda #$06
jsr strobe_enable
ldx #3
jsr delay_ms
;end of initialization

lda #$00 ;turn display on
jsr strobe_enable
lda #$0f ;display on, cursor on, blink on
jsr strobe_enable
ldx #3
jsr delay_ms

lda #$00 ;CURSOR HOME
```

```
                jsr strobe_enable
                lda #$02 ;cursor home
                jsr strobe_enable
                ldx #3
                jsr delay_ms

                jmp write_hello_world

write_hello_world:
                 ;send H
                lda #$14
                jsr strobe_enable
                ldx #5
                jsr delay_ms

                lda #$18
                jsr strobe_enable
                ldx #5
                jsr delay_ms

                ;send E
                lda #$14
                jsr strobe_enable
                ldx #5
                jsr delay_ms

                lda #$15
                jsr strobe_enable
                ldx #5
                jsr delay_ms

                ;add in extra strobe for reading 8 status bits
                lda #00
                jsr strobe_enable
                lda #00
                jsr strobe_enable

                ;send L
                lda #$14
                jsr strobe_enable
                ldx #5
                jsr delay_ms

                lda #$1C
                jsr strobe_enable
                ldx #5
                jsr delay_ms
```

```
;send L
lda #$14
jsr strobe_enable
ldx #5
jsr delay_ms

lda #$1C
jsr strobe_enable
ldx #5
jsr delay_ms

;send O
lda #$14
jsr strobe_enable
ldx #5
jsr delay_ms

lda #$1F
jsr strobe_enable
ldx #5
jsr delay_ms

;send space
lda #$12
jsr strobe_enable
ldx #5
jsr delay_ms

lda #$10
jsr strobe_enable
ldx #5
jsr delay_ms

;send W
lda #$15
jsr strobe_enable
ldx #5
jsr delay_ms

lda #$17
jsr strobe_enable
ldx #5
jsr delay_ms

;send O
lda #$14
jsr strobe_enable
ldx #5
```

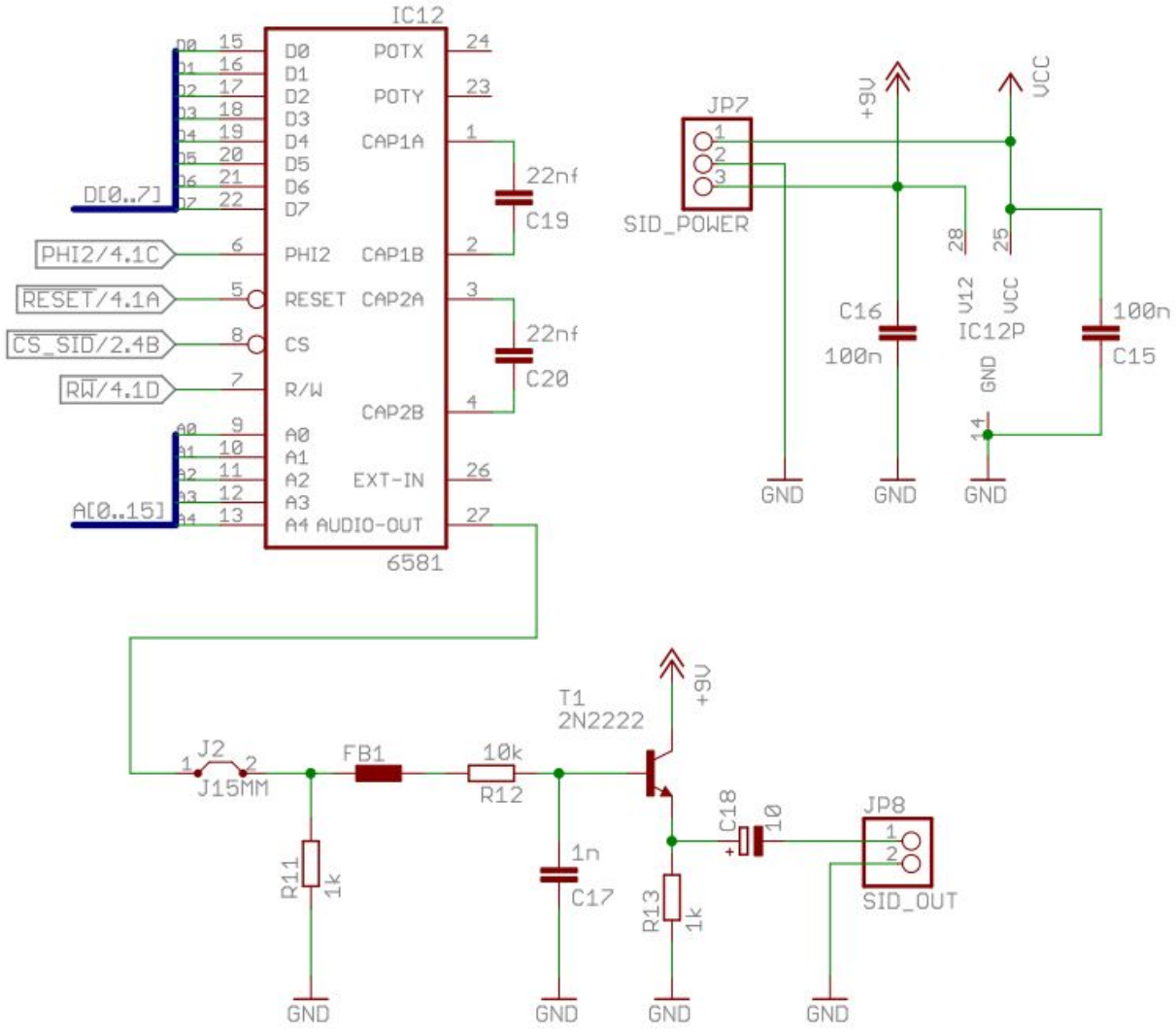
```
jsr delay_ms

lda #$1F
jsr strobe_enable
ldx #5
jsr delay_ms

;send R
lda #$15
jsr strobe_enable
ldx #5
jsr delay_ms

lda #$12
jsr strobe_enable
ldx #5
jsr delay_ms

;send L
lda #$14
jsr strobe_enable
ldx #5
jsr delay_ms

lda #$1C
jsr strobe_enable
ldx #5
jsr delay_ms

;send D
lda #$14
jsr strobe_enable
ldx #5
jsr delay_ms

lda #$14
jsr strobe_enable
ldx #5
jsr delay_ms

;send !
lda #$12
jsr strobe_enable
ldx #5
jsr delay_ms

lda #$11
jsr strobe_enable
```

```
                ldx #5
                jsr delay_ms

                jmp loop

strobe_enable:
                ora #$20        ;add enable bit
                sta VIA2_ORA    ;send value out to via
                ldx #10
                jsr delay_ms
                and #$DF        ;mask to take out enable
                sta VIA2_ORA    ;take out enable bit and send to via

                rts

; Delay the number of miliseconds specified by X
; This is hardcoded for a 1 MHz system clock
delay_ms:   pha       ; 3
            txa       ; 2
            pha       ; 3
            tya       ; 2
            pha       ; 3

            ldy $00     ; 3 (dummy operation)
            ldy #190    ; 2
@loop1:     dey       ; 190 * 2
            bne @loop1    ; 190 * 3 - 1

@loop2:     dex       ; 2
            beq @return   ; (x - 1) * 2 + 3

            nop       ; 2
            ldy #198    ; 2
@loop3:     dey       ; 198 * 2
            bne @loop3    ; 198 * 3 - 1

            jmp @loop2    ; 3

@return:    pla       ; 4
            tay       ; 2
            pla       ; 4
            tax       ; 2
            pla       ; 4
            rts       ; 6 (+ 6 for JSR)
```



resources:
https://www.sparkfun.com/datasheets/LCD/HD44780.pdf
http://web.stanford.edu/class/ee281/handouts/lcd_tutorial.pdf
https://www.arduino.cc/en/Tutorial/HelloWorld
http://www.bipom.com/documents/appnotes/LCD%20Interfacing%20using%20HD44780%20Hitachi%20chipset%20compatible%20LCD.pdf
http://web.alfredstate.edu/weimandn/lcd/lcd_initialization/lcd_initialization_index.html

Adding SID (sound interface device) Chip



For the audio out circuitry I stuck a 1k resistor from the 6581 pin 27 to ground and then ran the signal over to an amplifier built from an lm386 circuit found here: https://www.youtube.com/watch?v=PPQ87kjEgBA . I was thinking on the eventual board I could stick a switch between the gain boost 10uf cap and the resistor between pins 1 and 8 of the lm386. This would allow me to turn the gain on for driving large speakers, but then turn it off when listening to headphones etc. Instead of an lm386, since those are now obsolete, I could use this chip: http://www.digikey.com/product-detail/en/NJM386D/NJM386D-ND/805736, which is a drop in replacement for the lm386.

resources:
http://www.grappendorf.net/projects/6502-home-computer/sid-sound.html
http://www.acsu.buffalo.edu/~robertsz/projects/SID/index.html

http://archive.6502.org/datasheets/mos_6581_sid.pdf
http://www.deblauweschicht.nl/tinkering/mos6581_1.html
http://www.sidmusic.org/sid/sidtech3.html (For best results, the ground line between SID and the power supply should be separate from ground lines to other digital circuitry. This will minimize digital noise at the audio output.)

On another forum I read that you should ground the ext-in pin of the sid if it was not being used. This should prevent extra noise from entering the system.

http://csdb.dk/forums/index.php?roomid=7&topicid=36766&showallposts=1

This method is pretty effective atleast on 6581 chips, it insulates 6581 from the data and address busses when CS (Chip select) line for sid is not active.

Combined with filtered and separate +12v and +5v voltages and own separate grounding it improves output quality alot. Also grounding potx, poty and ext-in lines with small (1nF) capasitor helps littlebit on the way. Also building separate audio out buffer (pre amp) and making own audio out connector helps on the way too.

http://www.lemon64.com/forum/viewtopic.php?t=47932&sid=5920081b88b25de7e4688e5584dcfafe

http://www.commodore64site.nl/schematics/251469-1of2.gif

https://www.c64-wiki.com/images/5/5b/PRG_Schematic_%28right%29.gif
(c64 schematic showing sid circuitry)

also 8580 sid chip (later version of 6581)
another possible sound chip: https://en.wikipedia.org/wiki/General_Instrument_AY-3-8910
AY-3-8910, AY-3-8912, YM2149
YM2203

Adding Compact Flash

resources:
https://www.sparkfun.com/datasheets/BreakoutBoards/c0201mspdf.pdf
http://read.pudn.com/downloads52/ebook/178846/S72032.pdf
http://forum.6502.org/viewtopic.php?f=4&t=2877
http://www.msarnoff.org/6809/
https://www.sparkfun.com/datasheets/BreakoutBoards/c0201mspdf.pdf

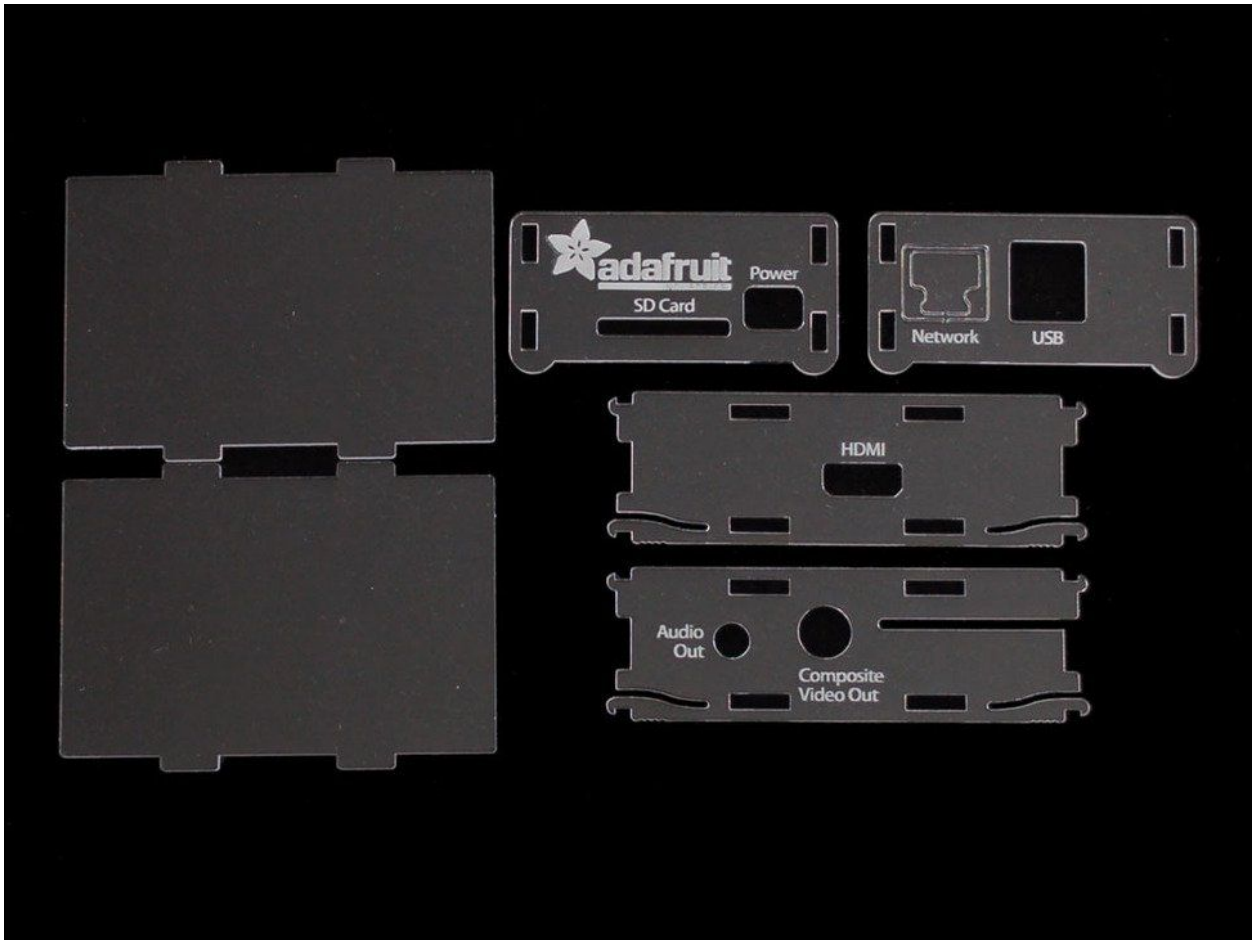
http://rumkin.com/reference/aquapad/media/cfspc3_0.pdf
http://sbc.rictor.org/io/IDE.html
http://dreher.net/?s=projects/CFforApple1&c=projects/CFforApple1/main.php
https://cowgod.org/replica1/applesoft/
http://www.microchip.com/forums/m364311.aspx
http://elm-chan.org/fsw/ff/00index_e.html

Adding a Keyboard

Should I use a decade counter?
https://mechanicalkeyboards.com/shop/index.php?l=product_list&c=40&show=100

Adding a Case
polycase.com
https://www.shapeways.com/model/upload-and-buy/4251306#materials
http://www.makexyz.com/help/faq
https://www.3dhubs.com/3dprint#?place=639%20Weaver%20Hill%20Ln,%20Colfax,%20CA%2095713,%20USA&latitude=39.012399599999995&longitude=-120.97612729999997&distanceUnit=miles
http://www.ponoko.com/home
http://lasergist.com/
protocase.com
emachineshop.com
bigbluesaw.com
https://learn.adafruit.com/laser-cut-enclosure-design/overview
http://parts-badger.com/

something like this: https://www.adafruit.com/products/859



I'm thinking of naming it STACHE I. And making the lettering in 8 bit with a 3d texture (like http://orig05.deviantart.net/ae1f/f/2013/265/1/8/8_bit_moustaches__the_wario_by_mattcantdraw-d6ncbh5.jpg).
Or another name might be RETRO I. or retro 1 (all lowercase like apple ii)

Adding BASIC
http://www.asciimation.co.nz/bb/2013/10/02/orwell-keyboard-input-and-basic

Adding real time clock
https://github.com/ytmytm/c64-ds12c887
https://courses.engr.illinois.edu/ece391/references/mc146818.pdf