Experiment 26

Introduction to the ARM Microprocessor

Department of Electrical Engineering & Electronics

February 2022, Ver. 7

Experiment Specifications

Module(s)	**ELEC211**
Experiment code	26
Semester	2
Level	2
Lab location	On campus: PC Labs as timetabled
Work	Individual
Timetabled time	7 hours, 1 lab day. Allocate further time to write your report. *Advice: take screenshots so you can write the report as you go.*
Subject(s) of relevance	Microprocessor Systems, the ARM Microprocessor
Assessment method	1. Pre-lab questions **[10%].** Online questions are available on Canvas. 2. Experiment results **[90%].** The Submission template is available on Canvas (Exp 26-Submission Template file) in the Lab Scripts folder.
Submission deadlines	7 days after lab session**, submitted online via Canvas.**

Important: Marking of all coursework is anonymous. Do not include your name, student ID number, group number, email or any other personal information in your report or in the name of the file submitted via Canvas. A penalty will be applied to submissions that do not meet this requirement.

Experiment 26

Introduction to the ARM microprocessor

1. Objective

To explore the basic features of the ARM Cortex M0 microprocessor.

2. Equipment

- Freescale Freedom FRDM-KL46Z, USB cable and MWS PC
- Keil uVision software
- Stopwatch (you are required to bring a stopwatch with you or you can use your mobile phone)

Instructions:

- Read this script carefully before attempting the experiment.
- Answer the pre-lab questions before attending the lab (worth 10%).
- Have a look at the document "Exp 26-Submission Template" available on Canvas to have an idea about the experimental results that you have to submit. Experimental results submission is worth 90% of the experiment mark.
- Keep a record of all screenshots, results, answers, comments made and graphs plotted in a logbook. When you submit your work, make sure that all the results, screenshots, etc., are clear and readable; otherwise, you'll lose marks.
- **Important**: When taking snapshots from the screen during the simulation, for each and every snapshot, please **FIRST** use **PrintScreen** under Microsoft Windows (which shows the entire desktop including date and time and helps to identify this as uniquely your work) **AND THEN ALSO** use the **Snipping** tool (Microsoft Windows) to show only the relevant part of your simulation. If the entire desktop is not visible in your first screenshot, the associated 'focussed' screenshot will be ignored, and the corresponding section of your report may receive a mark of zero.
- Including screenshots of other people's work is considered academic malpractice and will be penalised in accordance with the University Codes of Practice.
- **All code and text should be included as text and not as screenshots.**

3. Introduction

The ARM Cortex M0 microprocessor is a modern 32-bit processor designed by a British company, ARM Holdings PLC, based in Cambridge. The major advantage of an ARM processor is its low power consumption. This makes it the ideal choice for battery-powered devices which require digital computation. For example, the great majority of all modern digital mobile phones use an ARM processor. The ARM microprocessor is manufactured by most semiconductor companies under licence; the one used in this experiment is made by Freescale (formerly called Motorola). The purpose of this experiment is to become familiar with the ARM Cortex M0 microprocessor, and this will be achieved by examining the operation of simple computer programs that have already been written. This experiment will help you on being more familiar with the assembly language and see practically how the ARM microprocessor works. This will also help you to recall how simple instructions work, arithmetic and logic operations, including negative numbers, branches, flags and conditional executions.

This experiment is based upon the Freescale integrated circuit, KL46Z256, containing the ARM Cortex M0 processor which is mounted on a printed circuit board known as the Freescale Freedom FRDM-KL46Z. This development board is connected to a host computer so that programs can be prepared on the PC and downloaded and executed (or run) on the ARM processor. In this experiment, you will use the Keil uVision environment for writing code and debugging. The Keil uVision software is a professional package used in industry and can be used to link C and C++ programs with assembly language programs. The Keil uVision debugger is a powerful tool for finding bugs in programs and is useful for investigating the characteristics of the ARM Cortex M0 microprocessor.

4. Getting started

During the on-campus lab session, the Freescale Freedom FRDM-KL46Z board will be used. This development board can be connected to a host computer (e.g. MWS PC) so that programs can be prepared on the PC and downloaded and executed (or run) on the ARM processor. The Freedom board can also be connected to other boards, commonly called ‘shields’, through a header that is Arduino compatible. One of these boards is the MBED shield. The Freescale Freedom board connects to a PC as a serial device over the USB connection. In this section you are going to make the board ready to be used for the experiment. Power on an MWS PC and log in into it by using your university credentials. Connect the MWS PC to the Freescale Freedom board using the OpenSDA USB connection as shown in Figure 1. The board should

be mapped by Windows as a new USB drive called "DAPLINK". Check that the DAPLINK drive is present by using File Explorer.

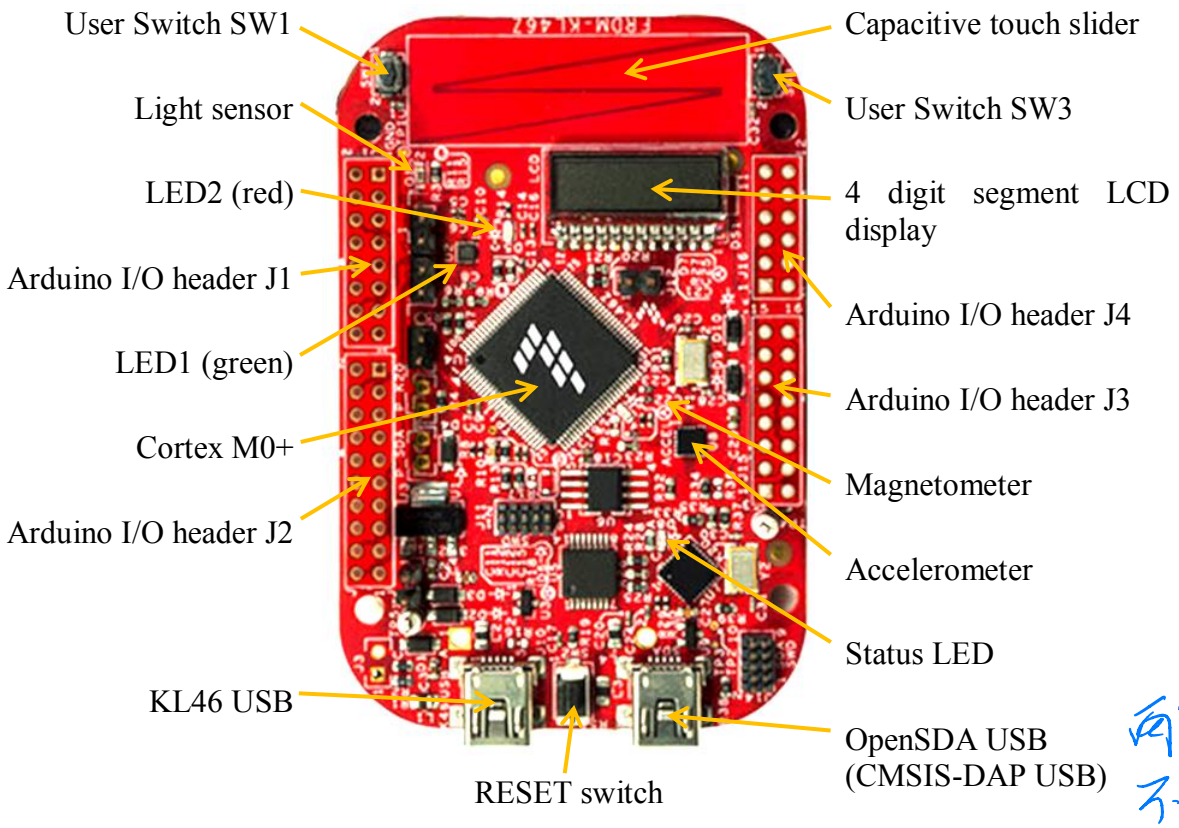


Figure 1: Freescale Freedom development board.

Find the 'Exp 26 (ARM microprocessor) - Demo code.zip' file on Canvas for your module (ELEC222, ELEC224 or ELEC273) by using the browser in the MWS PC. Save the zip file to a location of your choice, then open it, **extract it** and place the program directory at a suitable location. Open the directory and double click on the "uvprojx" file to start Keil uVision. This will launch the 'Exp 26 (ARM microprocessor) - Demo code' project.

5. Assembly language program

The 'Exp 26 (ARM microprocessor) - Demo code' project has two source files, 'main.c' written in the C programming language and 'exp26_asm.s' written in assembly language or 'mnemonics'. Other files under "RTE\Device" are used by the Keil uVision environment to work with the "FRDM-KL46Z" board based on the ARM Cortex M0+ microprocessor. Take a look at the C language program in the Keil uVision environment by double-clicking on "main.c" in the "Project" window. Move the cursor to the line of the main function (line 14) and insert, if not present, a breakpoint by pressing F9. This can also be done by selecting 'Insert/Remove Breakpoint' from the 'Debug' drop-down menu or by clicking on the icon

in the toolbar. Take a look at the assembly language program in the Keil uVision environment by double-clicking on "exp26_ams.s" in the "Project" window.

To run the program on the board you need to **carefully** follow these steps:

1) In the Keil uVision environment, right-click on 'frdm kl46z' in the 'Project' window and select 'Options for Target'. Click on the 'Target' tab and then select from the drop-down menu 'ARM Compiler' on the right the 'Use default compiler version 6' compiler version. Click on the 'Linker' tab and select the radio button 'Use Memory Layout from Target Dialog'. Click on the 'Utilities' tab on the far right and then click on the 'Settings' button. A new window entitled 'CMSIS-DAP Cortex-M Target Driver Setup' should appear. Make sure that in the dialog "Programming Algorithm" the 'MKXX 48Mhz 256kB Prog Flash' is present, if not, click on the 'Add' button and from the list select 'MKXX 48Mhz 256kB Prog Flash'. Click on 'Add' to close the window and then 'OK' once to close another window. Now, in the 'Options for Target' window, click on the 'Debug' tab. The radio button next to 'Use' should be selected (i.e. not the radio button next to 'Use Simulator') and 'CMSIS-DAP Debugger' should be showing in the adjoining box. If it is not, then select it from the drop-down list. Finally click on 'OK' to close the 'Options for Target' window.

2) Then build the project by selecting 'Build target' in the 'Project' drop-down menu. F7 can also be used. Then you need to download the program to the Freescale Freedom board by selecting 'Download' in the 'Flash' drop down menu. F8 can also be used.

3) Start debugging the program in the debug window by selecting 'Start/Stop Debug Session' in the 'Debug' drop-down menu. Control-F5 can also be used.

The two important windows in the debugger view are the 'Disassembly' window and the 'Registers' window, as shown in Figure 2. In the 'Disassembly' window, there are 4 columns which, from left to right, represent (i) the memory address (note that it increments by 2 or occasionally 4), (ii) the machine code – normally 4 hexadecimal digits but occasionally 8 hex digits, (iii) the mnemonic of the instruction and (iv) the operands of the instruction – either registers or immediates or both. In the 'Registers' window there should be a list of 17 registers (R0 to R15 and the xPSR, 'program status register') and the 4 byte value that is held in each register. Click on the ⊞ next to the xPSR, and you should see the flags (N, Z, C and V) and their state (1 for set, 0 for clear).

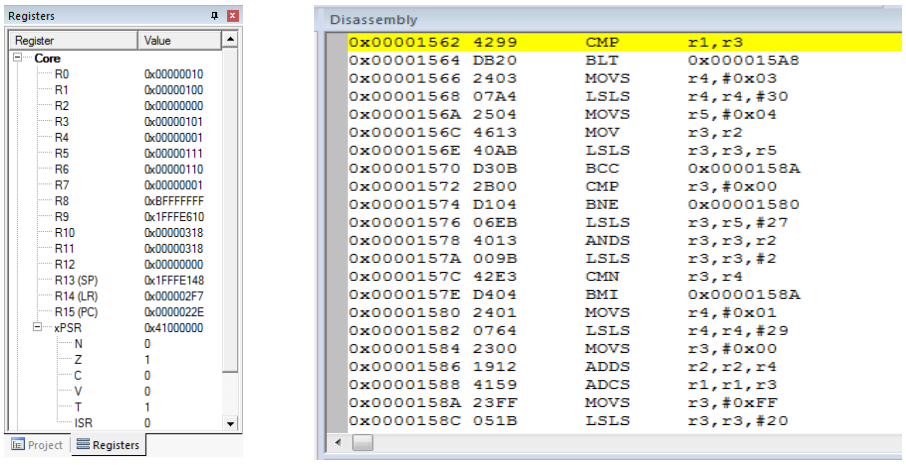


Figure 2: 'Registers' and 'Disassembly' windows.

6. Initial Address (IA)

You should see the yellow arrow pointing to the same instruction (a PUSH) of the corresponding breakpoint in the C program, 'main.c'. This instruction is highlighted in the 'Disassembly' window, as shown in Figure 3 (**in this example** at memory address 0x00000238). Executing instructions by pressing F5 is known as 'Run', and it allows you to continue executing the program until the next active breakpoint is reached. This can also be executed by selecting 'Run' from the 'Debug' drop-down menu or by clicking on the  icon in the toolbar.

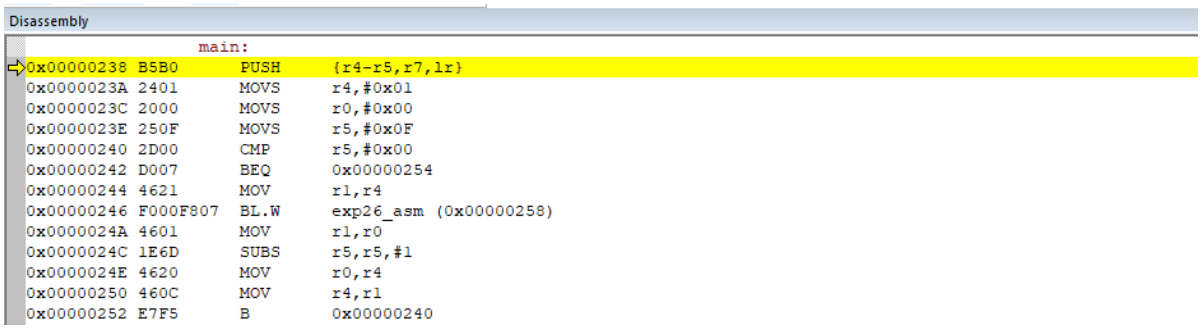


Figure 3: Example of PUSH instruction at address 0x00000238.

The next instruction to be executed by the microprocessor is shown by a yellow arrow, and this points to the first instruction in the C program, 'main.c'. Press F11 (single stepping) once and the yellow arrow moves to the 'for' instruction, you have just executed the first few instructions. Before you press F11 again, see if you can predict which registers will change in value and what the new values will be before the "CMP" instruction is executed. Press F11 again, and the instructions on lines 15, 16 and 17 are executed. In the C program, the next instruction to be executed is the 'for()' instruction and in the 'Disassembly' window, the equivalent mnemonic is 'CMP r5, #0x00' (**in this example** at memory address 0x00000240). In the 'Registers'

window, the value held by register R5 is 0x0000000F as the block of code is repeatedly executed until the conditional expression is true 'i<15'.

Now press F11 twice until the 'Disassembly' window jumps to the assembly language program, 'exp26_asm.s', highlighting the first instruction 'ADDS r7, r0, r1' as shown in Figure 4 (**in this example** at memory address 0x00000258). This memory address will be called from now on **Initial Address (IA)** and will be used as a reference memory address. It is important to note that the IA may vary depending on the board used. Therefore, **only** in this example, the IA is equal to the memory address 0x00000258.

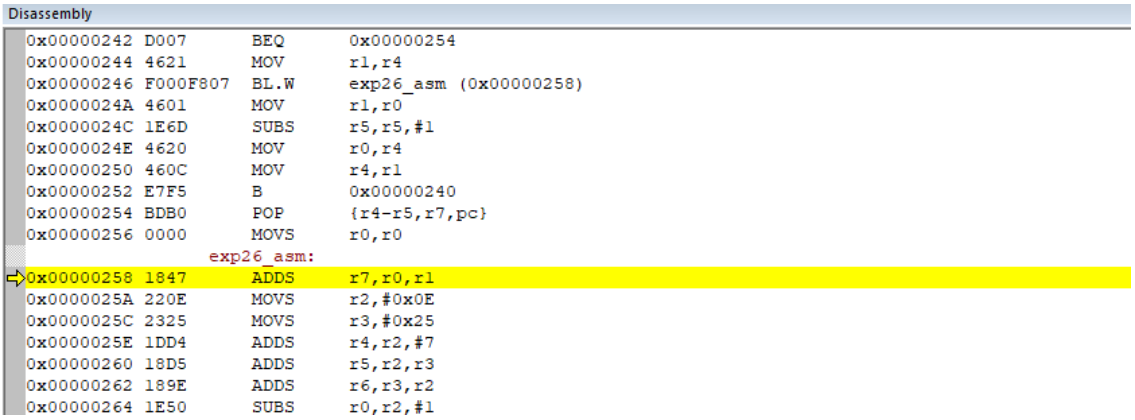


Figure 4: 'Disassembly' windows for the 'exp26_asm.s'.

Table 1 shows the main addresses used in this example. You need to calculate them depending on your IA. This will help you to quickly find and execute instructions.

Table 1: main addresses used in **this example**.

Address	Resulting address (in this example)	Address	Resulting address (in this example)
IA	0x00000258	IA+0x50	0x000002A8
IA+0x6	0x0000025E	IA+0x54	0x000002AC
IA+0x12	0x0000026A	IA+0x60	0x000002B8
IA+0x1C	0x00000274	IA+0x62	0x000002BA
IA+0x2C	0x00000284	IA+0x64	0x000002BC
IA+0x30	0x00000288	IA+0x68	0x000002C0
IA+0x32	0x0000028A	IA+0x6C	0x000002C4
IA+0x3A	0x00000292	IA+0x7E	0x000002D6
IA+0x44	0x0000029C	IA+0x82	0x000002DA
IA+0x46	0x0000029E	IA+0x84	0x000002DC

The main C program has 'called' the assembly language program and has passed two parameters, j and k, using registers r0 and r1 by convention.

Single step the next 3 instructions and notice that registers R2 and R3 hold the values 0x0000000E and 0x00000025 respectively. These values were moved into the registers by executing the first two MOVS instructions of the assembly program. Also, notice that the program counter, R15 or PC, increments by 2 as each instruction is executed. Look at the machine code for the first two MOVS instructions shown in the disassembly window.

mnemonic MOVS r2, #0x0E

Q1. Which part of the machine code identifies the register used?

machine code 2 | 2 | 0E

2nd MSB

Q2. Which part of the machine code gives the number to be moved into that register?

3rd and 4th MSB

Write the answers to all questions in your notebook. You will need these answers at the end of the lab to complete the test sheet using the submission template available on Canvas.

7. Arithmetic Operations

Single step the instruction at address 0x0000025E and note the contents of R4.

ADDS r4, r2, #7 where r2 contains value, 0x0E

Q3. What is this equal to?

$0x0E + 7_{10} = 0x15$

Continue single stepping until you reach address IA+0x12 (in this example address 0x0000026A). Note the contents of the registers after each instruction and work out what each instruction does.

> continued on next page

Q4. Is the order of the registers in the mnemonics important? Why?

Note that negative numbers are given in 2's complement format. For example, to find the 2's complement format for -1,045,387,883$_{10}$ first convert +1,045,387,883$_{10}$ to hexadecimal which gives 0x3E4F5A6B. Then convert each digit to its 1's complement equivalent using the following hexadecimal inversion table.

Original digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Inverted digit (1's complement)	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

0x00000260 ADDS r5, r2, r3

Add r3 and r2 and place the result in r5

2 ADDS r6, r3, r2

r2 r3 r6

4 SUBS r0, r2, #1

r2 - #1 and place the result in r0

6 SUBS r1, r2, r3

r2 - r3 r1

8 SUBS r2, r3, r2

r3 - r2 r2

A MOVS r2, #0x11

Move 0x11 into r2

Q4.

~~Yes, because each location has its meaning. For example, the first location after mnemonic SUBS means the result's destination, the second means minued of the subtraction, and the third means subtrahend of the subtraction.~~

For mnemonic SUBS and MOVS, the order of each reg is important. However, for the last two regs of mnemonic ADDS, the order is not important, but the first reg should always be the destination of the result.

So the 1's complement of -1,045,387,883$_{10}$ is 0xC1B0A594. Next, add 1 to find the 2's complement so the 2's complement format for -1,045,387,883$_{10}$ is 0xC1B0A595. You can easily check this using the Windows calculator in the "Programmer" view.

Next, we want to run the program again, but with different values in registers R2 and R3. In the 'Registers' window, double left click on the contents of R2 - this will allow you to modify the contents of the register - change it to 0x00000064. Similarly, change R3 to 0x0000000A and change the contents of the program counter to IA+0x6 (in this example to 0x0000025E). Now single step the program from address IA+0x6 (in this example address 0x0000025E) to address IA+0x12 (in this example address 0x0000026A). Make a note of the contents of the registers.

Q5. Are these the values you expected?

Single step the program from address IA+0x6 (in this example address 0x0000025E) starting with numbers of your choice in R2 and R3 and note the contents of the registers after execution.

Q6. Again are these the values you expect? skip

8. Logical Operations

Continue single step through the program to address IA+0x1C (in this example address 0x00000274). Looking in the registers window, the last four digits in R2 and R3 should be 0011 and 0101 respectively (when in hexadecimal format). Draw a truth table for input variables A and B (where A is given by a value of a digit in R2 and B is given by the value of a digit in R3) and output variable C given by the value of the digits held in R4.

Q7. What logic function has been performed? and r2 r3

Single step the next 8 instructions and note the contents of R5, R6, R0 and R1. Identify each logic function executed. Clearly, the order of the registers in the mnemonic for bit clear (BIC) is important.

Q8. Is the order important for the other functions?

They are commutative. Therefore, change the order will not affect the result but affect the destination of the result.

r2 = 0x64
r3 = 0x0A

r4 := r2 + 7₁₀
:= 0x6B

r5 := r2 + r3
:= 0x64 + 0x0A
:= 0x6E

r6 := r3 + r2
:= 0x6E

r0 := r2 − 1₁₀
:= 0x63

r1 := r2 − r3
:= 0x64 − 0x0A
:= 0x5A

r2 := r3 − r2
:= 0xFFFFFFA6

MOVS r2, #0x11
⇒ r2 := 0x11

MOVS r2, #0x11
r2 := 0x11

MOV r3, r2
ADDS r3, r3, #0xF0
r3 := r2
:= 0x11
r3 := r3 + 0xF0
:= 0x11 + 0xF0
:= 0x101

MOV r4, r2
ANDS r4, r4, r3
r4 := r2
:= 0x11

0x11 0000 0001 0001

0x101 0001 0000 0001

and → 0000 0000 0001 = 0x1

r2 = 0x11
r3 = r2 + 0xF0 = 0x101
r4 = r2 and r3 = 0x1

From the last section,

r2 := 0x11

r3 := 0x10

```
MOV   r5, r2
ORRS  r5, r5, r3
```

r5 := 0x11

r5 := r5 or r3

:= 0x111

```
MOV   r6, r2
EORS  r6, r6, r3
```

r6 := 0x11

r6 := r6 exclusive or r3

:= 0x110

```
MOV   r0, r2
BICS  r0, r0, r3
```

Rd

The BIC (Bit clear) instruction performs an AND operation on the bits in Rn with the complements of the corresponding bit in the value of Operand2

```
MOV   r1, r3
BICS  r1, r1, r2
```

developer.arm.com/documentation/

is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation

0x11 0000 0001 0001

0x10 0001 0000 0001

or → 0001 0001 0001 = 0x111

0x11 0000 0001 0001

0x10 0001 0000 0001

or → 0001 0001 0000 = 0x110

r3: 0x10 0001 0000 0001

complements → 1110 1111 1110

0x11 0000 0001 0001

↓ and

0000 0001 0000 = 0x010

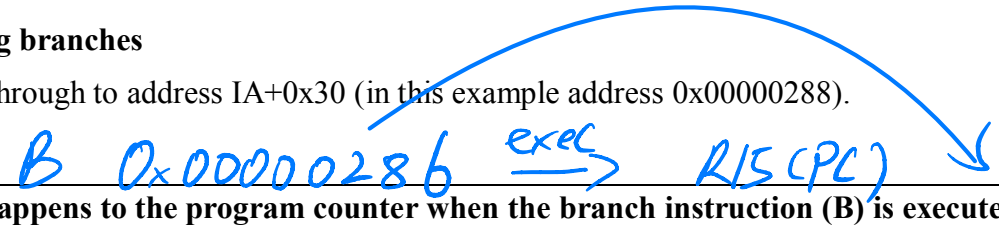
r2: 0x11 0000 0001 0001

0x10 0001 0000 0001

0001 0000 0000 = 0x100

9. Using branches

Single step through to address IA+0x30 (in this example address 0x00000288).



Q9. What happens to the program counter when the branch instruction (B) is executed?

PC will store the address after the B

Q10. If the instruction at address IA+0x30 (in this example address 0x00000288) is executed 5 times, what would the contents of R2 be?

The next section requires a stopwatch, you can use a mobile phone app. Clear the stopwatch so that it reads 0:00:00. Reset R2 to zero. Set the stopwatch running at the same time as you start the program running using F5. After approximately 20 seconds, stop the watch and the program simultaneously - the program can be stopped by either clicking on the  icon in the toolbar or selecting 'Stop' from the 'Debug' drop-down menu. Record the content of R2 and the exact time on the stopwatch. Set the program and the stopwatch going for a further 20 seconds and again stop them simultaneously. Again record the time and value in R2. The value held by R2 should now be approximately twice the value after the initial 20 seconds.

Q.11 Why?

Calculate the number of times R2 has been incremented in 1 second and the time taken for R2 to be incremented once. The answers calculated after approx. 20 and 40 seconds should be roughly the same.

Reset the program counter to IA+0x32 (in this example 0x0000028A). The program starting from address IA+0x32 (in this example 0x0000028A) is identical to the program starting at IA+0x2C (in this example address 0x00000284) except that it has one additional instruction in the loop (which does nothing). Repeat the experiment again and find the time taken for this program loop to be executed once.

Next set the program counter to IA+0x3A (in this example 0x00000292) and repeat the experiment again. **Draw a graph of the time taken for the program loop to execute on the vertical axis against the number of instructions in the loop including the branch instruction on the horizontal axis (use MS Excel or MATLAB…etc.).** There should be three points on your graph, which are roughly in a straight line. The add instruction takes one clock cycle to execute, whereas the branch instruction takes more than this.

```
    MOVS  r2, #0
loop1
    ADDS  r2, r2, #1
    B     loop1
```

Execute B loop1 5 times means adding 1 to r2 which its value is 0 5 times.

20.27 R2: 0x10BAE9B4

40.07 0x21679B4A

Q11

The microprocessor will operate at one clock cycle per instruction, 就以同一时长执行的指令数量是固定的。

clock cycle 代表一秒固定的时间，就每个指令执行的时间是固定的，

20.27 R2: 0x0CBB644F

40.13 0x19356228

20.34 R2: 0x0A305374

40.29 0x14306EE4

Q12. What is the time taken by one clock cycle?

Q13. How many clock cycles are required for the branch instruction?

10. Flags

The ARM Cortex M0 processor has four 'flags', which can give information about the action of the previous instruction e.g. if the sum of two numbers is zero, then the zero flag, Z, is set. The other flags are negative, N, carry, C, and overflow, V. Reset the program counter to IA+0x44 (in this example 0x0000029C) and single step through the program to IA+0x50 (in this example 0x000002A8) making a note of the values held in the registers and also the state of the flags in registers window after each instruction. Note that only instructions with a mnemonic ending with an 'S' actually affect the flags - e.g. the first instruction, MOVS r2, #0x00, sets the zero flag to 1 the second instruction, MOVS r3, #0x01, clears the zero flag to 0. However, the next instruction, MOV r4, r2, leaves the flags unchanged, but the following one MOVS r4, r2 sets the zero flag. Write down the most significant binary digit of R4, R5 and R6 and compare it with the negative flags.

Change the value held in R2 to 0xFFFFFFFF, reset the program counter to and single step from address IA+0x46 (in this example address 0x0000029E). Note values held in the registers and the state of the flags after each instruction.

Single step the program from address IA+0x46 (in this example address 0x0000029E) with 0x7FFFFFFF in R2.

11. Conditional execution

Flags can be used to determine if a branch instruction is executed or not e.g. for a mobile phone this could be 'if credit is less than 10p then disable outgoing calls'. This is known as conditional execution. Look at the machine code for the 8 branch instructions at addresses IA+0x62 (in this example address 0x000002BA) to IA+0x7E (in this example address 0x000002D6) inclusive. The second hexadecimal digit of the machine code determines the condition of the instruction e.g. CS, CC, EQ, NE, etc. Single step the program from address IA+0x50 (in this example address 0x000002A8) to IA+0x82 (in this example address 0x000002DA). Note that the addition ADDS at memory address IA+0x60 (in this example address 0x00002B8) either sets or clears the flags and the carry and zero flags have been set in this case. The following ADD instructions at IA+0x64, IA+0x68, IA+0x6C etc. (in this example at 0x000002BC, 0x000002C0, 0x000002C4 etc.) do not change the flags.

		N	Z	C	V
MOVS	r2, #0	0	1	0	0
MOVS	r3, #1	0	0	0	0
MOV	r4, r2	Not update			
MOVS	r4, r2	0	1	0	0
ADDS	r5, r2, r3	0	0	0	0
SUBS	r6, r2, r3	1	0	0	0
	↓ 0xFFFFFFFF				
MOVS	r2, #FFFFFFFF	NaN			
MOVS	r3, #1	0	0	0	0
MOV	r4, r2	Not update			
MOVS	r4, r2	1	0	0	0
ADDS	r5, r2, r3	0	1	1	0
SUBS	r6, r2, r3	1	0	1	0
	↓ 0xFFFFFE				
MOVS	r2, #7FFFFFFF	NaN			
MOVS	r3, #1	0	0	1	0
MOV	r4, r2	Not update			
MOVS	r4, r2	0	0	1	0
ADDS	r5, r2, r3 ← 0x80000000	1	0	0	1
SUBS	r6, r2, r3	0	0	1	0
	↓ 0x7FFFFE				

Why not updated?

D000	BEQ
D100	BNE
D200	BCS
D300	BCC
D400	BMI
D500	BPL
D600	BVS
D700	BVC

Q.14

BEQ	next1	If zero flag set
BNE	next2	X clear
BCS	next3	carry
BCC	next4	X
BMI	next5	X negative
BPL	next6	
BVS	next7	X overflow
BVC	last	

N	Z	C	V
0	1	1	0

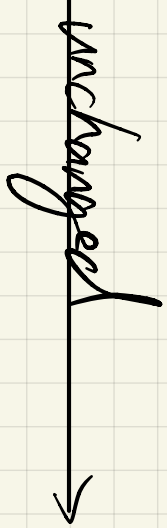
unchanged

Q.15

BEQ	next1	X
BNE	next2	
BCS	next3	X
BCC	next4	
BMI	next5	
BPL	next6	X
BVS	next7	
BVC	last	X

N	Z	C	V
1	0	0	1

unchanged



Q14. Which of the conditional branch instructions have been executed and why?

Now change the contents of register R1 to 0x01000000 and single step the program again from address IA+0x54 (in this example address 0x000002AC) to address IA+0x82 (in this example address 0x000002DA). The negative and overflow flags should now be set.

Q15. Have the conditional branch instructions executed as expected? Why?

12. Getting familiar with Mbed

Remote IDE

First, you need to create an account on the Mbed website (https://os.mbed.com/), the page should look something like Figure 5.

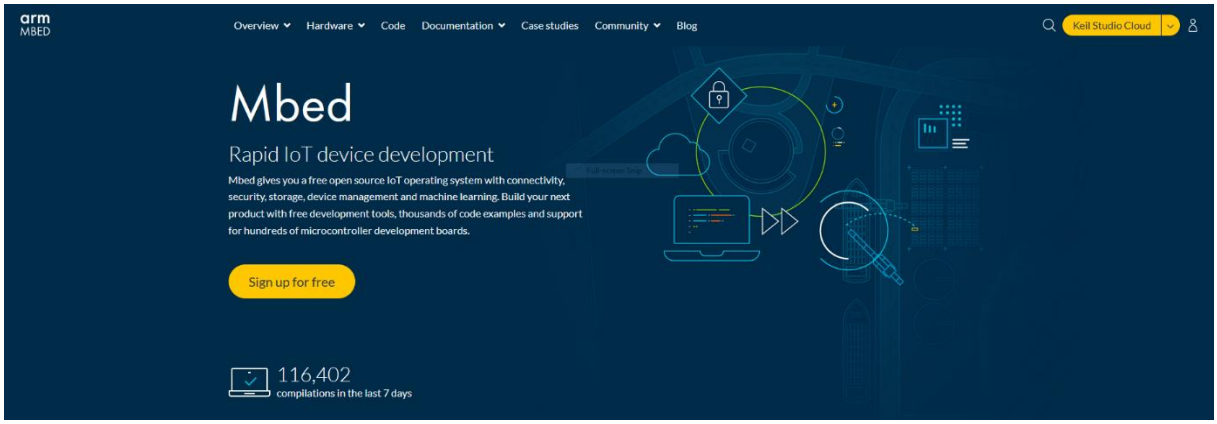


Figure 5: Mbed website.

After you have created the account, you need to log in to your email account and verify your Mbed email address. After you have finalised the creation of your account, make sure you are logged in into the Mbed website. Next, select "Hardware => Boards" from the top of the web page. By using the filter section on the left, filter on 'Cortex-M0+' in "Target Core", and NXP Semiconductors' in "Target vendor". Then select the FRDM-KL46Z board, as shown in Figure 6.

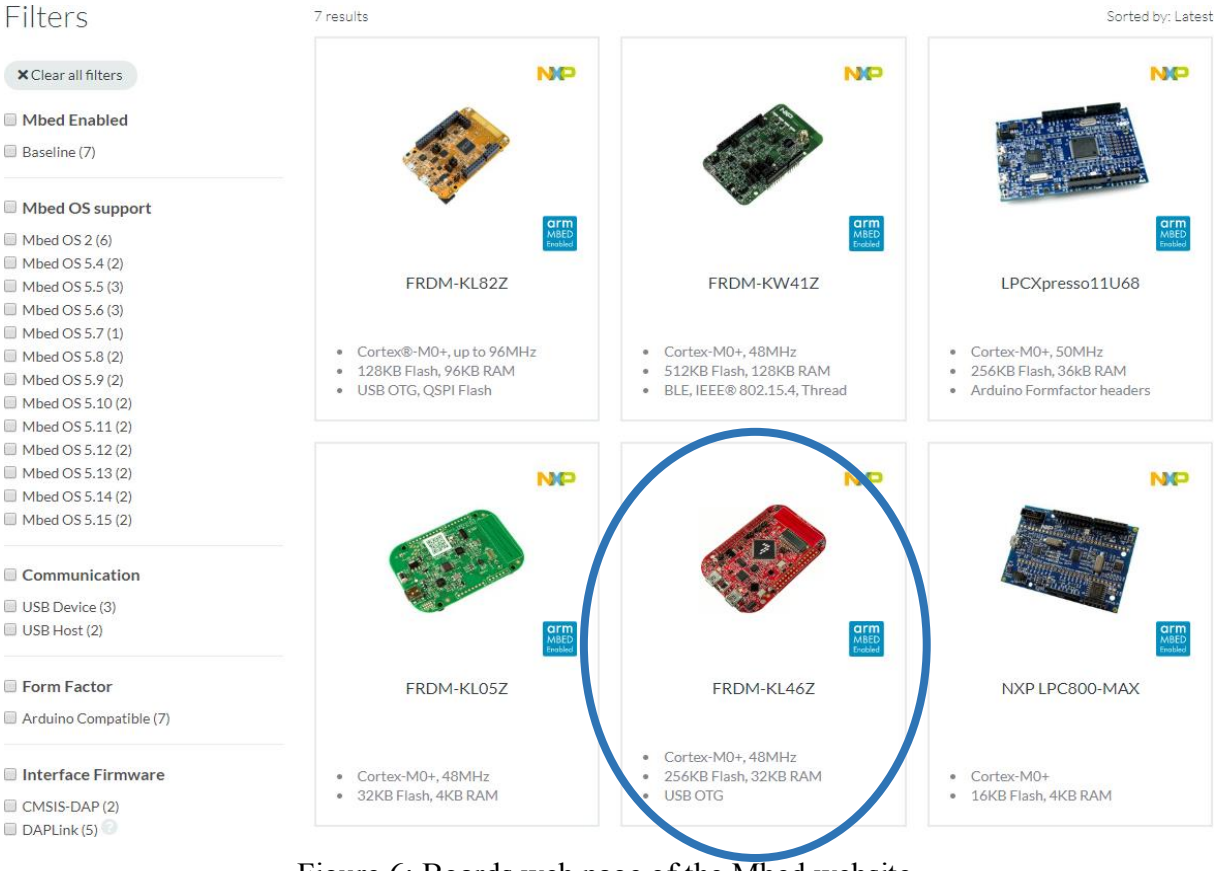


Figure 6: Boards web page of the Mbed website.

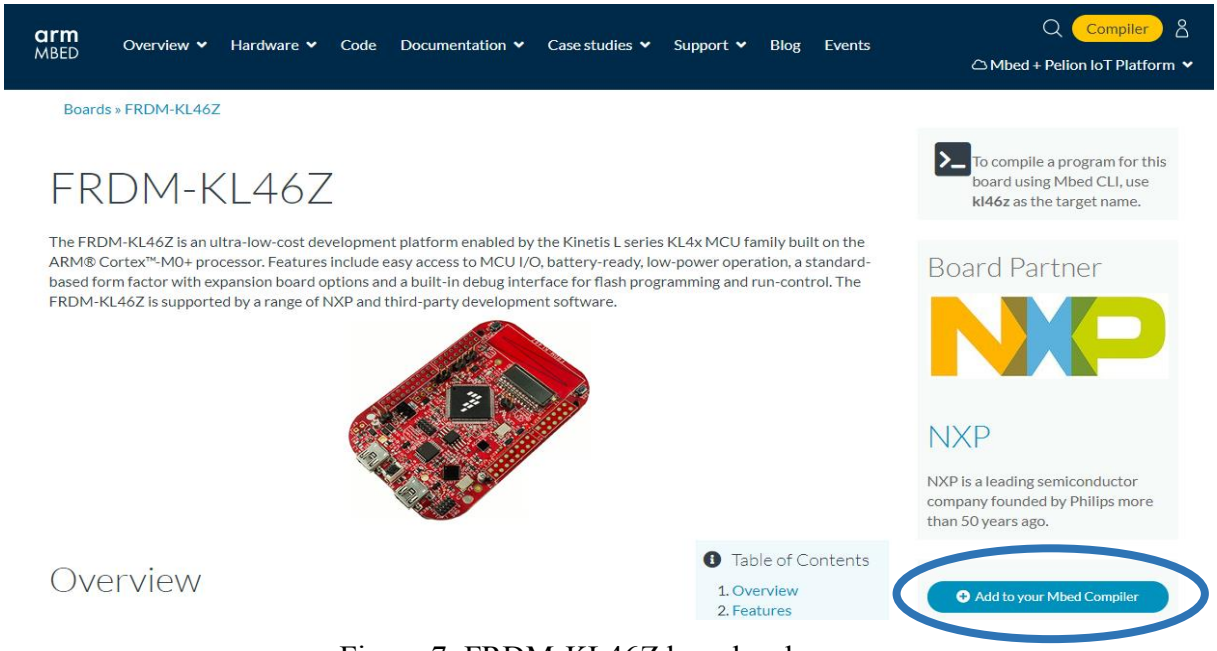


Figure 7: FRDM-KL46Z board web page.

At this point, add the FRDM-KL46Z board to your Mbed Compiler by clicking the “Add to your Mbed Compiler” button as shown in Figure 7; you should see the “Platform 'FRDM-KL46Z' is now added to your account!” message at the top of the page.

Read through the information on how to get started in the "Getting Started with mbed" section. From the web page for the FRDM-KL46Z, import the "mbed_blinky" program by clicking on the light blue 'Import program' button, Figure 8.

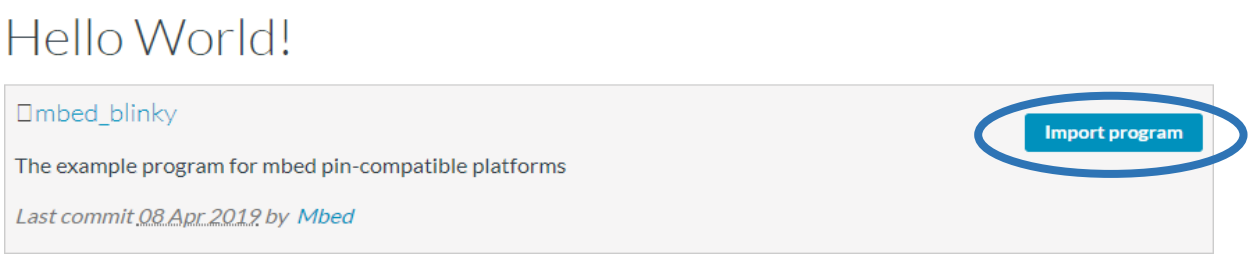


Figure 8: Hello Word program.

The MBED compiler environment should automatically open when you import the program. Then you need to click on the "Import" button to import the program into the compiler by **making sure** to check "Update all libraries to the latest revision". Select "Compile" from the toolbar at the top of the page to compile the C program. This will then create and ask to download the binary executable (mbed_blinky_KL46Z.bin), which can be used with the real board.

The binary executable should be saved on the DAPLINK drive. On Internet Explorer you can use the arrow in the "Save" button to choose "Save as". On Google Chrome a window will appear giving you the possibility to save it directly on the DAPLINK drive. While on Google Chrome and Mozilla Firefox the binary file will be downloaded locally, usually on your "Documents" folder, and you need to copy it into the DAPLINK drive by using File Explorer. After copying the file, the board should restart and the LED1 should flash. In the MBED Compiler environment, edit the main C++ program by changing LED1 to LED2 on line 3. Recompile the program and save the binary file to the board. Observe the new behaviour. Change the program again so that the red and green LEDs flash alternately. Next change the delays (wait) in the program so that the LEDs will flash at a different rate.

Q.16 What's the smallest time you can go to that allows you to see the LEDs flashing? What frequency does this correspond to? f = 30Hz 0.01s 0.01s

13. Exporting to the Keil environment Local IDE

A program can be exported from the Mbed Compiler to the Keil environment. This has the advantage that you can make use of the Keil debugger to debug the application. To export a program:

I. In the Mbed Compiler, in the 'Program Workspace' on the left-hand side of the web page "right-click" on the program and select "Export Program" as shown in Figure 9.

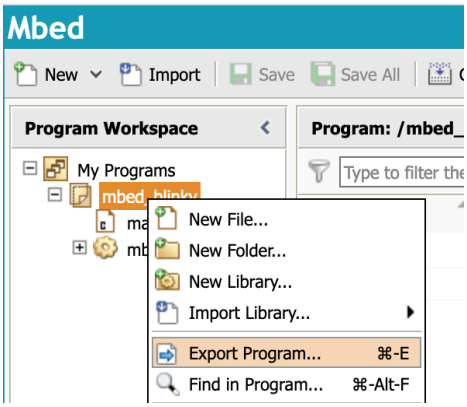


Figure 9: Export a program menu from the Mbed Compiler.

II. Select uvision5-armc6 as the "Export Toolchain" and click on "Export" as shown in Figure 10.

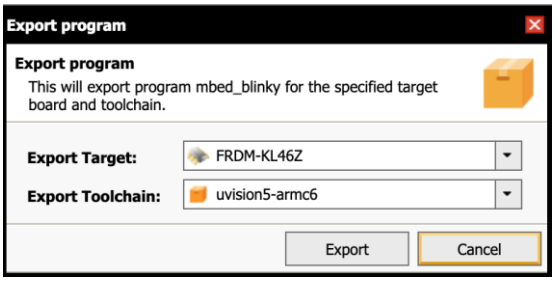


Figure 10: "Export program" dialog window from the Mbed Compiler.

III. Save the zip file to a location of your choice, then open it, **extract it** and place the program directory at a suitable location.

IV. Open the directory and double click on the "uvprojx" file to start Keil uVision.

14. Further investigate the Keil environment

You will further investigate how the Keil environment works. After opening the Hello Word project in start Keil uVision (Step IV in Section 13), perform the following steps:

1) Edit the main C++ program by changing the delays (wait) so that the LEDs will flash together at the same rate.

2) Select 'Options for Target' in the 'Project' drop down menu (Alt-F7 can also be used). Click on the 'Target' tab and then select from the drop-down menu 'ARM Compiler' on the right the 'Use default compiler version 6' compiler version. Click on the 'Linker' tab and select the radio button 'Use Memory Layout from Target Dialog'. Click on the 'Utilities' tab on the far right and then click on the 'Settings' button. A new window

entitled 'CMSIS-DAP Cortex-M Target Driver Setup' should appear. Make sure that in the dialog "Programming Algorithm" the 'MKXX 48Mhz 256kB Prog Flash' is present, if not, click on the 'Add' button and from the list select 'MKXX 48Mhz 256kB Prog Flash'. Click on 'Add' to close the window and then 'OK' once to close another window. Now, in the 'Options for Target' window, click on the 'Debug' tab. The radio button next to 'Use' should be selected (i.e. not the radio button next to 'Use Simulator') and 'CMSIS-DAP Debugger' should be showing in the adjoining box. If it is not, then select it from the drop-down list. Finally click on 'OK' to close the 'Options for Target' window.

3) Then build the project by selecting 'Build target' in the 'Project' drop down menu. F7 can also be used.

4) Download the program to the Freescale Freedom board by selecting 'Download' in the 'Flash' drop down menu. F8 can also be used.

As you can see the LEDs are not flashing as when you have copied the binary executible file inside the DAPLINK drive. This because the board is reprogrammed and automatically restarted by Windows. While by downloading the program with Keil, the board is reprogrammed but not restarted. Therefore, to let the board run the program, you need to manually restart it by pressing the 'RESET switch' button (Figure 1).

15. Serial communication

In addition to the two LEDs which can be controlled by a user program, the Freescale Freedom board connects to a PC as a serial device over the USB connection. In this section you are going to create a program that sends data over the USB serial link and displays it on the PC. For the terminal program running on the PC, you should use "PuTTy 0.70" which is available to install from the University's managed windows service by using the "Install University Applications" application from the Desktop.

I. On the toolbar in the MBED Compiler environment select "New"->"New-Program". This should open a "Create New Program" dialog window.

II. In the "Template" selection box, select "frdm serial example for the Freescale freedom platform", leave the name as "frdm_serial" and select "OK" as shown in Figure 11.

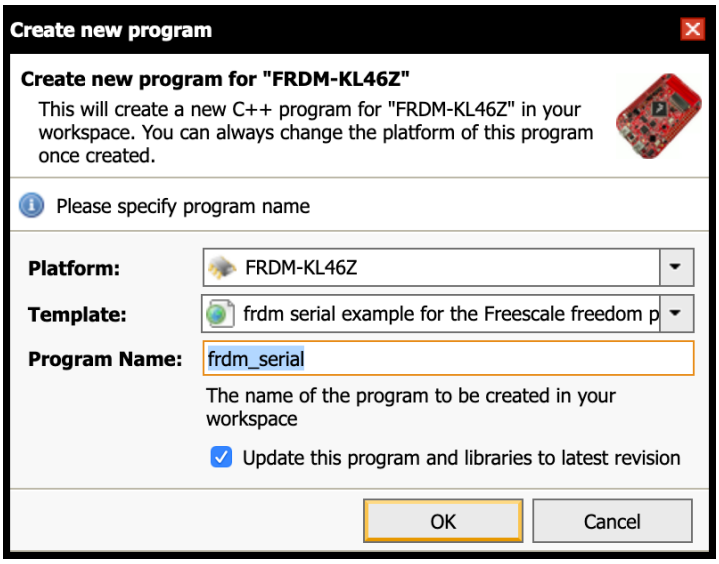


Figure 11: "Create new program" dialog window.

III. Examine the main.cpp program. You should see that it creates an object of type "Serial" and calls it "pc" with the USBTX and USBRX pins as inputs. To use the serial port, the standard "printf" function can be used although you must identify the port using "pc.printf".

IV. Compile and download the program to the board.

V. Next you need to find the serial port number of the board on your PC, to do it open the Windows' Control Panel, then "Device Manager" and expand the Ports (COM and LPT) section (in this example it is "COM4" as shown in Figure 12).

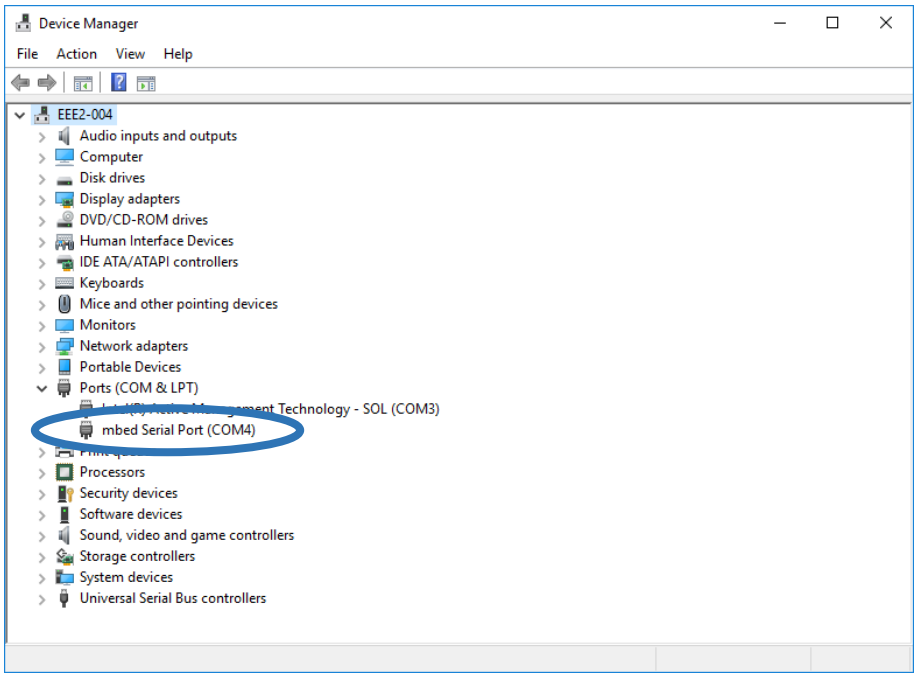


Figure 12: Device Manager window.

VI. Next you need to open a serial terminal on the PC to display the output. Start PuTTy and in the "Configuration" window, select serial at the bottom of the "Category" box on the left-hand side. This should open a new window as shown in Figure 13.

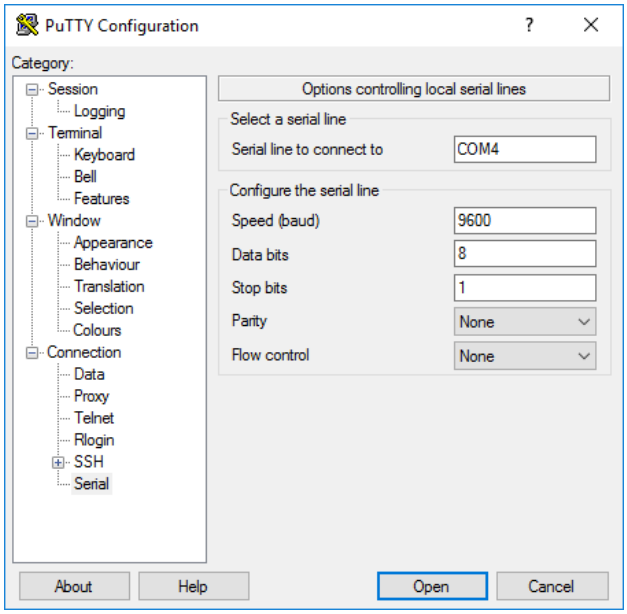


Figure 13: Serial configuration for PuTTy.

VII. You need to select the serial port allocated by the PCs operating system (in this example it is "COM4"), set the speed (baud) to 9600, 8 data bits, 1 stop bit, parity set to 'None' and flow control set to 'None'.

VIII. In the PuTTY configuration window, click on "Session" under "Category" and select the "Serial" radio button (Figure 14).

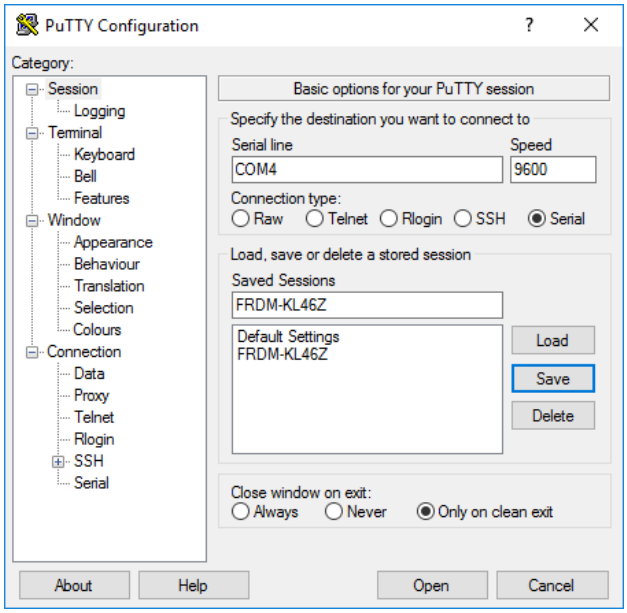


Figure 14: Session category of PuTTy.

IX. Then select “Open” to start the terminal and reset the board to see the messages. Note that the messages appear to “walk” across the screen. This is because the “\n” in C++ is newline rather than newline and carriage return. To add a carriage return, by using the MBED Compiler environment, change the printf in the main C++ program to use “\n\r” instead of just “\n”.

X. A baud of 9600 is quite slow. It can be changed in the main C++ program using the baud command. For example “pc.baud(115200);”. Note that you will need to restart Putty and change its parameters. This can be done by right clicking on the PuTTy window and selecting “Change settings”.

XI. Compile and download the program to the board again to see the effects of the changes.

16. Using the LCD and the Capacitive Touch Slider

The Freescale Freedom board also has a small LCD display and a capacitive touch slider. Go to the Mbed web page and select “Code”, in the search box towards the top of the page, search for ‘FRDM-KL46Z_LCD_DEMO’. Click on the first link and then click on the yellow box that reads ‘Import into compiler’ as shown in Figure 15.

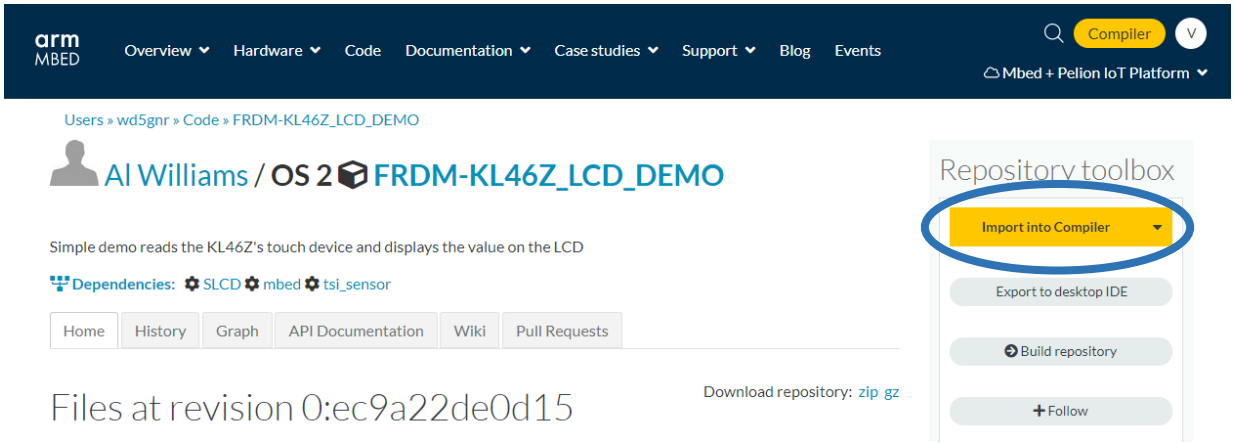


Figure 15: FRDM-KL46Z_LCD_DEMO web page.

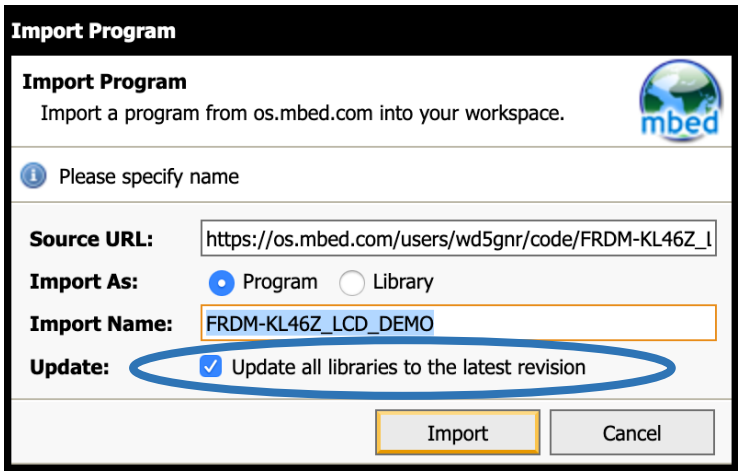


Figure 16: Import of the program FRDM-KL46Z_LCD_DEMO.

In the MBED Compiler environment, import this program by making sure to check "Update all libraries to the latest revision" as shown in Figure 16. Then compile the program and download the binary file to the board. Observe the LCD display whilst touching the capacitive touch slider (Figure 1). Modify the program so that the green LED is illuminated when you touch the slider anywhere between the midpoint and the right-hand end and the red LED is illuminated when you touch the slider anywhere between the midpoint and the left-hand end.

Reminder: the C++ construct for if-then-else is

if (condition) {action if true} else {action if false};

17. Assessment and Marking Scheme

This experiment is assessed by means of a pre-lab test and practical work results submission using the provided template.

The marking scheme for this workbook is as follows:

- The pre-lab test: 10 Marks.
- The Practical work results: 90 Marks (**detailed marking scheme of this part is available in the submission template on Canvas**).

18. Plagiarism and Collusion

Plagiarism and collusion or fabrication of data is always treated seriously, and action appropriate to the circumstances is always taken. The procedure followed by the University in all cases where plagiarism, collusion or fabrication is suspected is detailed in the University's Policy for Dealing with Plagiarism, Collusion and Fabrication of Data, Code of Practice on Assessment, Category C, available on:

http://www.liv.ac.uk/tqsd/pol_strat_cop/cop_assess/appendix_L_cop_assess.pdf.

Follow the following guidelines to avoid any problems:

(1) Do your work yourself.

(2) Acknowledge all your sources.

(3) Present your results as they are.

(4) Restrict access to your work.

Version history

Name	Date	Version
Dr V Selis	April 2022	Ver. 7
Dr V Selis	March 2021	Ver. 6.1
Dr V Selis	March 2020	Ver. 6
Dr V Selis	February 2020	Ver. 5.4
Dr M López-Benítez	September 2019	Ver. 5.3
Dr V Selis	February 2019	Ver. 5.2
Dr L Esteban	March 2018	Ver. 5.1
Dr J Marsland and Dr A Al-Ataby	November 2015	Ver. 5
Dr A Al-Ataby	November 2014	Ver. 4.1
Chao Zhang, Prof J S Smith and Dr A Al-Ataby	November 2013	Ver. 4.0
Dr A Al-Ataby	October 2012	Ver. 3.2
Dr J S Marsland and Dr A Al-Ataby	November 2011	Ver. 3.1
Dr J S Marsland	October 2010	Ver. 3.0

Feedback:

If you have any feedback for this experiment (e.g. timing, difficulty, clarity of script, demonstration...etc.) and suggestions to how the experiment may be improved in the future, please write them down in the space below. This feedback is important for future versions of this script and to enhance the laboratory process, and will not be assessed. If you wish to provide this feedback anonymously, you may do so by sending via email this page to the Student Support Office at studyenq@liverpool.ac.uk.

Script re-writing award
If you think that this experiment could do with enhancement or changes and you have some ideas that you'd like to share, why not re-write this script