E85: Digital Electronics and Computer Engineering

Lab 6: Linear Algebra in C on a Microcontroller

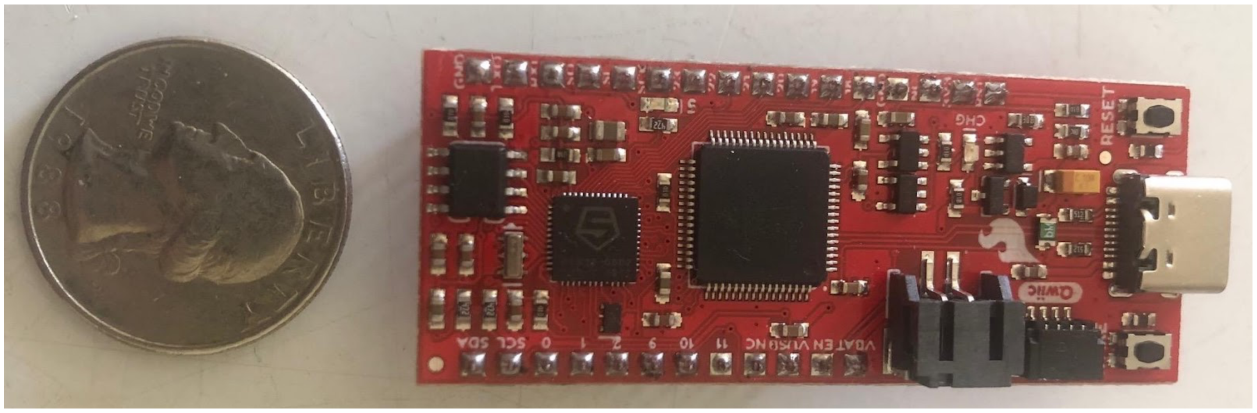
Objective

The purpose of this lab is to familiarize yourself with programming and debugging in C on an embedded microcontroller. Specifically, you will write some linear algebra routines that will help you become comfortable with loops, arrays, and pointers.

1. Welcome to the SparkFun RED-V

In this and subsequent labs, you will be using the SparkFun RED-V Thing Plus ("Red-Five") microcontroller board.

The board is roughly the size of a stick of gum and costs about $30. It is based around a 32-bit RV32IMAC Freedom E310 core and uses the RISC-V instruction set architecture (ISA). It is in a form factor which makes it easy to plug into a breadboard to connect to other circuit elements and can be programmed over USB through a USB-C connector. The board also gets power from USB, or from an optional battery, DC transformer, or other 3.6-6 V supply. The FE310 core runs at 150 MHz which makes it relatively fast among microcontrollers. It also features several useful peripherals such as UARTs (serial ports), QSPI (serial peripheral interfaces), PWMs (pulse width modulation), and timers, making it convenient to control other sensors and actuators.



Unfortunately, the FE310 chip is out of production and thus the boards are no longer available. HMC bought up all we can and has a lab set. Please handle your board carefully. If you think a board is malfunctioning, please notify the instructor or a lab assistant; don't just leave it at the station to confuse other people.

2. Getting Started with Segger Embedded Studio

SEGGER Embedded Studio is an IDE (Integrated Development Environment) for programming embedded microcontrollers. It is a professional-grade tool, but is free for educational and personal use. We will use it for programming assignments in this course. An IDE is more powerful than a text editor like Sublime, because an IDE has features like debugger and hardware integration, allowing users to build, test and upload code all in one place. In this course, we will be using SEGGER to practice C programming syntax on the desktop in Lab 6, then move onto embedded C programming to code for your FE310 microcontroller in Lab 7 - 9.

Installing Embedded Studio

Segger Embedded Studio is already installed on computers in the Digital Lab. You can also run it on your own Windows, Mac, or Linux computer. To download it on your personal device, go to the Segger Embedded Studio website and download the "Embedded studio for ARM and RISC-V" v8.18 (or later) for your operating system. Older versions had separate installations for ARM and RISC-V so make sure to install the latest version. Click through the installation process.

Create a Project

Open Segger Embedded Studio. The left pane shows the location of folders and paths of your files and code. You can navigate to your computer's files and open them directly from the folder structure here. The middle space is your code editor where the majority of your time will be spent on in the lab assignments. The bottom pane is the terminal where error messages will be presented when you compile your code. Feel free to poke around to become familiar.

To create your first project, select File >> New Project. Select Create a project in a new solution >> C/C++ executable for a RISC-V processor. Enter the project names, such as lab6_xx, where xx are your initials. Avoid spaces or other special characters in file names. Select next and scroll down to SiFive and choose the FE310 processor as the Target Device. Click through the rest and finish. By default, you will find your projects and its associated files under "C:/Users/yourname/Documents/SEGGER Embedded Studio for RISC-V Projects".

If you want to continue working on an existing project, use File >> Open Solution, and navigate to your solution file (file type .project) of that project, and click OK.

You should now see a main.c program under your "_projectname_/src" folder. Feel free to look into the provided starter code before replacing them with your own. You can simply paste lab code into this file. Save your work often!

Write a Program

Write a simple Hello World program in project named hello_xx. The main file will be named main.c Replace the provided code with the code below. Save.

```
#include <stdio.h>
#include <stdint.h>
int main(void) {
     printf("Hello world!\n");
}
```

When you are ready to test out your newly written code, click Build >> Compile main.c (Ctrl-F7). Look for a success / fail message at the bottom terminal. Debug any syntax errors.

For later labs, if you have issues with libraries not found when you compile, make sure your .h header files are in the same src folder as your C code.

Simulate the Program

You can run your program in a built-in simulator or on the actual hardware. We will first run in the simulator because you don't need to connect your board to the computer to do this.

Choose Build >> Build and Debug (F5). When Segger prompts you about whether you would like to connect to a J-Link via TCP/IP because No probes are connected via USB you can select **No**. It will then ask if you would like to use the simulator. Your program will show up in the center pane with the current line highlighted. The assembly and machine language version will show up in the left pane and register and variable watch windows will show up in the right. The output console at the bottom will show progress and the print statements. If any of the debug panes are not shown you can add them by going to View and selecting the appropriate pane under the Debug submenu.

Choose Debug >> Step Over (F10) twice to step through the printf statement. You'll see Hello world! show up in the Debug Terminal.

If your program had variables, they would show up in the Locals pane on the right.

Arrays have a small triangle next to them that you can click on to see the contents. By default, you see only the first element of the array. Right-click on the variable and select Array from the pop-up menu, then enter the number of elements so you can see all of the elements.

Look at the other Debug menu options about how to stop, restart, or step through your program. When your program contains function calls, Step Over steps through the entire function at once rather than line-by-line. Step Into goes into the function so you can watch it line-by-line. The Go command runs a program to completion or until it hits a breakpoint. If you run past the end of the program, Segger will hang in an infinite loop. Set breakpoints at the final return statement or anywhere along the way to help debug.

Run on a RED-V Board

To run your program on the physical hardware, plug your board into the computer with a USB cable. In the left pane, right click (or on a Mac, hold the option button while clicking) on your project folder ("Project hello_xx") and choose Options… Under the Debug section, click on Debugger. Make sure that J-link is selected as the current Target Connection. J-link is great for debugging directly on the board which is our current goal. However, in the future, if you're just looking to use the simulator, you might want to switch the target connection to "Simulator." This way, you can avoid Segger from prompting about not having a board connected. Click OK.

Click Target >> Connect J-Link to connect to the board. Look at the bottom pane to see the connection made. If any error messages show up, check that your board is plugged in.

Choose Build -> Build and Debug again. This time the program is downloaded to the board. You can still step through just as if it were on the simulator, but if your program connects to LEDs or other peripherals on the board, you'll see them light up.

3. Debugging in Segger Embedded Studio

In this section, you will practice running and debugging code with arrays.

- Launch Segger Embedded Studio
- Create a new project (A C/C++ executable for a RISC-V processor). Call it tutorial_xx, where xx are your initials. Select the SiFive FE310 as the device.
- Paste the code below into main.c and save it.
- Always include your name, email, and date in your code.
- The code should compute the dot product of [3 4 5] and [1 2 3]. Predict the answer. Remember that a dot product is the sum of the products of corresponding elements of the two vectors: 3*1 + 4*2 + 5*3.
- Build and debug the program. Step through it and check the result.

- Take the warnings in your code seriously. Fix the comparison, which should have been if (i == 0) and recompile. Or better yet, move the initialization out of the loop to "double sum = 0;" Rebuild and check that the warnings are gone.
- You should find and fix another bug in the program besides the comparison.

```
// tutorial.c
// Your Name, date, email
// Dot product code to learn the PlatformIO tools

#define DIM 3
double dotproduct(int n, double a[], double b[]) {
    volatile int i;
    double sum;
    for (i=0; i<n; i++) {
        if (i=0) sum=0;
        sum += a[i]*a[i];
    }
    return sum;
}

int main(void) {
    double x[DIM] = {3, 4, 5}; // x is an array of size 3(DIM)
    double y[DIM] = {1, 2, 3}; // same as y
    double dot;
    dot = dotproduct(DIM, x, y);
    return dot;
}
```

---

3. Linear Algebra.

Now it is your turn to write some code.

Create a new project such as lab6_xx and a new file named lab6_xx.c, with xx being your initials. Paste in the code below. You'll see functions for matrix addition, linear combination, transpose, equality, and multiplication are empty. Write these functions using good coding style. Predict what the results should be, especially for m8. Run the program and debug any discrepancies.

Debugging hint:

When you are using the Segger debugger to test your linear algebra code, you may want to view arrays defined by pointers. For example, the add function uses three arrays: double*A, double *B, double *Y. In the locals window, right click on the variable (e.g. Y). (On a mac, control-click). Choose Array. Set the number of elements (e.g. 9 for a 3x3 array). Then click on the little triangle next to Y to see all of the elements.

```
// lab6_xx.c
// Your Name Email Date

#include <stdlib.h> // for malloc

// Add two m x n matrices and get an m x n sum: Y = A + B
void add(int m, int n, double *A, double *B, double *Y) {

}

// Y = sa * A + sb * B
void linearcomb(int m, int n, double sa, double sb, double *A, double *B,
double *Y) {

}

// At = transpose(A). A is an m x n matrix, so At is an n x m matrix
void transpose(int m, int n, double *A, double *A_t) {

}

// return 1 if all elements of A are equal to the corresponding elements of
B, 0 otherwise
int equal(int m, int n, double *A, double *B) {

}

// Y = A * B
// A is m1 x n1m2. B is n1m2 x n2 Y is m1 x n2
void mult(int m1, int n1m2, int n2, double *A, double *B, double *Y) {

}

// The following functions and main() are provided for you
double* newMatrix(int m, int n) {
	double *mat;

	mat = (double*)malloc(m*n*sizeof(double));
	return mat;
}

double* newIdentityMatrix(int n) {
	double *mat = newMatrix(n, n);
	int i, j;

	for (i=0; i<n; i++)
	for (j=0; j<n; j++)
	mat[j+i*n] = (i==j);

	return mat;
}
```

```
int main(void) {
      double v1[3] = {4, 2, 1}; // 1x3 vector
      double v2[3] = {1, -2, 3}; // 1x3 vector
      double dp = dotproduct(3, v1, v2); // compute v1 dot v2
      double m1[9] = {0, 0, 2, 0, 0, 0, 2, 0, 0}; // 3x3 matrix
      double *m2 = newIdentityMatrix(3); // 3x3 identity matrix
      double *m3 = newMatrix(3, 3); // 3x3 matrix
      double m4[6] = {2, 3, 4, 5, 6, 7}; // 3x2 matrix
      double *m5 = newMatrix(3, 2); // 3x2 matrix
      double m6[6] = {6, 2, 5, 8, 2, 7}; // 2x3 matrix
      double *m7 = newMatrix(3, 2); // 3x2 matrix
      double *m8 = newMatrix(3, 2); // 3x2 matrix
      double expected[6] = {2, 1, 0, 1, 0, -1}; // expected result matrix
      int eq;

      add(3, 3, m1, m2, m3); // m3= m1+m2
      mult(3, 3, 2, m3, m4, m5); // m5= m3*m4 (3x2 result matrix)
      transpose(2, 3, m6, m7); // m7= m6^t
      linearcomb(3, 2, 1, 1-dp, m5, m7, m8); // m8= 1*m5 + (1-dp)*m7
      eq = equal(3, 2, m8, expected); // check if m8 is as expected

      return eq; // return 1 if so; 0 otherwise
}
```

4. Extra Credit: Solving Systems of Linear Equations

If you have time to spare and enjoy linear algebra, extend your linear algebra library to solve systems of linear equations. This isn't easy, but it's quite interesting. For extra credit, email the instructor your code and a demonstration of the solution to the system of equations.

Write a C function to invert an n x n matrix. If the matrix is singular, the function should return 0 and Y is a don't care; otherwise, the function should return 1 and Y is A^{-1}. Use the following function declaration:

```
int invert(int n, double *A, double *Y);
```

Then write a function to solve for x in Ax=b for a system of n variables.

```
int solve(int n, double *A, double *b, double *x);
```

The function should again return 0 if A is singular. Be thoughtful about your use of memory because there isn't too much on this processor.

Use your function to solve the following system of linear equations:

$$\begin{aligned} 3a + b + c + d &= 6 \\ 2a + 3b + 4c - d &= 12 \\ -4a + d &= 8 \\ 3b - 2c &= 0 \end{aligned}$$

What to Turn In

1. Please indicate how many hours you spent on this lab. This will be helpful for calibrating the workload for next time the course is taught.
2. Code for `add`, `linearcomb`, `transpose`, `equal`, and `mult`.
3. What does your code produce for m8? Does it match your expectations?
4. Extra credit, if applicable. Give your code and a, b, c, and d.

Please indicate any bugs you found in this lab manual, or any suggestions you would have to improve the lab.