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

The SparkFun RED-V Thing Plus board is circuit board roughly the size of a stick of gum which 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 and the onboard NXP K22 ARM Cortex-M4 processor. The FE310 core runs at 150 MHz which makes it one of the fastest microcontrollers currently on the market. It also features several useful peripherals such as UARTs, QSPI, PWMs, and timers.

2. Platform IO Tutorial

PlatformIO is an integrated development environment (IDE) for embedded systems which is built on top of Microsoft Visual Studio (VS) Code. It is cross-platform and includes a built-in debugger.

In this section, you will learn to write, compile, and debug programs with the PlatformIO IDE.
- Launch VS Code from the start menu.
- PlatformIO is an extension for VS Code and all extensions are installed on a per user basis. The first step is to find and install the PlatformIO extension from within VS Code. Open VS Code and click on the blocks symbol in the Activity Bar on the left edge of the VS Code window. Enter "PlatformIO IDE" in the search window and install the PlatformIO IDE extension. After the installation completes (you should see some output in the VS Code terminal window and a notification when it completes) restart VS Code and select the alien head PlatformIO icon in the Activity Bar. If you are unfamiliar with the VS Code interface you may find the overview here helpful for getting oriented.
- Choose **New Project** from the Quick Access menu in the center right of the PIO Home window.
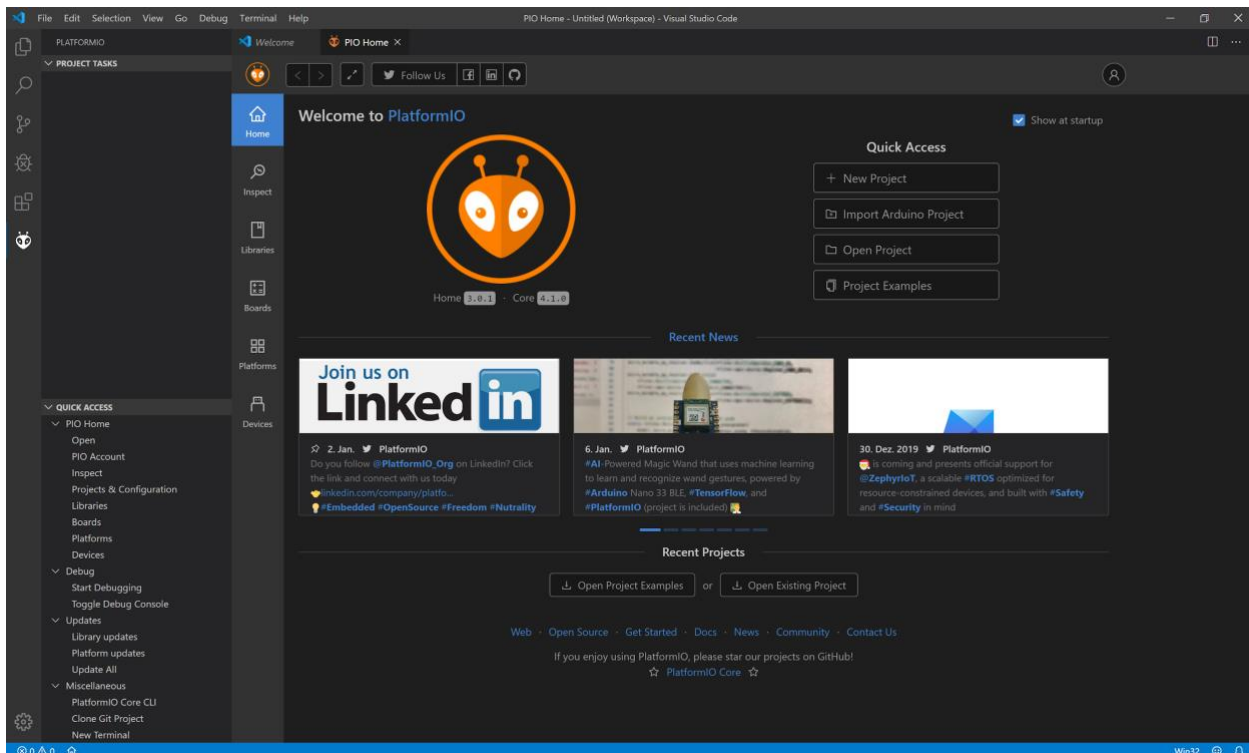
- Name your new project 'tutorial', select the 'SparkFun RED-V ThingPlus' in the 'Board' field using the search box, and select 'Freedom E SDK' as the Framework. Uncheck the "Use default location" checkbox and select a new folder on your Charlie directory (in //tempstorage.hmc.edu/eng/e85) for storing the project files. Scroll down and click the "Finish" button.
- After you click "Finish" PlatformIO will make sure that you have the appropriate toolchains installed for the selected platforms and boards and install them if necessary. Then it will open the project folder you just created in your VS Code workspace, accessible through the Explorer tab.

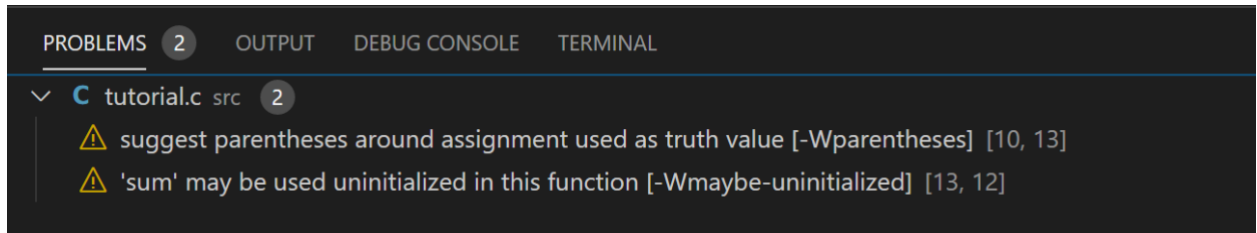Now we will create a program that computes the dot product of two vectors.
- Create a new file (**File -> New...**) and enter the following code below. Note that it intentionally contains some bugs. Save your program as 'tutorial.c' in the 'src' directory. You may also download the file from the E85 webpage.

```c
// 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;
}
```

- Calculate the dot product of [3 4 5] and [1 2 3] to predict the output of your program.
- Select the PlatformIO icon in the Activity Bar. In the top of the Side Bar you will see a list of 'Project Tasks'. Choose **Build** to compile (you can also use the keyboard shortcut Ctrl + Alt + B or the checkmark icon found in the Status Bar.) When you compile the project, you should see a terminal window appear in the Panel and some text scroll by as the project is compiled. If all goes as expected, you should see a message that confirms the build was successful. However, you will see two warnings in the Problems tab of the Panel about an assignment as a truth value where the (i = 0) comparison should have been (i == 0) and that the value of 'sum' may be used uninitialized. Fix the errors, rebuild the code, and ensure there are now errors or warnings.

```
PROBLEMS  2    OUTPUT    DEBUG CONSOLE    TERMINAL

∨  C tutorial.c  src   2
     ⚠ suggest parentheses around assignment used as truth value [-Wparentheses]  [10, 13]
     ⚠ 'sum' may be used uninitialized in this function [-Wmaybe-uninitialized]  [13, 12]
```

- To run and debug your program you have two options: debugging on actual hardware or in the simulator. Instructions for both are provided below.
  a) **Hardware:** Make sure a SparkFun RED-V ThingPlus board is plugged in and connected via a USB port (note that when you plug the board in, you should see a new USB drive appear in Explorer named "HiFive." If you do not see this, then it is likely that the USB port you plugged it into is not working properly and you need to try a different port. The USB ports on the rear of the machine tend to be more reliable than those on the front).
  b) **Simulator:** In the project folder, open the 'platformio.ini' file and comment out the block of code starting with the line [env:sparkfun_thing_plus_v]. This block of code is used to configure the build and debug environment for PlatformIO. Then add the following block below the now commented block. This block of code enables software emulation of the FE310 chip.

```
[env:e310-emulation]
platform = sifive
framework = freedom-e-sdk
board = e310-arty
debug_tool = qemu
```
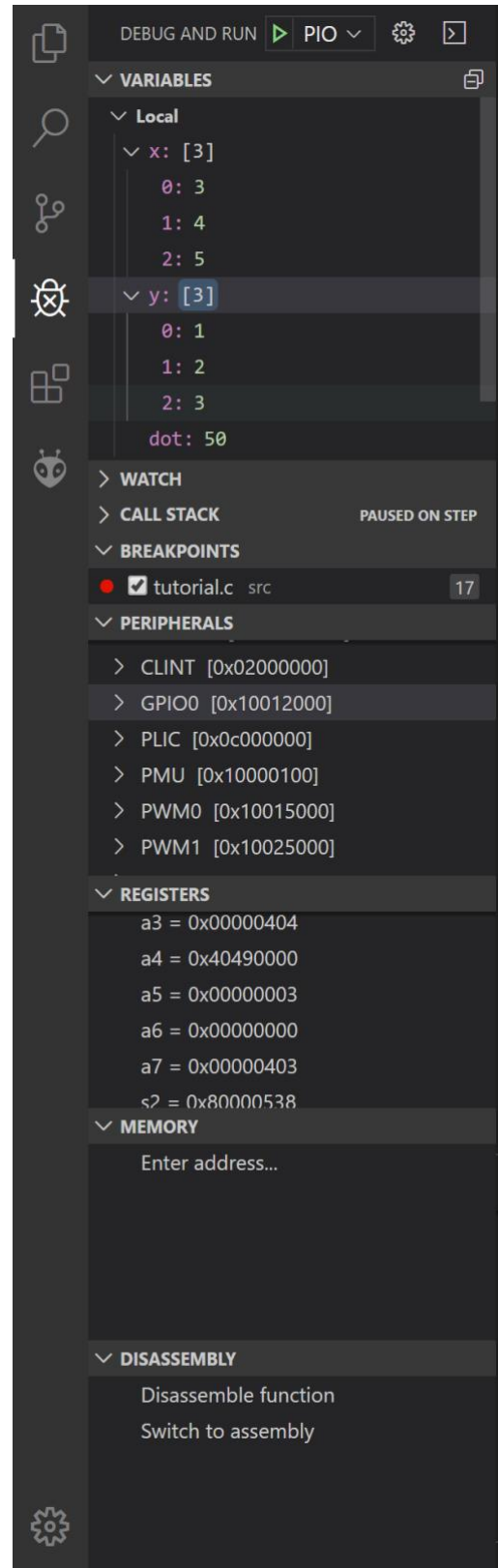
- Choose **Debug -> Start Debugging** or press F5 to begin a debugging session. If it does not automatically come to the forefront of your window, you may also need to click on the debugging icon on the left of the interface in the Activity Bar. You should see the debugger jump through a few screens and then end at your 'tutorial.c' source code with a yellow arrow pointing to the beginning of the main function. (Note: If working with the hardware to debug, your computer must have the Segger J-Link drivers installed to properly upload and debug programs on the RED-V board. The lab computers should be properly configured with the appropriate drivers, but if you get an error when going to debug related to J-Link please contact one of the teaching staff for assistance.)
- To control your debugging session, you can use the debugging toolbar which appears near the top of the editor. The **Continue** button will allow the program to execute until the next breakpoint. Breakpoints can be added by clicking to the left of the line number in the editor. The **Step Over** button will execute the current line and then stop. The **Step Into** button will execute the current line and if it includes a function call, will jump into that
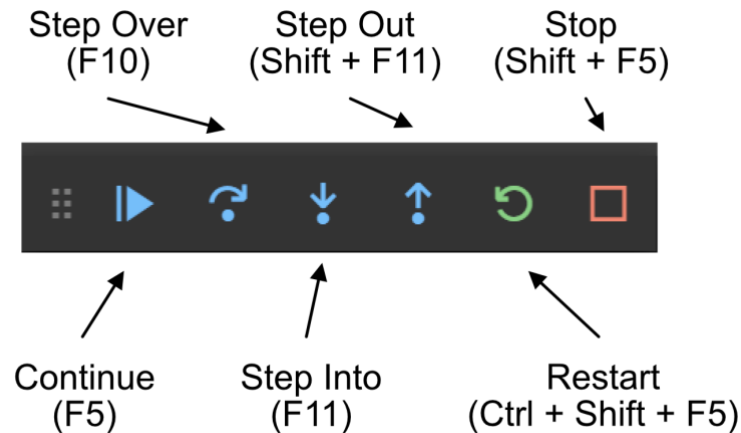
function and stop. The **Step Out** button will execute all the code in the function you are in and then stop once it returns. The **Restart** button will go back and restart the debugging session from the top. Finally, the **Stop** button will stop the debugging session and return to normal editing mode.
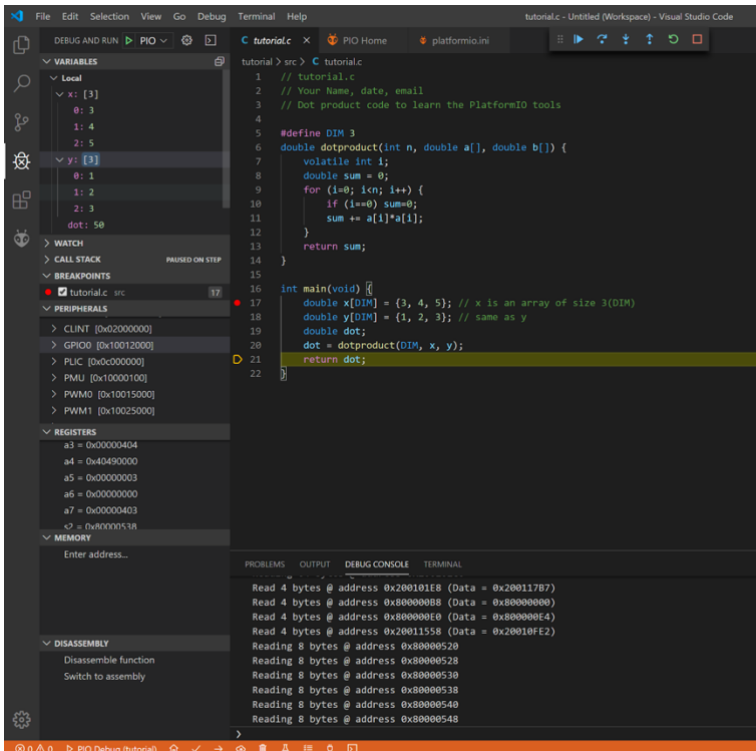
The Debugger Side Bar

The **Variables** section lists out the various local, global, and static variables present in your program along with their values. The **Call Stack** section shows you the current function being run, the calling function (if any), and the location of the current instruction in memory. The **Breakpoints** section shows you any set breakpoints and highlights their line number. Breakpoints can be managed in this section and also helpfully can be temporarily deactivated without removing them by toggling the checkbox. The **Peripherals** section (if debugging on hardware) allows you to see the status of the registers of the memory-mapped peripherals of the device (we will cover these in more detail in later labs). The **Registers** section lists the current values present in each of the registers of the processor. The **Memory** section displays the contents of a specific address of memory. Finally, the **Disassembly** section allows you to view the assembly code for a specific function or switch to the assembly view for debugging the instructions one-by-one.

Step Over (F10) / Step Out (Shift + F11) / Stop (Shift + F5) / Continue (F5) / Step Into (F11) / Restart (Ctrl + Shift + F5)

- Use the **Step Into** (F11) command to step through your code one line at a time. You can watch the variable addresses and values in the Side Bar change as you step through the code. Expand the x and y arrays so you can see their values. As you step through the first two lines, you should see arrays getting initialized. The compiler aggressively optimizes the code, so sometimes you will see weird things in the debugger or variables not changing when you expect them to change. The variable i is declared as volatile in this code to force it to be preserved by the optimizer and appear in the debugger.
- Find the bug that causes the dot product to be incorrect. Fix your code. Stop the debugger, rebuild, and redownload the fixed code.

3. Linear Algebra

The goal in this section is to write a library of linear algebra routines in C. This will help you get accustomed to loops, arrays, and pointers in C, and it is good to understand these routines because they are fundamental building blocks of signal processing code.

Mathematical operations that you will be writing include matrix addition, linear combination of matrices, matrix transpose, matrix equality, and matrix multiplication.

The following functions operate on matrices of m rows and n columns (m x n). You may assume that the result matrix has already been allocated prior to the function call. The transpose function produces an n x m result. Function declarations are given below:

```
void add(int m, int n, double *A, double *B, double *Y); //Y=A+B

void linearcomb(int m, int n, double sa, double sb, double *A, double *B, double *Y);
```

```
//Y=sa*A + sb*B

void transpose(int m, int n, double *A, double *A_t); //A_t=transpose(A)

int equal(int m, int n, double *A, double *B); //returns 1 if equal, 0 if not
```

The last function multiplies an m1 x n1m2 matrix A by an n1m2 x n2 matrix B to produce an m1 x n2 matrix Y. Y should already be allocated and the contents will be overwritten. The following is the function declaration.

```
void mult(int m1, int n1m2, int n2, double *A, double *B, double *Y); //Y=A*B
```

Now is your turn to program.
- Write a C program to complete five operations given above
- Test your program with the following code, using the newMatrix and newIdentityMatrix code from lecture. Remember that you will need to include the standard library that has the malloc function using the statement: #include <stdlib.h> at the beginning of your code.

```c
#include <stdlib.h> // for malloc

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
}
```

- Predict what each of the matrices should be and particularly check that m8 matches your expectations.
- You'll find it frustrating in the debugger watch window that when you look at a pointer variable such as m2, you only see the $0th$ element. However, the watch section of the debugger Side Bar allows you to enter expressions. You can type in particular elements such as m2[4] to have them displayed for you.

4. Extra Credit: Solving Systems of Linear 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.

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

$$3a + b + c + d = 6$$
$$2a + 3b + 4c - d = 12$$
$$-4a + d = 8$$
$$3b - 2c = 0$$

Note that your new program may hang from lack of memory. Comment out the test code from the previous part or use free to free up some space. Be careful in your memory allocation and deallocation, particularly if you are using a recursive determinant function. A malloc that fails for lack of memory will return a NULL pointer (with an address of 0).

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.