CE-2812, Lab #2, Knight Rider Lights

1 PURPOSE

The purpose of this lab is to write a complete simple application that interacts with STM32F4 peripherals.

2 PREREQUISITES

- The Nucleo-F446RE board had been mounted onto the Computer Engineering Development board.

3 ACTIVITIES

3.1 PROJECT REQUIREMENTS

Write a program that sweeps the LED lights on the dev board back and forth similar to K.I.T.T. in TV's Knight Rider. For those of you who are so unfortunate to be unaware of K.I.T.T., be sure to watch the show's into: https://www.youtube.com/watch?v=oNyXYPhnUIs.

While we will eventually want to structure our code for reuse, let's focus on functionality this time around and restructure later as needed.

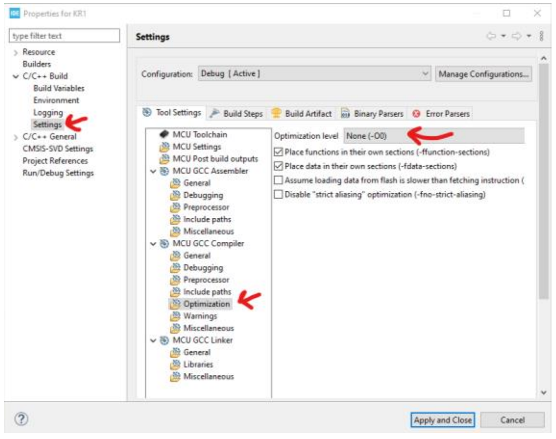
As a first step, create a new project from the template project you set up last week. We will not necessarily need the serial console for this project, but it does not hurt to have it ready to go if we want to use it to print debugging messages or the like.

For simplicity, I suggest implementing this entire project in a single source file. Also, you may "hardcode" just about everything and we will improve later.

3.1.1 Delay Subroutine

We will need a delay subroutine to slow down the action a bit. Much like in Embedded I, we will start with a "busy wait" using a simple loop.

Create a subroutine **`void delay_100ms(void)`**. The purpose of this routine should be self-explanatory. In it, create a loop (for / while / or do-while) that will create a 100-millisecond delay. How many iterations for 100 milliseconds? You will need to experiment with that a bit. Note, this sort of busy wait will only work if our 'C' compiler is not trying to "optimize." Be sure optimization is disabled. This should have been the default in CubeIDE, but it cannot hurt to verify. The optimization setting is under project properties, MCU GCC Compiler, Optimization. Set to None.



Configuration: Debug [Active]

Optimization level: None (-O0)

- [✓] Place functions in their own sections (-ffunction-sections)
- [✓] Place data in their own sections (-fdata-sections)
- [ ] Assume loading data from flash is slower than fetching instruction (
- [ ] Disable "strict aliasing" optimization (-fno-strict-aliasing)

As mentioned, for simplicity, include this routine within your main 'C' source file. It will need to be encountered by the compiler before attempting to call it, or you will need to provide a prototype.

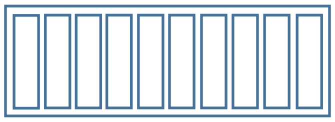
3.1.2 LED Driver Subroutine

While we can achieve the Knight Rider effect by bit shifting, we may also benefit from a generic subroutine that simply lights up the LEDs based on a number and the LED will essentially display that number in binary. Since we have 10 LEDs, a number between 1 and 1023 will light up 1 to all of the LEDs. In other words, the parameter supplied will be displayed in binary on the LEDs. No "conversion" to binary is needed – a number is a number…

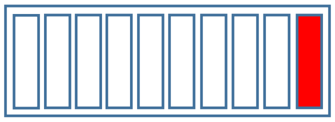
Create a subroutine **`void light_LED(XXX number)`** that will use the parameter, number, to determine which LEDs to light. In the prototype, XXX represents the type of the parameter, and it is up you to determine what actual type to use.

As an example, the following should be the behavior of this routine:

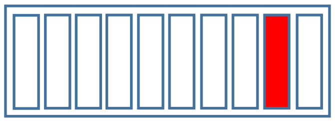
light_LED(0); ➔



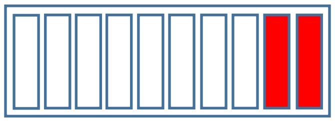
light_LED(1); ➔



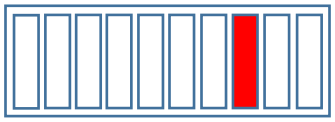
light_LED(2); ➔



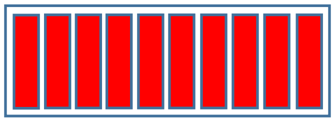
light_LED(3); ➔



light_LED(4); ➔



light_LED(1023); ➔



Your light_LED routine should use various operators (masks and shifts) to manipulate the argument's variable. You should NOT have a series of if/else statements or similar logic.

3.1.3 LED Driver Initialization Subroutine

Recall that just because we are using a different programming language than last quarter, that doesn't mean the peripherals behave any differently than before. We still need to initialize the GPIO ports, which includes enabling the clocks, setting the mode to output, etc. It will be best to do this in a subroutine, such as **`void light_LED_init(void)`** which you will need to call near the beginning of your program. You must create pointer(s) as local variables, assign the I/O register's address to the pointer, then dereference the pointer(s) to modify the I/O register(s). Be sure to read-modify-write when necessary. As mentioned in lecture, there are “better” ways to access registers, but this is the technique we are using now.

3.1.4 Main Logic

Now that you have the support routines in place, tie it together with your main logic. You will need to initialize the LEDs first, then you can enter a loop that takes an integer variable and manipulates it and passes it to **`light_LED`** on each iteration. For this assignment, it will be acceptable to contain logic within main() although later on we may structure our programs a bit differently.

Just to practice, let’s also employ the console. At a minimum, print to the console how many complete cycles of your algorithm has completed at the end of each iteration. We have not talked in detail about printing to the console, but there are numerous examples in lecture and of course, you can refer to the book (The ‘C’ Book, section 9.1.1). Don’t forget to initialize the UART at the beginning of your program as shown in the template program.

You “Knight Rider logic” must employ loops! We have not discussed loops in lecture and may not at all. In ‘C,’ loops are virtually identical to Java, a language with which you have considerable experience.

I suggest two loops, one for when LEDs are sweeping left, and one for when LEDs are sweeping right. You can use bit shift (<< or >>) to modify the pattern on each iteration of the loops.

3.2 Hints and Tweaks

- You will find the debugger to be fully functional and able to inspect the values of variables as well as registers and memory. Use it to verify your expressions.
- You will likely find it necessary to do some common operations on variable/numbers. The regular mathematical operations (**+,-,*,/**) work like you would expect them to. You might also be interested in bitwise OR (**|**), bitwise AND (**&**), bitwise NOT (**~**). There are bit shift operations as well: shift-left (**<<**) and shift-right (**>>**). The C Book covers these in section 2.8.2 accompanied by numerous examples.
- 'C' does not have bit field clear (BFC) or bit field insert (BFI) equivalent operations. Forget they exist, forget you ever learned them. All bit manipulation will need to be done with masking and shifting operations.
- Remember our discussion on prototypes? Assuming all of your code is in one file, the three routines you need to write will either need to appear before main() in the source file, or, you will need prototypes at the top of the source file if the bodies of the methods are below main().
- Don't forget how the LEDs work. There is a gap in the port pin assignments. Refer to the schematic as needed. Your "Knight Rider logic" does not need to know this – all accommodations for this should be handled within light_LED().
- Need more help getting started? A template for this lab's main .c file is provided at the end of this document.
- You very correctly learned in Embedded I to work with peripheral registers by using a "base address" and an "offset." Avoid that for now as you have a 50/50 chance of doing it incorrectly. We will see soon how to do this properly in 'C.'
- We covered working with I/O registers in lecture. A couple of details were omitted. One is that the following line will likely produce a compiler warning:

```
uint32_t* ahb1enr = 0x40023830;
```

  The reason is that the compile know ahb1enr is a pointer-type and can only hold addresses. However, the compiler sees 0x40023830 as a regular integer number, not necessarily an address. So, we have to tell the compiler that we know what we are doing. We do that with a "cast." The following line will work properly:

```
uint32_t* ahb1enr = (uint32_t*) 0x40023830;
```

- Want to turn on a bit? Use an OR mask. The following lines will read from a pointer, modify what has been read by turning on selected bit(s), and write the modified value back to the register:

```
uint32_t temp = *ahb1enr;
temp = temp | 0b10;
*ahb1enr = temp;
```

- Want to turn off a bit? Use an AND mask. The following lines will read from a pointer, modify what has been read by turning off selected bit(s), and write the modified value back to the register:

```
uint32_t temp = *ahb1enr;
temp = temp & (~0b10);
*ahb1enr = temp;
```

3.3 Deliverables

It is intended that you can complete this exercise by the end of the lab period. However, you will have an additional week if needed.

When completed:

1. Submit to Canvas a **single pdf** printout of your completed source code to Canvas. **Include in a comment block at the top of your code a summary of your experience with this project. Your submission to Canvas must be made prior to demoing.**

2. Ask to demo your lab to instructor. You can do this via writing your name on the whiteboard.
   a. If you demo during lab in Week 2, you will earn a 10% bonus on this lab.
   b. If you demo during lab in Week 3, you will be eligible for full credit.

- Demos are ONLY accepted during lab periods. If you are unable to demo by the end of lab in Week 3, you lose the 10% of the assignment attributed to the demo (per syllabus). You should come to lab in Week 3 ready to demo as there will be a new assignment, but there is time to put in a few tweaks if necessary.

- Demos must be ready a reasonable amount of time before the end of the lab period. If you write your name on the board at 9:45 and lab ends at 9:50, and there are five names in front of yours, you will be unlikely to complete your demo by the end of lab and hence lose the bonus or demo points.

3.4 Grading Criteria

For full credit, your solution must:

- Implement the delay_100ms() routine as described.
- Implement the light_LED_init() routine to initialize the appropriate GPIO peripheral to control the dev board's LEDs. **Registers are to be accessed via pointer local variables as discussed in lecture.**
- Implement light_LED() routine to display the provided number on the LEDs as specified. **Registers are to be accessed via pointer local variables as discussed in lecture.**
- Register accesses must utilize Read-Modify-Write when appropriate.
- Implement a "main loop" to output a sequence resulting in lit LEDs bouncing back and forth on LED bar.
  - Loop will need to use a variable to keep track of current position of LED.
  - Loop should use operators to manipulate the variable to calculate next position to be lit.
  - You will need a way to detect if sequence has reached one end or the other and reverse direction.
- Minor errors usually result in a deduction of ~ 3 points (three such errors results in about a letter grade reduction)
- Major errors, such as not achieving a requirement, usually result in a deduction of 5 to 10 points.

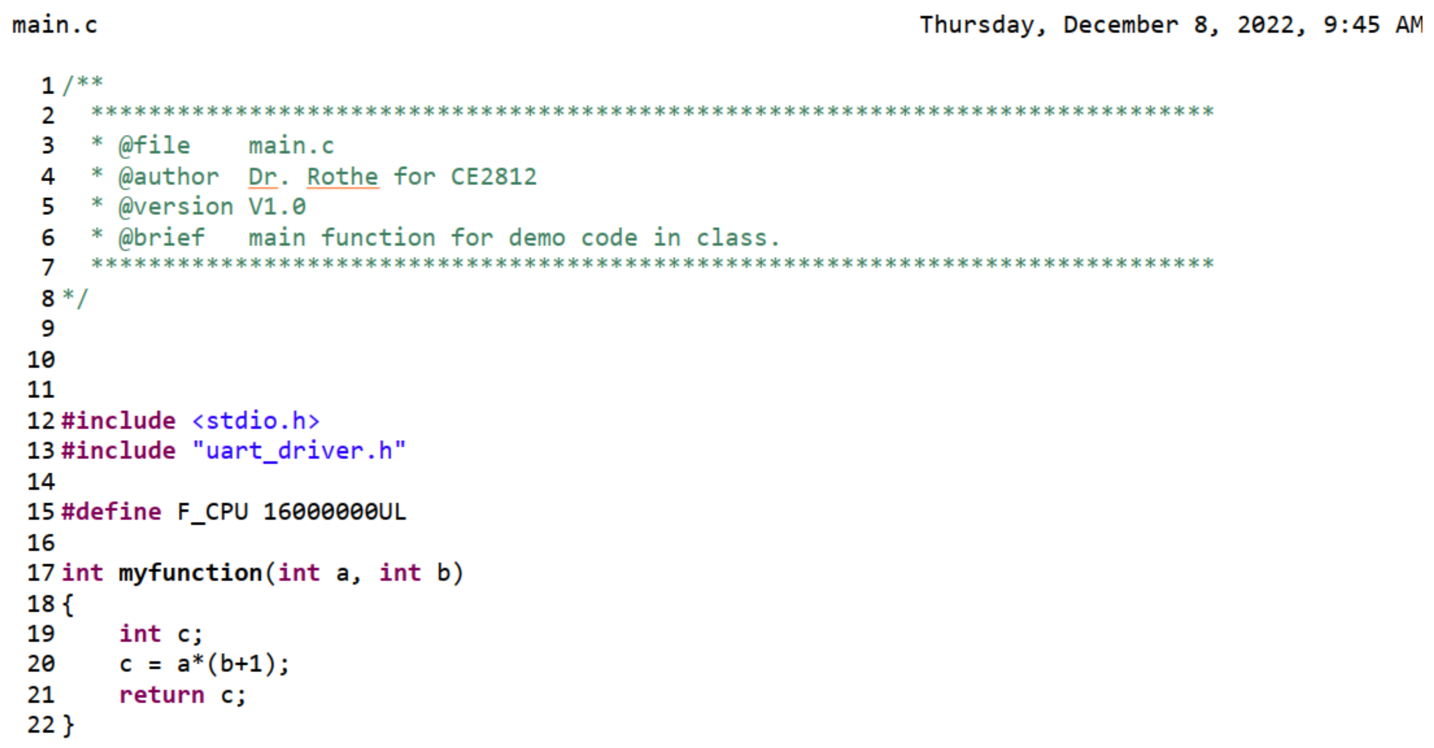
3.5 STARTER-CODE TEMPLATE

```c
#include <stdio.h>
#include "uart_driver.h"

#define F_CPU 16000000UL

/* need banner comment here */
void delay_100ms(void)
{
	// busy wait
}

/* need banner comment here */
void light_LED_init(void)
{
	// enable clock to GPIOB peripheral
	// set PB5 - PB15 to output
}

/* need banner comment here */
void light_LED(xxx number)  // replace xxx with a proper type
{
	// write number to PB5 - PB15 (skipping PB11)
}

/* need banner comment here */
int main(void){
	// get console going for user interaction
	init_usart2(57600,F_CPU);

	// contain main logic in "forever loop"
	for(;;)
	{

		// your main logic goes here

	}

	// never get here but compiler will complain without return
	return 0;
}
```

3.6 Lab Submissions

Please review your lab submissions before submitting. A couple of things:

#1 - CubeIDE can print code to pdf and by default and it looks pretty good. If you have line numbers enabled in CubeIDE, they will will be on the printout, which I appreciate as it allows me to be specific when I point or good or bad lines of code. The default will be colorized as well. All of this will help me read your code and be happy while reading it. Example:



```
main.c                                                    Thursday, December 8, 2022, 9:45 AM

 1 /**
 2   ******************************************************************************
 3   * @file    main.c
 4   * @author  Dr. Rothe for CE2812
 5   * @version V1.0
 6   * @brief   main function for demo code in class.
 7   ******************************************************************************
 8 */
 9
10
11
12 #include <stdio.h>
13 #include "uart_driver.h"
14
15 #define F_CPU 16000000UL
16
17 int myfunction(int a, int b)
18 {
19     int c;
20     c = a*(b+1);
21     return c;
22 }
```

this is the default, and is perfectly fine. Not sure why some deviate...

#2 - CubeIDE has a "dark mode" which is pleasing enough and many of you are using it. However, the color pdfs it produces are somewhat unreadable:

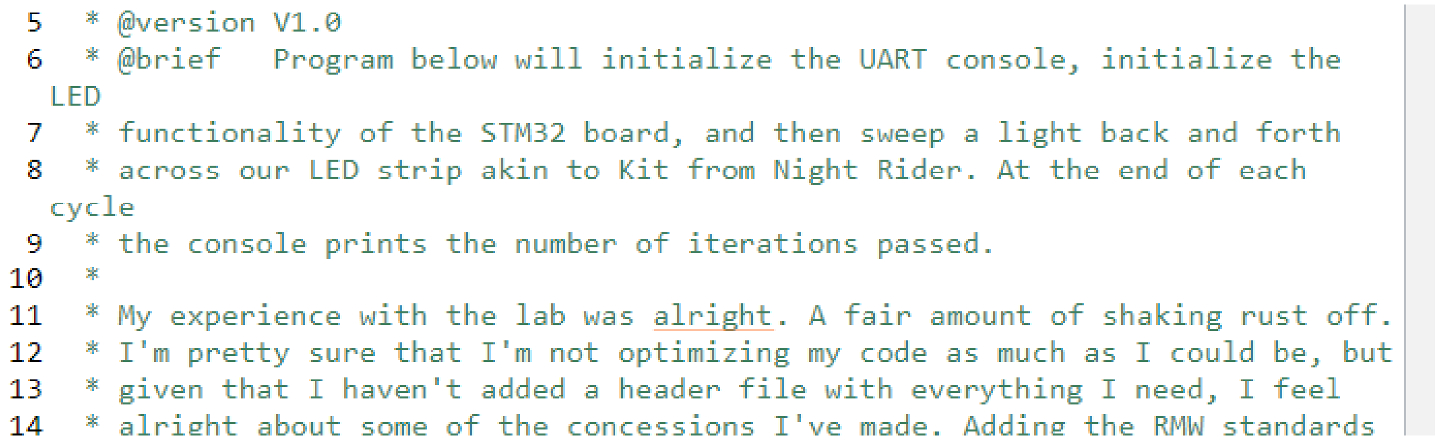
main.c

```c
/**
  ****************************************************
  * @file    main.c
  * @author  Dr. Rothe for CE2812
  * @version V1.0
  * @brief   main function for demo code in class.
  ****************************************************
*/

#include <stdio.h>
#include "uart_driver.h"

#define F_CPU 16000000UL

int myfunction(int a, int b)
{
    int c;
    c = a*(b+1);
    return c;
}


// main
int main(void)
{
    // call initialization function here
```

I cannot see the curly braces and certain other features. THIS IS NOT ACCEPTABLE.

I would not prefer b&w code either… Black & white make it hard for me to distinguish comments and various types. Since the default output (on non-dark theme) is nicely colorized, that is my minimal expectation.

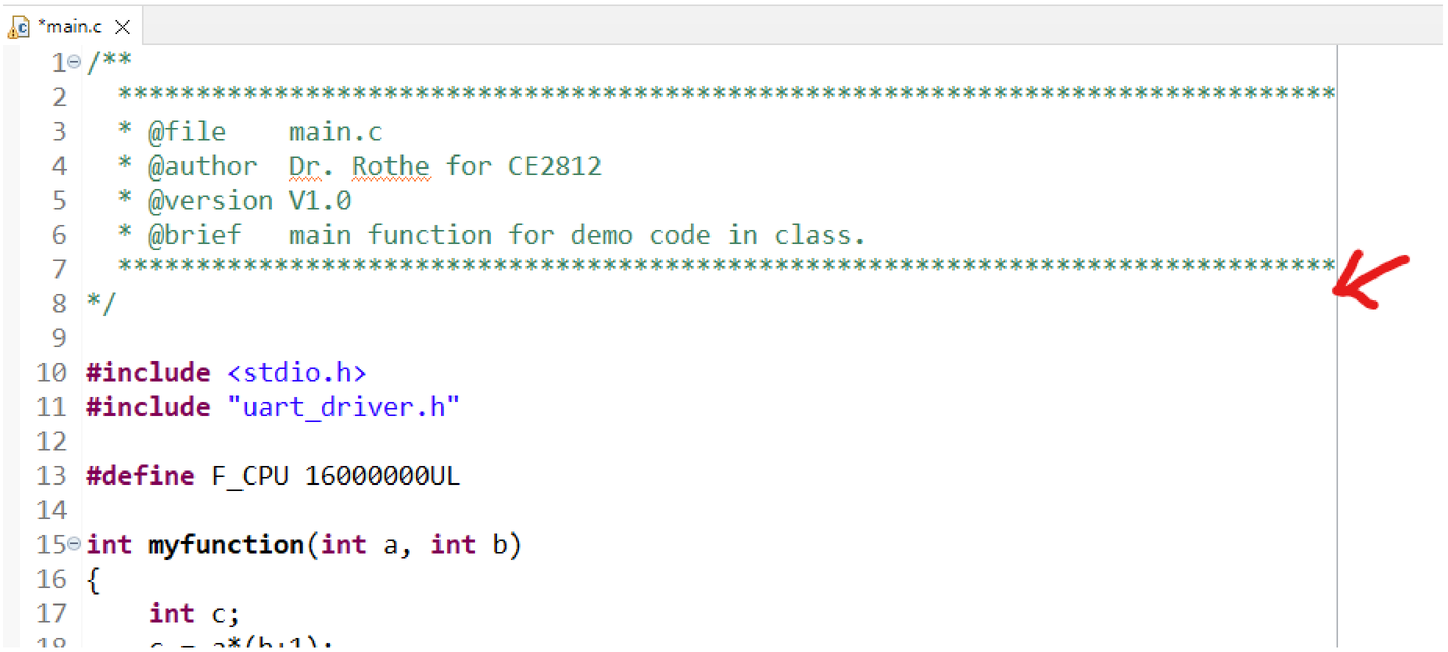
Please also make sure you do not have long lines, especial in block comments. If you do, they wrap to the next line and make the comment block difficult to read. For example:



```c
  * @version V1.0
  * @brief   Program below will initialize the UART console, initialize the
LED
  * functionality of the STM32 board, and then sweep a light back and forth
  * across our LED strip akin to Kit from Night Rider. At the end of each
cycle
  * the console prints the number of iterations passed.
  *
  * My experience with the lab was alright. A fair amount of shaking rust off.
  * I'm pretty sure that I'm not optimizing my code as much as I could be, but
  * given that I haven't added a header file with everything I need, I feel
  * alright about some of the concessions I've made. Adding the RMW standards
```

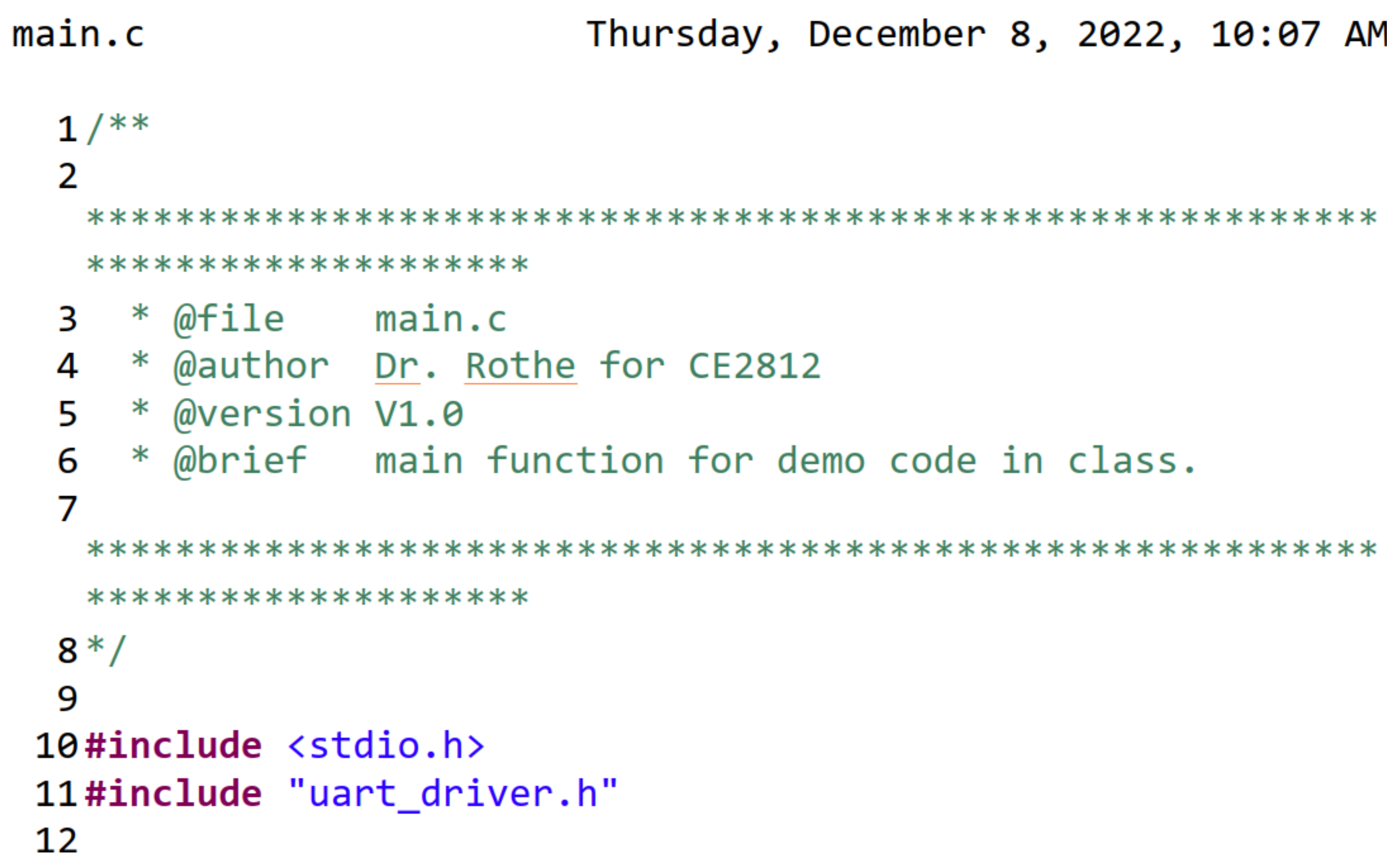
Lines 6 and 8 are too long and break to the next lines making it somewhat difficult to read quickly. Simple solution - look at your pdf before you submit it and make sure it is work that you are proud of.

Incidentally, there is a setting for the editor to place a line at the print margin (Windows->Preferences->General->Editors->Text Editor) to Show print margin. Enabling that will place a line in the editor that your lines should not cross.

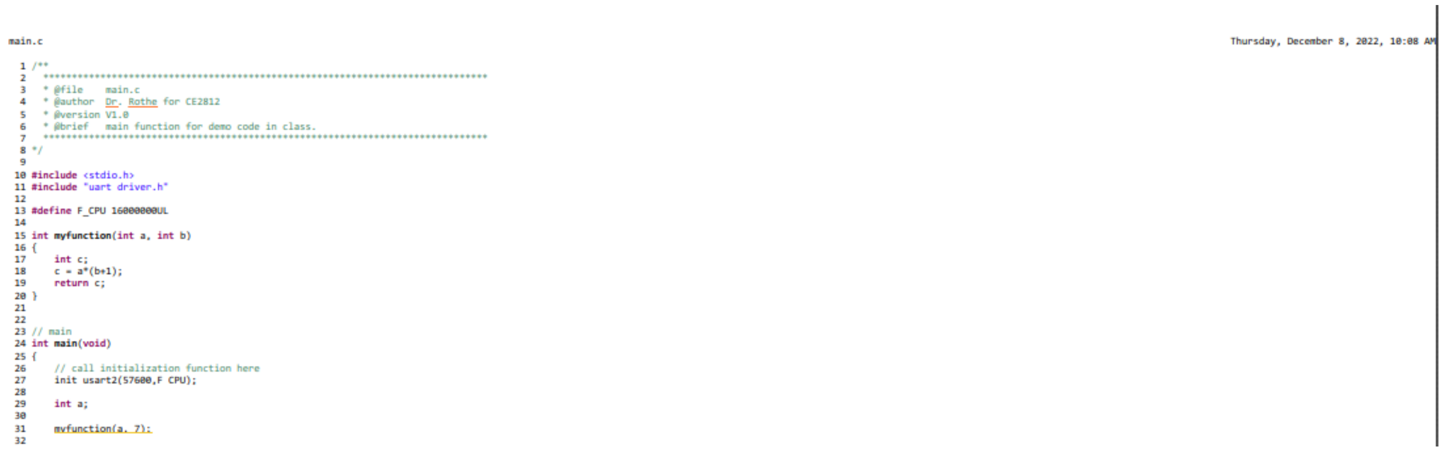


```
 1 /**
 2   ******************************************************************************
 3   * @file    main.c
 4   * @author  Dr. Rothe for CE2812
 5   * @version V1.0
 6   * @brief   main function for demo code in class.
 7   ******************************************************************************
 8 */
 9
10 #include <stdio.h>
11 #include "uart_driver.h"
12
13 #define F_CPU 16000000UL
14
15 int myfunction(int a, int b)
16 {
17     int c;
```

Oh, and one more. If you are zoomed in or out of your editor window in CubeIDE (ctrl-+ or ctrl- -), that scale factor will be applied to your printout. Cannot imagine why... Below, same source file zoomed in then out:



```
main.c                    Thursday, December 8, 2022, 10:07 AM

/**
 *************************************************************
*******************
 * @file    main.c
 * @author  Dr. Rothe for CE2812
 * @version V1.0
 * @brief   main function for demo code in class.
 *************************************************************
*******************
*/

#include <stdio.h>
#include "uart_driver.h"

```



```
main.c                                                    Thursday, December 8, 2022, 10:08 AM

/**
  ******************************************************************************
  * @file    main.c
  * @author  Dr. Rothe for CE2812
  * @version V1.0
  * @brief   main function for demo code in class.
  ******************************************************************************
*/

#include <stdio.h>
#include "uart driver.h"

#define F_CPU 16000000UL

int myfunction(int a, int b)
{
    int c;
    c = a*(b+1);
    return c;
}


// main
int main(void)
{
    // call initialization function here
    init usart2(57600,F CPU);

    int a;

    myfunction(a, 7);

```

Non-100% zoom will potentially trigger word wrap when scaled up or tiny code if scaled down. How can you prevent this? Look at your pdf before submitting. If it is not easily readable, fix it.