ECEN 3320-002
Assembly Language Programming
Lab Assignment # 3.1

Cameron Biniamow
University of Nebraska-Lincoln
Department of Electrical and Computer Engineering
Peter Kiewit Institute
Due: 11/06/2020

Summary

Lab 3.1 introduces ARM assembly through the STM32F103C8 Blue Pill Development Board. This lab focuses on the development tools and therefore the ARM assembly code is provided. Three activities are included in Lab 3.1 and run through the hardware/software setup, simulating, and live debugging on the Blue Pill. The program provided will toggle an LED on the Blue Pill on and off. The programming, simulating, and uploading to the Blue Pill is completed through the Keil µVision5 software.

Background

The STM32F103C8 Blue Pill, based on an ARM Cortex-M3 processor. The Blue Pill is powerful microcontroller with a 32-bit CPU. ARM registers are 32-bits, which differ from previously used microcontrollers. ARM uses 15 registers, R0 – R15, where R0 – R12 are general purpose registers. ARM uses assembly directives to instruct the assembler. These directives are as follows: AREA, END, EQU, and INCLUDE.

For Lab 3.1 the ARM assembly code is already provided. The objective of the provided code is to control and toggle an LED (PC13) on the Blue Pill. This can be seen directly on the board following the uploading of the code or during simulation can be seen in the "Peripherals" under GPIOC_ODR. This allows the user to view the input/output of the ports.

Procedure

Activity 1 introduces the hardware and software setup in order to write a program and upload to the Blue Pill. Initially, to use the ST-Link V2 programmer, it is connected to the PC being used. Using the Device Manager, the programmer is detected and verified by the PC. Upon the programmer being verified, the Keil ARM-MDKIDE is installed. This is performed by visiting the Keil website and locating the MDK-ARM package. Once the installation finishes, the Pack Installer opens automatically. Within the Pack Installer, STMicroelectronics is selected followed by selecting STM32F1 Series. In the right side of the window, Keil::STM32F1xx_DFP is located and the Install button directly to the right is selected. Once the installation is complete, the install button will change to "Up to Date". Finally, the contents of the CMSIS include directory is copied into the Device include directory. This is performed by locating the CMSIS include directory with a path similar to the following: "C:\Users\...\AppData\Local\Arm\Packs\ARM\CMSIS\5.5.1\CMSIS\Include". The Device include directory path is similar to the following:

“C:\Users\...\AppData\Local\Arm\Packs\Keil\STM32F1xx_DFP\2.3.0\Device\Include”. At this point, the Keil µVision5 software is installed on the PC.

Activity 2 begins work with the Keil µVision5 software and simulates an ARM assembly project. In order to create a new project, the Keil IDE is opened and from the Project menu, New uVision project is selected. A new folder is created and labeled with a relevant name. The project is named similar to the folder name and the project is saved. The “Select Device for Target ‘Target 1’…” window opens, and the STMicroelectronics tree is expanded. STM32F103 is located then selected and the window is closed. Within the Project window on the left side of the Keil IDE, right click on the “Source Group 1” folder and select “Add New Item to Group ‘Source Group 1’…”. Select “Asm File (.s)” and name the file “main” followed by clicking the “Add” button. Since this lab is intended to focus on the simulation and debugging of the Blue Pill, the assembly source code is already provided. This code is simply copied and pasted into the Keil IDE and saved. The program is then built and given that there are not any warnings or errors, setup for simulation begins. Simulation is setup by right clicking on the “Target 1” folder in the Project window and selecting the “Options for ‘Target 1’…” button. A new window appears, and the “Debug” tab is selected followed by selecting the “Use Simulator” option. Debugging begins by selecting the “Start/Stop Debug Session” button and the simulator is then running. As the simulator is run, the following error is received:

“*** error 65: access violation at 0x41004480: no ’write’ permission.”

Within the command window, the following command is entered after receiving the error:

MAP 0x40011000, 0x400113FF READ WRITE

In order to ensure that the simulation is operating as expected, the RCC and PORT C peripheral windows are opened. This is completed through the Peripherals – System Viewer – RCC and Peripherals – System Viewer – GPIO – CPIOC. A breakpoint is set at the instruction “LDR R1,=GPIOC_ODR” and the code is simulated.

For Activity 3, the program simulated in Activity 2 is uploaded to the Blue Pill for live debugging. This is performed by opening a new project in Keil uVision 5 and selecting the STM32F103C8 device. After pressing “OK”, the “Manage Run-Time Environment” window opens, and the “Device” tree is clicked. The “Startup” option is located and the box to the right is checked. Now the same steps listed in Activity 2 are followed in order to create the main file. In order to upload the program to the Blue Pill, the ST-Link is used. The ST-Link connects to the

PC through the USB socket and the pins SWCLK, SWDIO, GND, and 3.3V connect from the ST-Link to the Blue Pill. On the Blue Pill, the jumpers are set to 0 while programming. For the uploading of the program, the Projects tab is opened and the "Options for Target 'Target 1'…" button is clicked. Under the "Debug" tab, the "Use" option is checked, "ST-Link Debugger" is selected and the "Settings" button is clicked. Under the "Flash Download" tab, the "Reset and Run" option is checked. The program is built, and the "Download" button is pressed to upload the program to the Blue Pill.

Results

Activity 1 required the download and configuration of the Keil µVision5 software. Through following the provided material, the software was downloaded properly along with the ARMMDK-IDE package.

Activity 2 introduced simulation of ARM assembly code through the Keil software. After the provided code was copied into the Keil software and compiled, no warnings or errors were present. The simulation began by observing the values and statuses of the registers and peripherals. As can be seen in Figures II – V, it was determined that the program operated as expected and without error as GPIOC_ODR register toggled a HIGH and LOW output.

Activity 3 used the program simulated in Activity 2 and uploaded it to the Blue Pill for testing. Through following the provided material and the procedure listed in this report, uploading of the program to the Blue Pill was properly completed and without error. As can be seen in Figures VI – X, the PC13 LED on the Blue Pill toggled ON and OFF after uploading the program.

Through the completion of all three activities, it was determined that the listed procedure in this report provides the correct information needed to achieve proper downloading of the Keil µVision 5 software, simulation of ARM assembly code, and uploading of a program to a Blue Pill. All results achieved were expected and obtained without problem or error. It can be determined that Lab 3.1 was performed successfully.

Answers to Posted Questions

1. What is the default operating frequency of the STM32F103C8?
    - 72 MHz
2. What is the maximum operating frequency of the STM32F103C8?
    - 72 MHz

3. Consult the STM32F103C8 datasheet provided by ST.
    - How wide (in bits) are the I/O ports on this specific device?
        - 16-bits
    - How many PORTS are physically exposed to the user on this device (by way of its package pins?) 1 port? 2 ports? 3 ports? 4 ports?
        - 3 ports
4. How many registers are available to the programmer on the ARM Cortex-M3? List them.
    - 13 registers. R0 – R12.
5. What is so special about registers R13, R14, and R15?
    - R13 – Stack Pointer: Holds the address of the stack in memory
    - R14 – Link Register: Subroutine return address link register
    - R15 – Program Counter: Holds address of instruction being fetched
6. In the sample source code provided in the APPENDIX section there's a procedure called delay. Curiously, upon taking a closer look, you'll notice there's no CALL instruction and no RET instruction. Then, explain how exactly do you:
    - Invoke a procedure the 'ARM way'?
        - A procedure is "CALLED" by using the BL instruction followed by the name of the procedure. This instruction branches and links to the procedure that is being called.

          ```
          BL "Procedure Name"
          ```
    - 'Return' from a procedure the 'ARM way'?
        - Returning from a called procedure is performed through the following instruction:

          ```
          BX LR
          ```

          This instruction branches and exchanges instruction sets, which returns back to where the procedure was originally called.
7. Consult the STM32F10XXX datasheet. All of the I/O-port related special function registers (SFRs) are memory mapped to a region called 'AFB2'. Given this:
    - What is the starting address of all these registers (for AFB2)?
        - ADC3: 0x40013C00
        - USART1: 0x40013800

        - TIM8 timer: 0x40013400
        - SPI1: 0x40013000
        - TIM1 timer: 0x40012C00
        - ADC2: 0x40012800
        - ADC1: 0x40012400
        - GPIO Port G: 0x40012000
        - GPIO Port F: 0x40011C00
        - GPIO Port E: 0x40011800
        - GPIO Port D: 0x40011400
        - GPIO Port C: 0x40011000
        - GPIO Port B: 0x40010C00
        - GPIO Port A: 0x40010800
        - EXTI: 0x40010400
        - AFIO: 0x40010000
    - Within the AFB2 region, what is the starting address for all registers that are specifically related to Port B?
        - GPIO Port B: 0x40010C00
8. In the sample source code provided in the APPENDIX:
    - What does the AREA directive do?
        - Instructs the assembler to assemble a new code or data area.
    - How many such 'areas' are in the sample source code provided?
        - Three
9. When Cortex-M3 resets, what is the first address the processor uses to start executing instructions?
    - 0x0800004C
10. Provide further information on the following:
    - What is the Thumb instruction set?
        - A subset of the most commonly used 32-bit ARM instructions. The instructions are 16-bits and have 32-bit corresponding ARM instructions.
    - How is it different from [classical] ARM instruction set?

        - Thumb instructions are 16-bits versus the 32-bit ARM instructions.
- Why is one better than the other? Compare and contrast.
    - Thumb offers a long branch range and large address space. Additionally, Thumb is roughly 65% of the size of ARM code with 160% of the performance of ARM code.

Conclusion

Lab 3.1 introduces the ARM architecture and becoming familiar with the STM32F103C8 Blue Pill Development Board**.** This lab consisted of three activities which run through the setup and configuration of the Keil µVision5 software with the ARMMDK-IDE package, program simulation, and uploading a program to the Blue Pill. Through the completion of the three activities, valuable information was gained as to how to use the ARM instruction set, controlling the simulator, and uploading a program to the Blue Pill. It was determined that through implementation and testing, the program simulated and uploaded to the Blue Pill as expected and without error. Following the uploading of the program to the Blue Pill, proper operation was demonstrated as the PC13 LED toggled ON and OFF. All understandings gained within Lab 3.1 will prove to be useful in future labs in ECEN 3320 as well as future courses in the program.

Appendix

Figure I: ARM Assembly Source Code

```
;=================================================================
;  CAMERON BINIAMOW
;  10/30/2020
;  Lab 3-1
;  Description: A single file tutorial for blue pill programming
;            that will toggle the on-board LED ~1 time per second
;
;    Note: Be wary of indentiation. ARM Labels must be located
;        on the left margin, while instructions and directives
;        must be indented.
;=================================================================

;=======================Port Locations============================
RCC_APB2ENR EQU 0x40021018         ;Clock enable register

GPIOC_CRL     EQU 0x40011000      ;CRL = Control Register Low (Pins 0-7)
GPIOC_CRH     EQU 0x40011004      ;CRH = Control Register High (Pins 8-15)
GPIOC_ODR     EQU 0x4001100C      ;ODR = Output Data Register

;These are for the linker
       EXPORT Reset_Handler
       EXPORT __Vectors

;=======================Vector Area===================================
; Area will define the name of area, type (code or data), and access
; (READONLY, READWRITE)
       AREA VECTORS, DATA, READONLY
       THUMB
; When the controller boots, it reads the location for the top of stack
; (our stack pointer) at 0x0000 0000 and begins code execution at the
; address contained in 0x0000 0004. Therefore, we must define constants
; (DCD) with these values.
__Vectors
       DCD    0x20000190          ;Points to top of stack
       DCD    Reset_Handler       ;Points to our reset location

;=======================Startup Area==================================
; Here we provide the restart behavior of the device written with the
; THUMB instruction set
       AREA STARTUP, CODE, READONLY
       THUMB
; The reset handler can be used for many things, but our main focus
; is to enter the main area of our code
Reset_Handler    PROC
       ;-------------------------------------------
       ; ADDITIONAL RESET INITIALIZATION GOES HERE
       ;-------------------------------------------
       LDR    R5, =__main
       BX     R5                  ;Branch to our main code
       ENDP

;=========================MAIN CODE===================================
       AREA MAIN, CODE, READONLY
       THUMB
;Now that our setup is complete, lets get to the task at hand!
__main
       LDR    R1,=RCC_APB2ENR     ;Enable all of our GPIO clocks
       LDR    R0,[R1]
       ORR    R0,R0,#0xFC
       STR    R0,[R1]
```

```
        LDR    R1,=GPIOC_CRH       ;Each pin has a nibble in the control register
        LDR    R0,=0x44344444
        STR    R0,[R1]

LOOP
        LDR    R1,=GPIOC_ODR
        LDR    R0,[R1]             ;Move contents of ODR to R0
        EOR    R0,R0,#0x2000       ;Toggle value at position 13
        STR    R0,[R1]             ;Move contents of R0 to ODR
        BL     DELAY
        B      LOOP                ;Infinite loop

DELAY
        LDR    R0,= 50
L1      LDR    R1,= 50000          ; Play with these values to change delay time
L2      SUBS   R1,R1,#1
        BNE    L2                  ; Branch until R1 is 0
        SUBS   R0,R0,#1
        BNE    L1                  ; Branch until R0 is 0
        BX     LR                  ; Branch and exchange instruction set (return)

        NOP                        ;To remove alignment warning. Disregard
        END
```

Figure II: Activity 2 - ARM Simulation - LED OFF

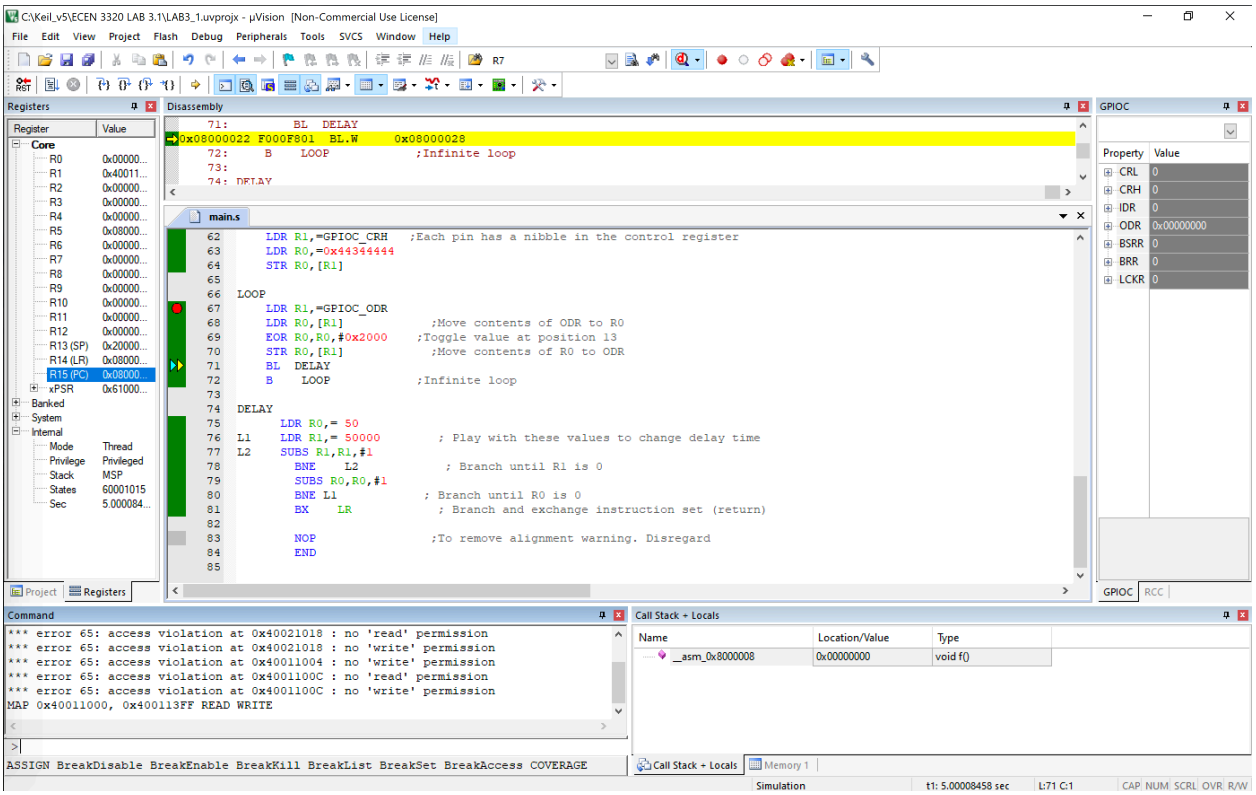


Figure III: Activity 2 - ARM Simulation - LED ON

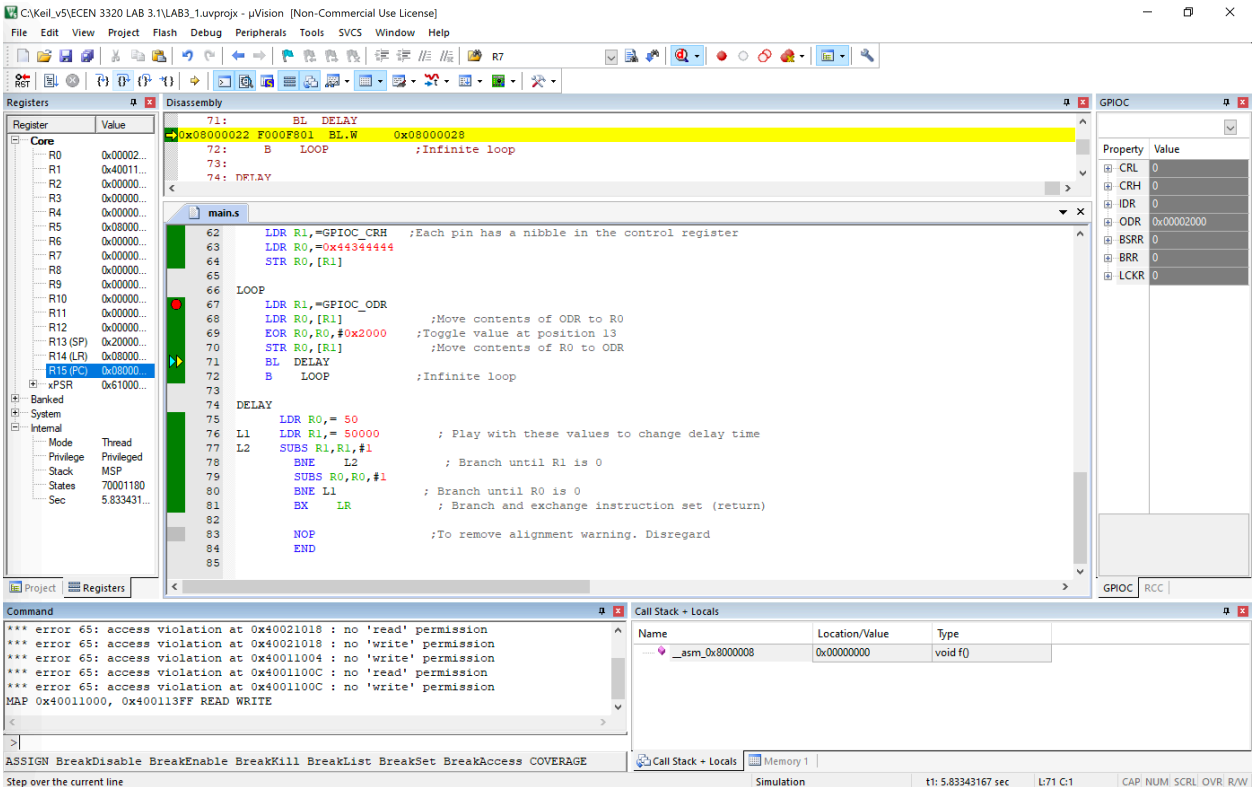


Figure IV: Activity 2 - ARM Simulation - LED OFF

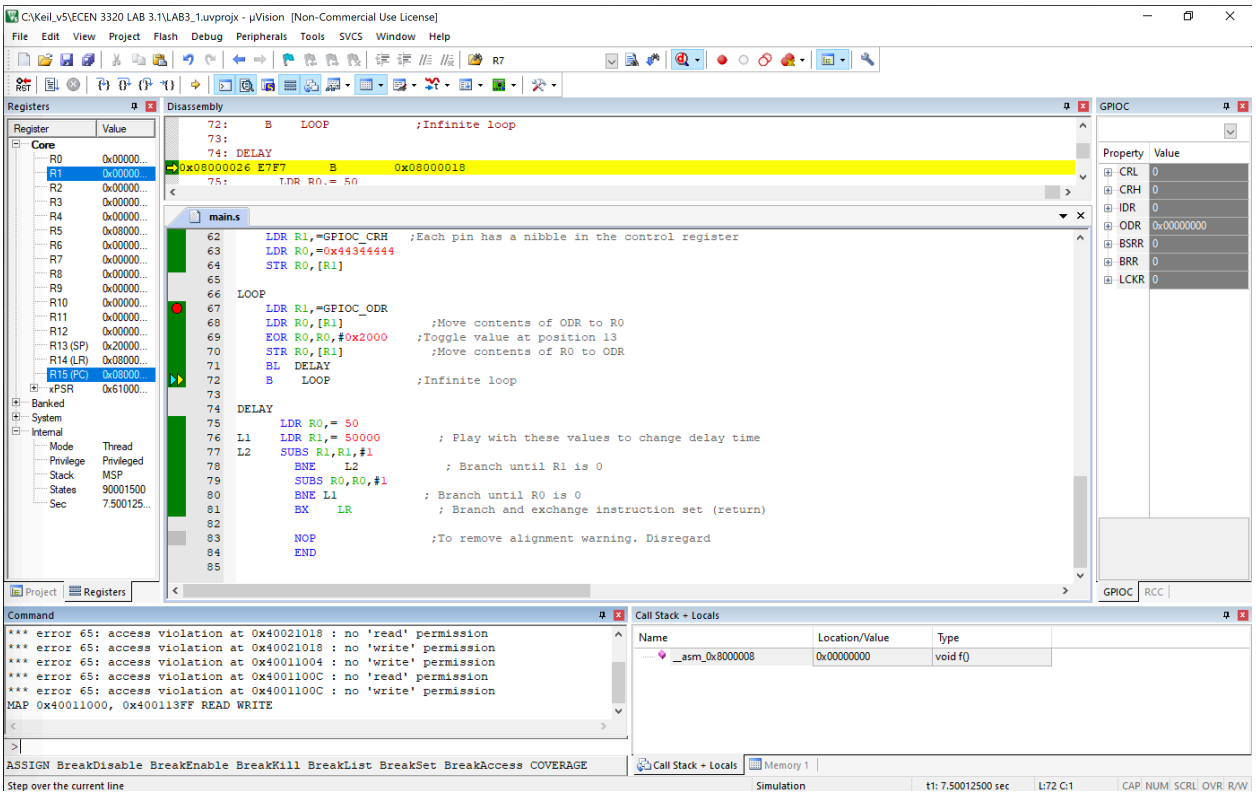


Figure V: Activity 2 - ARM Simulation - LED ON

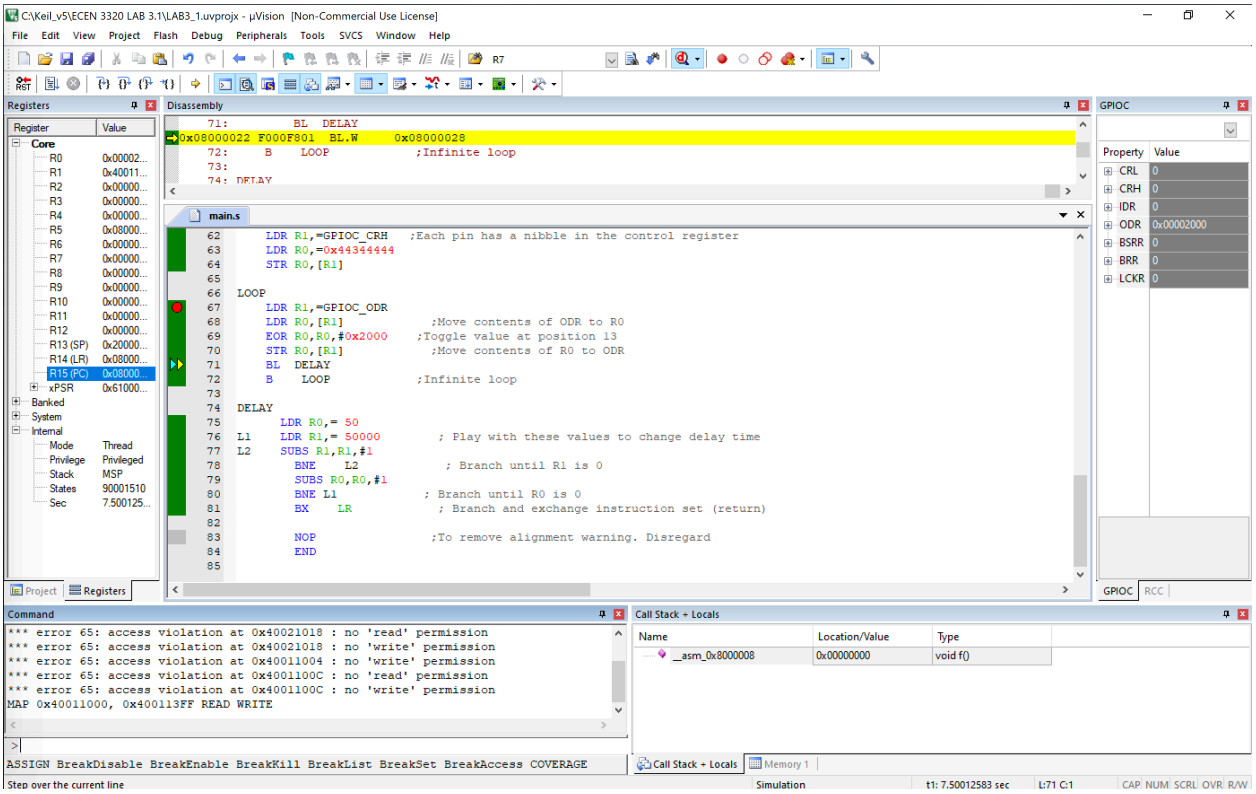


Figure VI: Activity 3 - Live Debugging on the Blue Pill - PC13 LED OFF

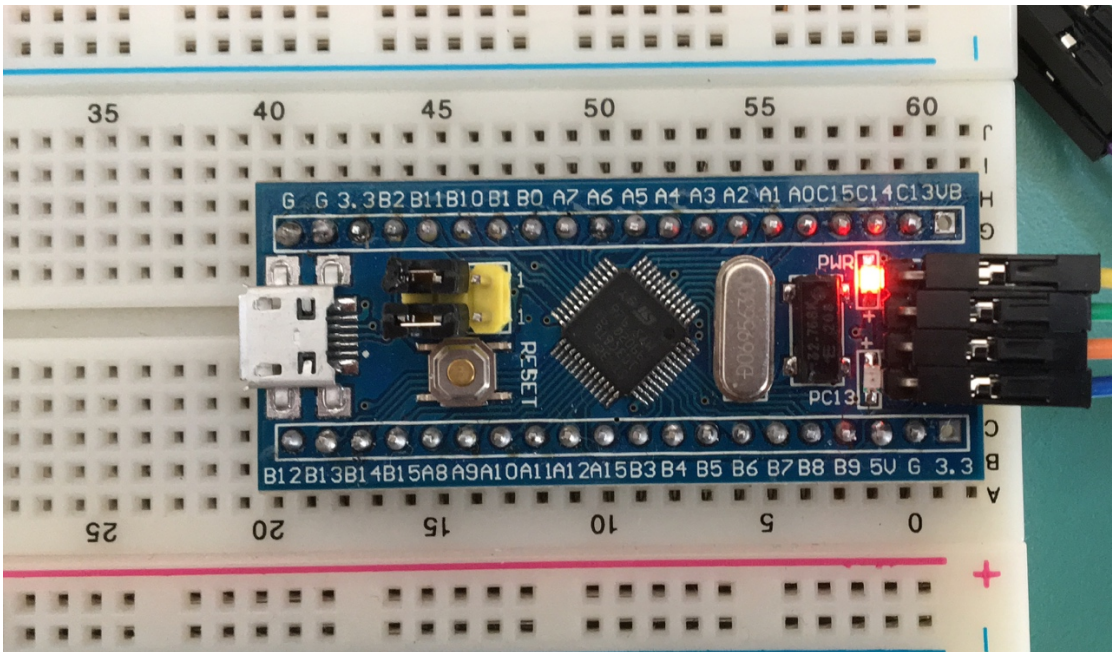


Figure VII: Activity 3 - Live Debugging on the Blue Pill - PC13 LED ON

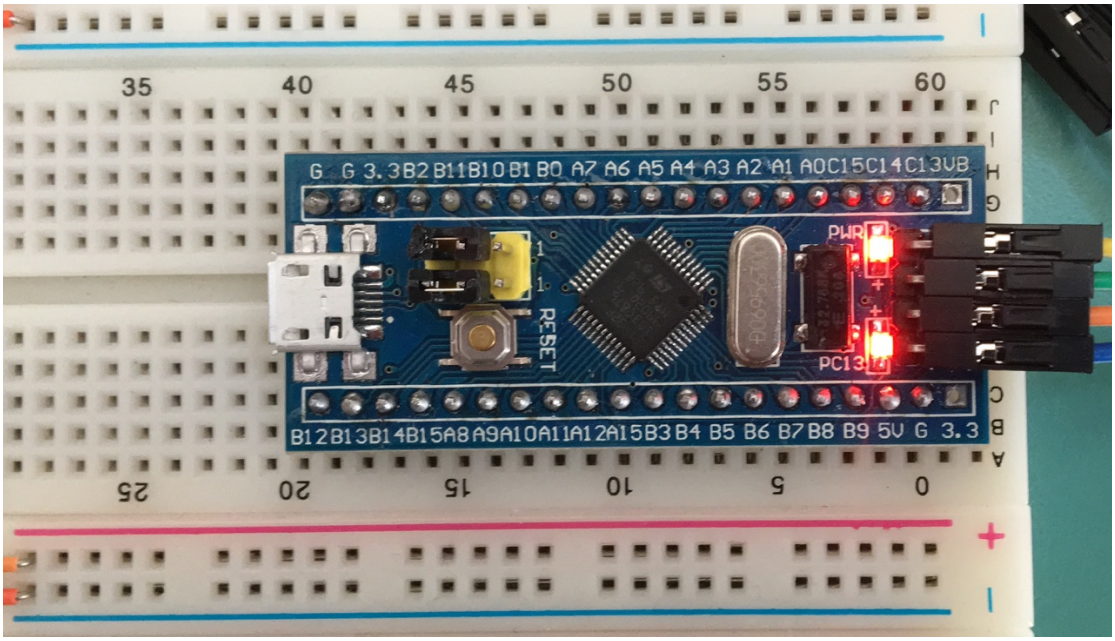


Figure VIII: Activity 3 - Live Debugging on the Blue Pill - PC13 LED OFF

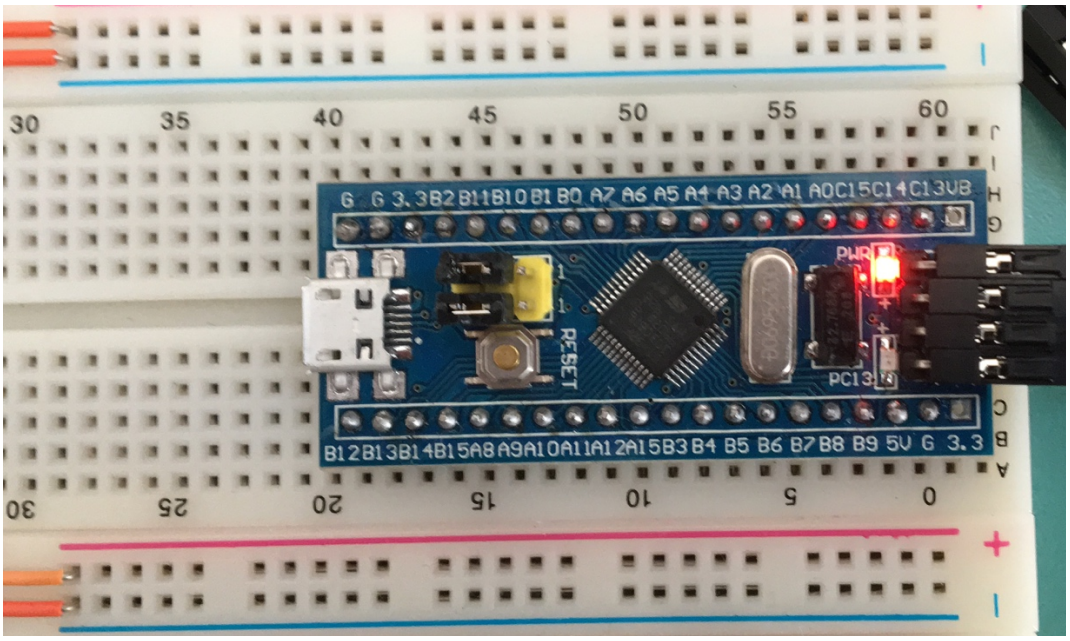


Figure IX: Activity 3 - Live Debugging on the Blue Pill - PC13 LED ON

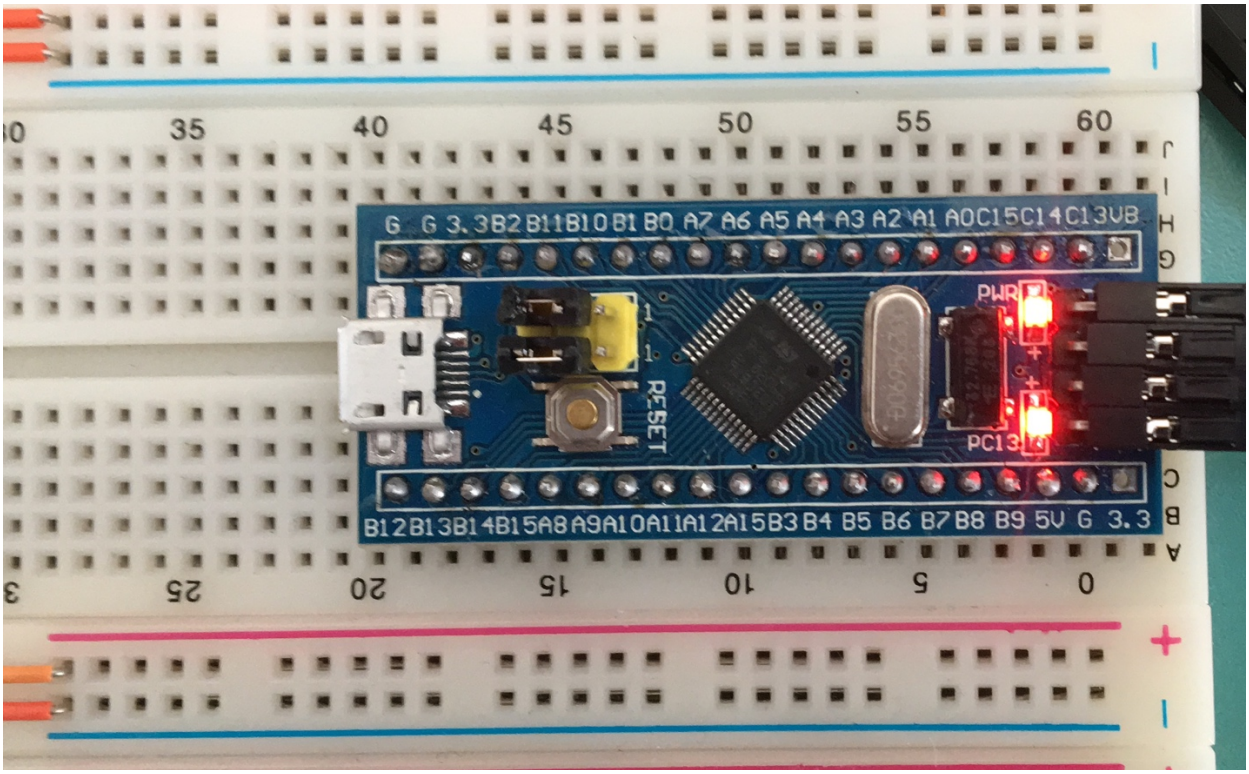
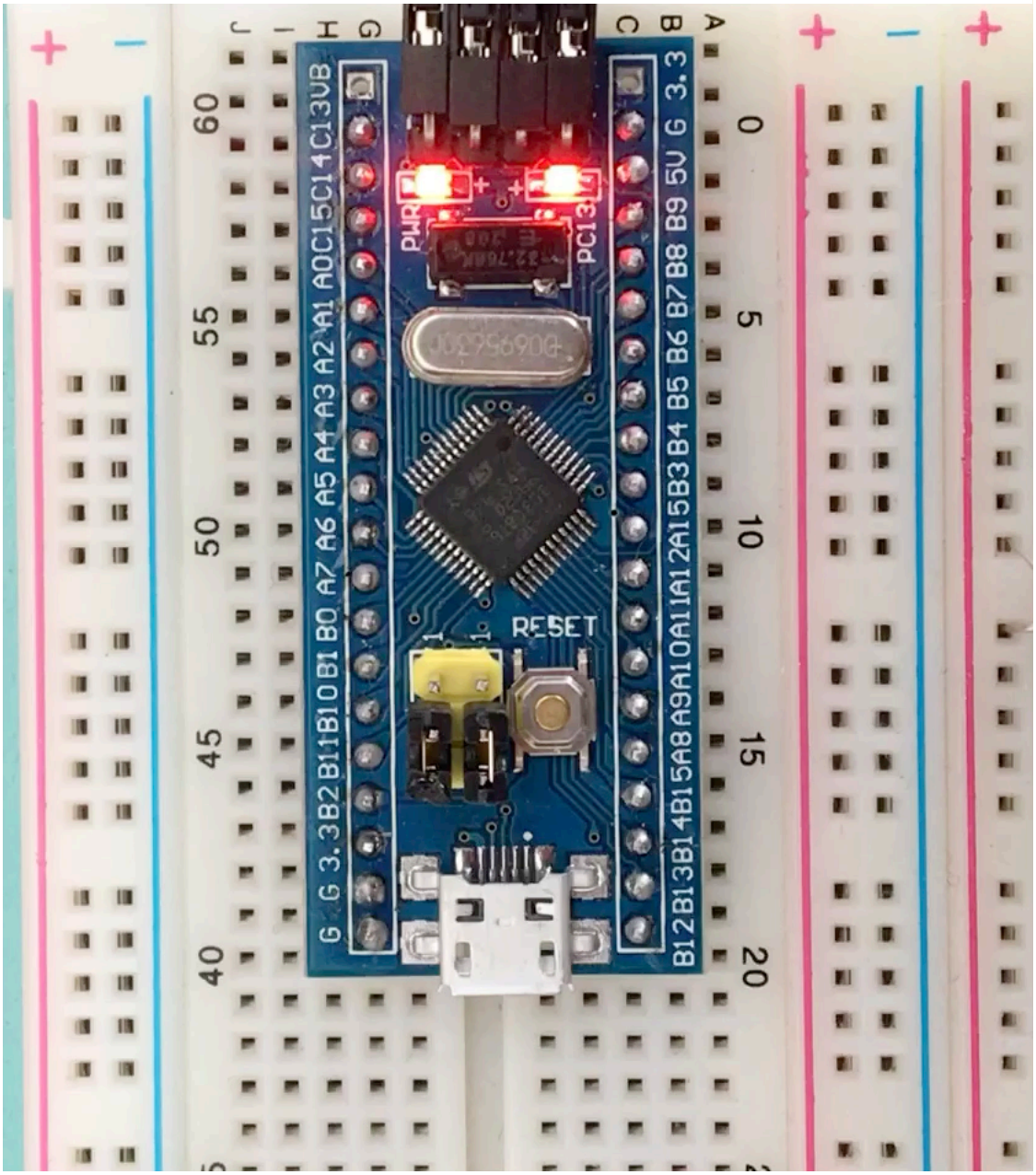


Figure X: Activity 3 - Live Debugging on the Blue Pill – Video



CAMERON BINIAMOW

"LAB 3.1"

ECEN 3320

STM32F103C8

LABELS ON LEFT

INSTRUCTIONS TABBED IN

ST-LINK PINS

+3.3V

GND

SWDIO

SWCLK

TOTAL TIME

3.5 HOURS