Lab 4:
Introduction to Microcontroller Programming

ELEC1710

1 Introduction

In this lab you will learn how to compile and debug code on the STM32F103C8T6 development board within the System Workbench development environment.

System Workbench is a free and open source software package which combines the Eclipse IDE (integrated development environment) with the GNU compiler toolchain (compiler, linker, assembler, etc) and comes pre-configured for STM32 software development.

It is available for download at `http://openstm32.org`, a free registration is required prior to download. The website also hosts a wiki and forums which can be useful sources of information, especially when encountering errors.

In this lab your primary tasks are as follows:

1. Load a project into System Workbench
2. Configure the debugging environment
3. Compile the project
4. Debug the code using various execution control tools

2 Knowledge Reference

1. Use of System Workbench is covered in a video on Blackboard. **Watch this video**.
2. Be aware that System Workbench uses development tools written and maintained by the GNU project:
   - GCC - The GNU C Compiler
   - GDB - The GNU Debugger
   - Make - A compilation control scripting language written by GNU
   - OpenOCD - The tool used to communicate with the ST-Link debug adapter
3. Detailed information about the supplied STM32 development board can be found on this wiki (mostly Arduino specific): `http://wiki.stm32duino.com/index.php?title=Black_Pill`
4. The schematic is here: `http://wiki.stm32duino.com/images/5/52/Black_Pill_Schematic.pdf`
5. The pinout of the board is labeled in white text along the edge of the PCB as follows:
   a) G is ground
   b) V3 is +3.3V
   c) A0, A1, etc are GPIOA pins 0, 1, ..., etc
6. Full reference data for the STM32F103C8T6 chip can be found here: `http://bit.ly/2xNgAZX`
   a) Refer to the programming manual (PM0056) for an Assembly reference: `http://bit.ly/2miQ9cA`

b) (Advanced) Refer to the datasheet for pinouts, memory map, etc: http://bit.ly/2nXTNox

c) (Advanced) Refer to the reference manual (RM0008) for detailed information about internal peripherals (configuration registers, detailed functionality etc) http://bit.ly/2nEuP1k

3 Procedure

3.1 Hardware Configuration

1. Connect the ST-Link debugger to the STM32 microcontroller. The debugger is packaged with four jumper wires requied to make this connection. The four connections are labeled on the ST-Link and development board as follows:

   - SWCLK - CLK
   - SWDIO - IO
   - GND - G
   - 3.3V - V3 (Do not connect to 5V)

   NB: The pins on the development board are NOT in the same order as the pins on the ST-Link.

   DO NOT CONNECT TO THE 5V PIN. THIS WILL DESTROY THE STM32.

   The pin connections are shown, with the same colour coding, in Figure 2. If you are not certain that you have the pins correctly wired please ask your demonstrator to check. Plugging into the USB with the incorrect connections can destroy the STM32.

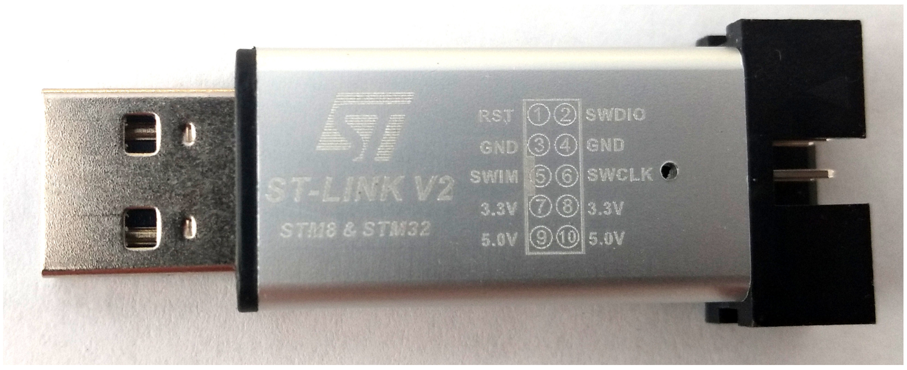
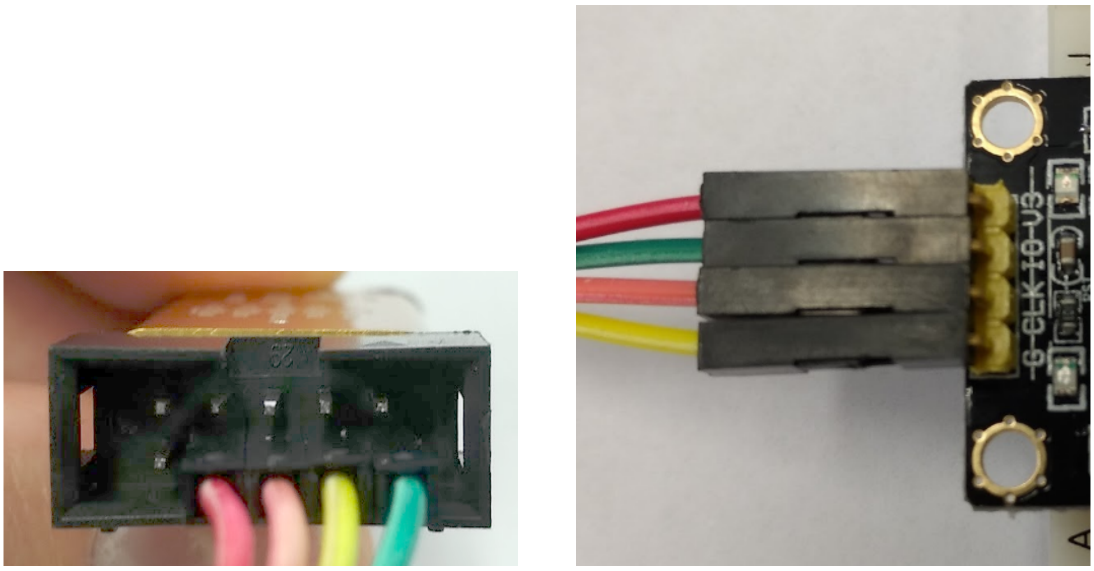


Figure 1: The Chinese ST-Link V2 debug adapter. Note the pinout specification written on the case.

2. Connect breadboard GND to one of the STM32 microcontroller's G pins. There are two of these pins, pick whichever is more convenient.
3. Connect a red LED in series with a 330 Ω resistor from pin C13 to GND. Remember that the longer LED pin is the anode (positive) pin and should be facing the GPIO pin.
4. Plug the ST-Link into a free USB port.



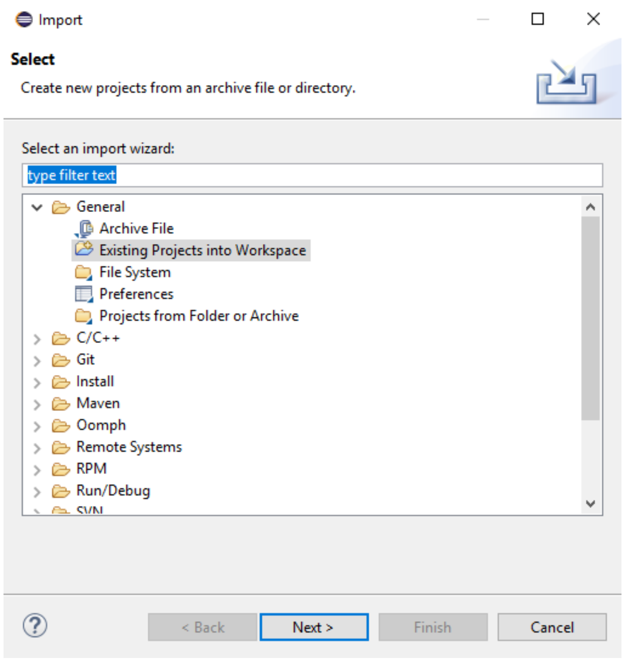
(a) Rear view of ST-Link (b) Development board debug header

Figure 2: Development board debugger connection

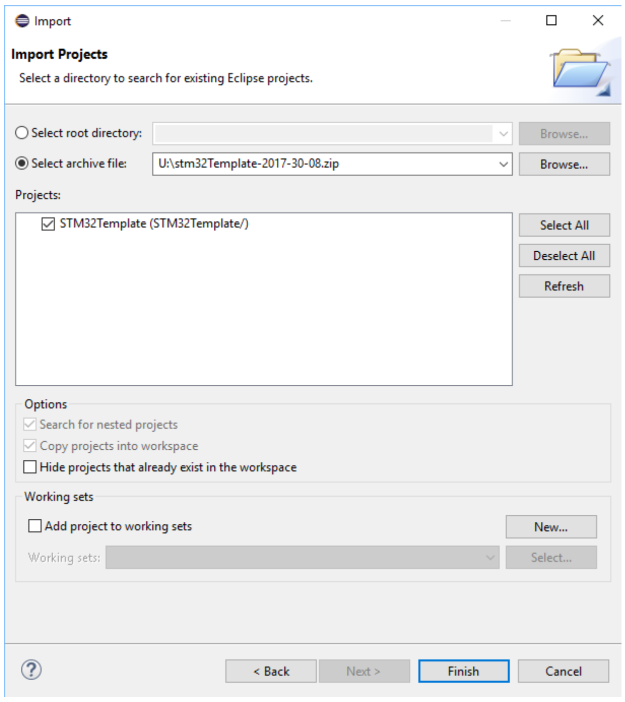
3.2 System Workbench

3.2.1 Configuring the Workspace and Importing a Project

1. Download `STM32Template.zip` from Blackboard.
2. Open System Workbench. If you see a “Welcome” screen close it by clicking the X next to the ‘Welcome’ label near the top of the window.
3. System Workbench will prompt you to set a “workspace” folder. Choose a folder in your student U:\which **does not contain spaces in the file path**. If the workspace path contains a space character your project will fail to compile.



(a) The import type dialog box.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory:

◉ Select archive file: U:\stm32Template-2017-30-08.zip

Projects:

☑ STM32Template (STM32Template/)

Options

☑ Search for nested projects

☑ Copy projects into workspace

☐ Hide projects that already exist in the workspace

Working sets

☐ Add project to working sets

Working sets:

(b) Importing a .zip archive.

4. Import the project into the workspace:
   a) Click 'File >Import', and select 'General >Existing Projects into Workspace' from the list as shown in Figure 3a. Click 'Next'.
   b) Select the "Select archive file" radio button at the top, click 'Browse', find `STM32Template.zip`, and open it. If done correctly you should see something similar to Figure 3b.
   c) Click 'Finish' to close the dialog then ensure that the project exists in the project explorer on the left side of the screen.
5. Compile the project by typing `CTRL+B` or clicking through the 'Project >Build All' menu. If this fails do it again 2-3 times or until it finishes without error.

3.2.2 Configuring the debug environment

1. Right click on the project name and select “Debug As >Debug Configurations...”, as shown in Figure 3.

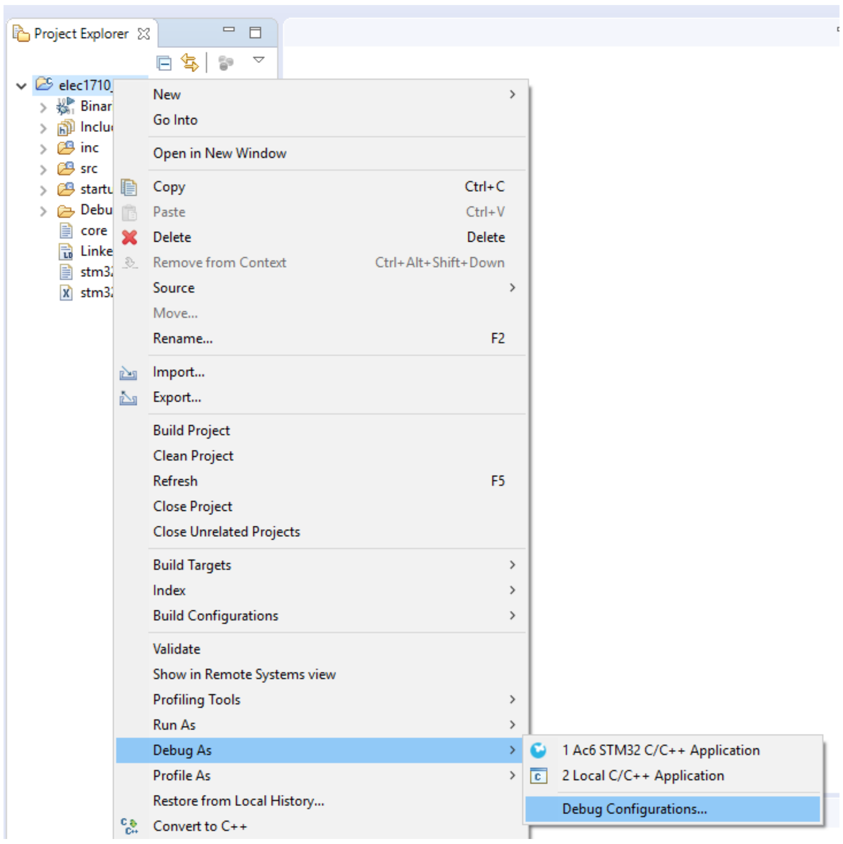
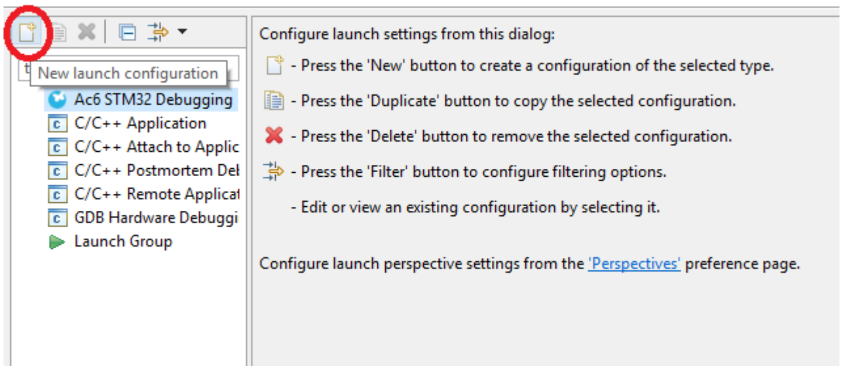


Figure 3: Opening the debug configuration.

2. Highlight ‘Ac6 STM32 Debugging’ and click the ‘New launch configuration’ button (highlighted in Figure 4).



Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

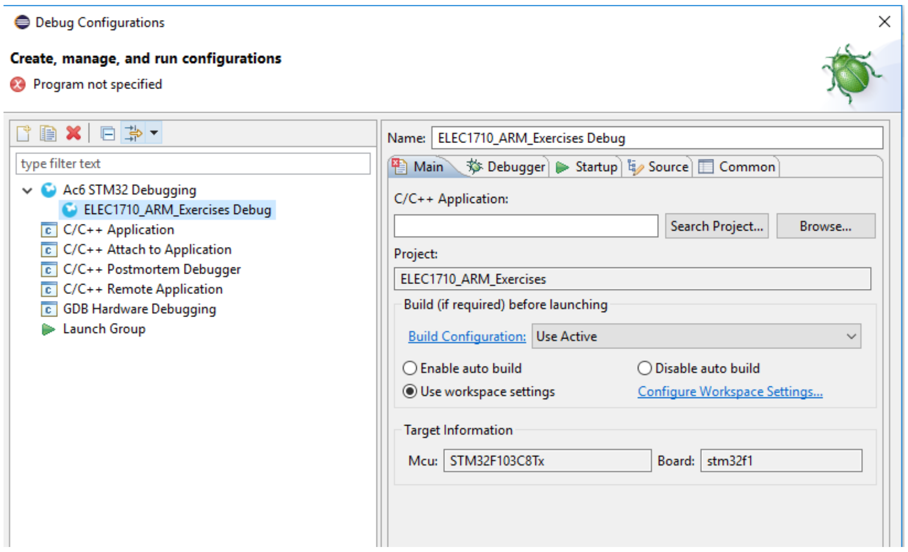
Configure launch perspective settings from the 'Perspectives' preference page.

Figure 4: Creating a new Ac6 STM32 debug configuration.

3. Select the new configuration and navigate to the 'main' tab as shown in Figure 5.

   If the box under C/C++ Application is empty, as it is in Figure 5, select 'search project' and choose the binary as shown in Figure6. After selecting ok the .elf file should now be shown as it is in Figure7.

   If the `.elf` file does not appear when clicking 'Search Project...' then the project did not compile correctly. Press `ctrl+b` again or, as a last resort, select the menu item 'Project - clean' from the top menus then attempt a build 2 or 3 more times.



Debug Configurations

Create, manage, and run configurations

Program not specified

type filter text

- Ac6 STM32 Debugging
  - ELEC1710_ARM_Exercises Debug
- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- Launch Group

Name: ELEC1710_ARM_Exercises Debug

Main | Debugger | Startup | Source | Common

C/C++ Application:

Search Project... Browse...

Project:

ELEC1710_ARM_Exercises

Build (if required) before launching

Build Configuration: Use Active

Enable auto build

Disable auto build

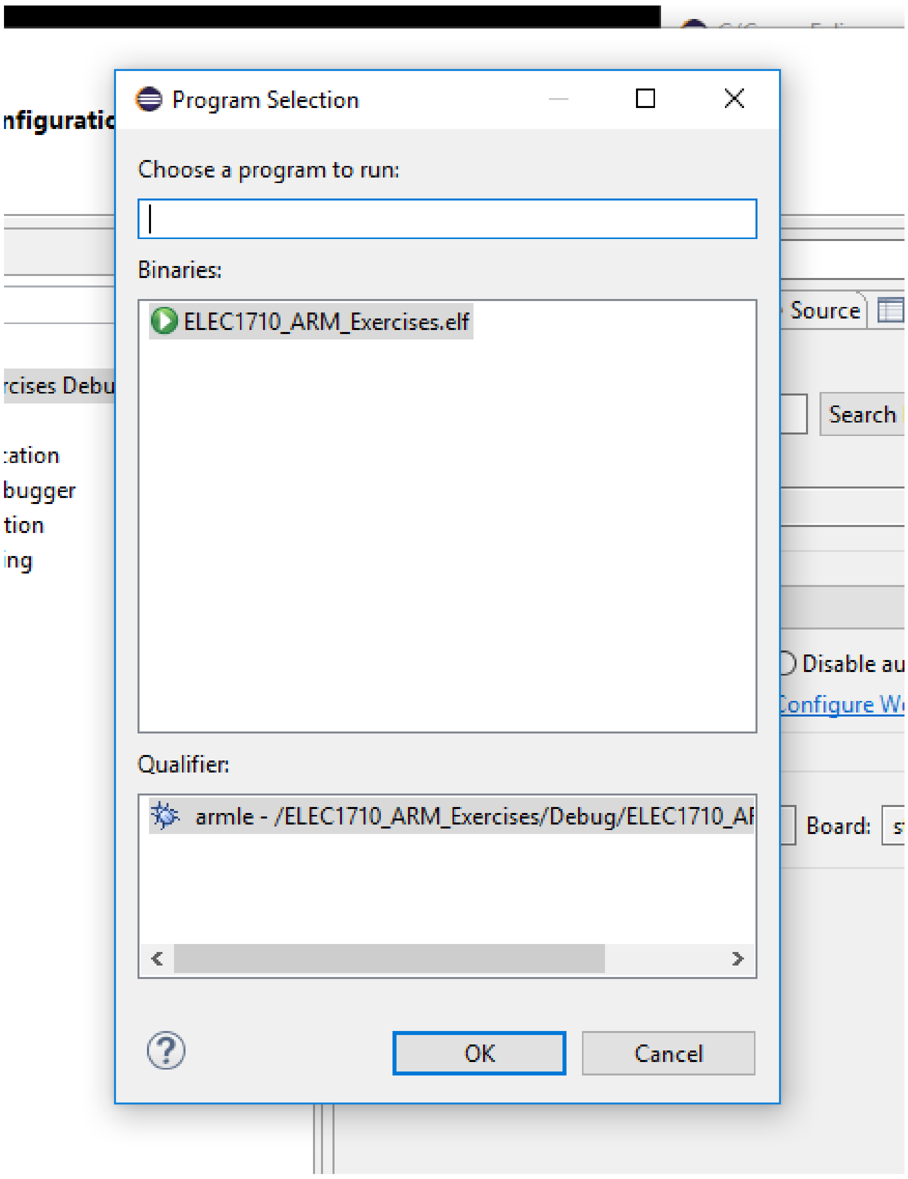
Use workspace settings

Configure Workspace Settings...

Target Information

Mcu: STM32F103C8Tx Board: stm32f1

Figure 5: The debug configuration window.



Program Selection

Choose a program to run:

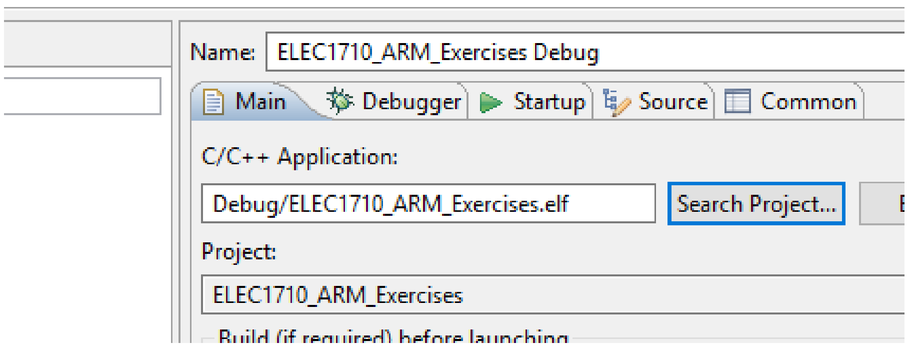
Binaries:

ELEC1710_ARM_Exercises.elf

Qualifier:

armle - /ELEC1710_ARM_Exercises/Debug/ELEC1710_AR

Figure 6: Manually finding the `.elf` binary file.



Name: ELEC1710_ARM_Exercises Debug

C/C++ Application:

Debug/ELEC1710_ARM_Exercises.elf

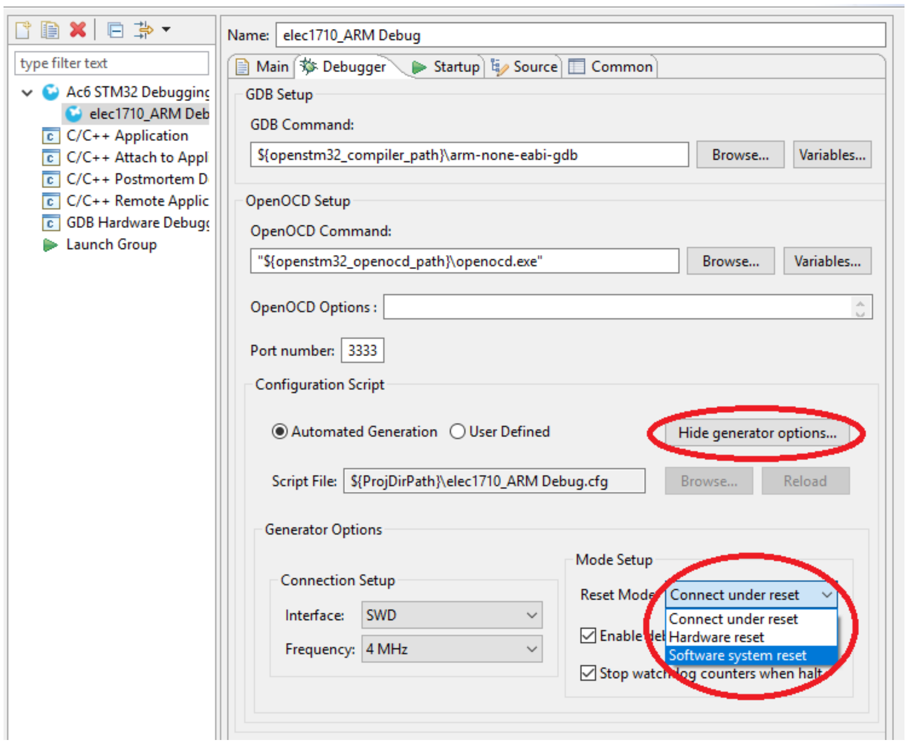
Project:

ELEC1710_ARM_Exercises

Build (if required) before launching

Figure 7: A correctly configured ‘Main’ tab.

4. Navigate to the ‘Debugger’ tab. Click the “Show generator options” button and select “Software system reset” from the “Reset Mode” drop down menu, as shown in Figure 8).



Name: elec1710_ARM Debug

GDB Command: ${openstm32_compiler_path}\arm-none-eabi-gdb

OpenOCD Command: "${openstm32_openocd_path}\openocd.exe"

OpenOCD Options :

Port number: 3333

(●) Automated Generation ( ) User Defined

Script File: ${ProjDirPath}\elec1710_ARM Debug.cfg

Interface: SWD

Frequency: 4 MHz

Reset Mode: Connect under reset

☑ Enable del

☑ Stop watchdog counters when halt

Figure 8: Configuring OpenOCD for the ST-Link debugger.

5. Ensure the ST-Link is plugged into the PC USB port. Click the ‘Debug’ button (bottom right), this will recompile the project then load the resulting binary onto the STM32 dev board and bring up a dialog box asking if you want to swap to the debug ‘perspective’. Click yes. You can swap between the debug and C/C++ perspectives by clicking the appropriate icons in the top right of the System Workbench window (ask a demonstrator to help locate them, they are small and subtle).

3.2.3 Debugging Code on the STM32

1. Execute the 'blinky' code using the debugger functions shown in the video (from 7 mins 10 sec). **NB:** Pressing the red square will terminate the exection, requiring a debugging session restart. Use the yellow vertical bar 'Pause' button to stop execution temporarily.
    - Use single instruction step mode
    - Use (`CTRL+r`) to run to a given line
    - Use breakpoints (`CTRL+SHIFT+b`) to stop the execution at a given line
    - Observe variables and registers in the upper right hand window
    - Manually modify register and peripheral contents
2. With the code running observe the LED on pin C13 flash
3. **Task:** Modify the assembly code to blink the LED soldered to the STM32 development board. It is connected to pin B12 (GPIOB, pin 12). See Appendix 4.2
    - `GPIOB_ODR` is at address `0x40010C0C`, add a `.equ` assembly directive to define this value. Using a `.equ` increased readability as the text `GPIOB_ODR` is used instead of a seemingly cryptic hexadecimal constant.
    - B12 is set to logic 1 with the constant `0x00001000`. Modify the appropriate `ldr` instruction to reflect this. **NB:** Pins are numbrered starting at zero.
    - The LED is wired in an active low configuration. Setting B12 to a logic 0 turns the LED on.

4 Appendix

4.1 Blinky Source Code

Listing 1: Blinky code listing

```
    .syntax unified
    .cpu cortex-m3
    .thumb
    .global blink
    // This defines a constant which is the STM32F103's address for the
    // port C output data register
    .equ        GPIOC_ODR,      0x4001100C
// NB: This function assumes that GPIOC was pre-configured as an output
blink:
        ldr r0, =0x00002000 // The "2" puts a 1 in bit 13, for output pin PC13
        ldr r1, =0x00000000 // A source of zero to turn off PC13
        ldr r2, =GPIOC_ODR  // Load r2 with the address of GPIOC_ODR

        str r0, [r2]    // Store the contents of r0 into the memory address
                        // in r2. This turns on PC13, turning OFF the LED.

        ldr r3, =0x00080000     // This function counts down from this bit number
delay1:
        sub r3, r3, #1          // Subtract 1 from r3 and store result back in r3
        cmp r3, #0      // Compare r3 to zero. Sets status flags
        it gt           // IT (if-then) block.
        bgt delay1      // If the comparison result was "greater-than"

        str r1, [r2]    // Store the contents of r1 into the memory address
                        // in r2. This turns off all of PORTC turning ON the LED.

        ldr r3, =0x00080000     // This function counts down from this bit number
delay2:
        sub r3, r3, #1  // Subtract 1 from r3 and store result back in r3
        cmp r3, #0      // Compare r3 to zero. This sets status flags
        it gt           // IT (if-then) block.
        bgt delay2      // If the comparison result was "greater-than"

        b blink         // Branch back to "blink" label
```

4.2 Modified Blinky Source Code

Listing 2: Blinky code listing

```
    .syntax unified
    .cpu cortex-m3
    .thumb
    .global blink
    // This defines a constant which is the STM32F103's address for the
    // port B output data register
   .equ GPIOB_ODR,      0x40010C0C

// NB: This function assumes that GPIOC was pre-configured as an output
blink:
          ldr r0, =0x00001000 // The "1" puts a 1 in bit 12, for output pin 12
        ldr r1, =0x00000000 // A source of zero to turn on pin 12
        ldr r2, =GPIOB_ODR  // Load r2 with the address of GPIOB_ODR

        str r0, [r2]    // Store the contents of r0 into the memory address
                        // in r2. This turns on PC12, turning OFF the LED.

        ldr r3, =0x00080000     // This function counts down from this bit number
delay1:
        sub r3, r3, #1          // Subtract 1 from r3 and store result back in r3
        cmp r3, #0      // Compare r3 to zero. Sets status flags
        it gt           // IT (if-then) block.
        bgt delay1      // If the comparison result was "greater-than"

        str r1, [r2]    // Store the contents of r1 into the memory address
                        // in r2. This turns off all of PORTB turning ON the LED.

        ldr r3, =0x00080000     // This function counts down from this bit number
delay2:
        sub r3, r3, #1  // Subtract 1 from r3 and store result back in r3
        cmp r3, #0      // Compare r3 to zero. This sets status flags
        it gt           // IT (if-then) block.
        bgt delay2      // If the comparison result was "greater-than"

        b blink         // Branch back to "blink" label
```

4.3 GPIO Memory Addresses

In order to calculate the memory addresses of the STM32's GPIO ports several resources need to be consulted. Each GPIO port is configured by multiple registers, each with a unique memory address. However, since each GPIO port (A through to E) has the same set of registers they are not all listed individually. Instead the *base address* of each GPIO port is published in the datasheet and the *offset* of each register from this base address is published in the reference manual.

The base address of `GPIOC` is shown in Figure 9 to be `0x4001 1000` (likewise `GPIOB`'s is `0x4001 0C00`. Figure 10 then shows that the output data register (`GPIOx_ODR`) is found at an offset of `0x0C` from this base address. So `GPIOB`'s output data register is at `0x4001 0C00` + `0x0C` = `0x4001 0C0C` and likewise `GPIOC`'s output data register is at `0x4001 100C`.

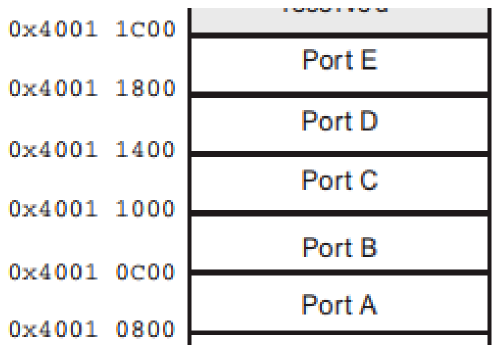
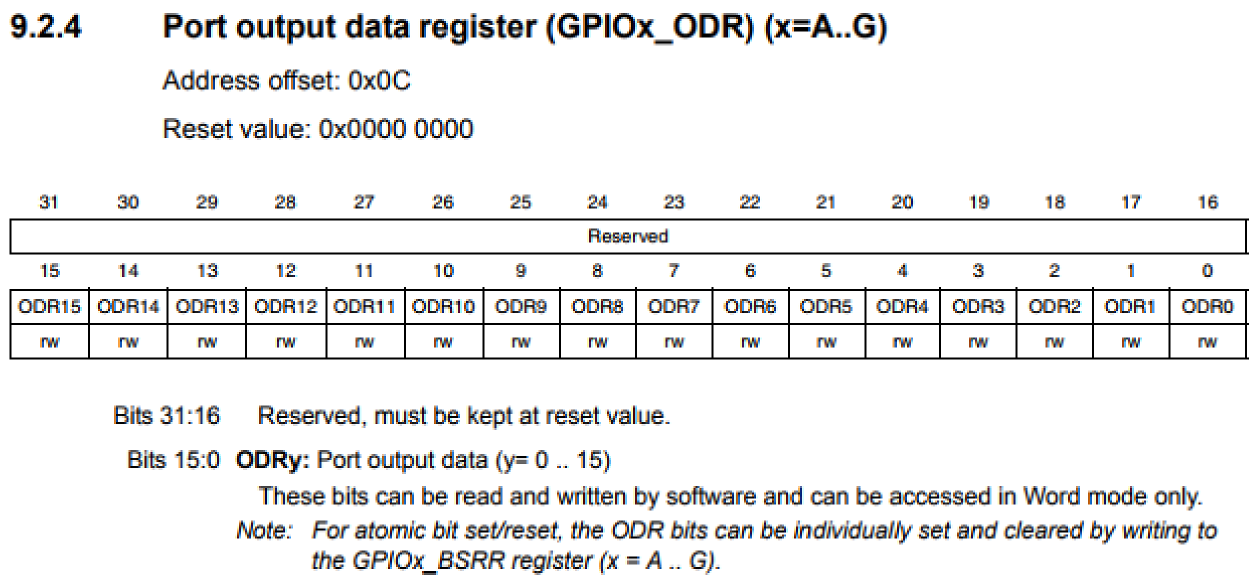


Figure 9: The GPIO memory base addresses are found in the *memory map*. Extracted from Figure 11 of the STM32F103x8 datasheet.



9.2.4 Port output data register (GPIOx_ODR) (x=A..G)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy:** Port output data (y= 0 .. 15)

These bits can be read and written by software and can be accessed in Word mode only.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. G).

Figure 10: The *output data register* (`GPIOx_ODR`) is listed as having an *offset* of `0x0C`. Extracted from the reference manual (RM0008).