ECEN 5613: Embedded System Design Lab 1

Trevor Sribar
https://canvas.colorado.edu/courses/130064/assignments/2624748

Table of Contents

Overhead

Understanding Lab 1

Lab 1 is split into 3 parts

Part 1 (due 1/16 2 PM):

- Basic assembly language program
- Understand of said program
- Practice using testing practices (setting of accumulator and registers)
- Practice making well-commented code
- Gain a strong understanding for the final project for this class
- Learn how to use Microchip (Atmel) WinCUPL
- Learn how to use WinSim
- Review Tutorials for the two above tools
- Review the example code for the two above tools
- Use WinCUPL to make basic code for the Atmel AT16V8C SPLD

Part 2 (due 1/23 2 PM):

- Buy a kit from a TA
- Verify kit has all parts & Acquire items for the semester
- Familiarize self with lab equipment locations
- Gain an understanding of soldering
- Learn how to treat electronics
- Gain an understanding of the schematics for our PCB
- Draw a schematic of your PCB
- Design the power, oscillator, and power on circuits

Part 3 (due 1/30 2 PM):

- Ensure PCB doesn't have manufacturing defects
- Plan future layout
- Label the PCB
- Solder wire wrap sockets to PCB
- Solder power jack to PCB
- Wire power jack and switch to PCB
- Solder RS-232 to PCB
- Create standard for color for wire wrapping
- Learn and use wire wrapping
- Verify integrity of solders and wraps
- Verify Power
- Create basic wiring for 8051
- Program 8051
- Verify 8051 integrity
- Verify VCC voltage & ripple
- Wire the '373 latch

- Test digital logic for AT16V8C
- Learn about O-Scopes
- Create steps to complete base board
- Learn about ARM/EFM8BB1 architecture

Gaining Lab Access

I already have lab access from being in the course as seen in Fig 0.1.

Fig 0.1: Access to Lab

Download Required Tools

The tools we will use are:
Keil uVision debugger for C51 or the SilabsEFM8BB1 dev board with Simplicity Studio
Canvas has the file location and demo videos of using them from the TAs

Part 1

Assembly Program

Requirements:

Implement the equation X*4 /Y = Z

- Do not use MUL or DIV
- Use shift(s) to implement X*4
- Program starts at 0x000h
- ENDLOOP exists and program always ends in it (infinte loop)

X, Y, and output Z are uint8s
X: Starts in the accumulator

- throws an error 0x02 in Error location if overflows after *2 and ENDS

Y: Starts in B register

- throws an error 0x01 in Error location if is 0 and ENDS

End conditions:
IRAM location 0x20 contains the value of (X)
IRAM location 0x21 contains the value of (X*4)
IRAM location 0x22 contains the value of (Y)
IRAM location 0x23 contains Z, the 8-bit quotient (in hexadecimal)
IRAM location 0x24 contains the 8-bit remainder (in hexadecimal)
IRAM location 0x30 contains the error code.

(IRAM means Internal Data Memory)

Questions

Does 0x20, 21, and 22 have how big of a size? A byte probably

Logic

Shift Left 2 in order to multiply X
See if there is an easy way to check 2 leftmost bits for being 1

- Shift left and check if non-zero
- And both bits and OR them
- Subtract after shift and see if is a smaller number?

Check if Y is zero
Then go through operations
Division

- Initialize 0x23 to 0

- Check if X-Y < 0 (we checked if X-Y causes carry to be set)
- If yes, set 0x24 to X & END (make sure to set 0x00 to error location)
- If no, let new X = X-Y and add 1 to 0x23, then repeat

Strange Parts of implementation

- We check for overflow in subtraction in order to find when to end the loop
    - This causes us to have no value to save to the remainder register
    - We have to add back Y to this overflowed W in order to get back the remainder

Test Program

setting the accumulator and B register to various initial values
Use combined code and data space in the 8051 (/overlay option if using Emily52)
How to set values in data space
Load into debugger => bits on left, double click to change

How I set values in code
Added a “Preconditions” Section, which can be seen in lab1Part1_1WithPre

Above and Beyond

Create the DIV and MULT version in a different location after our original run, then subtract the values and check if both are 0. If not, throw a new error.

Final Project

Requirements

8-9 minute demo (including a power point)
Establish goals

- Individual or Groups of 2
- Electronics & Firmware > Software, Mechanical, Packaging
- New hardware
- New firmware
- ~5 min PDR
    - Overview
    - Milestones
    - Deliverables
- No risk to students
- Each team member present for demos
    - This includes a presentation
    - Includes a detailed block diagram
    - Talk about design/implementation
    - Talk about struggles

    - Demon night class might (will) go late & start early
- A video of the working project
- All files are .pptx .mp4 or .pdf
- Final lab doc
- Honor code signed
- Files
    - All project files (code in source + pdf, header files, make files, ect., & other docs such as unique data sheets, and notes)
    - Organized fileset
- One copy of everything into canvas
- Report
    - Keep big stuff at end
    - Page numbers, cover page, ect.
    - Word editor
    - Don't include details about labs 1-4
    - Title (descriptive) names, ect.
    - Division of labor
- Don't copy onlines stuff
    - Or make sure it's legal & well documented before hand
- Code headers must have name, revision, date, important code

Timeline

Ideas by Late Feb
PDR Submission Sunday, March 8th
PDR Presentation Tuesday, March 10th
Final Demo by April 24th
Documentation by April 29th

Grading

38% of your grade
Must be complicated (no arduino)
Idk bro

Questions

Can I write my report in docs?
Where is the rubric?

Plans

Printer
Contactless doom (PS2 connectors)

Jane-street style algorithmic microtrades

- Trades are FPGAs
- Fall PLESD

References

Final project posted on Canvas: https://canvas.colorado.edu/…
Mp4s under canvas previous projects: (on canvas aswell)

SPLD

SPLD stands for
Download Atmel WinCUPL tools:
https://canvas.colorado.edu/courses/130064/pages/ecen-5613-reference-files-and-links

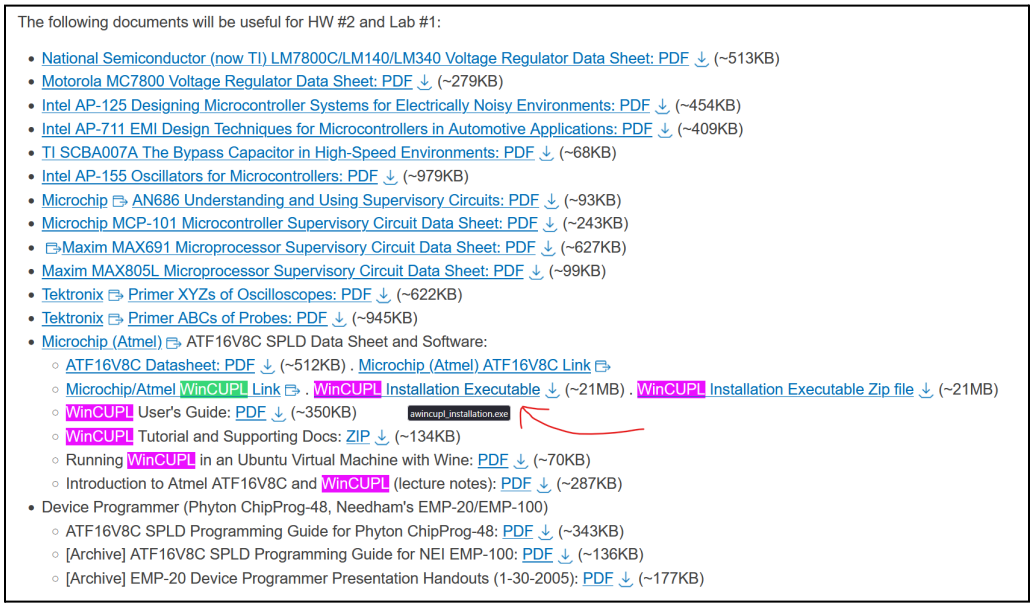


Fig 1.3.1: Where WinCUPL installation is on Canvas

Microchip (Atmel) WinCUPL

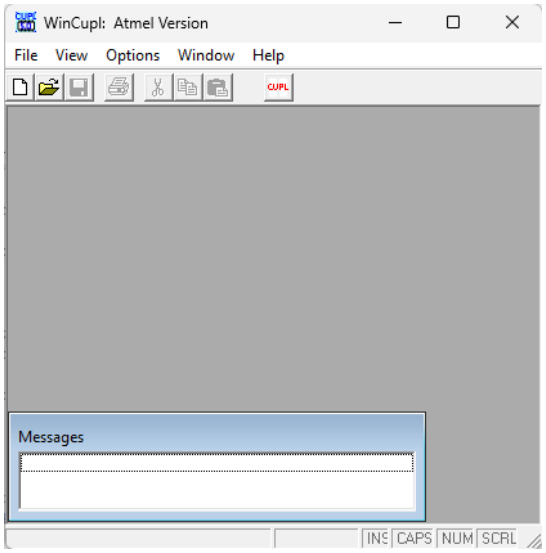


Fig 1.3.2: Initial view of WinCUPL

WinSim

File Structure (from the class handout found on canvas)

```
Name            Gates08;
Partno          ESD001;
Revision        01;
Date            1/22/2008;
Designer        I.M. Good;
Company         University of Colorado;
Location        None;
Assembly        None;
Device          g16v8a;
```

Tip: Header information for WinCUPL source file (.PLD) and WinSim input file (.SI) must match.

```
/*****************************************************************/
/*                                                               */
/*      General File Comments                                    */
/*                                                               */
/*****************************************************************/

/*
 * Order:  define order, polarity, and output spacing of stimulus and response values
 */


/*
 * Vectors:  define stimulus and response values, with header
 * and intermediate messages for the simulator listing.
 *
 * Note: Don't Care state (X) on inputs is reflected in outputs where appropriate.
 */
```

Fig 1.3.2: Header layout of file

Running/Compiling:

1. Go to Run => Device Specific Compile (check warnings/errors)
2. Double Click .si file

3. Simulator => Run Simulation

Creating Inputs/outputs

- Should be: pin (#) = (internal designation)
    - Can be [(char)#..0]

Operators (manual)

- & = and
- ! = not
- # = or
- $ = XOR
- ** = Exponential
- * = Multiplication
- / = Division
- % = Modulus
- + = Addition
- - = Subtraction

Creating Numbers (manual)

- ‘b’0 Small Binary
- ‘B’0000 Big Binary
- ‘D’55 Decimal
- ‘h’1F hex
- ‘H’AX11 Hex (X characters are for don’t cares/range of vals)

Fields (manual)

- Basically arrays
- FIELD ADDRESS = [AX..A0];
- FIELD DATA = [DX..D0];
- FIELD Mode = [Up, Down, Hold];

Logs & Flip-Flops are on pages 15&16 of the manual, hopefully you won’t need them

Test Vectors

```
ORDER: clock, input1, input2 , !output ;
VECTORS:
P X X 1     /* reset flip-flop */
            /* !Q goes to 1 */
            /* Q goes to 0 */
0 X X H     /* output is HI due to */
            /* inverting buffer */
```

Fig 1.3.3: Help menu documentation of Test Vectors

This does not work. I cannot find the file that this would be in. Instead, Do these steps

1. Have working code
2. Open the simulator by hitting the window with lines at the top

3. File->new
4. Select the project you have working code for using the box on the right
5. Add all signals (input/outputs) by clicking on the left/'addsignal" button on the top
6. Add the vectors you want by hitting the "add vector" button on the top
7. Manually click all signals into states that you want to see the output of

General Notes (From tutorial and class handout)

- If you get rid of any of the wanted "windows", just go to view
- Options
    - Compiler => CUPL compiler
        - Fuse Plot
        - Equations
    - Device => opens device select
        - Device (select ATF22V10B)
        - Mnemonic g22v10
    - Simulator => Sets simulation options
    - WinCUPL => Sets WinCUPL environment
- Help can be found at any time on the top
- Changing headers must be changed in both locations (or you will get warnings)
- Negative polarity on output pins doesn't work (but will compile)
- If you want to compare values across FIELDs (arrays), you must add bits as don't cares
    - CSROM = !(ADDR:0xxx # ADDR:1xxx # ADDR:2xxx # ADDR:3xxx # ADDR:4xxx # ADDR:5xxx);
    - CSROM = !(ADDR:[0xxx..5xxx]); /* CSROM low for addresses 0000h-5FFFh */

Coding Atmel AT16V8C SPLD

Assigning Pins

Requirements: Assign A15, A14, A13, A12, /RD, and /PSEN
Finding Pins: the datasheet for the AT16V8C doesn't have specified A15-12, so I will just use I/O pins, which 15-12 are conveniently open, so I will use those.
16 & 17 are also not used, so I will use those for /RD and /PSEN respectively
That did not work, as pin 16 got angry at me and A14 was an invalid input, trying with 8-5 as the "A"s and 2 & 3 as the /RD and /PSEN.

Generating of Outputs

Requirements: READ_n = RD_n & PSEN_n, and CSPERIPH_n = !(A15 & A14 & A13 & A12)
I don't know if we will need output pins, but I will use the last two, 18 and 19, on this side of the processor.

- _n means active low

Verification Vectors

Requirements:

- >= six test vectors to show correct logic functionality (well-chosen subset of input conditions)
- READ_n output of the SPLD should toggle on either /RD or /PSEN toggles
- CSPERIPH_n should be
    - high most of the time
    - low only for addresses in the range of F000h-FFFFh

Questions

Why did pin 14 not want to be an input if it is labeled as IO on the technical documentation? Why just that pin and not the others?

Above and Beyond

Created 16 vectors, one for each of the states

Checkoff

1. Show the TA your code and full listing (e.g. .LST) file
2. (Final project) Show understanding of and discuss your questions with the TA during the Part 1+2 signoff.
3. Show the TA your commented .PLD source file and .SI simulator input file
4. Explain how these files are structured

Upload Lab 1 Part 1 files in a zip file

- named: “lastname_lab_1_signoff_sw.zip”
- three folders
    - “Spld”
    - “8051”
    - “other_files”
- The WinCUPL code file must be placed in the "spld" folder
- The assembly .lst file must be placed in the "8051" folder
- Your answers to the following questions (in a PDF file) must be placed in the “other_files” folder.

How many bytes of code space does your program require?

Code Size? 51 bytes.

I found this by removing the preconditions (with them it would be 56 bytes) and building the project. Then I checked the build window for the byte size as seen in Fig 1.5.1:

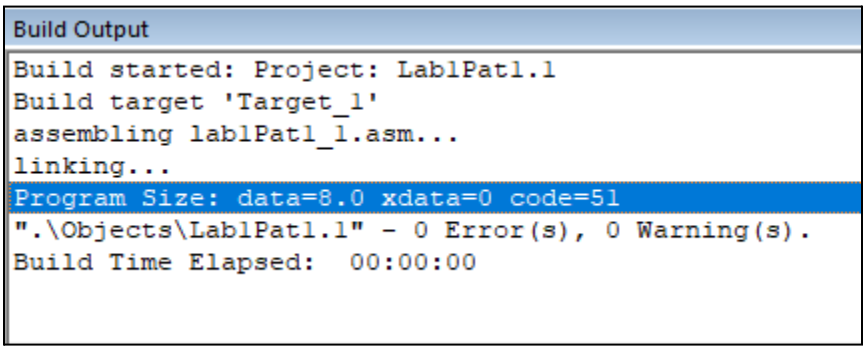


Fig 1.5.1: Build file output (without preconditions)

How long did your program take to execute for X=0x33 and Y=0x07? Assume an 11.0592 MHz clock and include the instructions executed from the beginning until you reach the ENDLOOP label.
Execution Time? 17.6323784722 us
Calculations can be seen in Fig 1.5.2

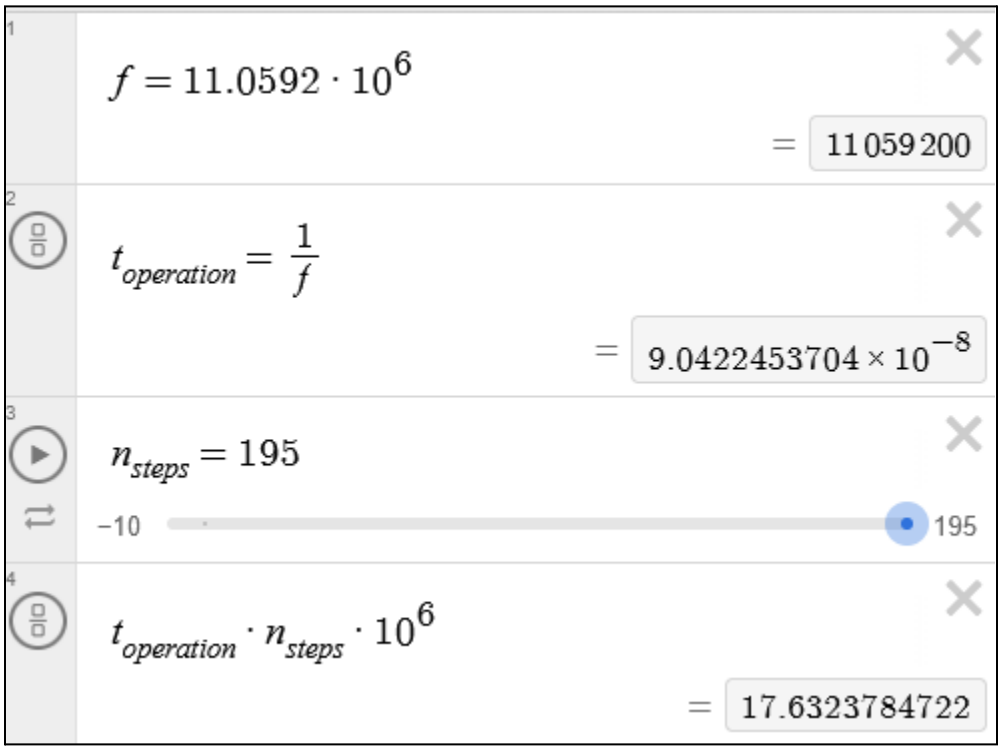


Fig 1.5.2: Detailed Calculation
The way n_{steps} was found was by running the code with a breakpoint at ENDLOOP and checking the machine states after reaching said loop, which was 195, as seen in Fig 1.5.3

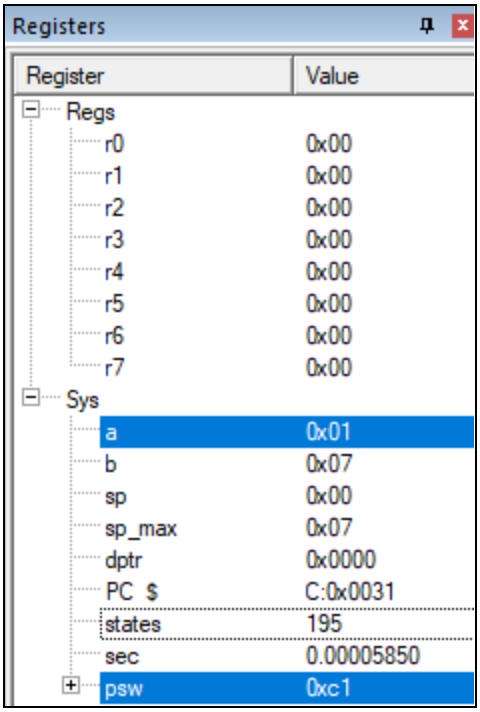


Fig 1.5.3: Register display after running code to ENDLOOP

Part 2

Obtaining Lab Kit

I have paid over VENMO as seen in Fig 2.1.

Fig 2.1: Evidence for paying for parts kit

BIG BAG

Printed Circuit Board 1:	1
16x4 LCD with HD44780 Compatible Controller:	1

SMALL BAG 1

40-Pin Wire Wrap DIP Socket (0.6" wide):	1
28-Pin Wire Wrap DIP Socket (0.6" wide):	1
20-Pin Wire Wrap DIP Socket (0.3" wide) x4:	4
16-Pin Wire Wrap DIP Socket (0.3" wide) x3:	3
14-Pin Wire Wrap DIP Socket (0.3" wide) x2:	2
8-Pin Wire Wrap DIP Socket (0.3" wide):	1
Pin Strip Socket, Single Row, 14 Contacts:	1
Pin Strip Header, Single Row, 36 Contacts x2:	2
Aluminum Standoff, Female/Female, 6-32 x4:	4
Aluminum Standoff, Male/Female, 6-32 x4:	4
DE-9 Serial Connector:	1
TO-220 Heat Sink:	1
2.1mm Male Power Connector:	1
Toggle Switch, SPST or SPDT:	1

SMALL BAG 2

T44 Wire Wrap Pins x20:	25 (forks)
Pushbutton Switch, Momentary x4:	4

Jumper/Shorting Block x5:	5 (Little blue squares)
7805T +5V Regulator, TO-220 Package:	1 (7805T)
11.0592 MHz Crystal:	1
27 pF Capacitor x3:	3
47 uF Electrolytic Capacitor x2:	2
10 uF Electrolytic Capacitor x2:	2
0.1 uF Monolithic Ceramic Capacitor (>=10V) x10:	10 104s
1.0 uF Monolithic Ceramic Capacitor (>=16V) x5:	5
4.7 uF Monolithic Ceramic Capacitor (>=10V) x2:	2
Diode, 1N4003 (1 Amp) x5:	5 (Black diodes)
Diode, 1N914 or 1N4148:	1 (Red diode)
10K 1/4W Resistor:	1 (Brown Black O G)
4.7K (or 2.7K) Pull-up Resistors, 10-pin SIP x2:	2
330 Ohm 1/4W Resistor x5:	5 (O O B G)
5.1K Ohm 1/4W Resistor x2:	2 (G B O G)
Potentiometer:	1
LED, Red x2:	2
LED, Green x2:	2
2N2222 NPN Transistor, TO-92 x2:	2 (PN 2222)
2N2907 PNP Transistor, TO-92 x2:	2 (IPS, 2907A)

Narrow Tube

74LS373, Transparent Latch, Tri-state, DIP-20:	1(GD74…)
74LS374, Octal D Flip-Flop, Tri-state, DIP-20:	1
MAX 232, RS-232 Line Driver/Receiver, DIP-16:	1
24C16 Serial EEPROM, DIP-8:	1
PCF8574 I2C I/O Expander, DIP-16:	1
SPI DAC (e.g. Microchip MCP4802), DIP-8:	1
SPLD (e.g. Microchip ATF16V8B/C), DIP-20:	1
8051, (Microchip/Atmel) AT89C51RC2, DIP-40	1 (isolated)

Basic Guide to Soldering

Tools:

- Soldering Iron
- Sponge to wipe iron off on
- Flux
- Exhaust Fan (on)
- Solder
- Mat
- Tip Tinner
- Solder Sucker

Pre-Conditions

Fig 2.2.1: Iron with condensate (link)
If the iron looks like the one above, take it to a file
When starting, melt a bit of solder onto the tip and then rotate it in the sponge (or use tip tinner)

Instructions

- Wet sponge (if using a wet sponge, distilled water)
- Turn soldering iron on to temp you want
    - Lead = 600 - 650°F
    - Non-lead = 650 - 700°F
- Heat what you want to add solder to
- Apply solder to side of the tip of the iron while right next to component
- Guide the solder onto where you want it

Why use flux?
Guide to Soldering: "Without flux the solder would oxidize rapidly, not flow efficiently, and result in poorer joints."
It is the stuff that smokes when you push solder onto the iron (apply solder fast!)

Why is flux used?
Flux is used as it pulls the layer of oxidation off the surface of metals, allowing them to properly merge together. It also aids "wetting", the ability for metal to adhere to solder. It also dissipates heat evenly across parts.

What type of flux should be used with electronic circuits?
Rosin, No-Clean, or Aqueous as followed by the IPC J standard.

Will flux remove grease from a connection point?
No, clean the board beforehand

What is a cold solder joint, and how is it created?
A solder that appears to bridge a connection but is unreliable/doesn't work at all. "Characteristics of cold solder joints include a rough surface area, rigidity, and a generally uneven appearance" Understanding Cold Solder Joints. They often form when the PCB or part is not heated enough.

Handling Power Plastic Transistors

When bending/cutting
- Hold where the pins will break off/bend (not the resin)
- Don't bend pins more than once
- Don't bend pins more than 90
- Don't bend pins "out" from the sides

PCB mounting
- Leave space between PCB & Circuit (spacers help)

Soldering
- Try to minimize high-temp exposure
- Remove excess flux, don't use trichloroethylene based products

Mounting Heat Sink
- Some have as screw you can use
- Others just use a spring clamp (paper binding)
- Don't mount them too tight (breaking plastic/copper)
- Silicone grease can help with conductivity (not too much)

PCB Layout

All of these are from the reference file

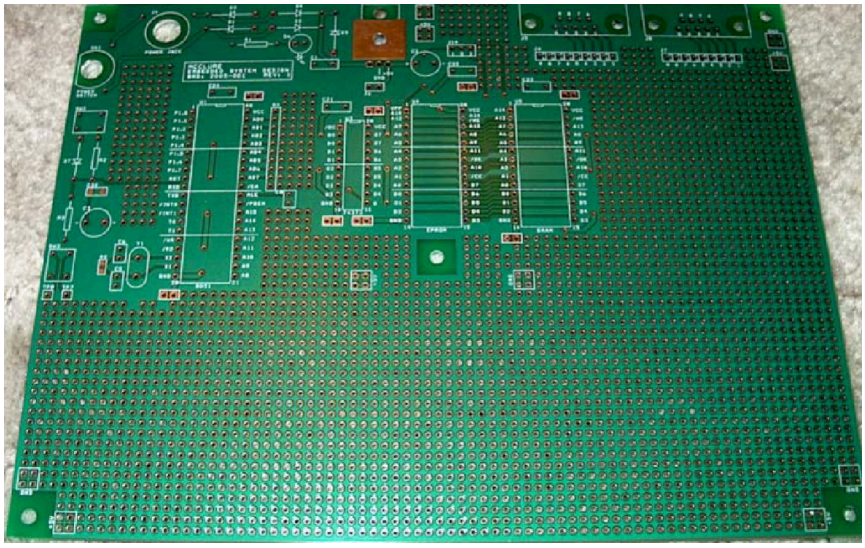


Fig 2.4.1: TOP side of PCB, notice that Decoupling capacitors on the top left

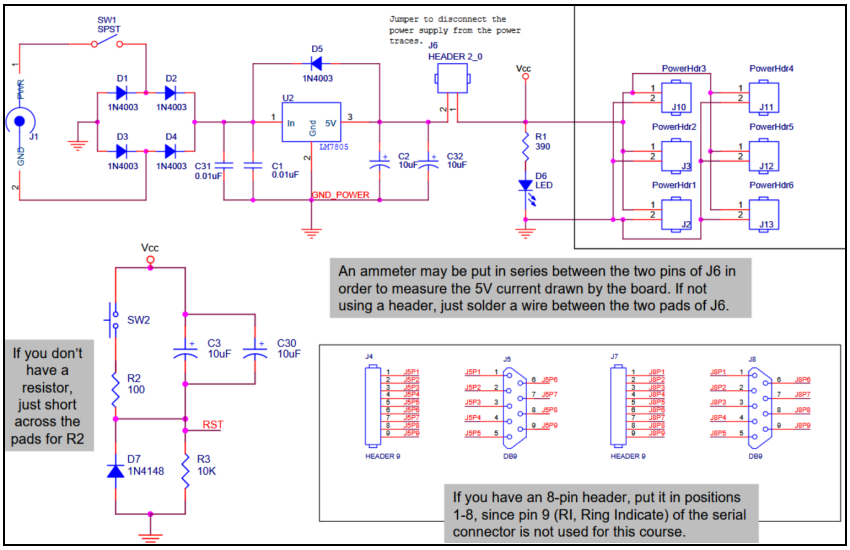


Fig 2.4.2: Basic Circuits used on PCB

Size is 6” by 8”

Initial Hardware Design

Software Choice: KiCad

KiCad Basics

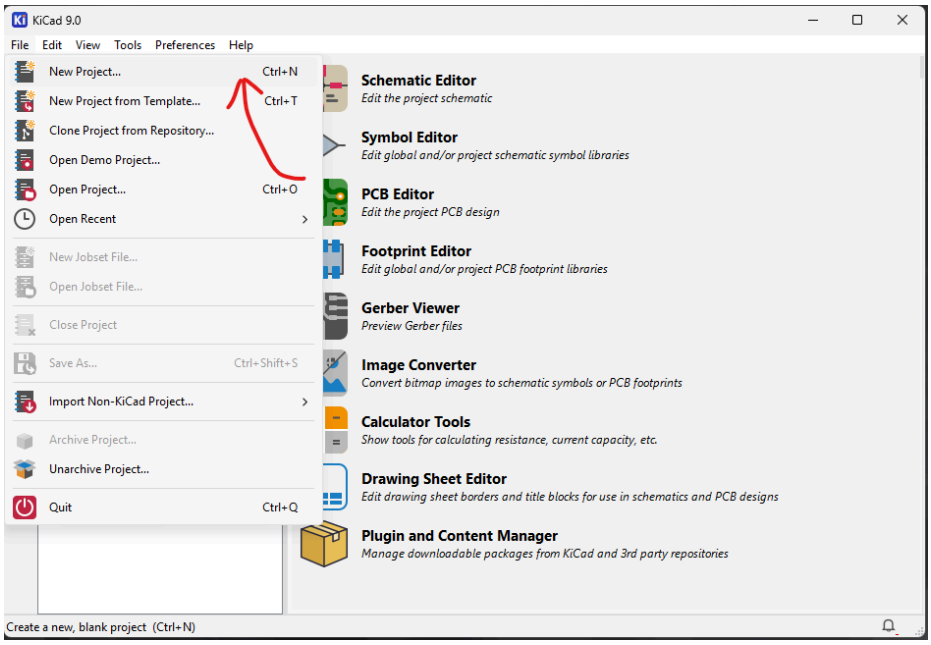


Fig 2.5.1: Making a new project

1. Find the directory you want the project in
2. Write the project name
3. Go into the Schematic editor for the project

Adding Symbols
1. Place => Symbols (or A)
2. Search up your symbol

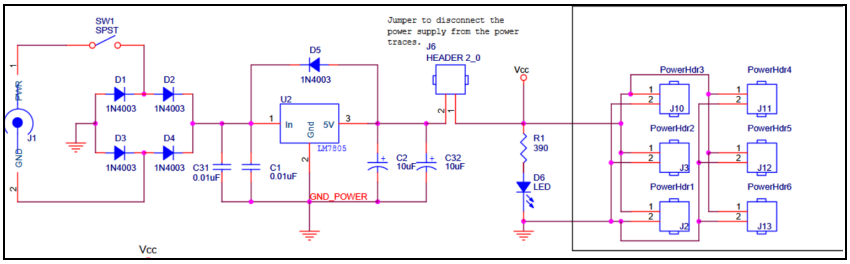
Renaming Symbols
1. Right click => Properties
2. Click to rename “Value”

Labeling Areas
1. Place => Draw Text Boxes
2. Click once, then drag to size you want, then click again
3. Name what you want it and click “OK”

Implement core microcontroller design

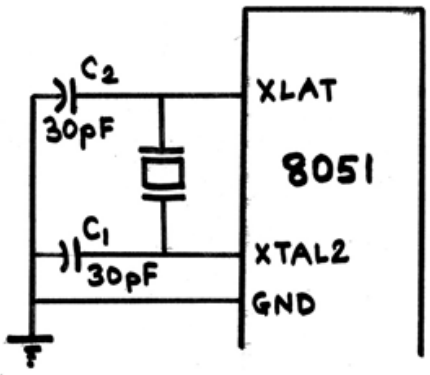
Basic Circuit Elements
- 8051
    - Power on circuit
    - Runtime circuit
- 11.0592MHz Oscilator
- 7805 (or 340T5) +5V regulator
- Hardware
    - PCB
    - Standoffs
    - Power Connector
    - 9-pin RS-232 Connector
    - Ect

Design:
- Power circuit
    - 2.1mm power jack (3 pin barrel)
    - power switch (DPDT)
    - Diode bridge (to allow two different wall adapter connector polarities – center positive or center negative) (1N4003)
    - 7805 voltage regulator
    - power-on LED



- C31 and C32 are not through-hole capacitors, so I did not add them
- Changed resistance to 330 as that is what I have in my kit

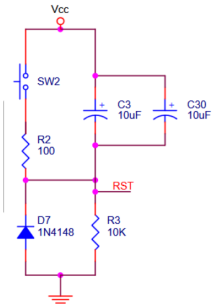
- Oscillator circuit
  - Crystal
  - 27pF capacitor



([link])

  - Replaced 30pF with 27pF

- Power On Rest Circuit
  - RC
  - Pushbutton



  - Replaced R2 with a 330 ohm and R3 with 5.1k ohm, as this is what our kit has
  - C30 is not a through-hole capacitor, so I did not add it

Finalized Schematic

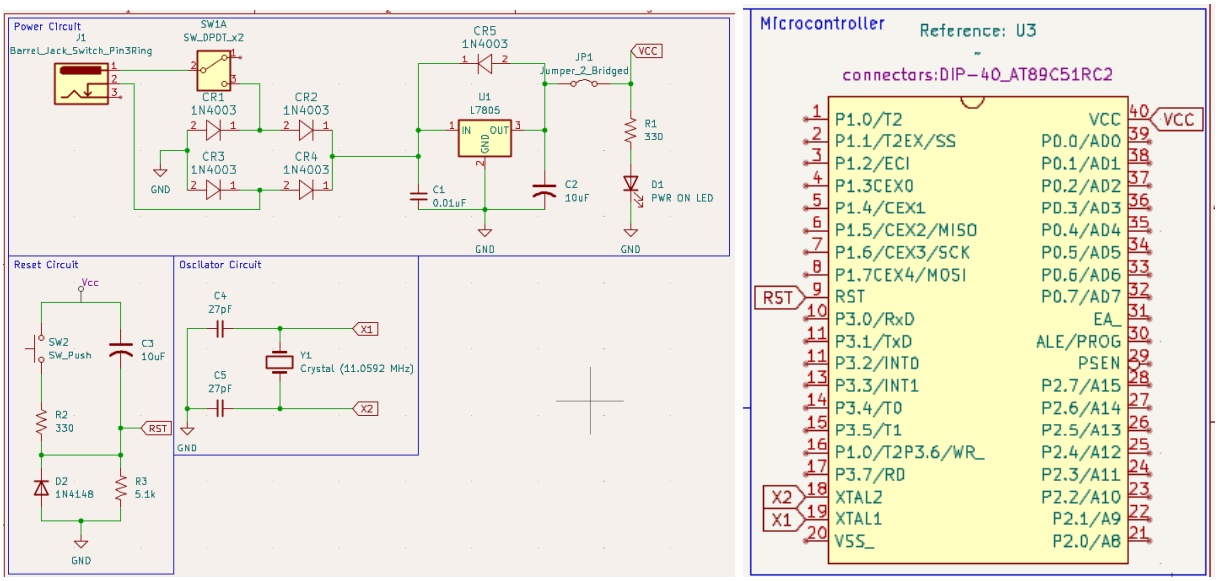


Fig 2.5.2: Schematic for part 2

Part 3

Validating PCB

I probed continuity to all GND lines and continuity of all 5 V lines, which all had continuity. Then I probed a ground to 5 V, meaning none of the ground lines have continuity with any of the 5 V lines.

Layout of PCB

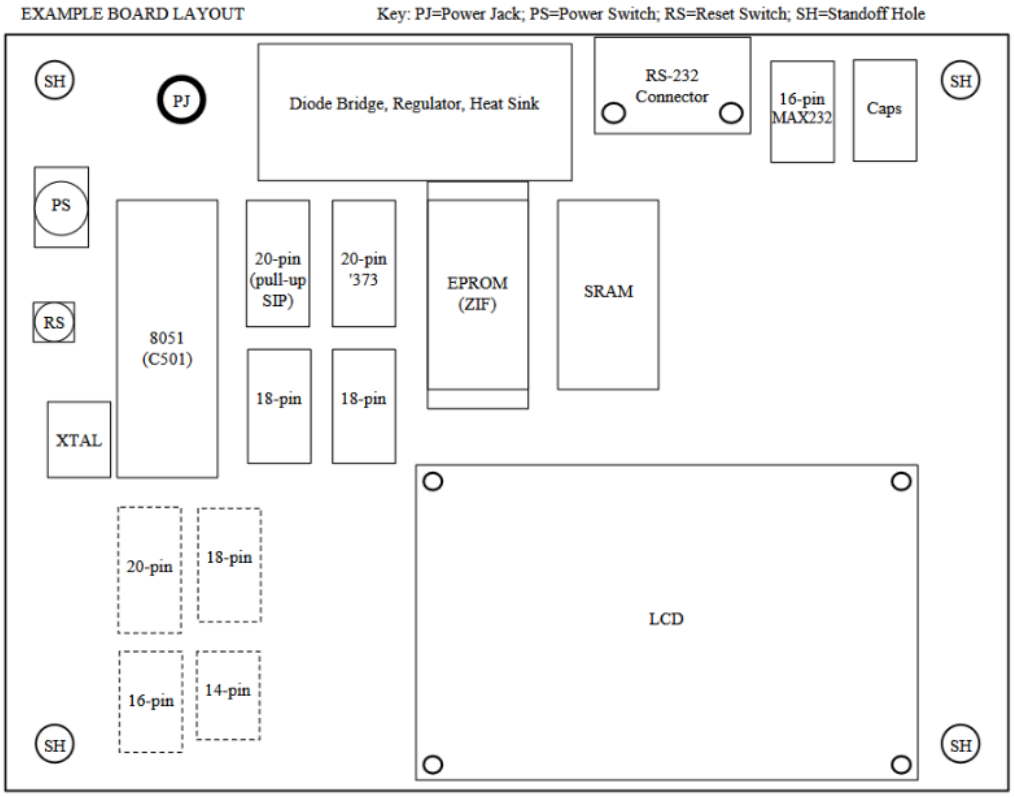


Fig 3.2.1: Example layout of PCB from Canvas

Notes

- leave a keep-out area (perhaps 0.5" to 0.8" wide) around the edge (cracking)
- Serial port connector and Power Supply Connector on the same edge of board for ez turing over
- Reset button and power supply SW near edge of board (EZ upside down reach)
- Buttons/Switches/Jacks EZ for fingers to reach (less components around them)
- Micro to pull-up resistors, 74LS08, 74LS373, SRAM, **oscillator** close
- 10-100nF for bypass caps for each chip **small loop**
- Drill early/not at all

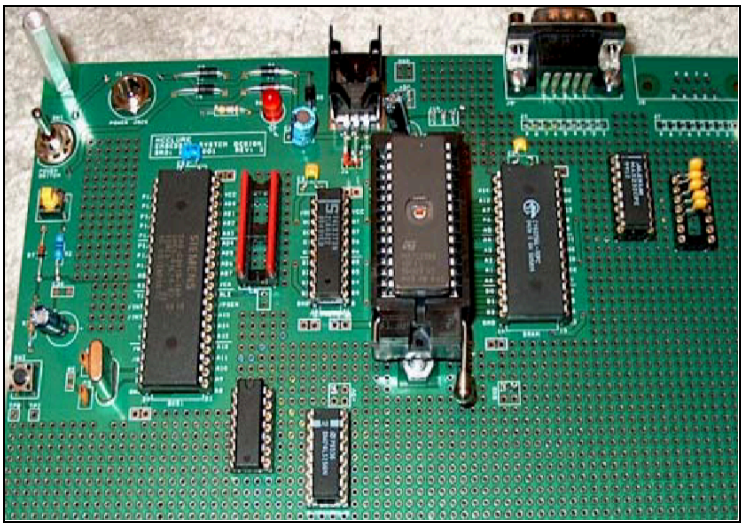


Fig 3.2.2: Example Board Setup (partial schematic canvas)

Due to not knowing all the components and what each will need, it is hard to make a final schematic, so I will try to put bypass caps at the heads of all the ICs as a convention, and follow this diagram when I am unsure of what to do next.

Labeling Your PCB

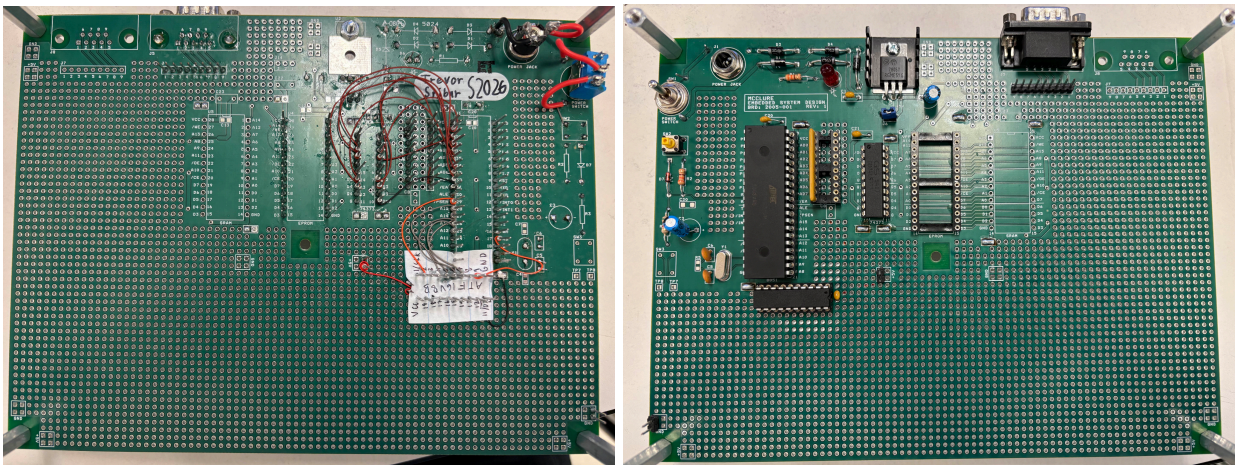


Fig 3.2.3: Images of labeled PCB

Inserting Wire Wrap Sockets

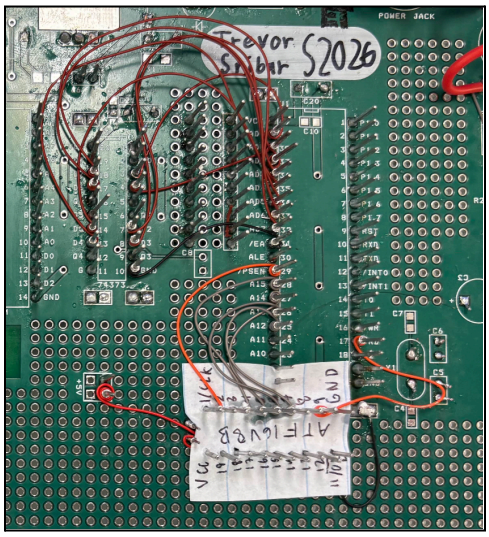


Fig: 3.2.4: image of circuit with wire wrapping

I am unsure of what the other parts are that I may need, I have added the obvious/non unlabeled wire wrap sockets, but have left the others off for now.

Power Circuit

Screw in:

- Power jack header
- Switch

Solder in:

- 8705T
- Diodes x5
- 330 ohm resistor R1
- RED power LED
- 100nF capacitor C1 (changed from 10nF as it wasn't in parts kit)
- 10uF capacitor C2

Solder for the switch:

- 330 ohm resistor R2
- Switch SW2
- Power Switch SW1 and attaching wires
- 1N4148 Diode D7
- 5.1k ohm resistor R3
- 10uF capacitor C3

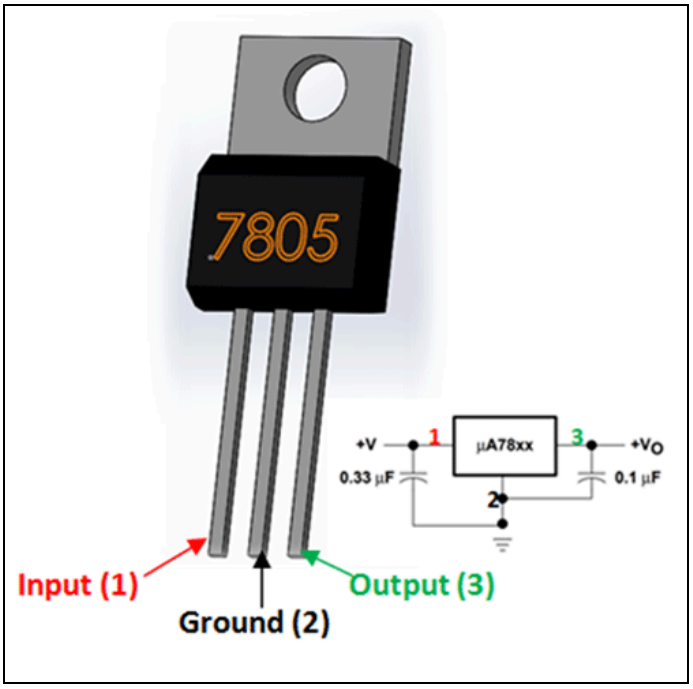


Fig 3.3.1: 7805 pinouts labeled (link)

We can also attach the heat-sinc. I made sure to not hold the housing when bending the pins

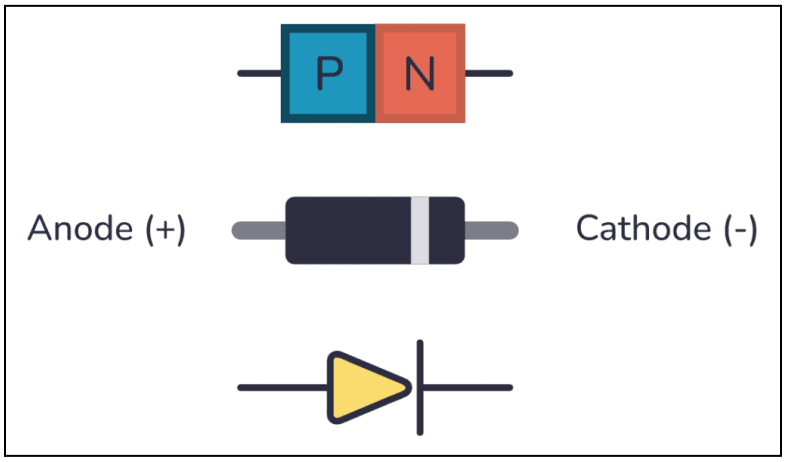


Fig 3.3.2: Diode Anode and Cathode Labeled (link)

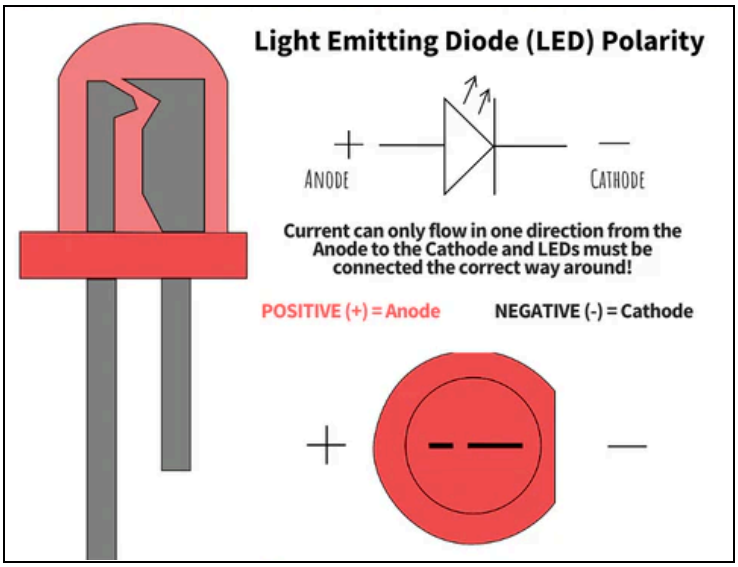


Fig 3.3.3: Diode Orientation (link)

My diode has the opposite of leg to flat edge, I will ask a TA. I ended up checking the continuity, as diodes have one-way (effective) short circuits. It ended up being the edge that is flat.

Mounting the RS-232

Setting the RS-232 was relatively easy, and I just broke off 9 pins from the long strands of pins I had. Make sure to put the long ends through the board, such that the black pieces end up on the top!
Future note: it may be easier to get another person/something to hold the pins in place for solder, it will prevent the tilting the occured.

Wire Standards

For this project I will follow the conventional wire colors specified from the lab instructions:

Signal Type	**Color**
Power	Red
Ground	Black
Multiplexed Address/Data	Grey
Buffered Data	Yellow

Latched Address	Blue/Dark Orange (as blue is not available)
Control Signals	Orange
Serial Port Signals	Green

Wire Wrapping

Wire Wrapping demon link
Instructions:

1. Get a wire wrapping tool
    a. Body
    b. Stripper
    c. Cap
2. Strip a wire with ~0.75” of metal exposed
3. Insert the wire into the small, semicircular hole (let it come out of the side of the wrapper)
4. Make sure the wire is long enough/has enough space to be wrapped around your other lead
    a. It may be easy to strip the other side of the wire at this point
5. Place the pin within the large hole
6. Turn any direction
    a. Be consistent in the direction you turn to take off wires easy
    b. I chose clockwise

Taking wires off:

1. Insert the pin over the large hole
2. Turn in opposite direction you tightened the wire in with
    a. This will be counter clockwise for me
3. Do not re-use unwrapped wires this way, as the ends are severely weakened

At this point for us, we have nothing to wire as we are using the board from 2005 that has wired GND and VCC for the controllers we have added so far.

Verification

After plugging in the power, I got no voltage drop across the power supply. I had checked the continuity before between the middle and side pins, one of which had continuity, and thus I assumed both were ground. After plugging in the power supply, they no longer had continuity, so the circuit did not work. I swapped the wires out (and now have a big solder blob on the back of my board), and the LED still did not come on. Turns out I forgot to add the jumper for power. Now the LED turns on just fine when the board powers up, and all the 5V rails are 4.996 V to

ground with the multi meter. (I also had unsoldered discharge capacitors that were shorting ground and 5V, but I fixed it when I realized the 7805T was getting too hot)

Fig 3.4.1: Stable 5V signal from O-scope

Oscillator circuit

Solder

- Oscillator
- 2x 27pF capacitor C5 & C6

Also add the decoupling caps for C20 and C21 (100nF)

Pre-Programming

Pull up port 0 pins: using line resistor:

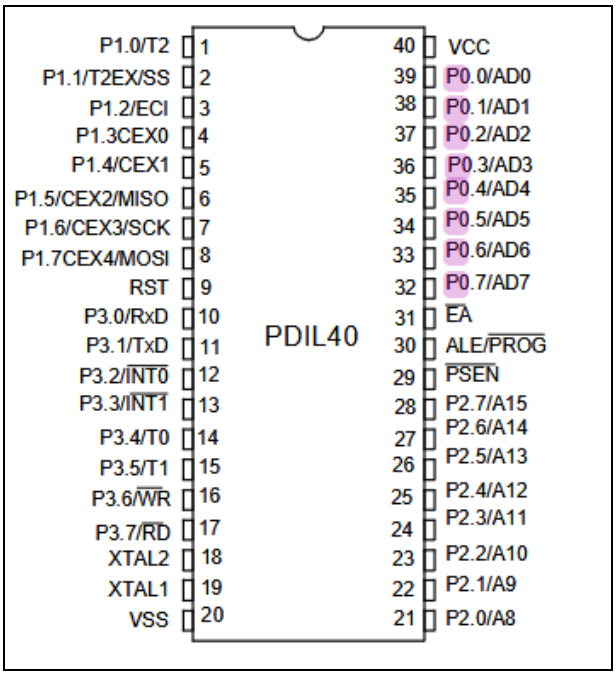


Fig 3.5.1: Port 0 pins (page 9 of datasheet)

Tie EA line high (part of the line of resistors we put in)

Programming

Download the hex file from canvas and take 8051 over to station and use Phyton to load the file onto the chip (see “Programming the Atmel ATF16V8C SPLD with the Phyton ChipProg-48 Device Programmer” on canvas.

Validation of Circuit

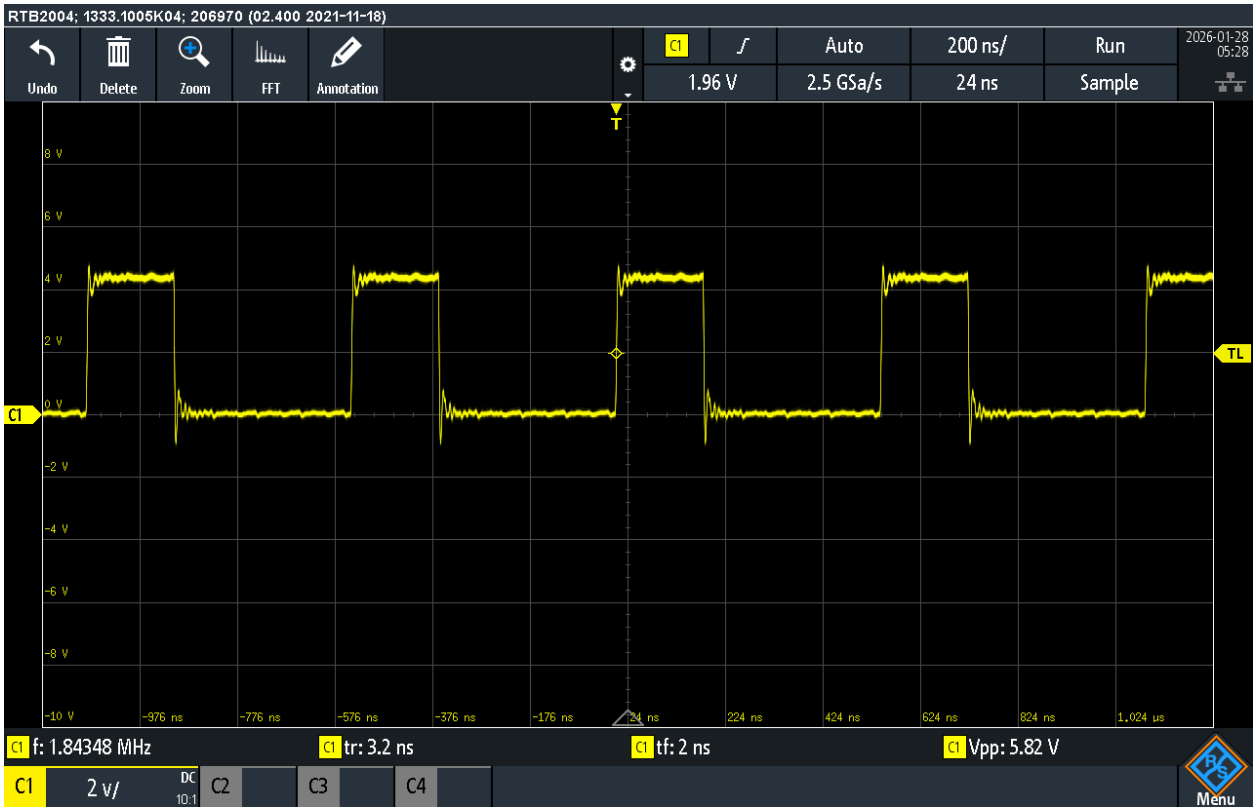


Fig: 3.6.1: Ale line output after flashing code

Notice the ringing that is occurring. This is due to the scope probe (and is not caused by compensation). I stole a screwdriver from capstone to check that I have returned. This ringing can also be seen on the compensation line.

Crystal oscillator: ~11.0592 MHz
Estimated ALE line frequency: ~1.8432 MHz
Measured ALE line frequency: 1.8435 MHz

This discrepancy is easily explained within nominal ranges for our environment

This circuit worked on every press of rest & off/on toggle, thus I did not compensate the capacitors

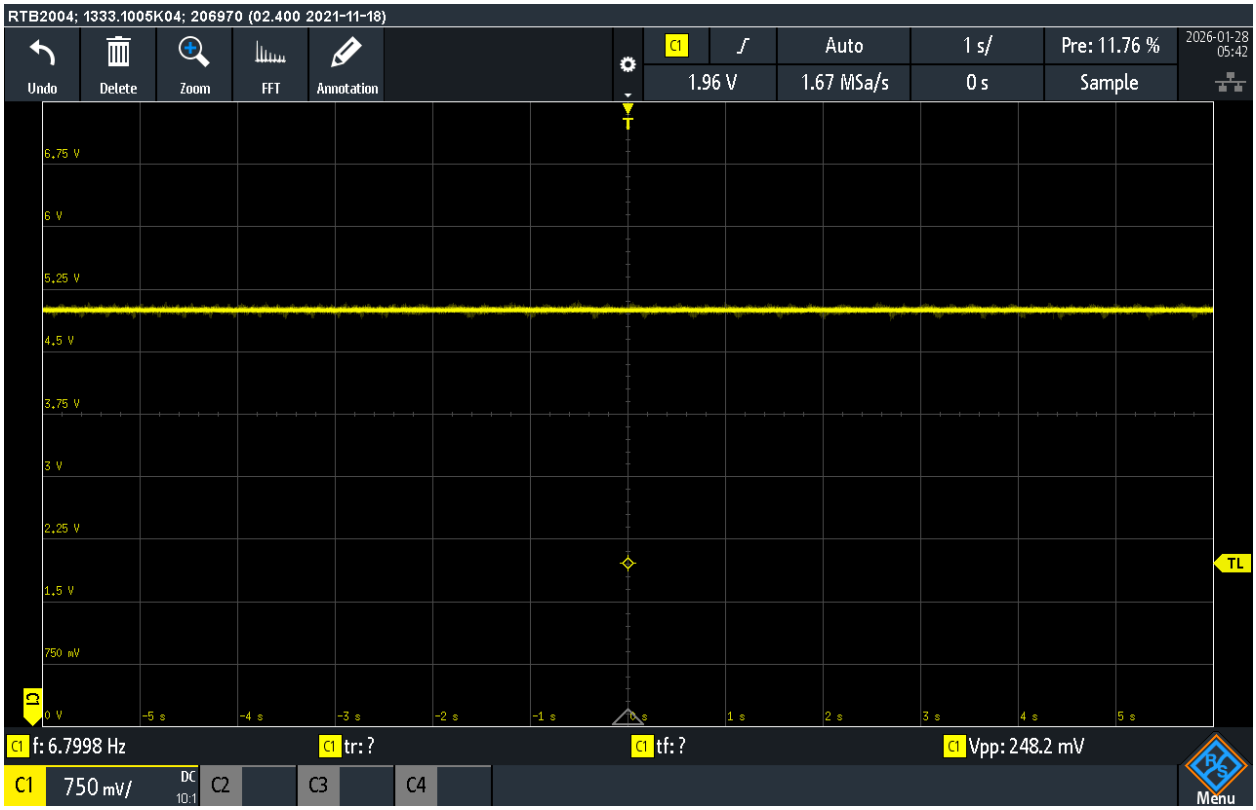


Fig 3.6.2: Scope probe of VCC and GND over 1s

We can see that the oscillation is much less than 800mV at 248.2 mV.

Wiring the ‘373

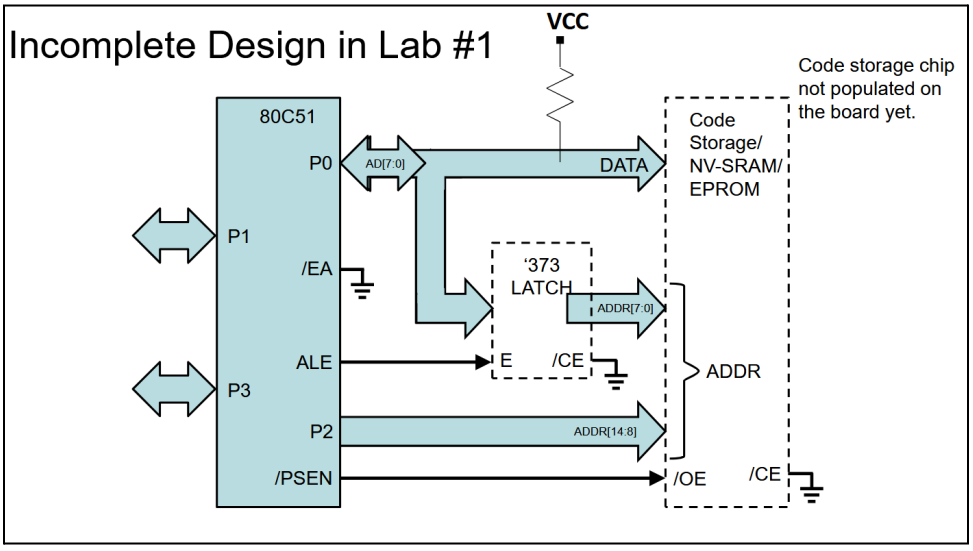


Fig 3.7.1: Design of the ‘373 latch circuit (from lecture 3 slide 76)

The D pins on the latch are the “input” (coming in from the microcontroller)
The Q pins are the output pins

- AD0 => D0
- AD1 => D1
- Ect.

These have a wire color of Blue/Dark Orange (as blue is not available) as this data is a Latched Address

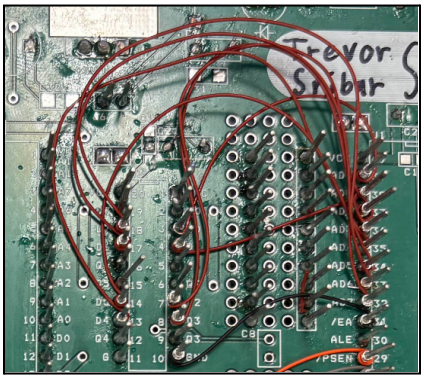


Fig 3.7.2: Wiring for ‘373 latch

Wiring the SPLD (F16V8B)

From the earlier WinCurl code, modified to fit board layout:
/* ********** INPUT PINS *********/
PIN 2 = PSEN_n
PIN 9 = RD_n
PIN 4 = A15
PIN 5 = A14
PIN 6 = A13
PIN 7 = A12

/* ********** OUTPUT PINS *********/
PIN 18 = READ_n
PIN 19 = CSPERIPH_n

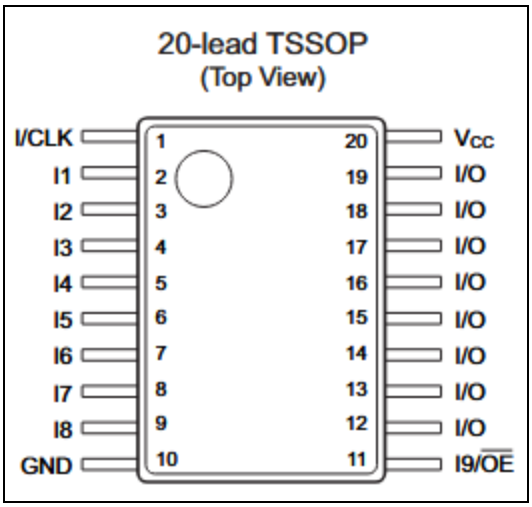


Fig 3.8.1: Schematic for ATF16V8B

Pins A12-A15 are address pins (Grey)
Pins PSEN_n and RD_n are control pins (Orange)

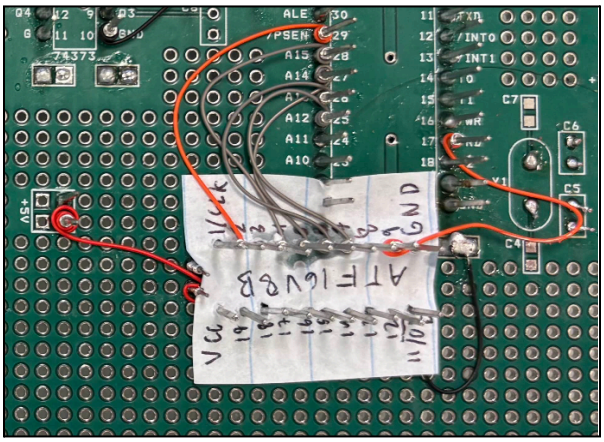


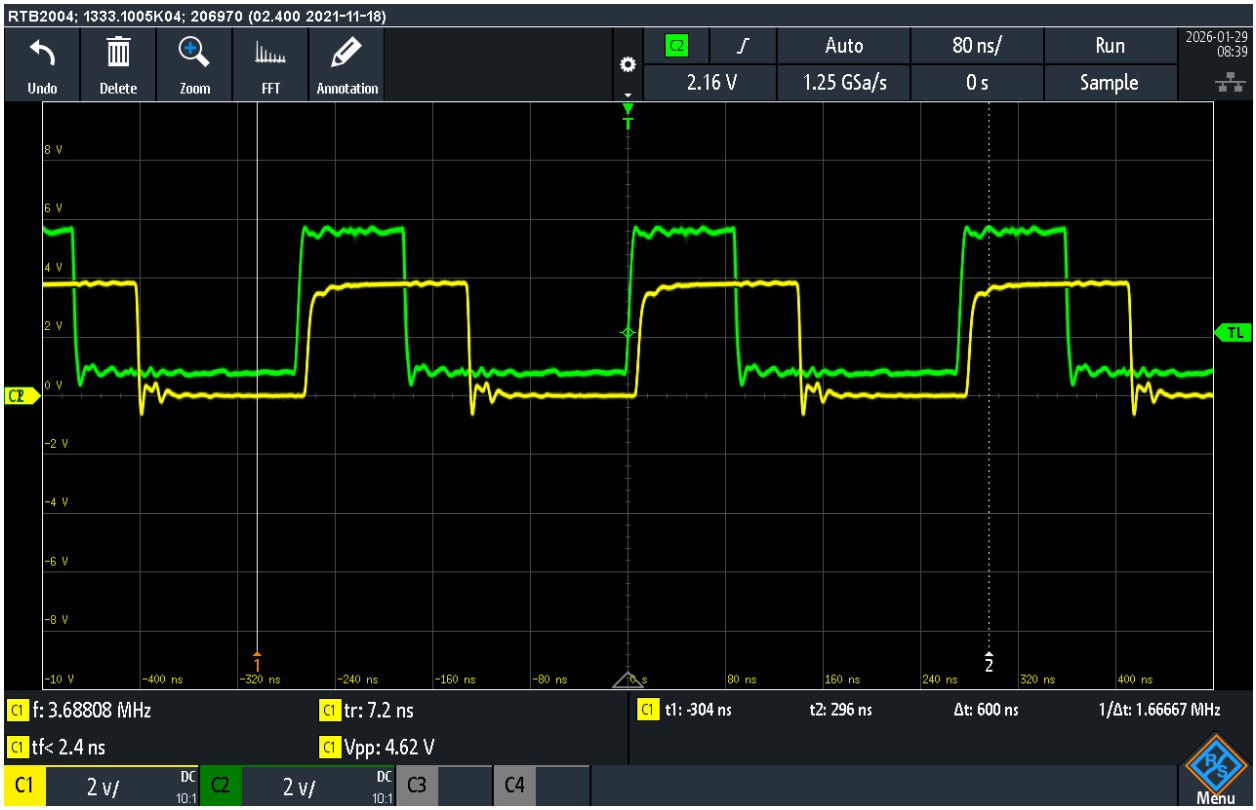
Fig 3.8.2: Wire Wrapped SPLD

Loading Code to SPLD

Follow the steps outlined in the PhytonProgrammingATF16V8C found on canvas, this is the file name I gave it for my google drive (should be under Life -> School -> colleg -> 3nd 2st -> in bed system design)

Verifying Output

I attached the latch, 8051, and the SPLD.
READ_n output is as seen below, compared to /PSEN:



test0124.hex

Fig 3.8.2: Image of READ_n with respect to /PSEN

CSPERIPH_n output is as seen below:

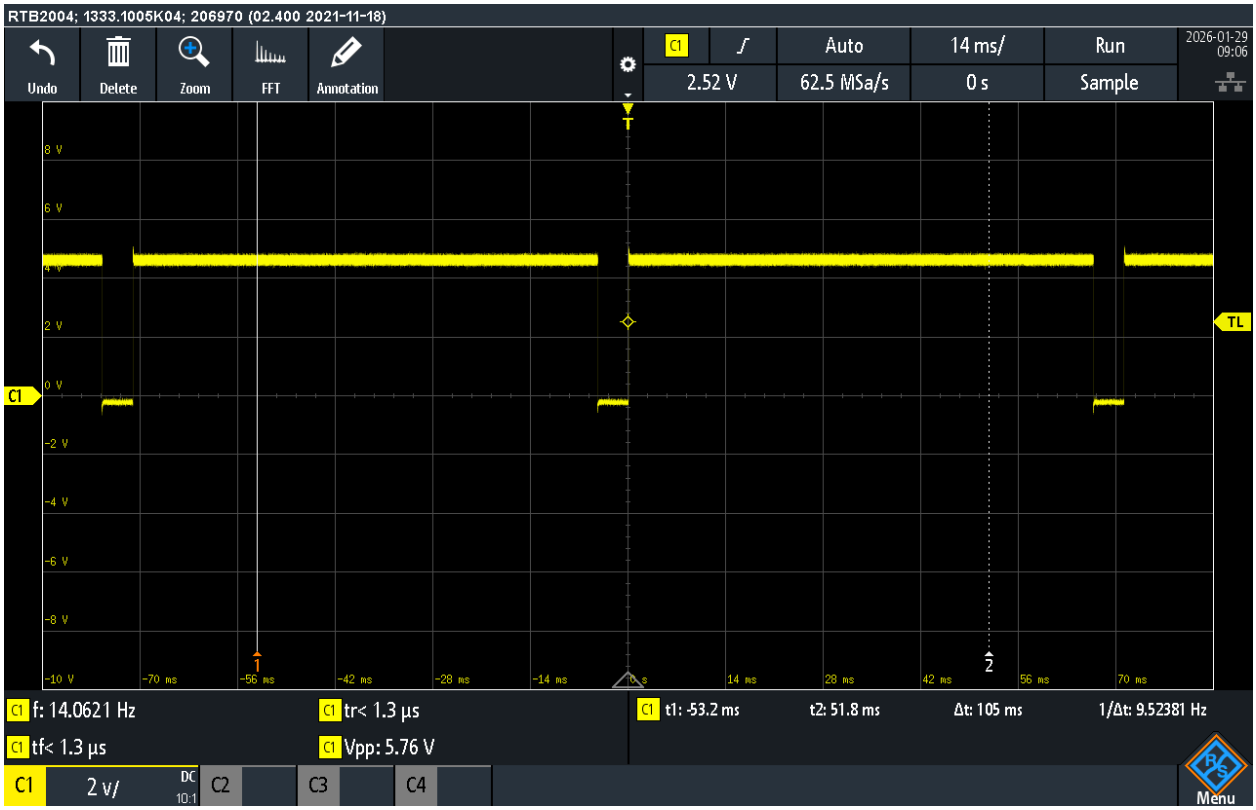


Fig: 3.8.3: Image of CSPERIPH (15/16 duty cycle)

Oscilloscope:

FFT does a fast fourier transform: plots the magnitude of all present frequencies

Future steps

For each element added

1. Check the closest VCC and GND
2. Check what you must connect to with the microcontroller
3. Build the expected final build into the schematic
4. Check the orientation of the chip (mirrored on the bottom of the board)
5. Check continuity after wiring/soldering

EFM8BB1/ARM

Basics of ARM

ARM is an instruction set architecture, so many processors can be arm processors, even if their functionality vastly differs. They are a part of RISC (Reduced Instruction Set Computer). They are light, low power, portable, making them good for embedded systems. It differed from the normal RISC processors by servicing interrupts, and has many other pieces that make them good for embedded systems.

Our ARM Chip

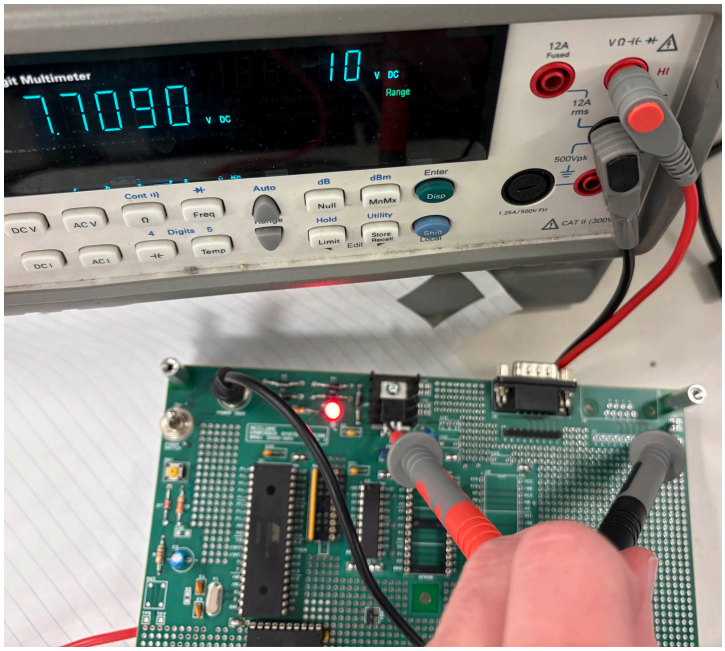
Our ARM chip is an STM32F411, which can be programmed with CUBE32IDE, which has many functions similar to VS code such as control clicking to find function definitions. It is usually used to flash C code to the chip.

Basic ARM Program

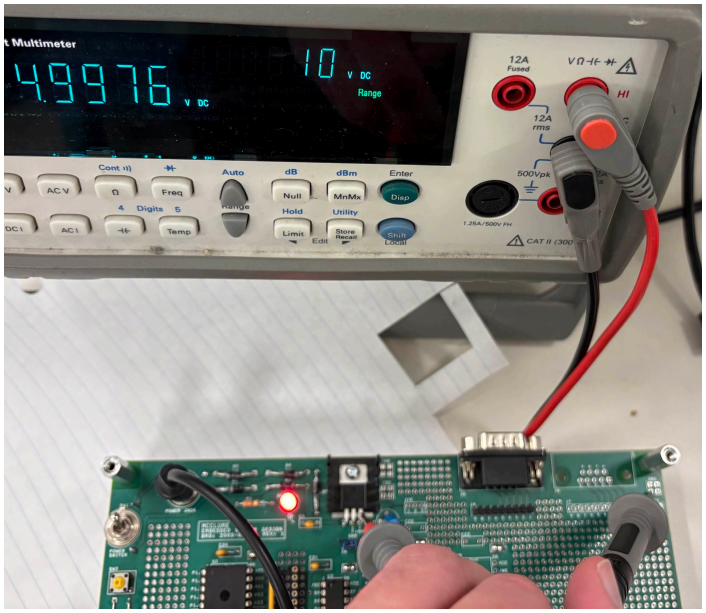
Logic: While the button is pressed, the LED will be on, otherwise it will turn off.
This code can be seen in the final lab submission.

Questions and Check in

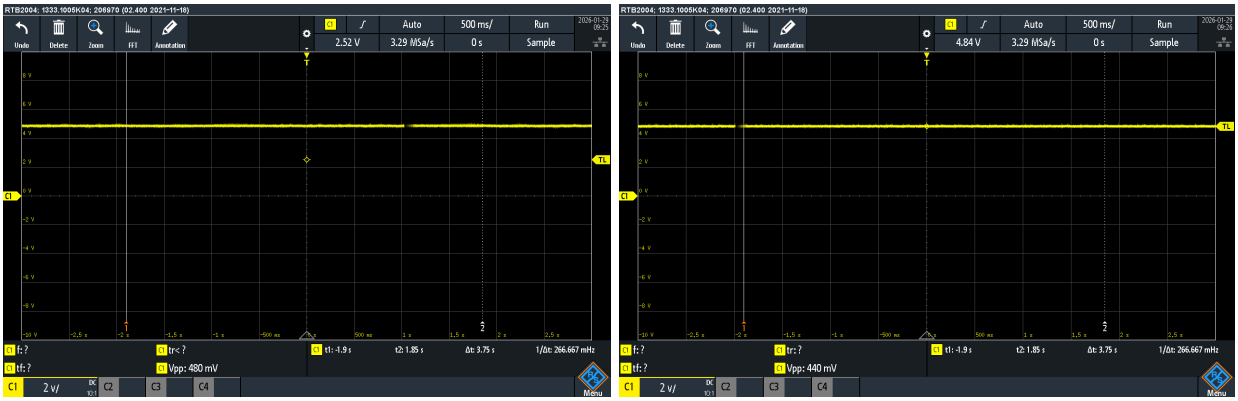
What voltage is present at the regulator input?
7.709 V



What voltage is present at the regulator output?
4.9976 V



What peak to peak noise is present across the processor VCC and GND?
Measured value at processor package pins on top side of board: 480mV
Measured value at wire wrap socket pins on bottom side of board: 440mV



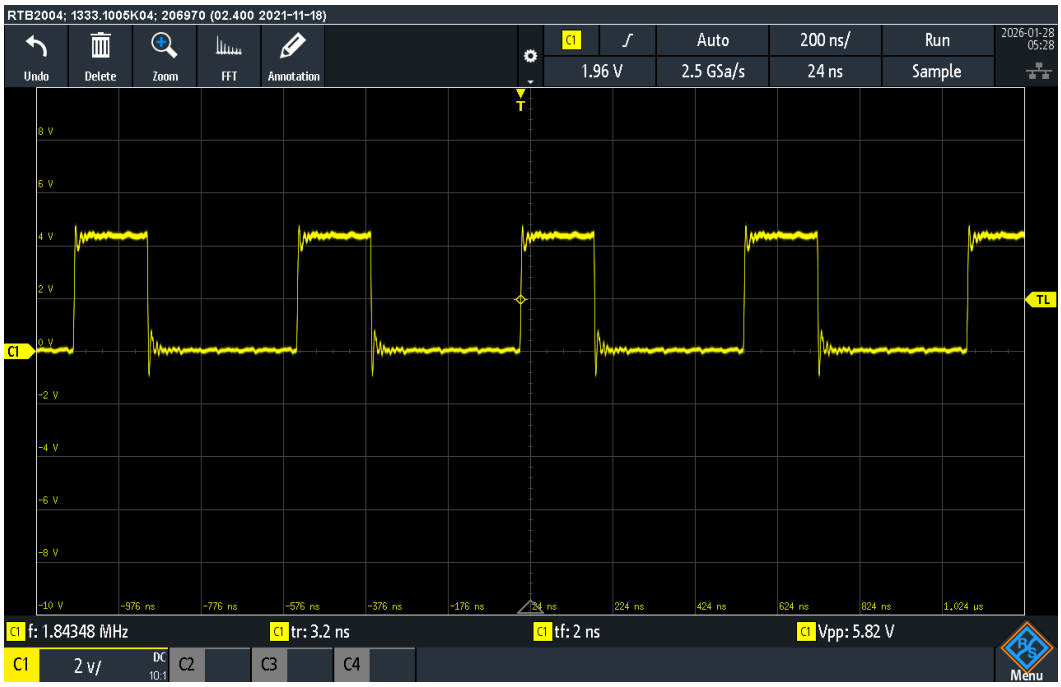
Left is top PIN (inserting probe into VCC location), Right is bottom side location of wire wrapping, that directly links into top pin.

How long is the processor held in reset after the run-time reset pushbutton is released? Use an oscilloscope and try to measure the time between the release of the pushbutton and the time when noise from ALE is observed on the RST signal.
Measured value: ~37.5 ms

C2 is the reset pin, C1 is the ALE pin

What frequency is present at the ALE pin?

1.84348 MHz



Probing of ALE pin

How much power is dissipated in the regulator, assuming a load current of 160mA? Assume that the regulator is drawing the max quiescent current shown in the data sheet (use the correct data sheet for the regulator you have on your board). Neatly show all your work and be able to explain how current flows into and out of the regulator.

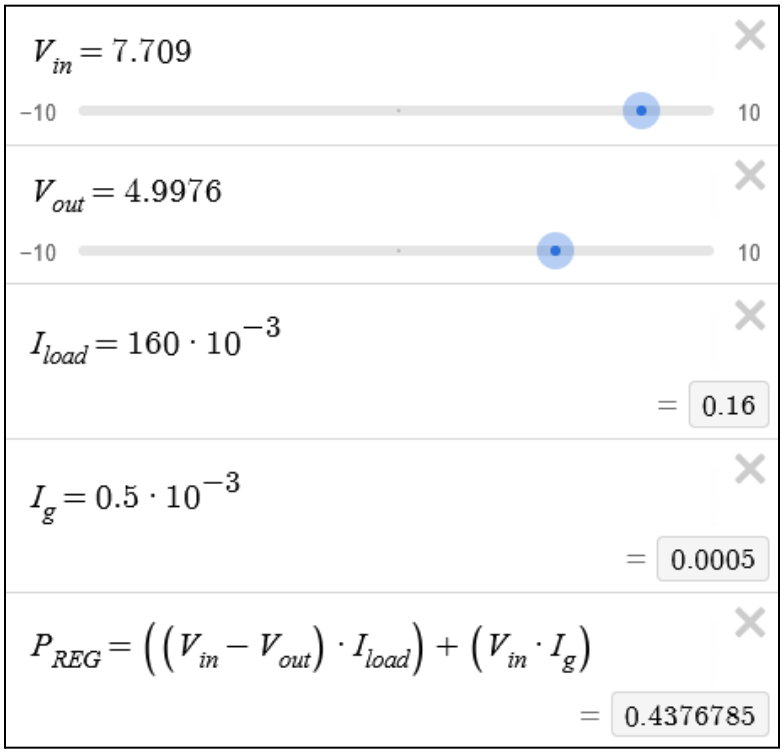
Calculated value: 437.6785 mW

From Slide 51 of Lecture 3:

$$P_{REG} = ((V_{IN} - V_{OUT}) * I_{LOAD}) + (V_{IN} * I_G)$$

We have all but I_G from the earlier parts, and from the datasheet:

ΔI_Q	Quiescent Current	5 mA ≤ I_O ≤ 1A	0.5	mA



Calculations for power dissipated in regulator

Which gives 437.6785 mW

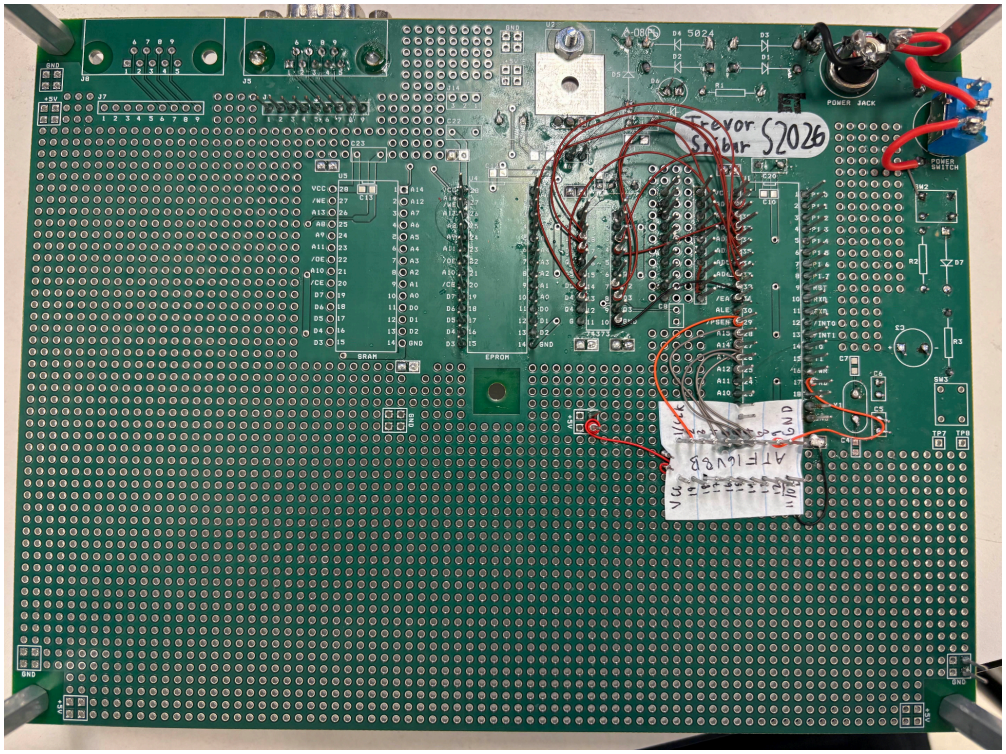


Fig 4.1.1: Final board picture BACK

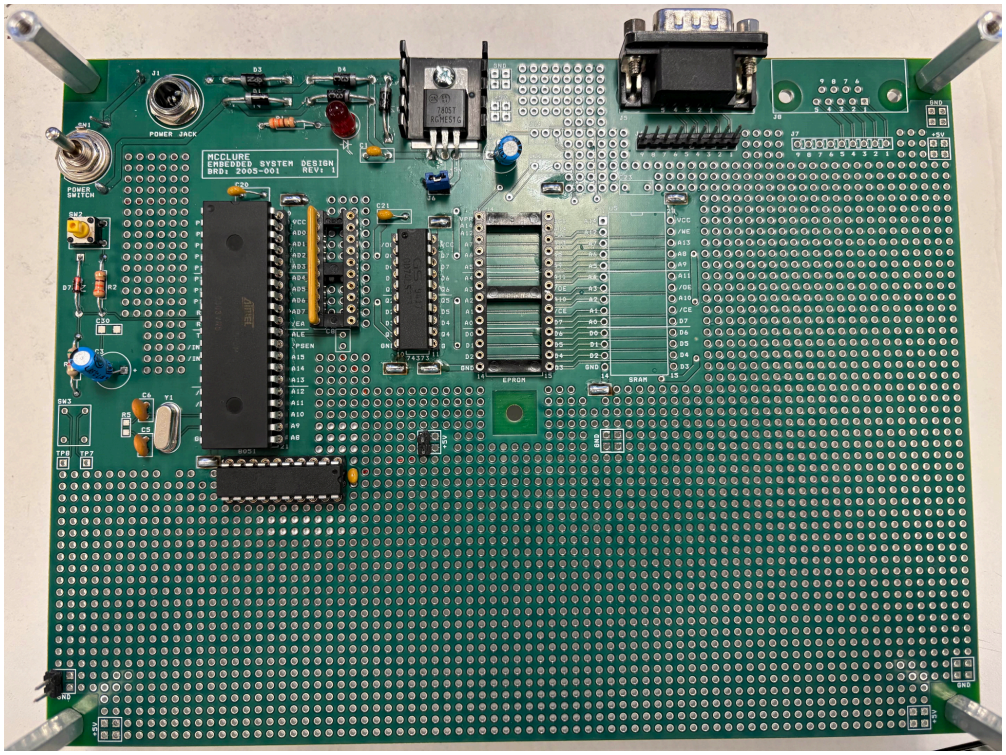


Fig 4.1.2: Final Image of Board Front