# LABORATORY MANUAL

# CENG 255

# Introduction to Computer Architecture

# Laboratory Experiment #0

This manual was prepared by

The many dedicated, motivated, and talented graduate students and faculty members in the Department of Electrical and Computer Engineering

The laboratory experiments are developed to provide a hands-on introduction to the ARM architecture. The labs are based on the open source tools Eclipse and OpenOCD.

You are expected to read this manual carefully and prepare **in advance** of your lab session. Pay particular attention to the parts that are **bolded and underlined**. You are required to address these parts in your lab report. In particular, all items in the **Prelab** section must be prepared in a written form **before your lab**. You are required to submit your written preparation during the lab, which will be graded by the lab instructor.

# **Laboratory Experiment 0: Introduction to Eclipse**

## 1.1 Goal

Eclipse is an integrated development environment (IDE) that can be used to develop applications with various programming languages such as C and C++. With the proper plug-ins, one can develop ARM assembly applications in Eclipse and execute/debug programs on ARM development boards. This is a tutorial to introduce Eclipse and the process of developing a C program to be executed on the STM32F0 Discovery Boards in the lab.

## 1.2 Part 1: Create a Blinky C Project

#### 1.2.1 Configure Eclipse for STM32F0

You develop and store your source code as projects. To create a project, go to Eclipse menu, File ->New, and select C Project:



Figure 1.1: New C project

Inside the C Project window:

- In the Project name: field enter the name of a new project, for example, Blinky
- In the Project type: section expand the **Executable** type and select **STM32F0xx** C/C++ **Project**
- In the Toolchains: select Cross ARM GCC
- Click the **Next>** button

C Project		- <b>O</b> X
C Project Create C project of selected type		
Project name: Blinky		
☑ Use default location		
Location: C:\Users\rexlei\workspace\Blinky		B <u>r</u> owse
Choose file system: default 💌		
Project type:	Toolchains:	
GNU Autotools Executable Empty Project Hello World ANSI C Project Hello World ARM C Project Hello World ARM Cortex-M C/C++ Project Freescale Kinetis KLxx C/C++ Project Freescale Processor Expert C/C++ Project STM32F0xx C/C++ Project STM32F10x C/C++ Project STM32F3xx C/C++ Project STM32F3xx C/C++ Project STM32F3xx C/C++ Project STM32F3xx C/C++ Project Stm32F4xx C/C++ Project Makefile project I	Cross ARM GCC	
Show project types and toolchains only if the	ey are supported on the platfor	m
(?) < <u>B</u> ack	Next > <u>F</u> inish	Cancel

Figure 1.2: Project name and processor selection

In the Target processor settings window, use all the default values:

- Chip family: default value (STM32F051) is the target ARM board
- Flash size (KB): the flash size of our ARM board
- RAM size (KB): the RAM size of our ARM board
- Clock (Hz): the default value of our ARM board
- Content: use this default value Blinky (blink a LED) as this tutorial is to create a Blinky project. If you want to create another new project, you can switch to Empty (add your own content)
- Click the **Next>** button

🖨 C Project		x	
Target processor settings			
Select the target proces	Select the target processor family and define flash and RAM sizes.		
Chip family:	STM32F051	ן נ	
Flash size (KB):	64		
RAM size (KB):	8		
Clock (Hz):	8000000		
Content:	Blinky (blink a led)	-	
Use system calls:	Semihosting (POSIX system calls via host)		
Trace output:	None (no trace output)		
Check some warnings			
Check most warnings			
Enable -Werror			
Use -Og on debug	$\checkmark$		
Use newlib nano	$\checkmark$		
Exclude unused	$\checkmark$		
Use link optimizations			
?	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish Cancel		

Figure 1.3: Target processor settings

In the Folders settings window:

• Leave the default folders unchanged and click the **Next>** button.

C Project		
Folders _settings Define the project fold	lers and other options.	
Include folder:	İnclude	
Source folder:	src	
System folder:	system	
CMSIS library folder:	cmsis	
C library folder:	newlib	
Linker scripts folder:	Idscripts	
?	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish	Cancel

Figure 1.4: Project folder settings

In the Select Configurations window:

• Leave the default settings unchanged and click **Next>** button to the next step. button.



Figure 1.5: Project select configurations

In the Cross GNU ARM Toolchain window:

- Select the Toolchain name: GNU Tools for ARM Embedded Processors (arm-none-eabigcc)
- Click the Finish button



Figure 1.6: Cross GNU ARM toolchain





Figure 1.7: Blinking LED application

### **1.3 Build the project**

- Select the Blinky project in the Project Explorer
- Click the hammer icon, which is the shortcut to build the selected project.



Figure 1.8: First method to build a project

Or

- Right click the Blinky project in the Project Explorer
- Select Build Project in the popup window



Figure 1.9: Second method to build a project

The build process produces a listing in the Console window like this:



Figure 1.10: Build process

The files created by the build process are left in a Debug folder



Figure 1.11: Debug folder contents

## 1.4 Configure debugging

To set debugging:

- Select the forward button on the right side of the bug icon.
- Select Debug Configurations... in the popup window



Figure 1.12: Debug menu

In the **Debug Configurations** window:

• Double click GDB OpenOCD Debugging; this creates a project debug with the project name Blinky Debug

Debug Configurations			×
Create, manage, and run configurations			Ť.
Image: Second Secon	Name: Blinky Debug Main * Debugger * Startup * Source Common Project: Blinky C/C++ Application: Debug\Blinky.elf Build (if required) before launching Build configuration: Select Automatically Enable auto build Use workspace settings	Variables Search Project B	rowse
Filter matched 10 of 12 items		Apply	Revert
?		Debug	Close

Figure 1.13: Creating an GDB OpenOCD debug configuration file.

- Go to the **Debugger** tab
- In Executable: enter openocd.exe
- In Config options: fill in the following setting (f is a flag indicating the file to be used ) :

-f board\stm32f0discovery.cfg

Debug Configurations			×
Create, manage, and run conf	igurations		Ť.
Itype filter text C/C++ Application C/C++ Attach to Applic C/C++ Postmortem Del C/C++ Remote Applicat GDB Hardware Debuggi GDB OpenOCD Debugg C Blinky Debug Launch Group	Name: Blinky Debug Main Debugg OpenOCD Setup Start OpenOCD I Executable: op GDB port: 33 Telnet port: 44 Config options: -f Allocate console GDB Client Setup Executable: S{c Other options: set	Jer Startup Source Common locally penocd.exe 333 444 f board\stm32f0discovery.cfg e for OpenOCD Allocate consol cross_prefix}gdb\${cross_suffix} t mem inaccessible-by-default off	e for the telnet connection
← Ⅲ → Filter matched 8 of 8 items			Apply Re <u>v</u> ert
?			Debug Close

Figure 1.14: OpenOCD settings

Go to the Startup tab

• In the rectangle above the Enable ARM semihosting checkbox, fill in the following setting (this command tells the debugger to stop at the beginning of the main function so that the programmer can debug the project step by step).

monitor reset halt

Debug Configurations	
Create, manage, and run configuratio	ns total
Image: Second Secon	Name: Blinky Debug Main 32 Debugger Startup Source Common Initialization Commands Initialization Commands Initial Reset. Type: init monitor reset halt Image: Type: Init Image: Type: Type: Image: Type: T
Filter matched 10 of 12 items	Apply Re <u>v</u> ert
0	Debug Close

Figure 1.15: Startup tab

Go to the Common tab

- Tick the Debug checkbox in the rectangle below Display in the favorites menu
- Click the Apply button
- Click the Debug button to start debugging

Debug Configurations		
Create, manage, and run configuratio	ns	- <b>A</b>
Image: Second Secon	Name:       Blinky Debugger       Startup       Source       ⊆ common         Save as       Image: Startup       Source       Image: Source       I	Encoding Default - inherited (GBK) C Othgr ISO-8859-1
Filter matched 10 of 12 items		Apply Revert
?		Debug Close

Figure 1.16: Common tab

The debugging configuration wizard creates a debug setting for the Blinky project. The result of debugging installs the binary of the project into the ARM board (that is, the executable binary file is downloaded to the RAM on board). At the beginning of debug process, the program stops at the main function waiting for the programmer to debug the program.

Debug - Blinky/src/main.c - Eclipse		
<u>File Edit Source Refactor Navigate Search Project Run Window Help</u>		
🗃 🖛 📾   副 🖎 🕪 🗉 🔳 🖉 🌫 🗇 🗷 🖶 🧮 🕹 🏪 🚳 🖉 🗸 🕲 🕮	<b>∻ - []</b> ≎ \$ <b>1</b> - ₹ - ← - → -	Quick Access 😰 🛛 💀 C/C++ 🌾 Debug
🎄 Debug 🛛 🦌 🦉 🗖 🗖	(*)= Variables 😫 💊 Breakpoints 👭 Registers 🖳 Perip	oherals 🛋 Modules 📃 🗖
Blinky Debug (GDB OpenOCD Debugging)		£
a 🔐 Blinky.elf	Name	
Thread #1 (Suspended : Breakpoint)	IName	Type
main() at main.c:65 0x8000ba4	(x)= argc	int char **
🔎 openocd-x64-0.8.0.exe	p argv	uint22 t
📓 arm-none-eabi-gdb	(x)= loops	int
		^
	4	· ·
	(*	
🖻 main.c 🔀 💽 0x0		🗄 Outline 🔀 📃 🗆
630 int	*	💱 🖻 🛃 😿 🖌 🖷 🐺 🗢
e4 main(int argc, char" argv[])		stdio.h
66 // Show the program parameters (passed via semihosting).		🛀 stdlib.h
67 // Output is via the semihosting output channel.		diag/Trace.h
68 trace_dump_args(argc, argv);		Timer.h
70 // Send a greeting to the trace device (skipped on Release).		BlinkLed.h
71 trace_puts("Hello ARM World!");		# BLINK_ON_TICKS
		# BLINK_OFF_TICKS
73 // Send a message to the standard output. 74 puts("Standard output message."):		<ul> <li>main(int, char*[]) : int</li> </ul>
75		# LOOP_COUNT
76 // Send a message to the standard error.		
<pre>77 fprintf(stderr, "Standard error message.\n"); 78</pre>	-	
	Þ	
🖳 Console 🕱 🧔 Tasks 🔮 Problems 🜔 Executables 📋 Memory	<b>=</b> X	※  🗟 🔠   ☶   🖅 😕 🚽 📮 ▾ 😁 ▾ 😑 🗖
Blinky Debug [GDB OpenOCD Debugging] arm-none-eabi-gdb		
<http: documentation="" gdb="" software="" www.gnu.org=""></http:> .		*
For help, type "help". Type "apropos word" to search for commands related to "word".		
<pre>remporary preakpoint 1, main (argc=1, argv=0x20000090 <argv_but>) at/src/main.c:65 65 65</argv_but></pre>		E
		+
4		- F
	Writable Smart Insert 65 : 1	1

Figure 1.17: Debugging code

These icons labeled below in the toolbar of Eclipse are the ones used most frequently in debugging and they help the programmer to interact with the debugger. Experiment/Play with these icons and see how they work.

- Icon A: to skip all the breakpoints
- Icon B: to resume the program from debugging
- Icon C: to suspend the program and set it back to debug
- Icon D: to terminate the program
- Icon E: to step into a function
- Icon F: to step over a function
- Icon G: to step return to the function
- Icon H: to restart a process or debug target without terminating and re-launch



Figure 1.18: Debug icons

## 1.5 Part 2: Create an ARM Assembly Project

- Follow the wizards in Part 1 of this tutorial that create an ARM C project.
- Change the .c extension of main.c to .asm or .S and remove all the contents in that file.
- Write assembly codes in the main.asm/main.S to implement your application.
- The **build** and **debug** steps are the same as described in Part 1.

Creating an assembly project: (using Blinky project as an example)

- Remove all the files except main.c in the **src** folder in **Blinky** project and replace the extension of **main.c** by **.asm**.
- Clear the contents in **main.asm** and write some simple assembly codes for experimentation.



Figure 1.19: Assembly code

### **1.6 Part 3: Tips for using eclipse**

Changing displayed format in register tab: During the debugging process you will be examining registers in the processor. You can change the displayed format of a register by right clicking on the specific registers and selecting the Number Format option. You can also change a group of registers. Changing the format is show in the image below. If the register tab is not being displayed you can display it by selecting the Window option on the main menu then selecting Show View->Registers.

Note: In other sections of eclipse you find number format also called radix.



Figure 1.20: Changing number format

Changing cell size and format in the memory browser:

The memory browser allows you to examining sections of memory in you program. The cell size is the numbers Eclipse will use in each cell in the Memory Browser tab. To change your cell size right click on the displayed data in your Memory Browser tab and select the Cell Size option, then select the cell size to be used. This is shown in the image below.

4	Panes Endian Text	* * *	
Console 💭 Tasks 💌 Problems 🕥 Everytables 🗍 Memory Bri	Cell Size	+	1 byte
	Radix	+	2 bytes
(void *)&table	Columns Update Mode	+	4 bytes
x20000020 - (void *)table <traditional> 🛛</traditional>		•	8 bytes
x20000020 000000034 000000056 000000012 000000079		- 1	000000000000000000000000000000000000000
0x2000004C 000000000 00000000 0134221656 0000000000	Сору		0000 00000000 0000
0x20000078 00000000 00000000 00000000 00000000	Reset To Base Address		0 0536870964 2557
x200000A4 000000000 00000000 000000000 00000000	Refresh		0000 00000000 0000
9x20000000 00000000000000000000000000000	00000000 000000000 000	00000	00000000000 0000
x200000FC 000000000 00000000 00000000 00000000	00000000 000000000 000	000000	0000 00000000 000
x20000128 000000000 00000000 00000000 0000000 17-	46553641 1117408045 389	223628	38 1244416804 <b>1244</b>

18

Figure 1.21: Changing memory browser cell size

To change the formatted display also known as Radix right click on displayed data and select the Radix option then select displayed format. This is shown in the image below.

		Panes Endian Text Cell Size	* * *		
Console V	🔄 Tasks 🚉 Problems 🕡 Executables 🕕 Memory Bro	Radix	×		Hex
(void *)&table 0x20000020 - (void *)table <traditional> 🕱</traditional>		Columns Update Mode	•	Decimal Signed	
			•	Decimal Unsigned	
0x20000020 0x2000004C	0000000034 000000056 000000012 000000079 000000000 00000000 0134221656 000000000 0	Copy Reset To Base Address	•		Octal 8 Binary 8
0x200000A4	000000000 00000000 00000000 00000000000	Refresh		00	000000000 00000000 0000
0x200000D0 0x200000FC	000 00000000 00000000 0000000000000000	00000000000000000000000000000000000000	30000 30000	00 0 00 0	00000000000000000000000000000000000000
0x20000128	000000000 00000000 00000000 00000000 174	6553641 1117408045 3892	22362	88 1	244416804 <b>1244422163</b> 1261

Figure 1.22: Changing memory browser displayed radix

Disabling console switching: During the debugging process when the debugger steps or encounters a break point it will print information on the console tab. When this occurs Eclipse will automatically switch to the console window to display this information. In specific labs such as the bubble sort lab it would be more convenient to not have Eclipse switch to the console and just to stay in the Memory Browser tab. There are two ways to achieve this.

The first is to disable console switching. This option can be found in the windows-¿preferences dialog box. In the preference dialog expand the Run/Debug section and select console. Unselect the options "Show when program writes to standard out" and "Show when program writes to standard error" as shown in the image below. This is a global setting and will apply to all projects in the Eclipse workspace.

/pe filter text	Console	↓ ↓ ↓
<ul> <li>General</li> <li>C/C++</li> <li>ChangeLog</li> <li>Createrepo</li> <li>Help</li> <li>Install/Update</li> <li>Library Hover</li> </ul>	Debug Console Settings. Fixed width console Maximum character width: Limit console output Console buffer size (characters):	80 80000
> Mylyn	Displayed tab width:	8
Remote Development     Remote Systems     Run/Debug	Show when program writes to	o standard out o standard error
Console External Tools External Tools CopenOCD Peripherals views Perspectives String Substitution View Management View Performance Specfile Editor	Standard Out text color: Standard Error text color: Standard In text color: Background color:	
▷ Leam Terminal		
Taria		Restore Defaults Apply

Figure 1.23: Disabling console switching

The second method is by dragging the Memory browser to a vacant area on the computer screen not being used by any program. Eclipse will automatically create a new window specifically for the memory browser. This feature can be used with any view in Eclipse.

Disassembly not showing in disassembly window: If you are using the disassembly view there is a known issue with this feature. When you first enter debug mode you will notice that the window does not update. See picture on the left. The simple resolution is to close the current disassembly tab and open it from the main menu. Windows->Show View->Disassembly. See picture to the right.

🖶 Outline 🎬 Disassembly 🖾 🗖		🗄 Outline 🎬 Disassembly 🛛 🗖 🗖
Enter location here 👻 👔 🔄	2,	Enter location here 👻 👔 🔄 🔯
$\bigtriangledown$		$\bigtriangledown$
No debug context		₫4 mov r5, #4 🔺
	*	<pre>     0800058a: movs r5, #4     5     add r0, r4, r5     0800058c: adds r0, r4, r5     6         sub r0, r5     0800058e: subs r0, r0, r5     7     stop: nop     stop:     08000590: nop; (mov r8, r8)     8         bl stop     08000592: bl 0x8000590 <stop>     08000592: bl 0x8000590 <stop>     08000592: bl 0x8000590 <stop>     08000592: bl 0x8000590 <stop>     08000596: movs r0, r0    udivsi3:     0800059a: beq.n 0x8000606 <udivsi3 #0="" #1="" 0800059c:="" 0800059e:="" <="" movs="" pre="" r2,="" r3,=""></udivsi3></stop></stop></stop></stop></pre>
٠		4 m

Figure 1.24: Disassembly windows