University of Limerick

Autumn 2021 ET4011/ED5011 Lab Book 2

Dr Ciaran MacNamee, Dept ECE

ULStaff
20-10-2021

ET4011 Lab Book (Part 2)

Circuit Simulation Software Tool

In Academic Year 2021/22 we will use the Deeds Digital Circuit Simulator to build up circuits using logic gates and flip flops (memory elements). The website for the Deeds Simulator can be found at the following link: https://www.digitalelectronicsdeeds.com/index.html and you can download the software at: https://www.digitalelectronicsdeeds.com/downloads.html. You will find online tutorials showing how to download the software to your Windows computer and how to get started on Sulis (Week 4).

We will repeat this exercise in Week 9, in the Lab 5 Session in a simple demonstration.

Lab 5, Part 1: Four Bit Adder, Eight Bit Adder, Decoders

Aims

- To build a Four-Bit Adder from Half Adder and Full Adder designs

Half Adder

Recall the Half Adder Circuit you have used in the past. As seen in Deeds a half adder is given in Figure 1:

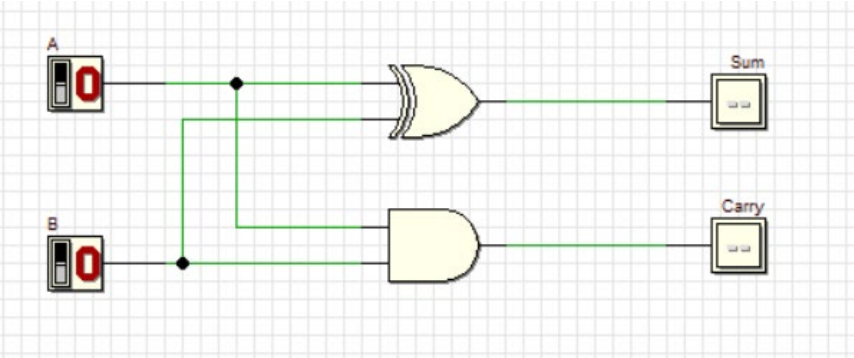


Figure 1: Half Adder Design

We can represent this is a reusable block (Figure 2):

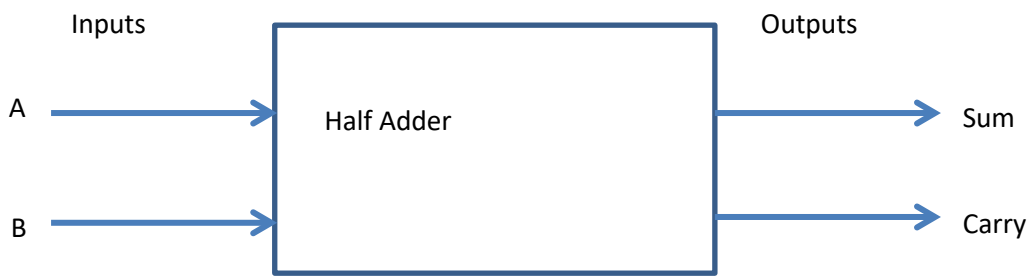


Figure 2: Block diagram of half adder circuit.

Exercise 1: Build a Half-Adder Circuit and use it to fill in the following Truth Table and Verify that it implements a Half Adder (note: you have done this in previous lab work).

A	B	Sum	Carry
0	0		
0	1		
1	0		
1	1		

Exercise 2: Download the file HalfAdder.CBE from Sulis and save it to the folder (directory) you are using for this lab. Build a Half-Adder Circuit using this Half Adder Circuit Block found on Sulis.

Open a New Circuit in Deeds or Start it again. You will get the following screen (Figure 3):

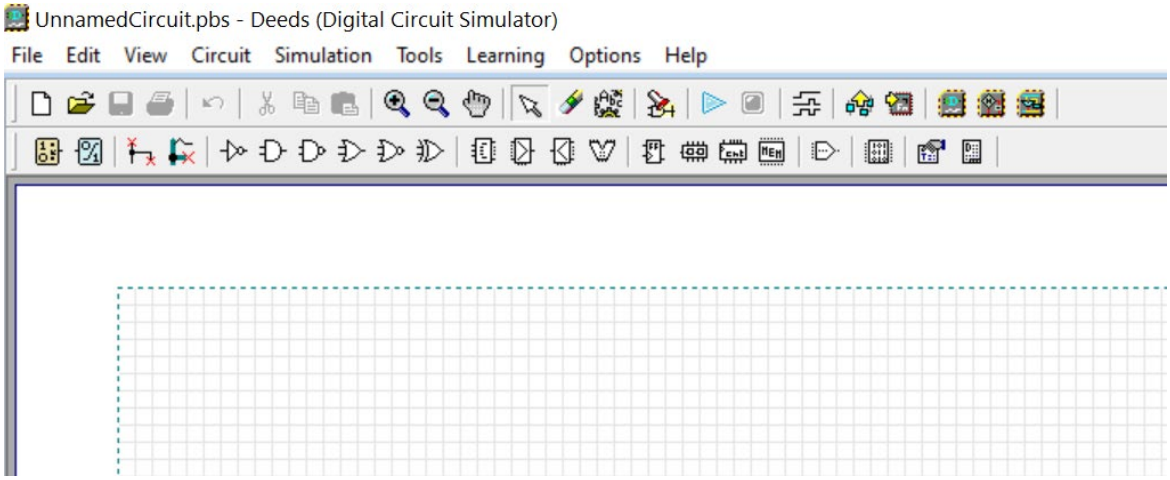


Figure 3: Deeds opening screen

Now select the Add Circuit Block Element as in Figure 4 below:

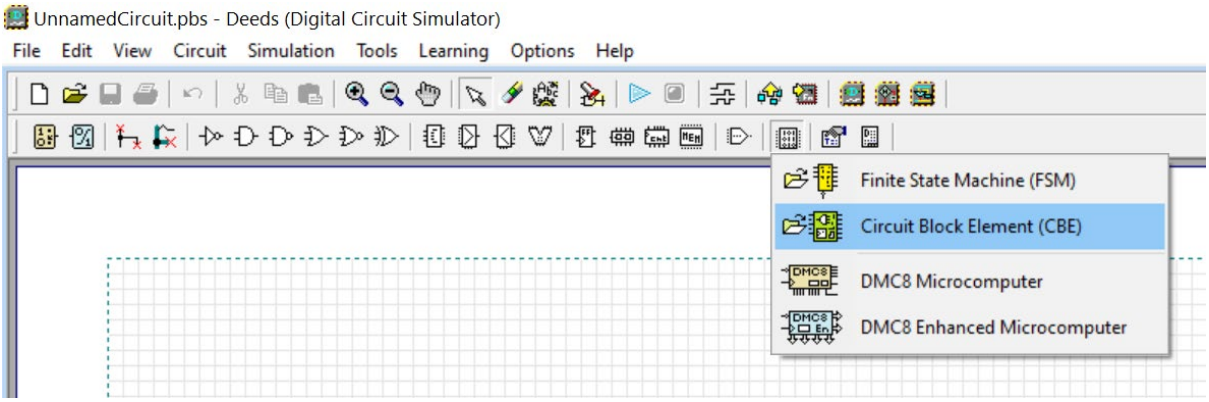


Figure 4: Adding a Circuit Block Element in Deeds

Browse to the folder where you saved the HalfAdder.CBE file and select it (I saved my file in a folder called Lab Week 7, but you may have used a different folder). See Figure 5.

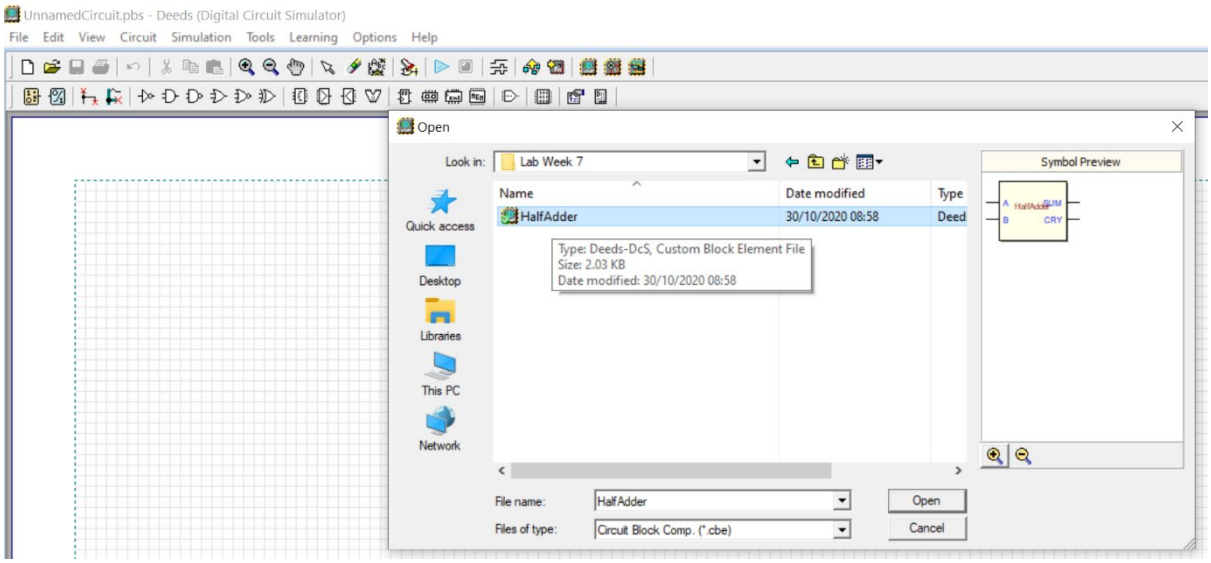


Figure 5: Finding a HalfAdder CBE to add to your circuit.

Click on Open to add this to your circuit and place it on the circuit in the same way that you place gates. You can add a label to identify the component and click on the Green Tick to add the block, as in Figure 6.

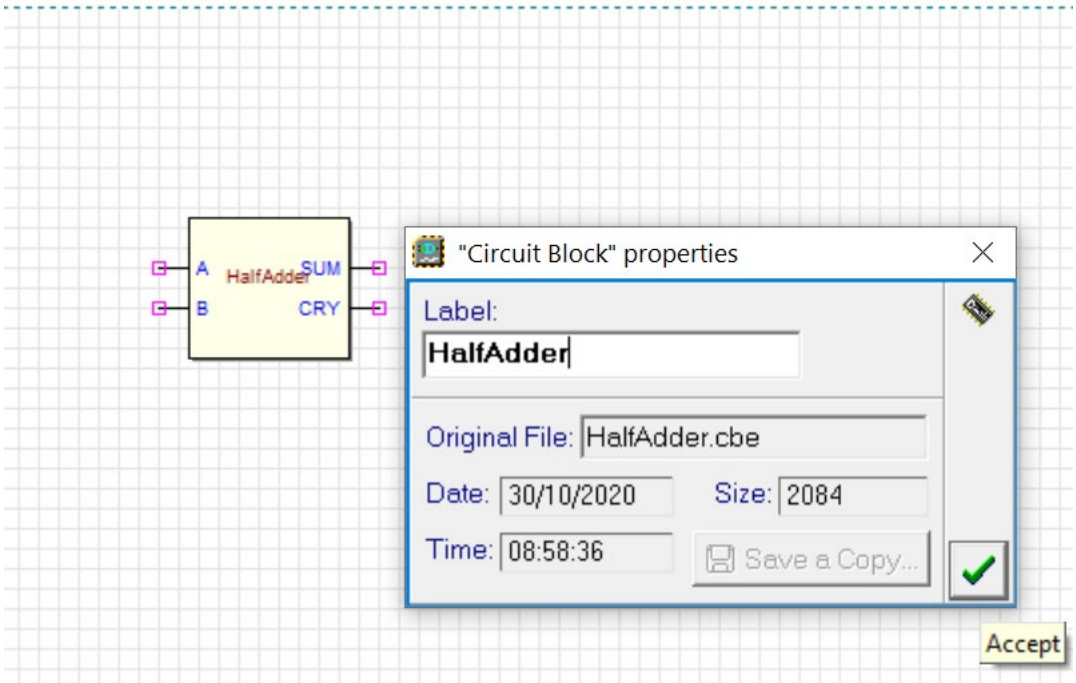


Figure 6. Adding the HalfAdder CBE to your circuit.

Finally add inputs and outputs (Figure 7) and Save your circuit.

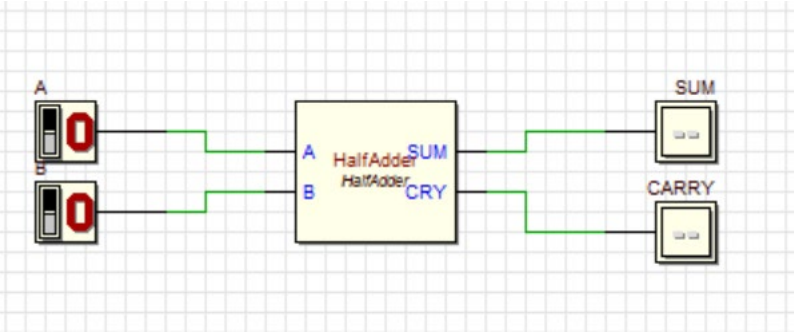


Figure 7. Completing the HalfAdder circuit.

Use the Simulator to fill in the following Truth Table and Verify that it implements a Half Adder.

Truth Table

A	B	Sum	Carry
0	0		
0	1		
1	0		
1	1		

While the Simulator is active, click on the HalfAdder Block, as in Figure 8, to see the circuit that it contains (this only works when the simulator is active):

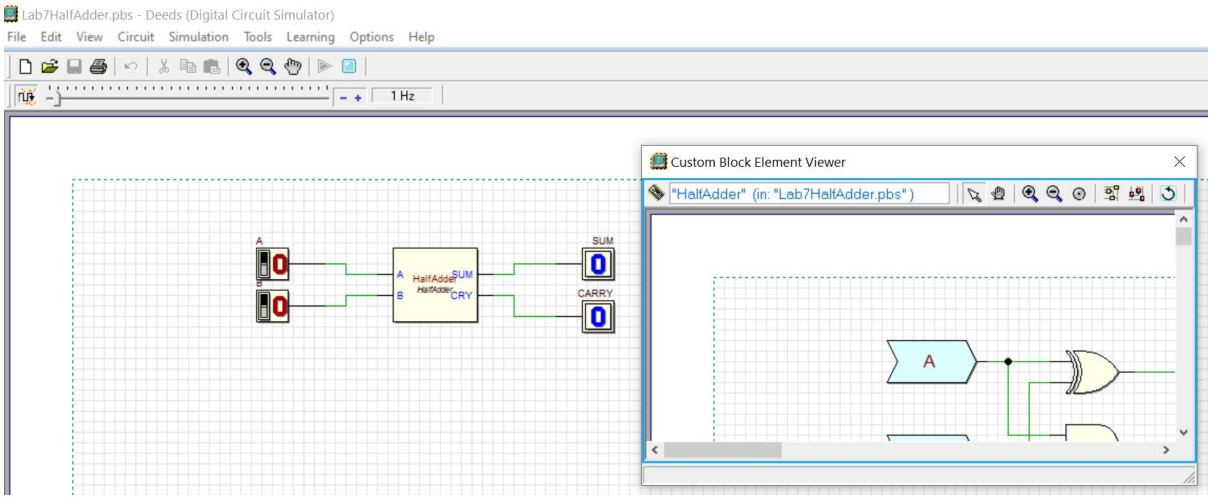


Figure 8. CBE Viewer

(You can expand the Custom Block Element Viewer if required).

Full Adder

We can combine a Half Adder with a second half adder to allow us to add a Carry In to the two bits we are adding. This is a Full Adder. This is shown in Figure 9 in block diagram form.

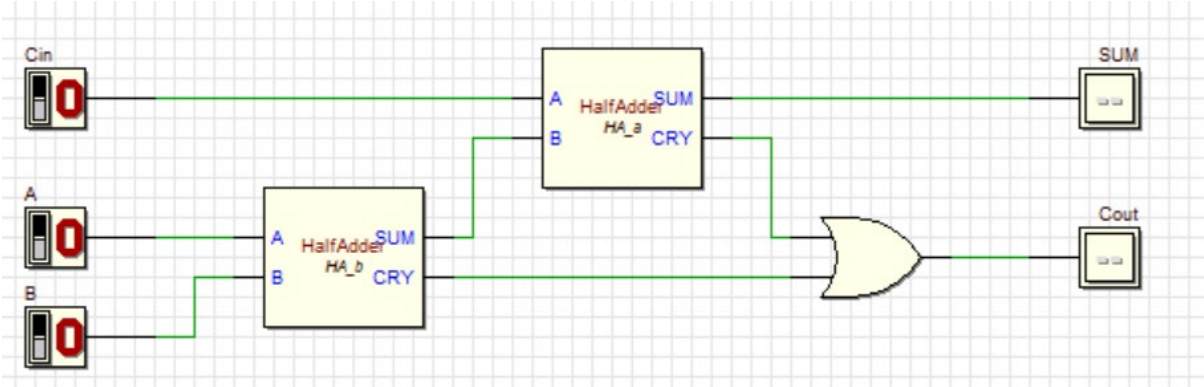


Figure 9: Block Diagram of a Full Adders

Exercise 3:

In Deeds, open a new circuit build this circuit, again using the half adder block, HalfAdder.CBE, this time added twice.

Verify that you get the correct result by filling in the Truth Table below.

A	B	Cin	Sum	Cout
0	0	0		
0	1	0		
1	0	0		
1	1	0		
0	0	1		
0	1	1		
1	0	1		
1	1	1		

More bits:

Download the Circuit Block, FullAdder.CBE from Sulis to the folder you are using, and use it to build the following circuit, Figure 10:

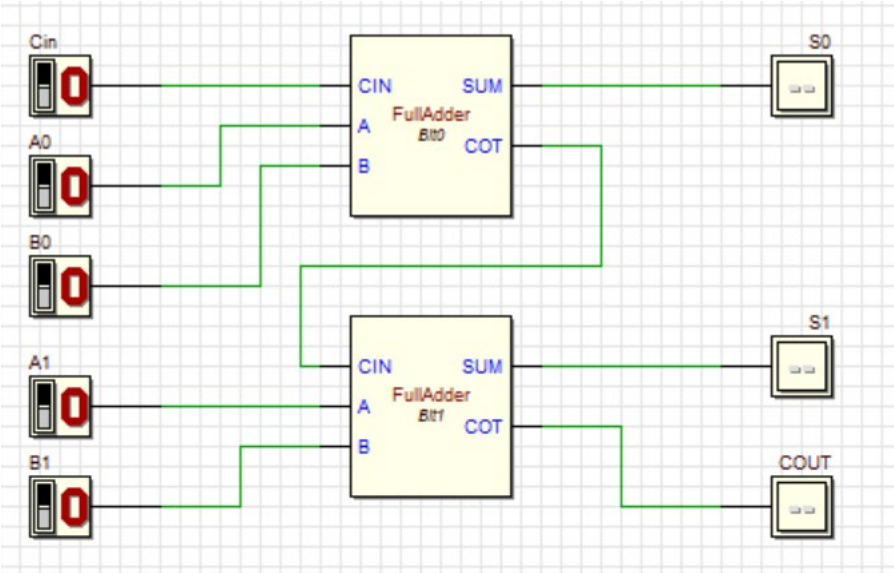


Figure 10: Two Bit Full Adder

Exercise 4:

Build this circuit and use it to verify that the adder is adding two bits correctly, using the following Truth Table. The first three rows are filled in as a guide. Cin is kept low (0) always. Not all possible inputs are used.

Exercise 4. Two bit Full Adder Truth Table

A (binary)	A (decimal)	B (binary)	B (decimal)	Sum (binary)	Sum (decimal)	Cout
00	0	00	0	00	0	0
01	1	01	1	10	2	0
01	1	10	2	11	3	0
01	1	11	3			
10	2	01	1			
10	2	10	2			
11	3	10	2			
11	3	11	3			

Four-Bit Adder:

Deeds also gives us a Four Bit Full Adder. You can find it under the Arithmetic Circuits Menu Item as in Figure 11.

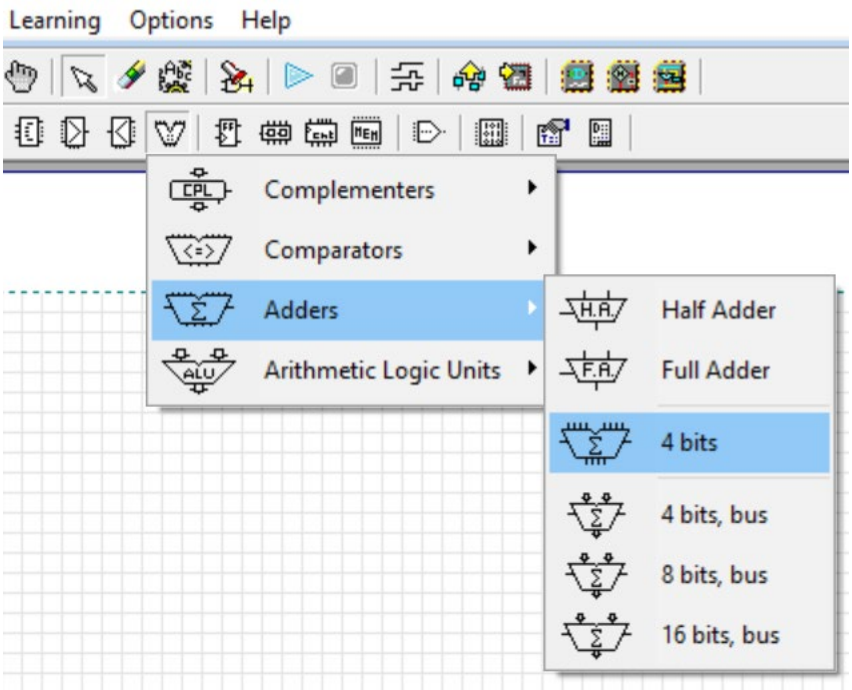


Figure 11. Finding the Four Bit Adder in Deeds

Place the 4-bit adder on a new circuit and connect as shown in Figure 12. You will find the '0' input under the input drop down. You may also load this circuit from Sulis – see FourBitFullAdder.pbs (Deeds).

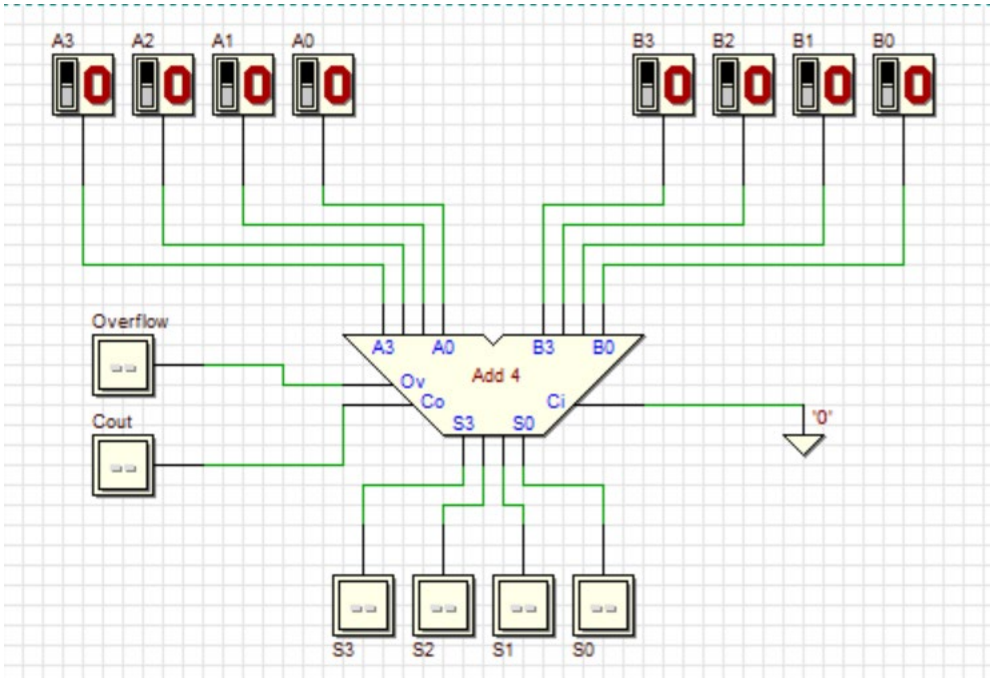


Figure 12. Four-bit full adder with inputs and outputs

Now use the Four-bit Adder to add the following numbers and verify that the result is what you would expect. The first two rows are filled in as a guide. We will ignore the Overflow output for now.

Exercise 5. Four Bit Full Adder Truth Table (partial)

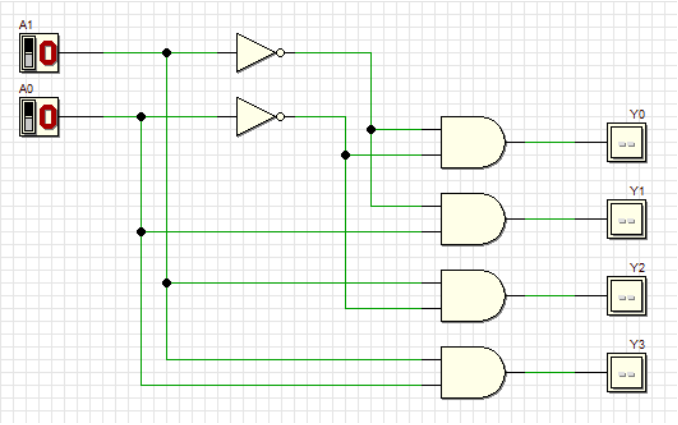
A (binary)	A (decimal)	B (binary)	B (decimal)	Sum (binary)	Sum (decimal)	Cout
0000	0	0000	0	0000	0	0
0010	2	0001	1	0011	3	0
0100	4	0110	6			
1001	9	0101	5			
0111	7	0111	7			
1010	10	0110	6			
1100	12	0101	5			
1110	14	0001	1			

Note: the four binary bits are numbered with the most significant bit as the leftmost bit. For example, if A is shown as 0101, this means A3=0, A2=1, A1=0, A0=1. Each number is also shown in decimal form for information purposes.

Lab 5, Part 2: Decoders, and Multiplexers

Decoders

A decoder is a logic circuit that ensures that only one output of 2^n possibilities is enabled for any given set of n inputs. A Two Input to Four Output Decoder is shown in Figure 1.



You do not need to build this circuit. Build the following one.

Figure 1. Two Input to Four Output Decoder

Very often, we want the option to make sure that no output is set to 1, so we may include an Enable input as in Figure 2.

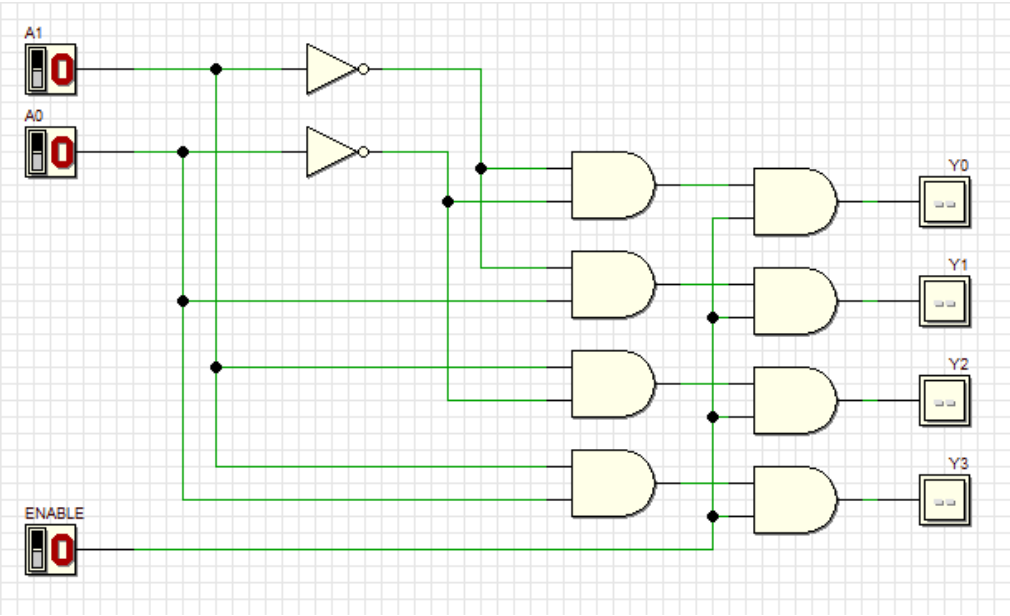


Figure 2. Two Input to Four Output Decoder with Enable

Exercise 1. Build the 2 to 4 Decoder with Enable shown in Figure 2 and use your circuit to fill in the following Truth Table

Truth Table 1. Two to Four Decoder with Enable

ENABLE	A1	A0	Y3	Y2	Y1	Y0
0	X	X				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

(where X means don't care – any value will do)

Multiplexers:

A multiplexer is a circuit that allows a decoder to select one of a number of input sources to appear at an output.

Build the circuit in Figure 3 or download it from Sulis (MUX_4_1.pbs), and use it to fill in the partial truth table, Table 4, (not all possible inputs are used).

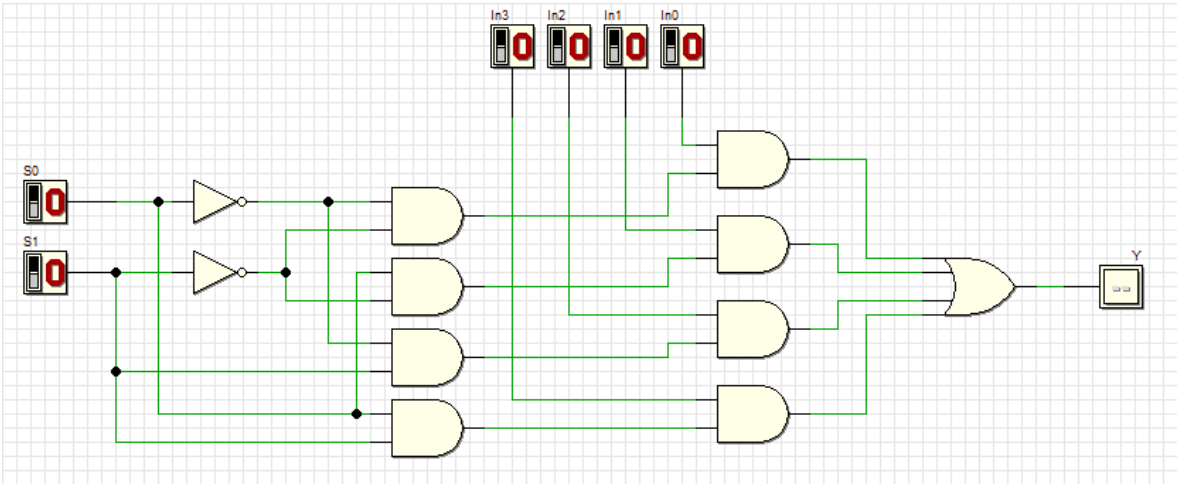


Figure 3 Four to one multiplexer

Table 2 Four to One Multiplexer Truth Table (partial)

S1	S0	In3	In2	In1	In0	Y
0	0	0	1	0	0	
0	0	0	1	0	1	
0	1	1	1	0	1	
0	1	0	0	1	0	
1	0	1	0	0	1	
1	0	0	0	0	1	
1	1	0	1	1	1	

Demultiplexers:

A demultiplexer is a circuit that performs the reverse operation to a multiplexer. It has a single input, 'n' selection lines and a maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines.

Build the circuit shown in Figure 4 and use it to fill in the Truth Table of Table 6.

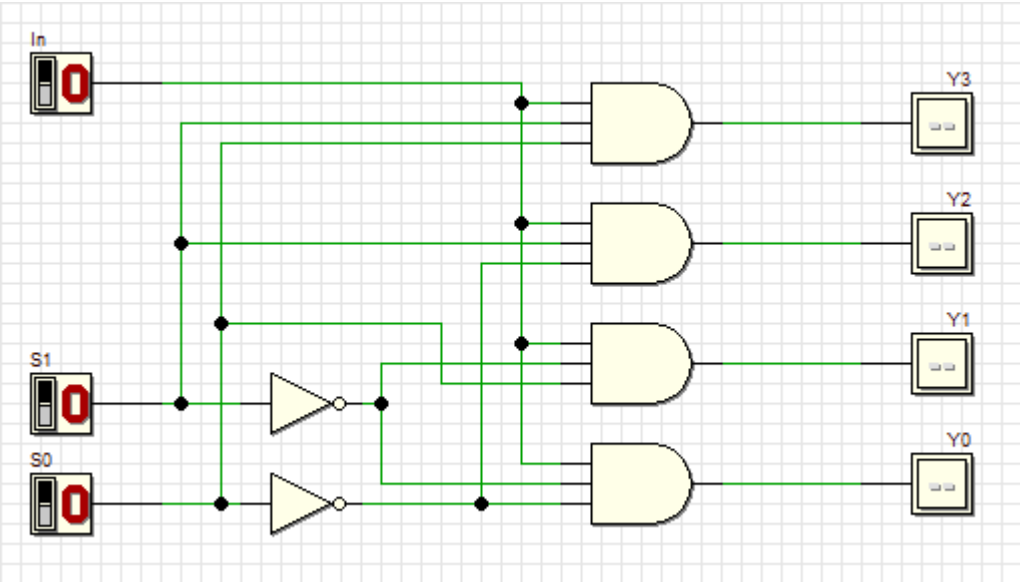


Figure 4. One to Four Demultiplexer

Table 3 One to Four Demultiplexer

S1	S0	In	Y3	Y2	Y1	Y0
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Lab 6: Digital Logic and ALU Implementations

Material for this lab is available on Sulis; you should upload your work to Sulis at the end of the lab session. The files for this session include an answer sheet. Fill in this sheet as you go through the exercises from 1 to 4.

Exercise 1

Download the file **Simple_ALU.pbs** from Sulis and open it in Deeds. Fill in the copy of Table 1 below in the Answers Sheet to verify that this simple ALU works as you would expect it. (Note that this is not a complete truth table).

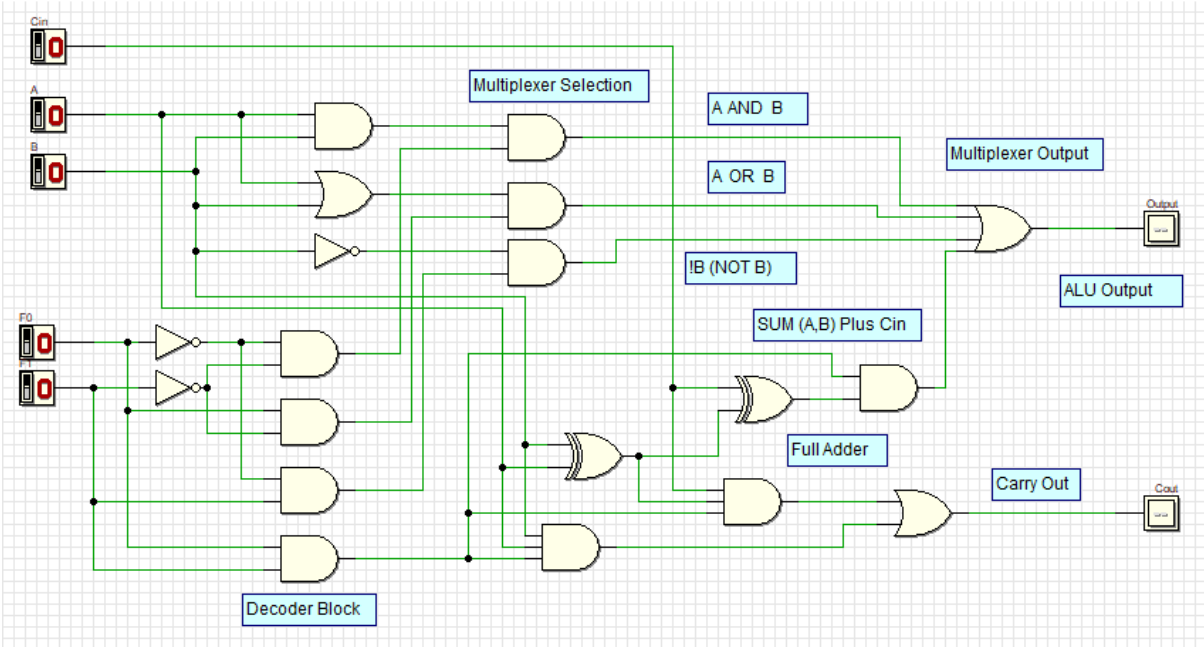
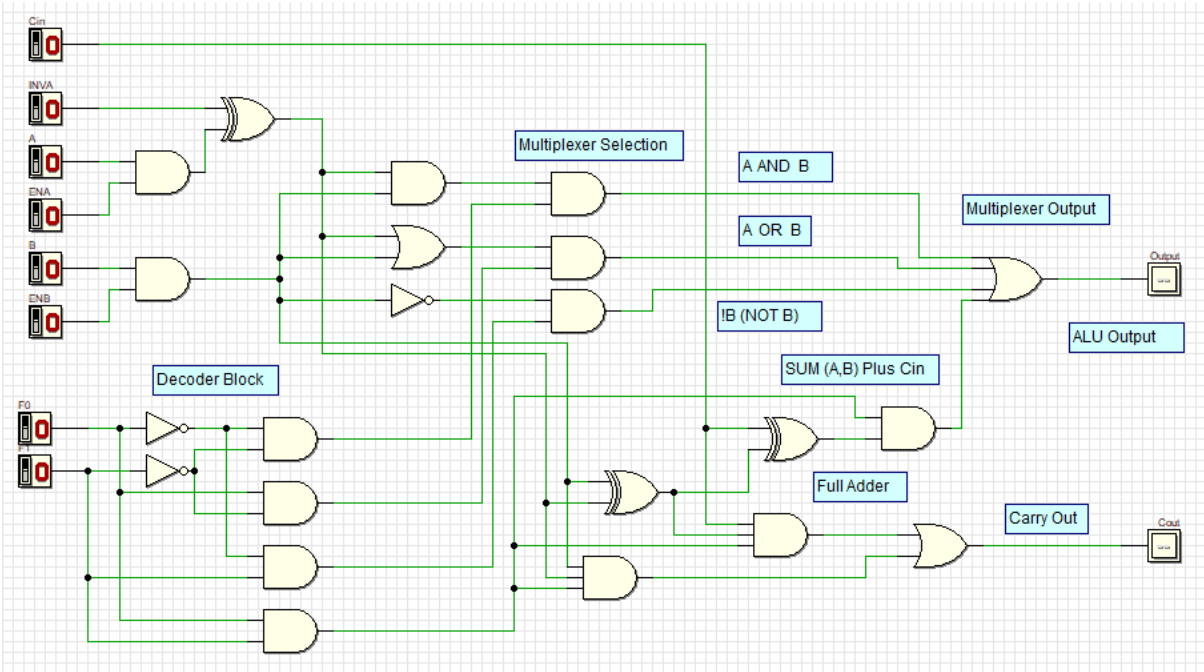


Table 1. Exercise 1 Truth Table (Incomplete)

F1	F0	Function Selected	A	B	Cin	Output	Cout
0	0		0	1	1		
0	0		1	1	1		
0	1		1	0	1		
1	0		0	1	0		
1	1		0	1	1		
1	1		1	1	1		

Exercise 2

Download the file **Bigger_ALU.pbs** from Sulis to your ET4011 Folder. Open the file in Deeds. Note the differences between this circuit and the previous one.



Fill in a copy of the following table and identify what functions are selected by each possible setting of F1 and F0.

Table 2. Exercise 2 Truth Table

F1	F0	Function Selected	ENA	INVA	A	ENB	B	Cin	Output	Cout
0	0		1	0	0	1	1	1		
0	0		1	0	1	1	1	1		
0	0		1	1	0	1	1	1		
0	1		1	0	0	1	0	1		
0	1		1	0	0	1	1	1		
0	1		1	0	1	1	1	1		
1	0		1	0	0	1	0	1		
1	0		0	0	1	1	0	1		
1	1		1	0	0	1	0	0		
1	1		1	0	1	1	0	1		
1	1		1	0	0	1	1	0		
1	1		1	0	0	1	1	1		
1	1		1	0	1	1	1	0		
1	1		1	0	1	1	1	1		

Exercise 3

Download the file **Four Bit ALU.pbs** from Sulis to your ET4011 Folder. Open the file in Deeds. The Four Bit ALU is built from 4 x 1-bit ALU modules of Exercise 2. The 1-bit ALU circuit can also do a subtraction operation (even though this is only sensible when using more than one bit). Thus, this Four Bit ALU can ADD, OR, INVERT, ADD, *and* SUBTRACT.

The Data inputs for the ALU are A3-A0, B3-B0 and Cin; the control inputs are F1, F0, ENA, ENB and INVA; the data outputs are Cout and Y3-Y0.

The function control settings are:

F1	F0	Function
0	0	Y = A.B
0	1	Y = A+B (OR)
1	0	Y = B' (NOT B)
1	1	Y = A plus B (addition)

The remaining control settings are:

ENA = 1, Enable A, otherwise A is 0

INVA = 1, Use Inverse of A (NOT) A, otherwise use A unchanged

ENB = 1, Enable B, otherwise B is 0

Use the ALU to carry out the following operations, recording your results in the Answers Sheet.

Note that A and B are organised as A3:A2:A1:A0 and B3:B2:B1:B0 below, ie A3 is the MSB.

Table 3 Exercise 3. A = 1011, B = 1110: Y = A.B (Example)

A = 1011, B = 1110: Y = A.B (Example)											
F1	F0	Function Selected	ENA	INVA	A (bits)	ENB	B (bits)	Cin	Y (bits)	Y (hex)	Cout
0	0	AND	1	0	1011	1	1110	1	1010	A	0
A = 0011, B = 1001: S = A+B (OR)											
F1	F0	Function Selected	ENA	INVA	A (bits)	ENB	B (bits)	Cin	Y (bits)	Y (hex)	Cout
			1	0	0011	1	1001	1			
A = 1011, B = 1110: Y = B’											
F1	F0	Function Selected	ENA	INVA	A (bits)	ENB	B (bits)	Cin	Y (bits)	Y (hex)	Cout
			1	0	1011	1	1110	1			
A = 1011, B = 1110: S = A’											
F1	F0	Function Selected	ENA	INVA	A (bits)	ENB	B (bits)	Cin	Y (bits)	Y (hex)	Cout
			1	1	1011	0	1110	1			
A = 0011, B = 1001: S = A plus B (arithmetic addition, 3 + 9), Cin =0											
F1	F0	Function Selected	ENA	INVA	A (bits)	ENB	B (bits)	Cin	Y (bits)	Y (hex)	Cout
			1	0	0011	1	1001	0			
A = 0011, B = 1001: S = A plus B (arithmetic addition, 3 + 9), Cin=1											
			1	0	0011	1	1001	1			
A = 0101, B = 0111: S = B minus A (subtraction, 7-5)											
F1	F0	Function Selected	ENA	INVA	A (bits)	ENB	B (bits)	Cin	Y (bits)	Y (hex)	Cout
			1	1	0101	1	1001	1			
A = 0101, B = 0010: S = B minus A (subtraction, 2-5)											
F1	F0	Function Selected	ENA	INVA	A (bits)	ENB	B (bits)	Cin	Y (bits)	Y (hex)	Cout
			1	1	0101	1	0010	1			
A = 0101, B = 0110: S = minus A											
F1	F0	Function Selected	ENA	INVA	A (bits)	ENB	B (bits)	Cin	Y (bits)	Y (hex)	Cout
			1	1	0101	0	0110	1			

Question: Why does the combination of INVA=1 and Cin=1 produce -A?

Answer: (Fill in your answer in the Table below)

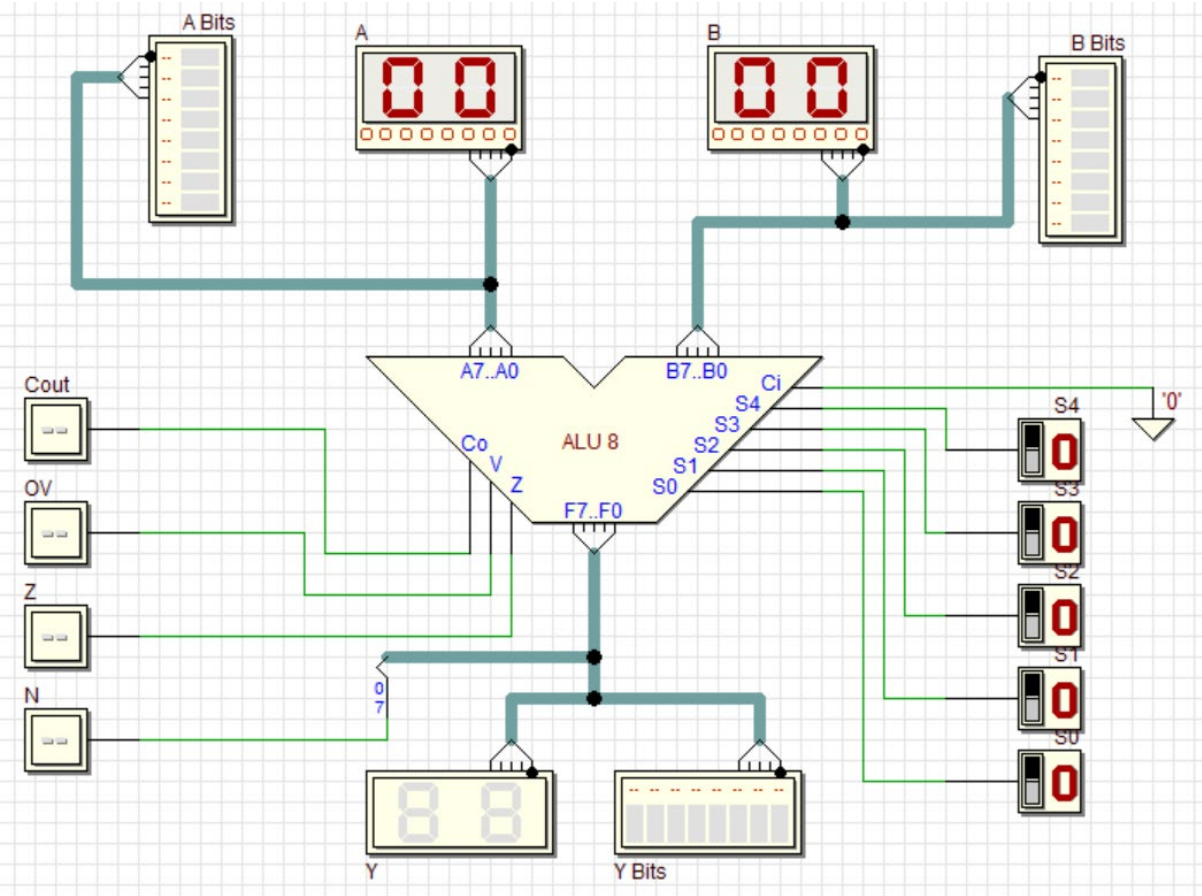
Question Exercise 3

Question:	Why does the combination of INVA=1 and Cin=1 produce -A?
Answer:	

Exercise 4. Using the Deeds ALU.

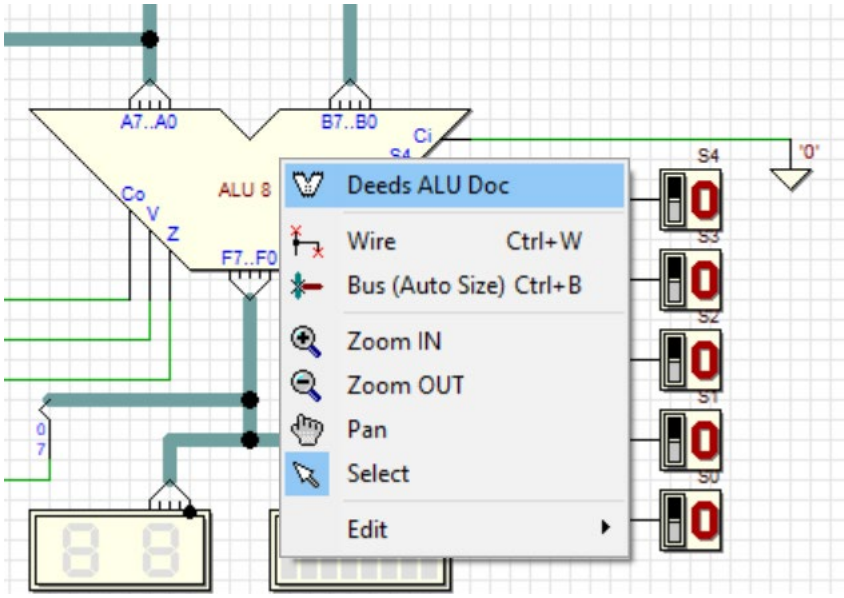
Download the file **Eight_Bit_Deeds_ALU.pbs** from Sulis to your ET4011 Folder. Open the file in Deeds. This circuit uses the Deeds ALU and uses buses to make the connections between multiple inputs and outputs easier to draw (remember that a bus is a set of wires).

The circuit is shown below:



A and B can be entered as hexadecimal characters. A Bits and B Bits are shown to represent A and B in bit form. This is also the case for Y and Y Bits.

This ALU has a different set of control inputs to the one we have used. Right-Click on the ALU in Deeds and then select Deeds ALU Doc to see what they are:



The five bits S4-S0 give you 32 options. A few are given below:

file:///C:/Program Files (x86)/DeedsSuite/guide/DocDDCS/DeedsALU.html

DEEDS - Arithmetic and Logic Units ("ALU 8" and "ALU 16")

ALU 8: A = A07..A00, B = B07..B00, F = F07..F00
ALU 16: A = A15..A00, B = B15..B00, F = F15..F00
S4..S0: Function Selection lines
Ci: Carry Input, **Co:** Carry Output, **Z:** Zero Condition, **V:** Overflow Condition

S4 .. S0	Function	Notes
00000	F = 0	
00001	F = +1	
00010	F = -1	
00011	F = - Negative Minimum	-2^{15} (16-bits version), or -2^{7} (8-bits version)
00100	F = A	
00101	F = B	

Use the description of the ALU to carry out the following operations:

A=0x7b, B=0xe2: F = A AND B, also F = -1 examples shown (Note that hexadecimal is not case sensitive, c and C mean the same thing).

Table 4 Exercise 4

A (Hex)	B (Hex)	S[4:0]	Function	Y (Hex)	Y (Bits)	N	Z	OV	Cout
7b	e2	01000	A and B	62	01100010	0	0	0	0
7b	e2	00010	-1	Ff	11111111	1	0	0	0
87	20		A or B						
87	29		2's comp of B						
35	21		A plus B						
35	a2		A plus B						
35	21		A minus B						
35	A2		A minus B						
35	21		B minus A						
35	A2		B minus A						
35	A2		Not (A) 1's comp						
64	53		Not (B) 1's comp						
27	53		A + 1 (A plus 1)						
27	60		B – 1						
A9	19		A exor B						

Lab 7 Part 1: Getting Started with DS-5 Community Edition and Linaro Bare-metal compiler

ARM v8 Labs: Introduction to the Development Environment

We use the ARM DS-5 Community Edition to develop our ARM Assembly Language programs. **This program should be already downloaded on your lab PCs**, but if you want to install it on your own PC, contact Ciaran MacNamee for a zipped file of the last available version of ARM DS-5. Note that the files are quite large. If you are doing this, unzip the file that you have downloaded. Then run the Setup.exe program to install the DS-5 Community Edition.

Linaro Bare-metal compiler (this section is for home use only)

NB: If the Linaro compiler and DS-5 Community Edition have already been installed on your computer, or if you are working in a UL ECE lab, skip ahead to '3 Getting Started Tutorial'.

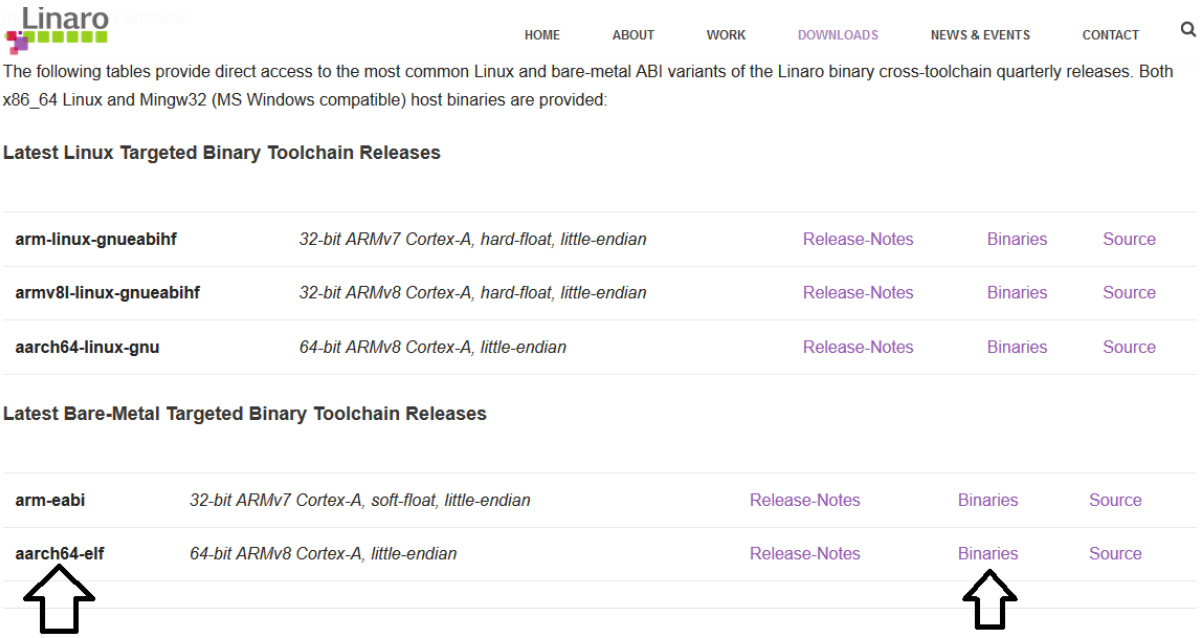
1. Download the Linaro compiler

We use the DS-5 Community Edition in our labs in bare-metal mode, so we need to download a compiler for this mode. You can find the linaro project compiler at this link:

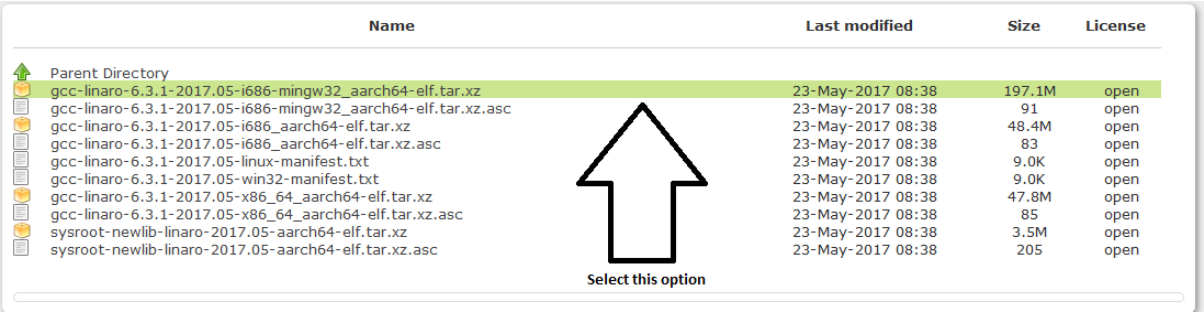
http://www.linaro.org/downloads/

There are a number of options, but the 64-bit ARMv8 Cortex-A, little-endian option is the one to select for this module.

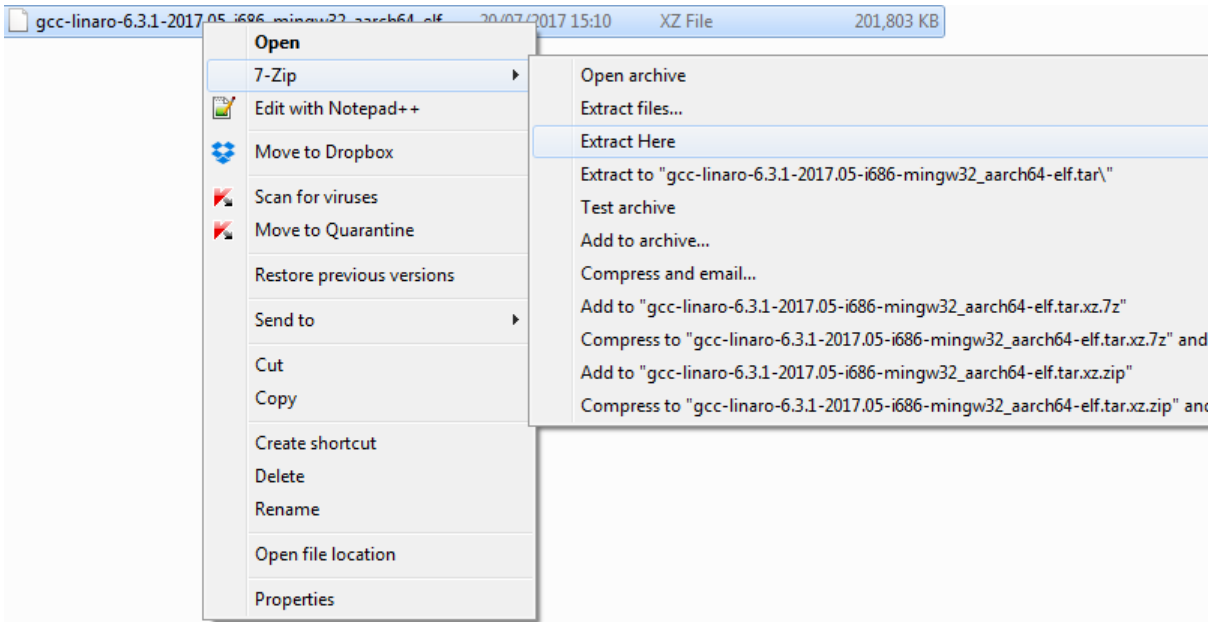
Note that Linaro is often updated so that the latest versions may be different from the ones shown in the screen shots below.



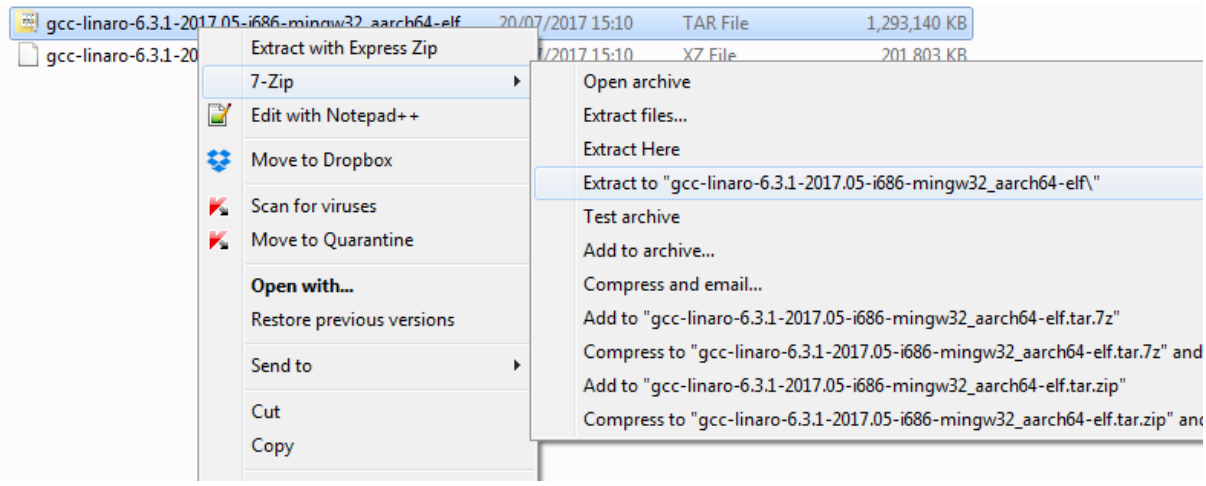
When you select the aarch64-elf release (Binaries) you'll come to the following screen:



Please note that the version you find on the Linaro website may be more recent than the one above. Save this file to a folder you will remember. I saved the file to MyDocuments\linaro. Next you need to extract the file, using winzip, 7-Zip or some other file extraction software. I used 7-Zip:



This will extract a 'tar' file, and you'll have to extract from the tar file again:



Finally you get the compiler files installed on your computer:

gcc-linaro-6.3.1-2017.05-i686-mingw32_aarch64-elf	20/07/2017 15:11	File folder	
gcc-linaro-6.3.1-2017.05-i686-mingw32_aarch64-elf	20/07/2017 15:10	TAR File	1,293,140 KB
gcc-linaro-6.3.1-2017.05-i686-mingw32_aarch64-elf...	20/07/2017 15:10	XZ File	201,803 KB

If you click on the File folder you should see the following folder underneath it:

Documents library
gcc-linaro-6.3.1-2017.05-i686-mingw32_aarch64-elf

Name	Date modified	Type	Size
aarch64-elf	20/07/2017 15:11	File folder	
bin	20/07/2017 15:13	File folder	
include	20/07/2017 15:11	File folder	
lib	20/07/2017 15:11	File folder	
libexec	20/07/2017 15:11	File folder	
share	20/07/2017 15:11	File folder	
usr	20/07/2017 15:11	File folder	
gcc-linaro-6.3.1-2017.05-win32-manifest	18/05/2017 00:25	Text Document	9 KB

2. Install DS-5 Community Edition (this section is for home use only)

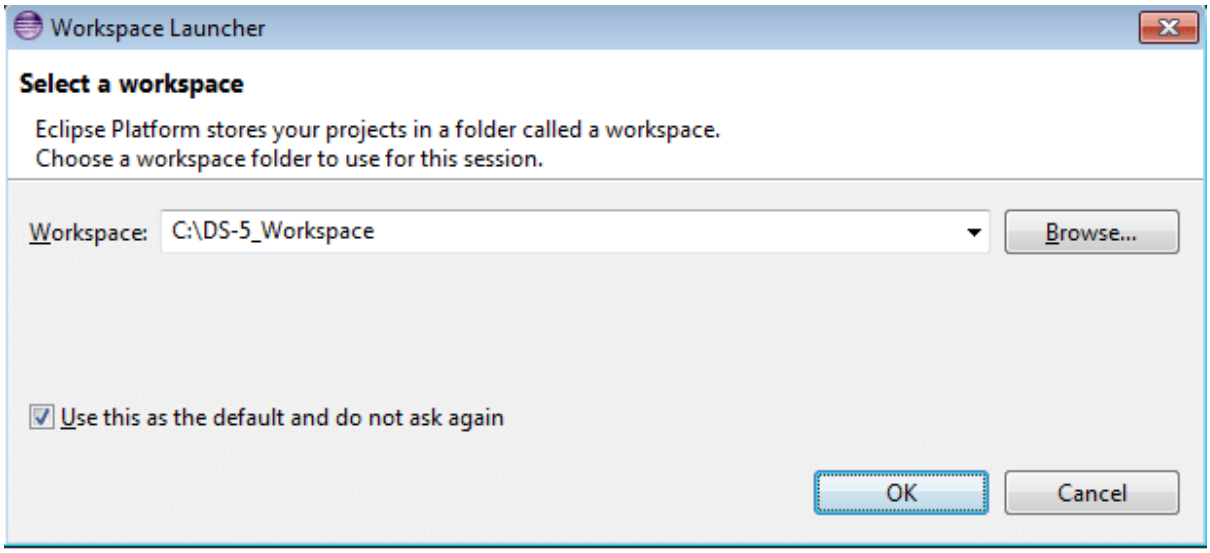
The link at the start of this document brings you to the DS-5 Community Edition page. When you click the Download Community Edition, you'll come to a registration page, which you'll have to fill in, and then you'll get an e-mail link allowing you to download the software. When you get the link, follow the installation instructions, and this will install the software on your PC. Note that the files are quite large so this may take a little time.

3. Getting Started Tutorial

Start DS-5 and set up your Workspace

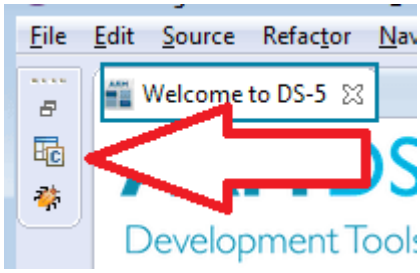
(Based on ARM DS-5 Getting Started Tutorial)

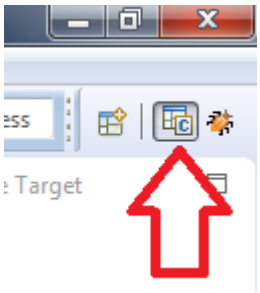
1. To start DS-5 select **Eclipse for DS-5 CE** from the installation folder or from Windows Programs Menu.
2. In the **Workspace Launche**r dialog, accept the default workspace, or click **Browse** and select a folder.



Note that the initial Workspace may be different on your computer.

3. Either close the **Welcome page**, or click **Go to workbench** to view the **C\C++ Perspective**.





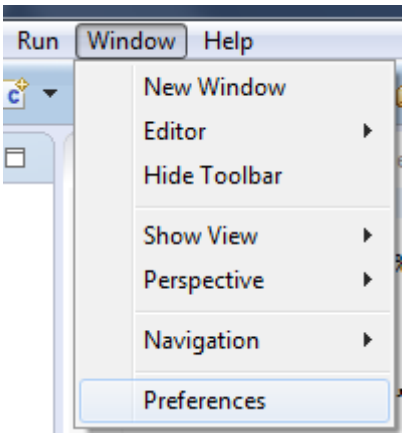
(The C/C++ could be in either of these positions.)

Import a bare-metal compiler

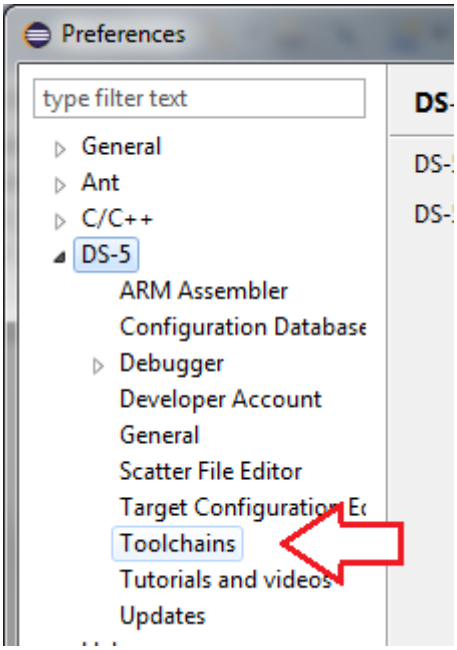
In order to build a project, setup a compilation toolchain in the DS-5 preferences. For this module, we use a bare metal compiler (Aarch64 little-endian) from Linaro. It should have been downloaded to your computer already if you're working in B2042 or B2043.

To setup the toolchain, follow these steps:

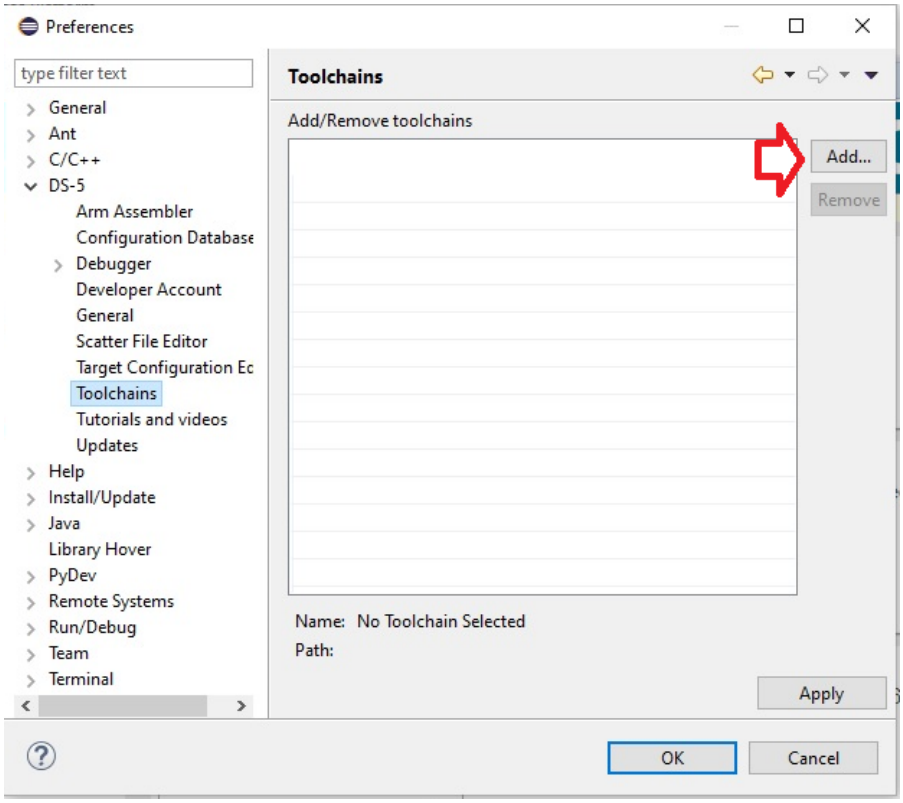
1. Select Window -> Preferences from the Toolbar



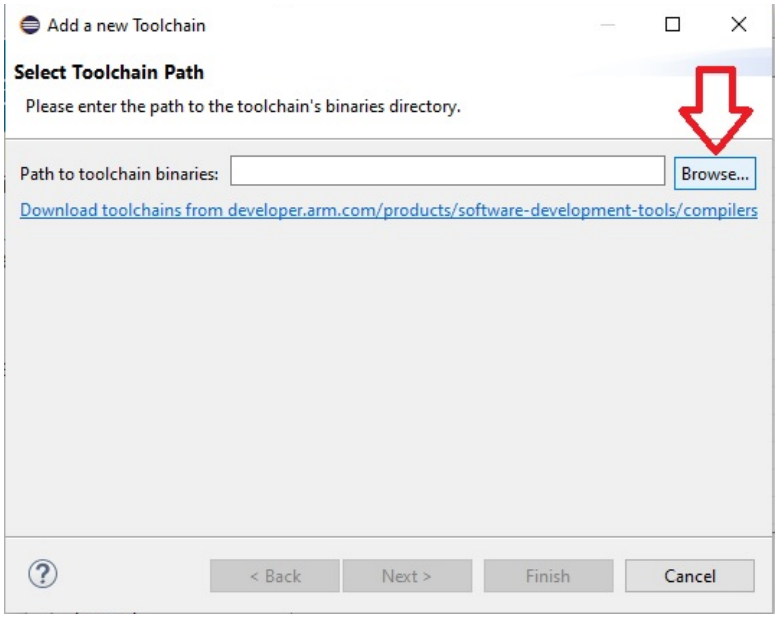
2. Expand DS-5 and Select Toolchain



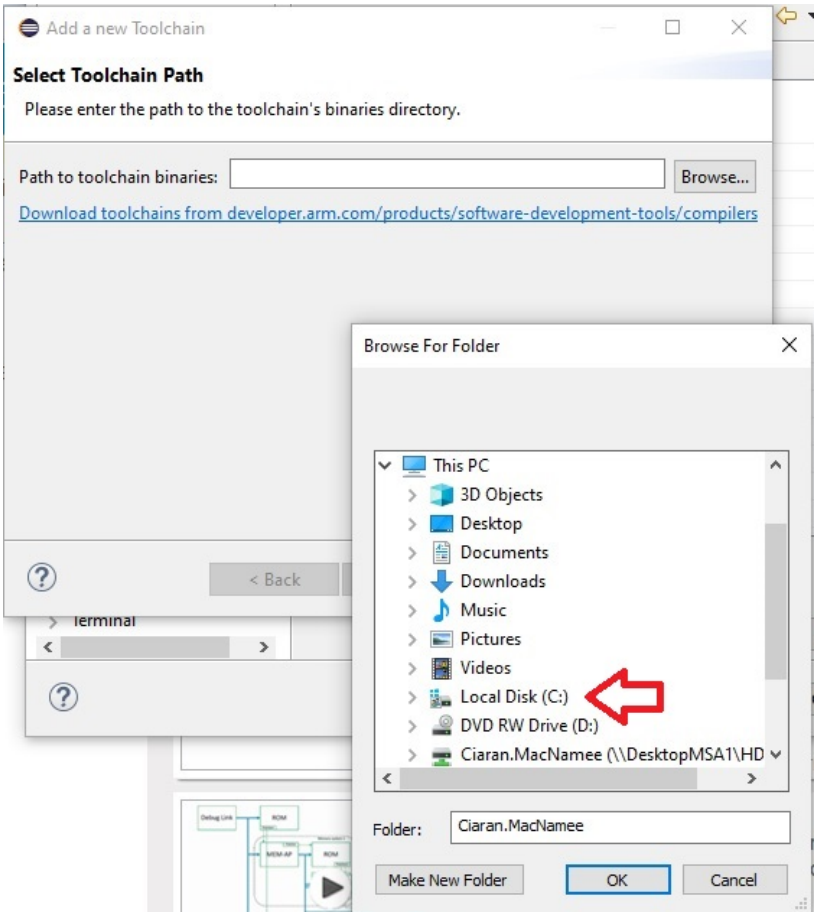
3. Select Toolchains and click Add... in the Add/Remove toolchains section.



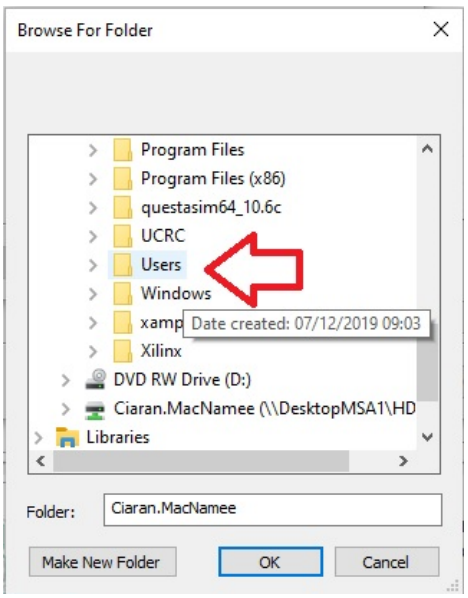
4. Select Browse to locate your toolchain folder.



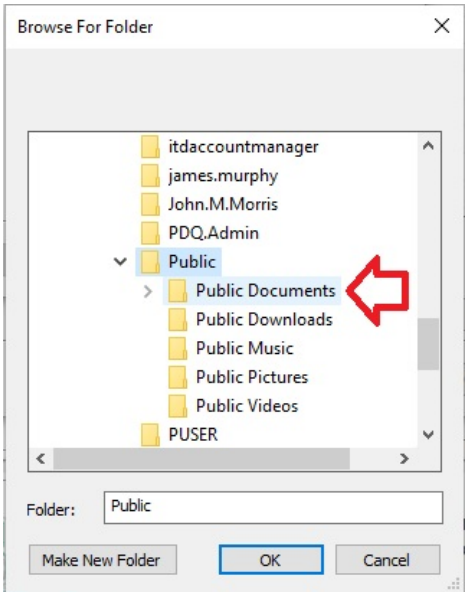
5. Select This PC then Local Disk C:



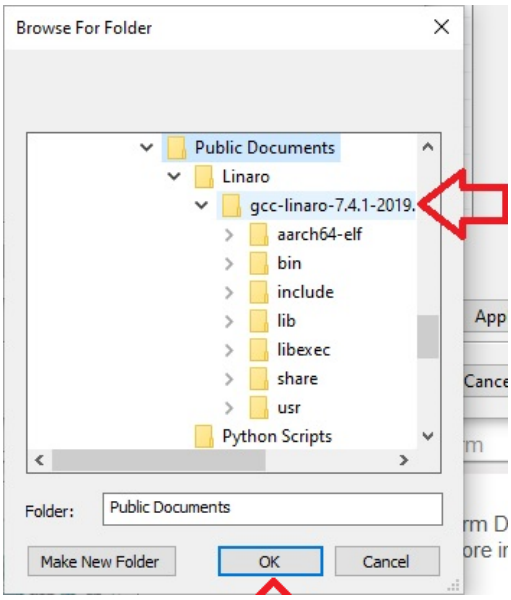
6. Under Local Disk (C:) Select Users



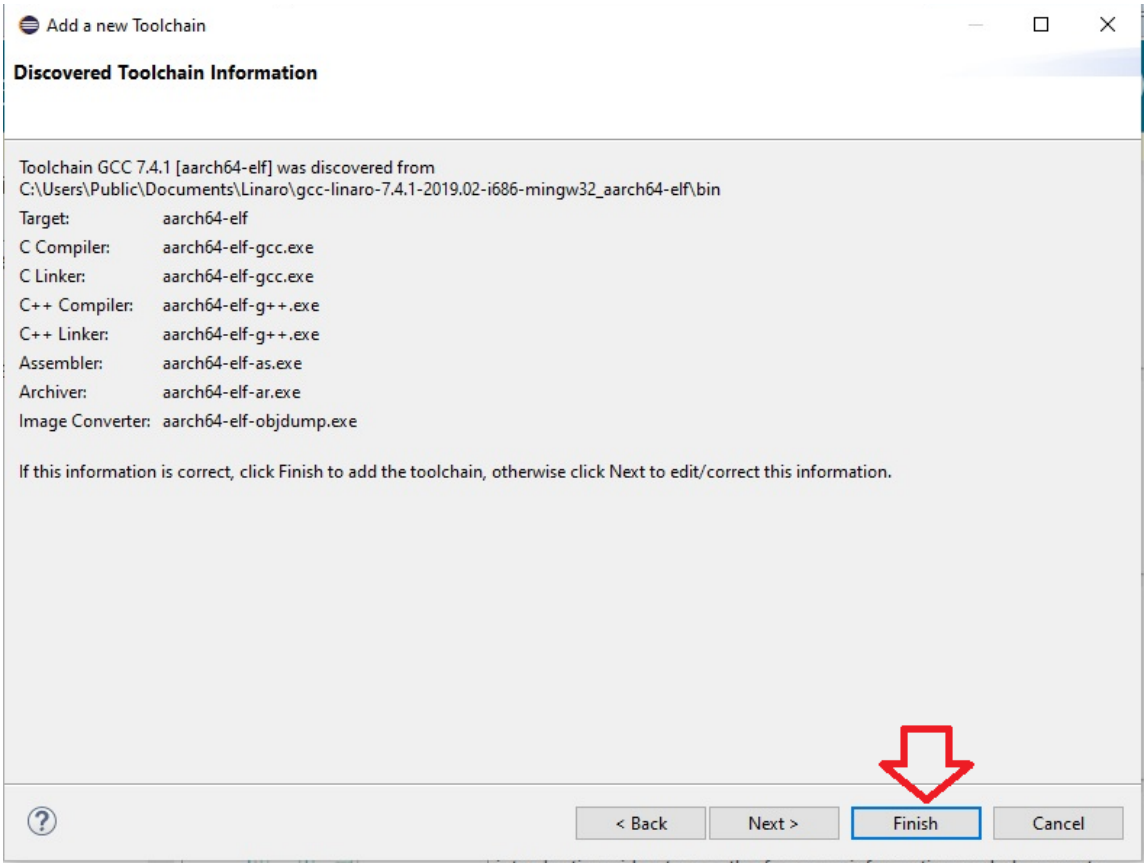
7. Next Select Public, then Public Documents



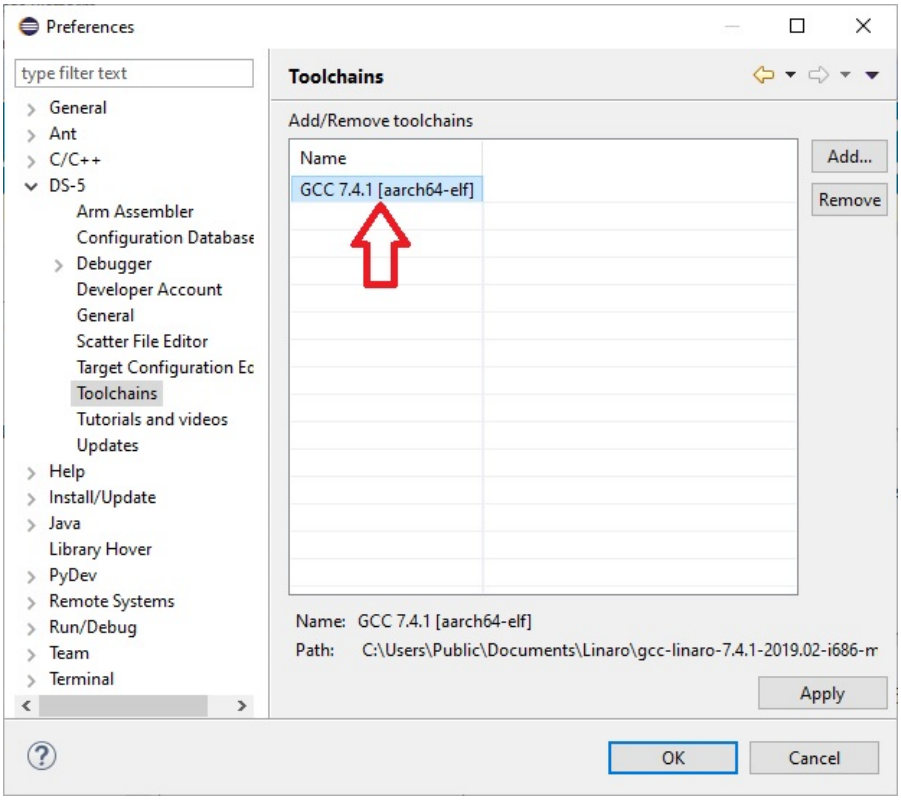
8. Finally Select Linaro, then gcc-linaro-7.4.1-2019 and press OK.



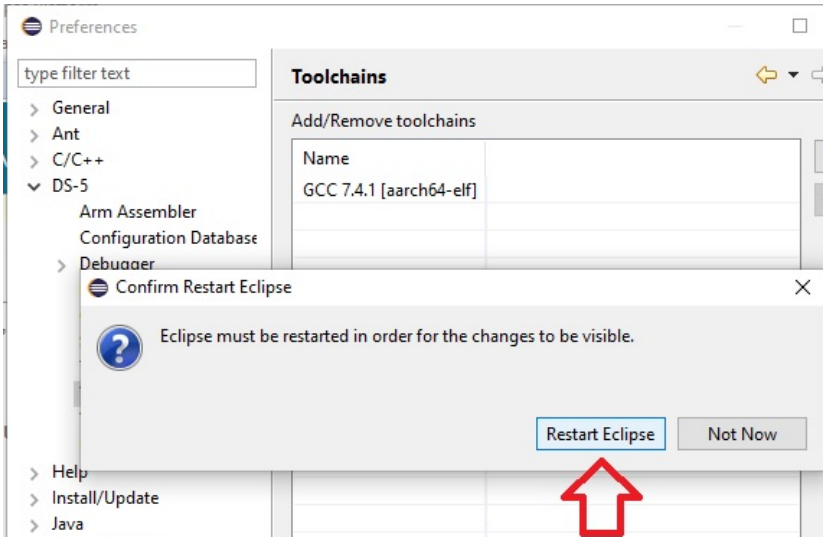
9. Select Next to autodetect the Toolchain. When the toolchain has been detected press Finish.



10. Select Finish when you have selected the Toolchain. Then Click on the ToolChain and Press OK.

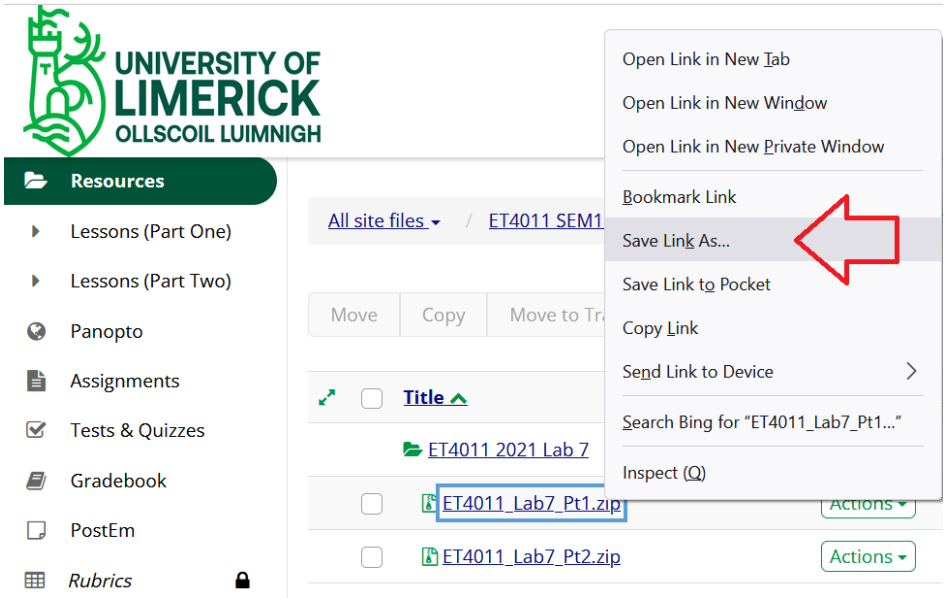


11. You may see a message to restart your DS-5 Community Edition. You will need to select this to complete the process.

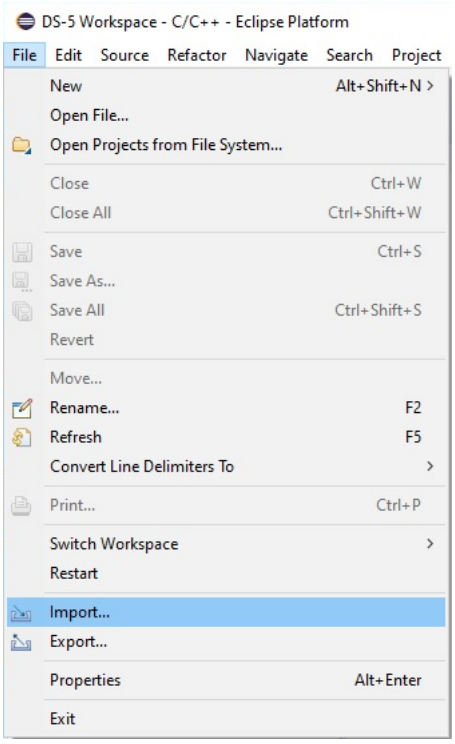


Import a Hello World type C project

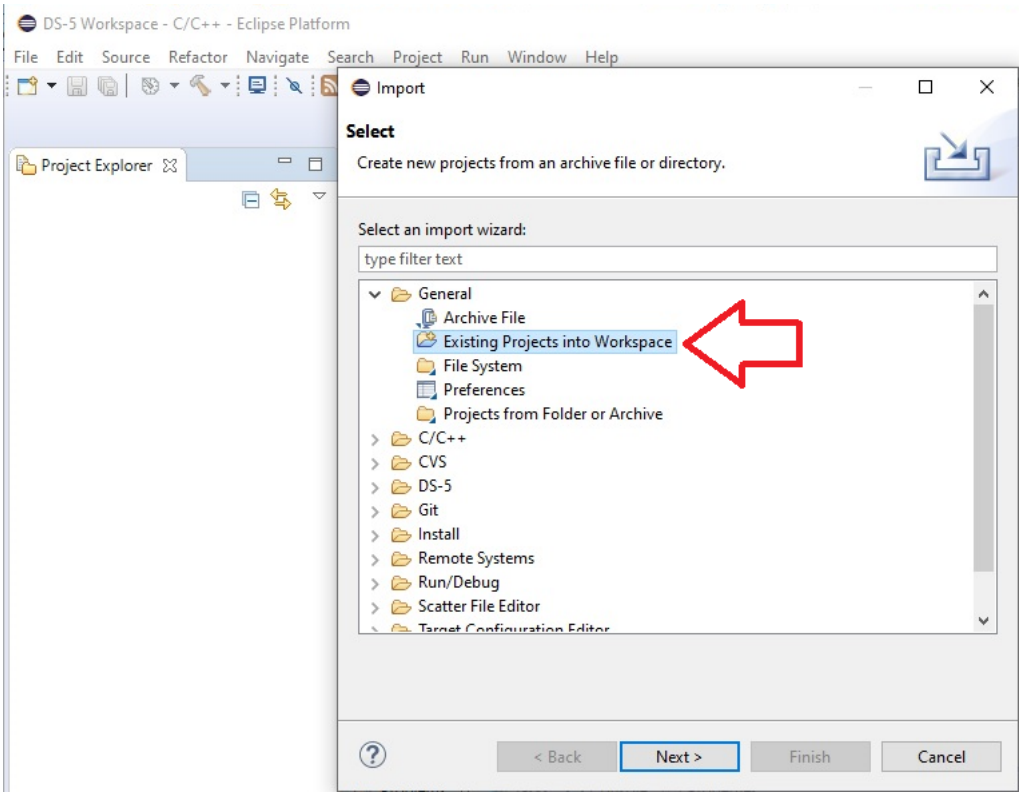
1. Go to Sulis for ET4011 and Select Resources-> Part 2 Labs -> ET4011 2021 Lab 7 and download the Zip file ET4011_Lab7_Pt1.zip to a folder on your computer.



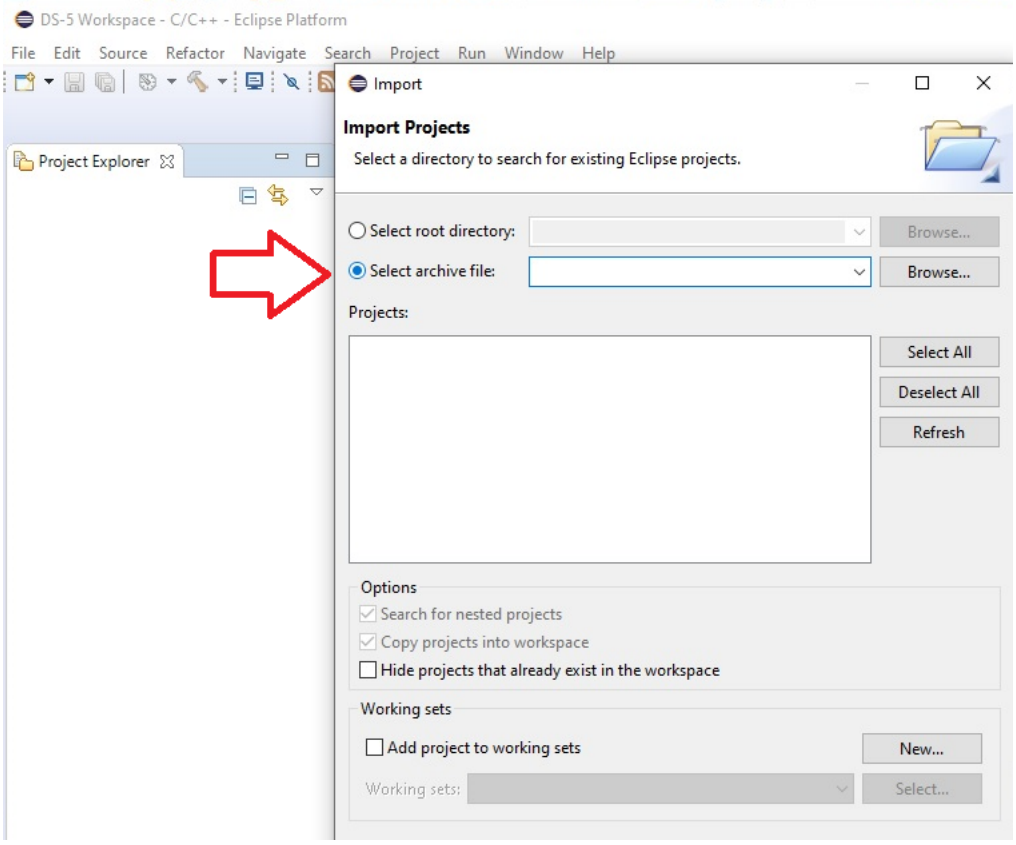
2. From the Ds-5 main menu, Select File -> Import to import the C project for this lab, from the zip file you downloaded in Step 1.



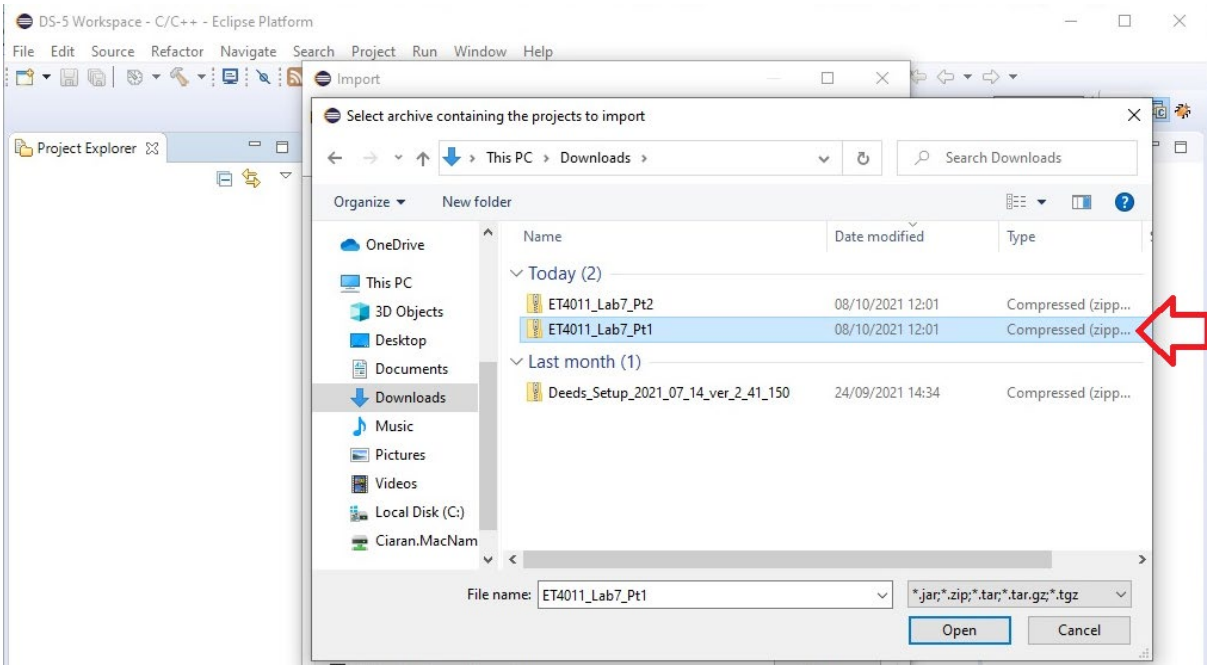
3. In the following menu, select 'Existing Project into Workspace'.



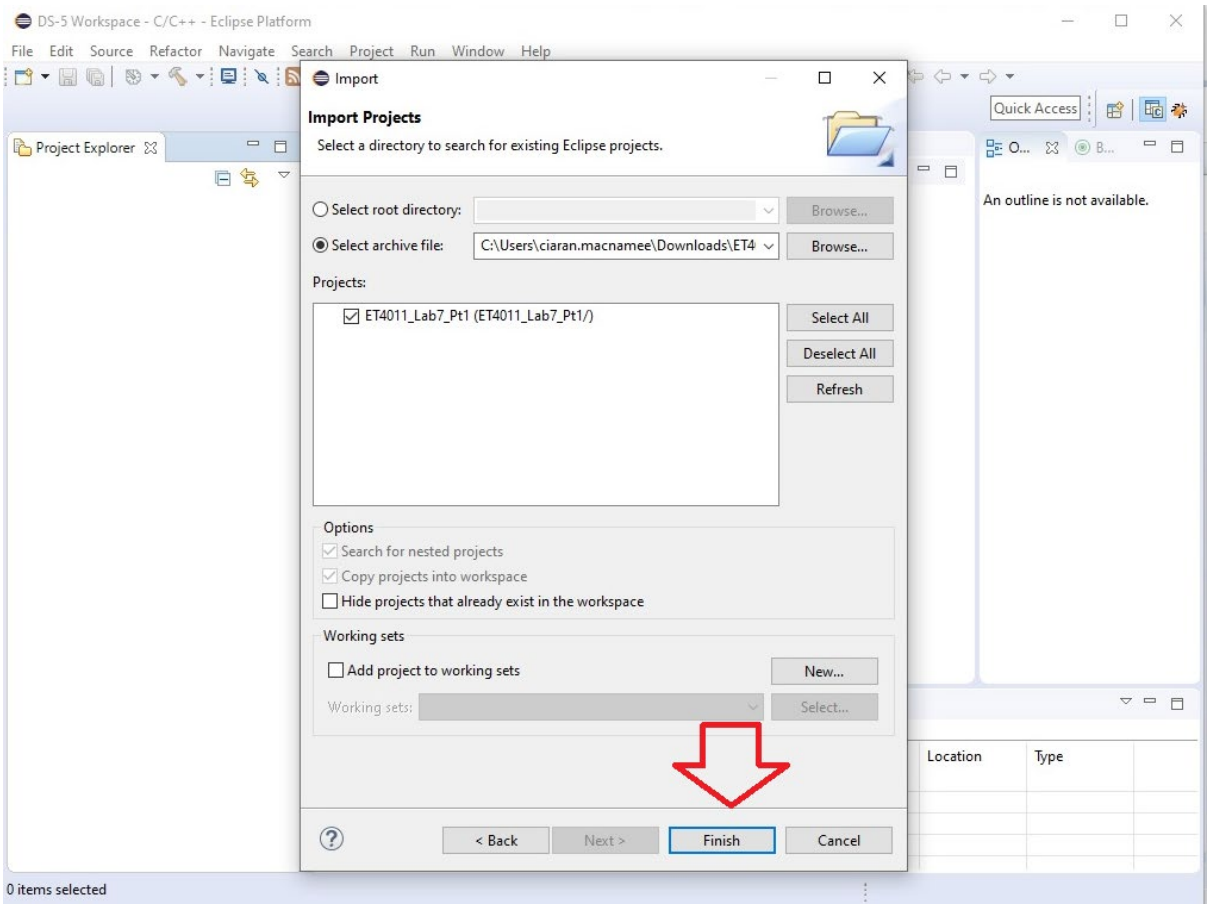
4. In the following Import Screen, select 'Select archive file'



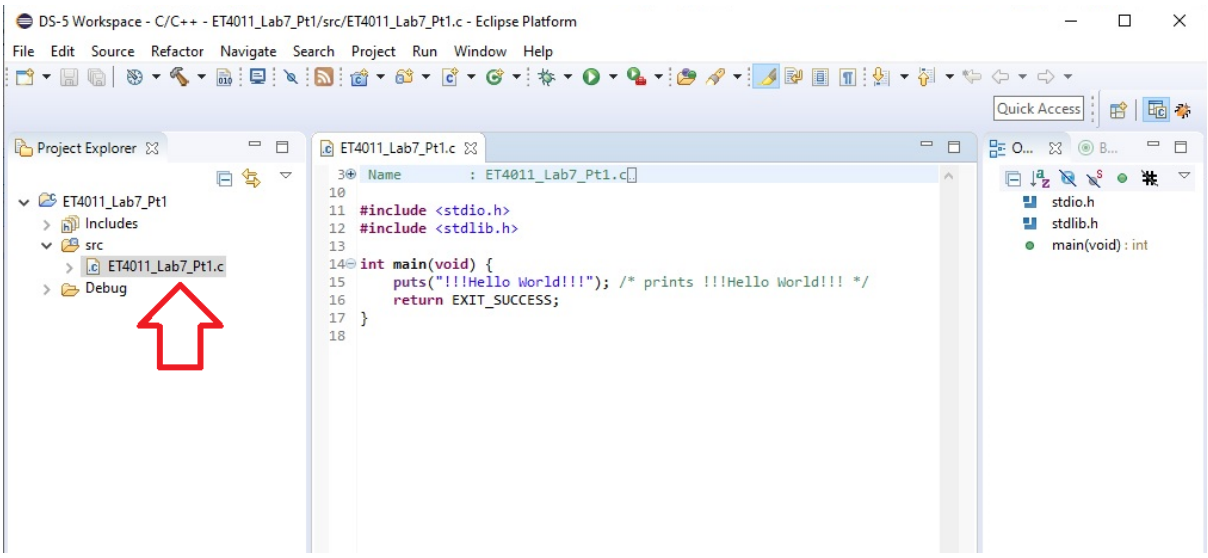
6. Browse to the folder where you downloaded the Zip file and select it.



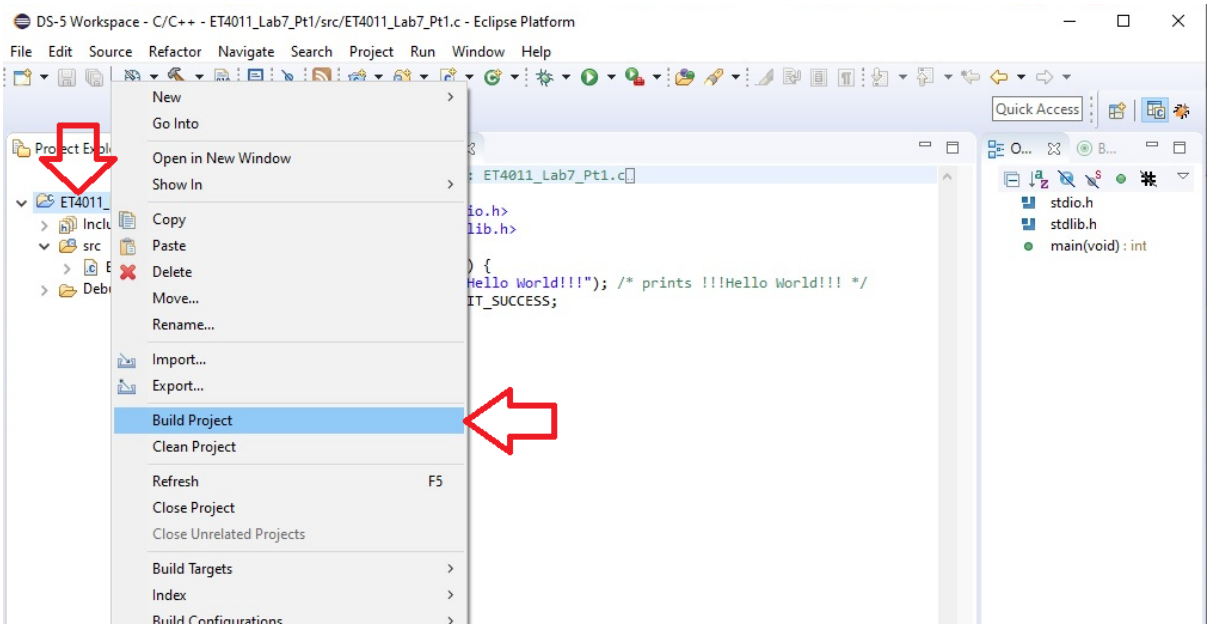
7. This should select the project as in the screen below. Press Finish to complete the Import process.



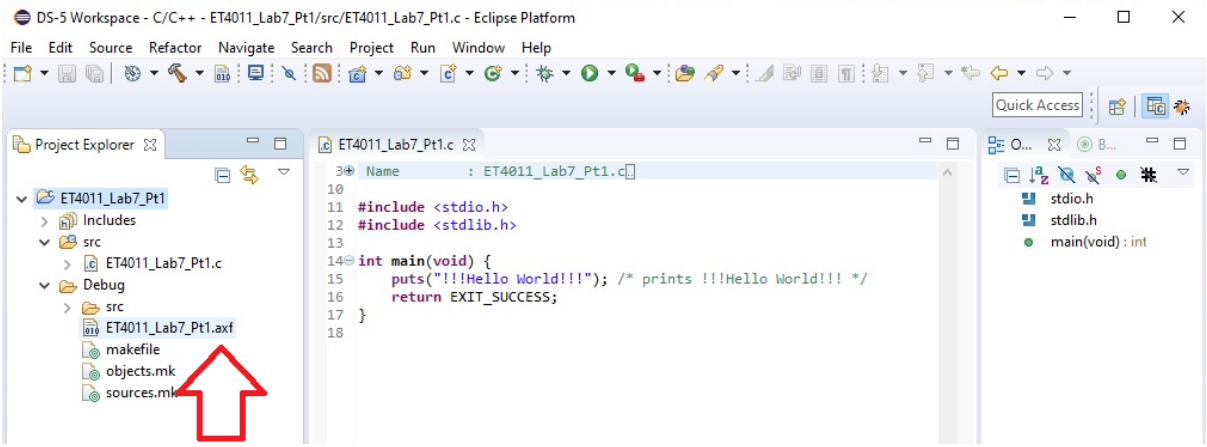
8. You should see a screen with the project imported. Next click on the .c file to display the source code as follows:



9. Next build the project. Do this by Right-Clicking on the Project Name and Selecting the 'Build Project' option in the following drop-down menu.

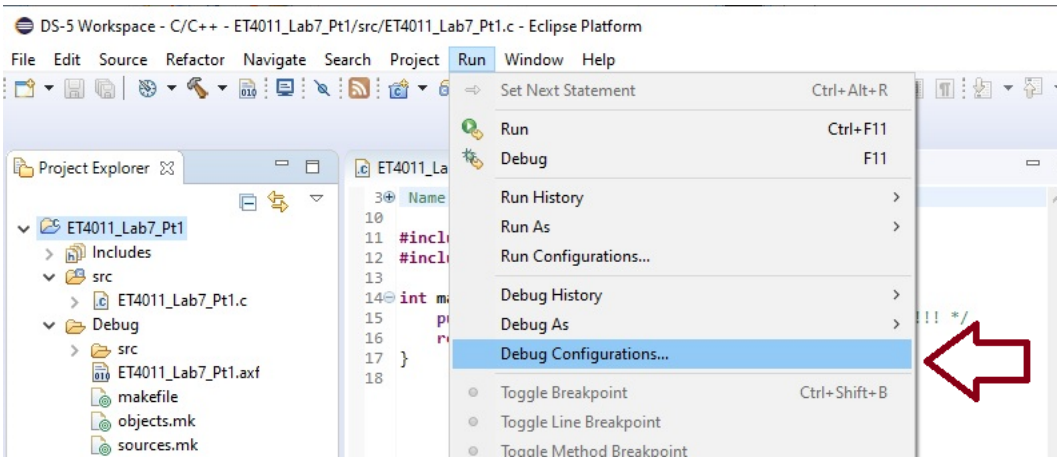


10. If the Build is successful, you should see a file called ET4011_Lab7_Pt1.axf in the 'Debug' folder in the project.



Create a DS-5 debug configuration

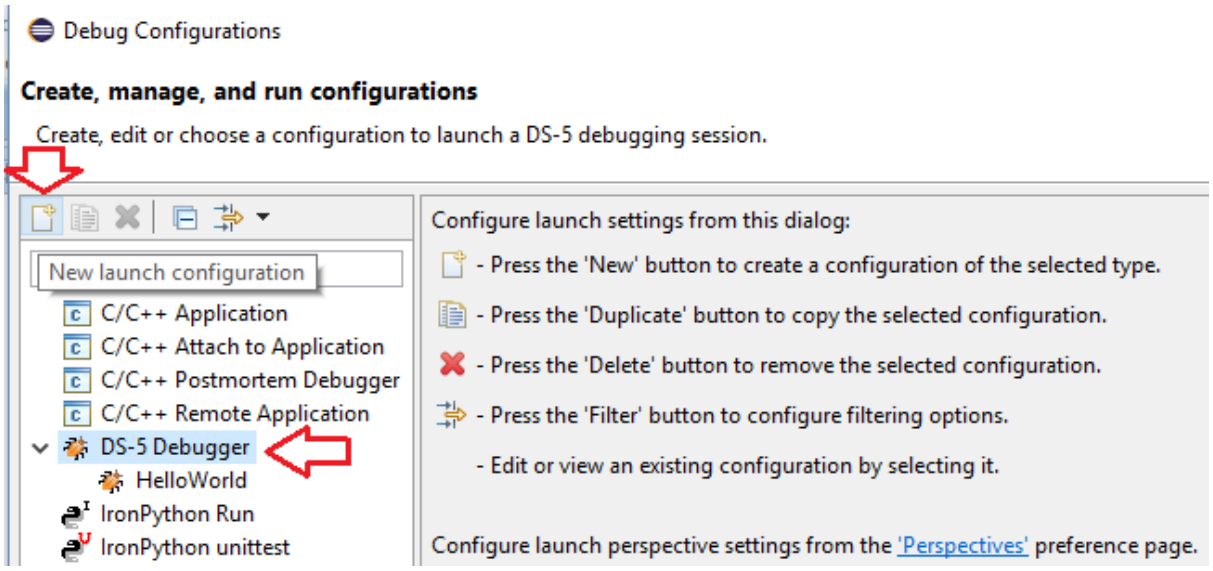
1. From the DS-5 main menu, select Run > Debug Configurations.



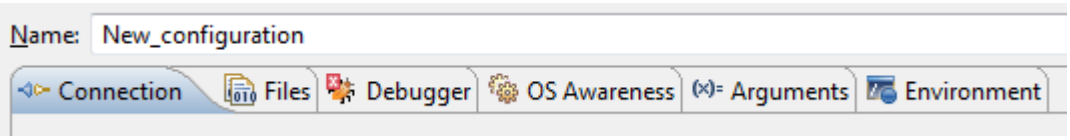
2. In the Debug Configurations dialog:

(a) Select DS-5 Debugger.

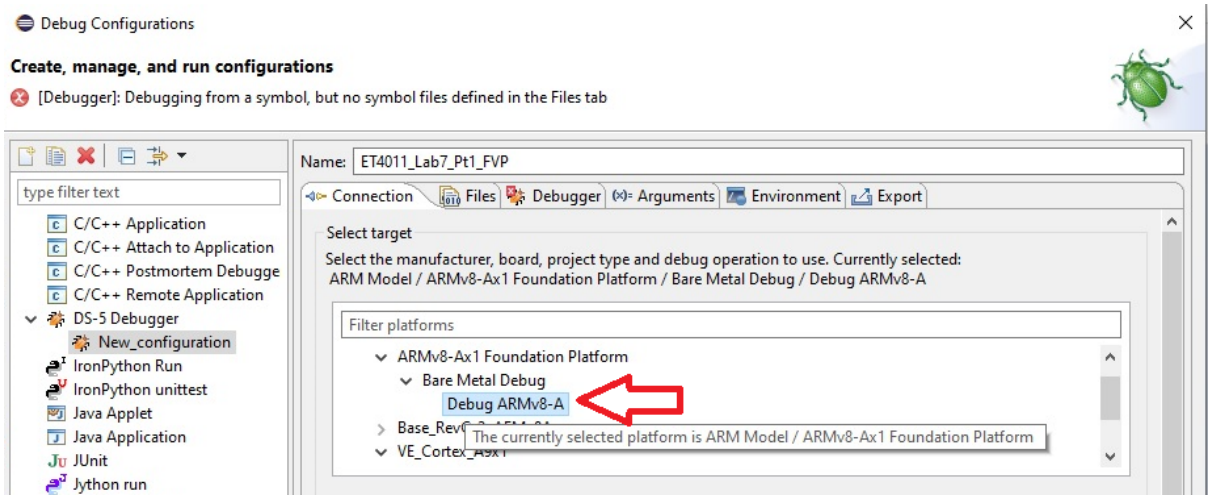
(b) Click the New launch configuration button.



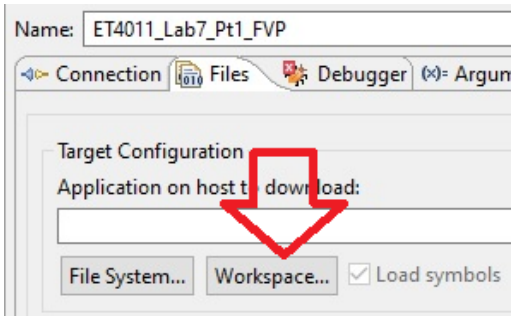
This creates a new DS-5 debug configuration and displays the various tabs required to specify settings for loading your application on the target.



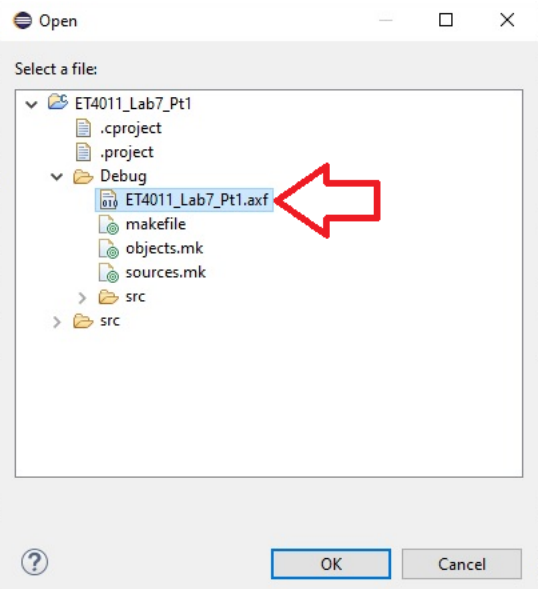
3. In the Debug Configurations dialog:
   (a) Give a name to the debug configuration. For example, ET4011_Lab7_Pt1_FVP.
   (b) In the Connection tab, under Select Target, browse and select ARM Model > ARMv8-Ax1 Foundation Platform > Bare Metal Debug > Debug ARMv8-A



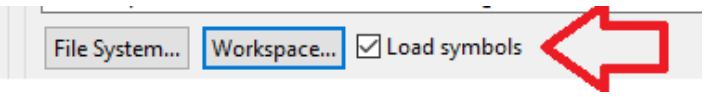
   (c) Select the Files tab, and under Target Configuration in the Application on host to download field, click Workspace.



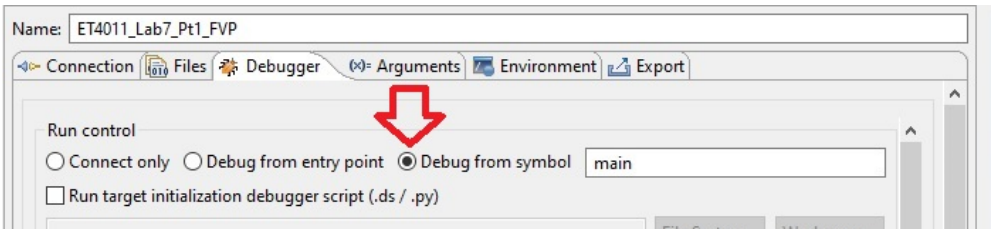
The workspace folder ET4011_Lab7_Pt1 -> Debug contains the ET4011_Lab7_Pt1.axf application file you created when you built the project. Select this file and press OK.



Note: Ensure that the Load symbols option is selected in the Files tab.



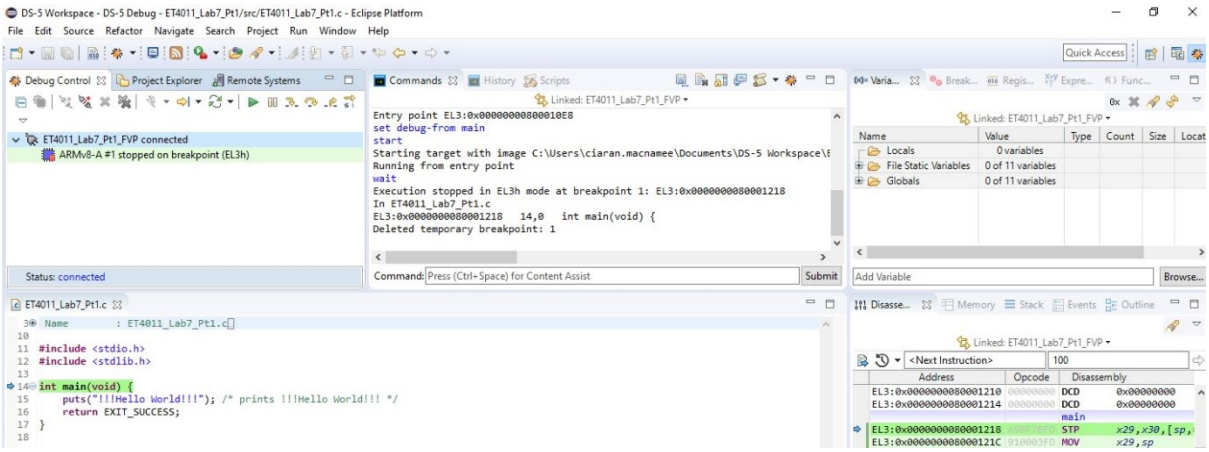
(d) Select the Debugger tab, and ensure the Debug from symbol option is selected and set to main.



(e) Click Debug to load the application on the target and to load the debug information into the debugger.

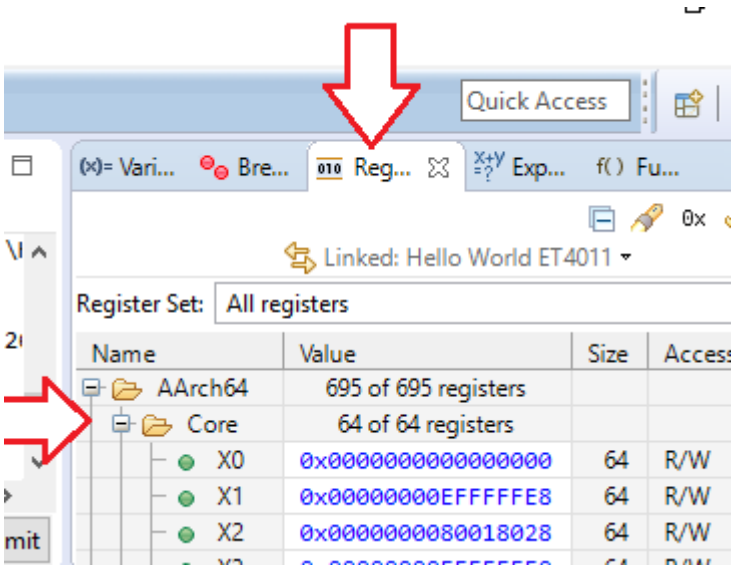
In the Confirm Perspective Switch dialog that appears, click Yes.

4. DS-5 connects to the model and displays the connection status in the Debug Control view.

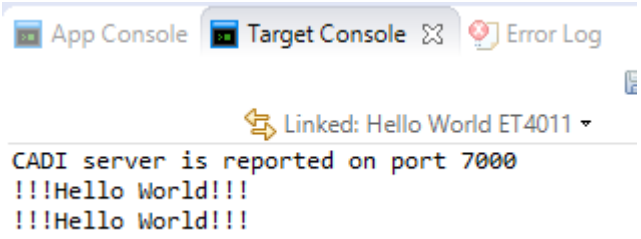


The application is loaded on the target, and has stopped at the main() function, ready to run.

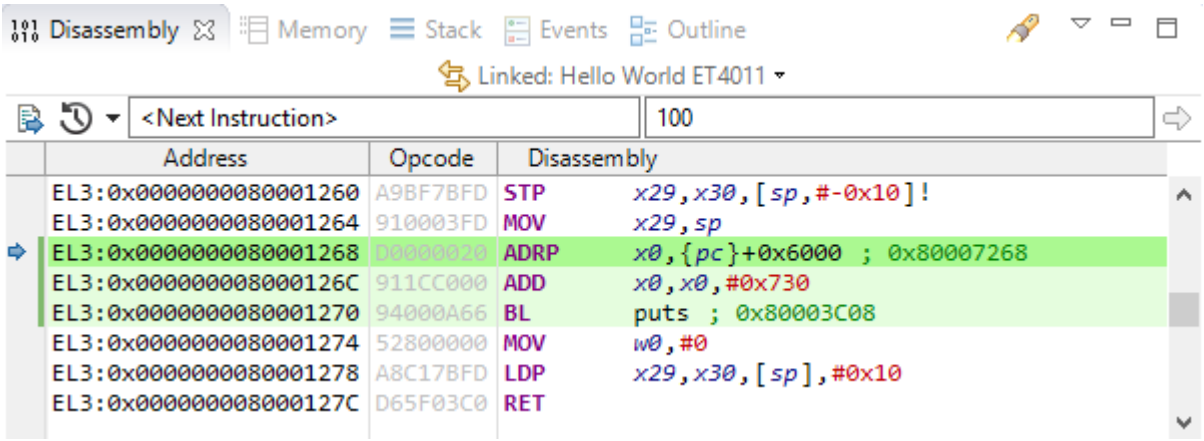
In the Variables View, Select Reg, Then AArch64 and Core:



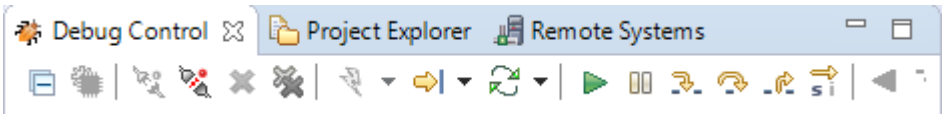
You can view the application output in the Target Console view.



Disassembly view shows the loaded program in memory as addresses and assembler instructions.



The Debug Control View shows the icons below:



Explore the debug control options:

- Click to continue processing code.

- Click to interrupt or pause processing code.

- Click to step through the code.

- Click to step over source line.

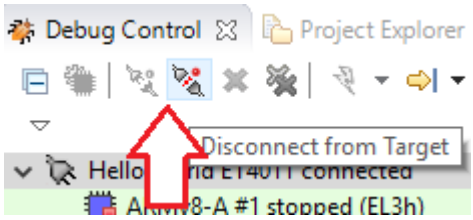
- Click to step out.

- This is a toggle. Select this if you want the above controls to step through instructions.

- Debug from main()

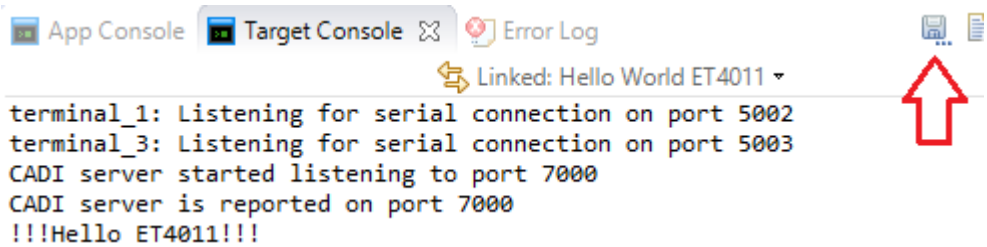
Lab 7 Part 1 outcomes

1. When you have used the debugger to run your program and step through it, Disconnect the debugger using the disconnect icon.



2. Go back to the C/C++ Perspective and modify the program to change the output string to "Hello ET4011". Save the program and build your program again. Then use the debugger to run it. (Use Run->Debug Configurations again, but you only need to press Debug this time.

3. The Console View should show your program output. Save the output as a text file, called ET4011_Lab7_Pt 1.txt using the save icon.



Upload two files to Sulis:

1. Your modified ET4011_Lab7_Pt1.c file
2. Your Terminal output file, ET4011_Lab7_Pt 1.txt
3. If you have time, please also upload the file 'ET4011 Lab 7 Answers'. See Exercises part 1 below.

Exercises:

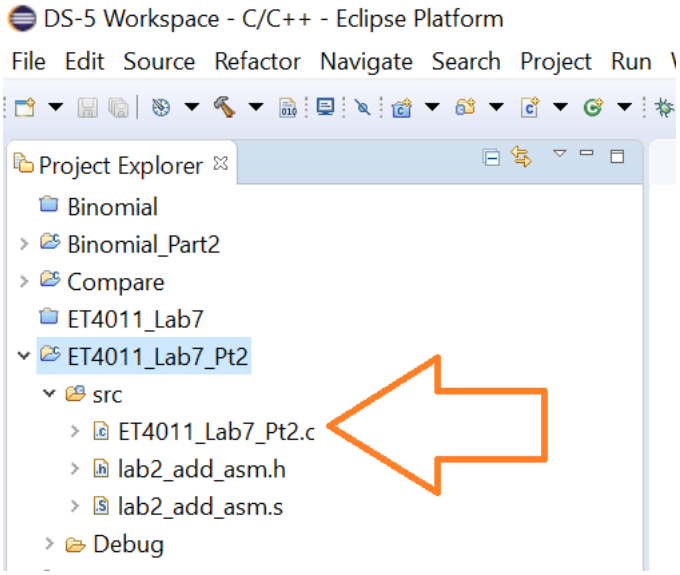
1. List the ARM v8 Assembly language instructions that implement the Hello World program, and fill in your results in 'ET4011 Lab 7 Answers'.
2. Change the output string from 'Hello World' to 'Computer Organisation Sem 1 2021'
3. Upload your files to Sulis.

Make sure you disconnect the debugger before quitting DS-5.

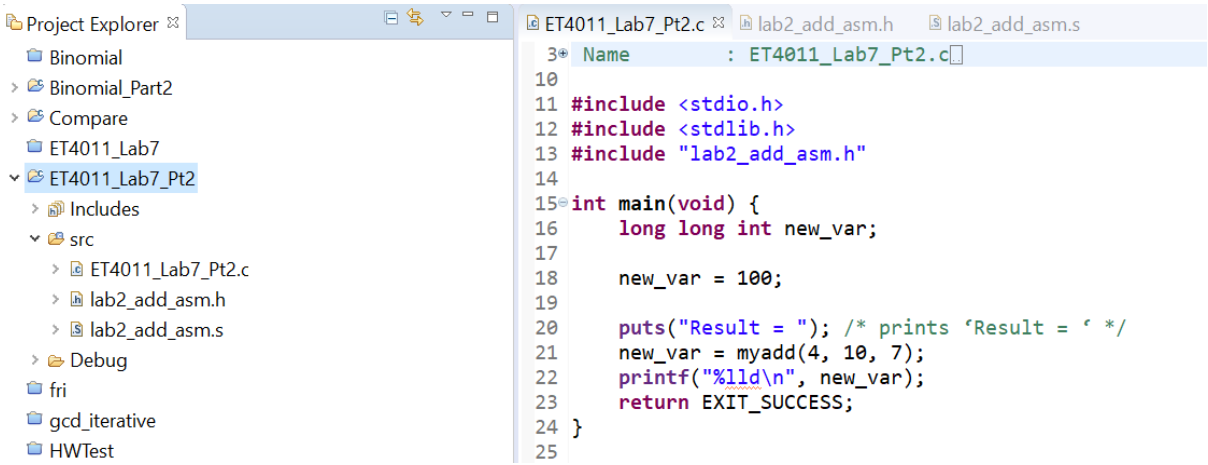
Lab 7 Part 2: ARMv8 Adding Integers using the CPU Registers

1. Repeat the steps in Part 1 but this time select ET4011_Lab7_Pt2.zip as the Project to Import.

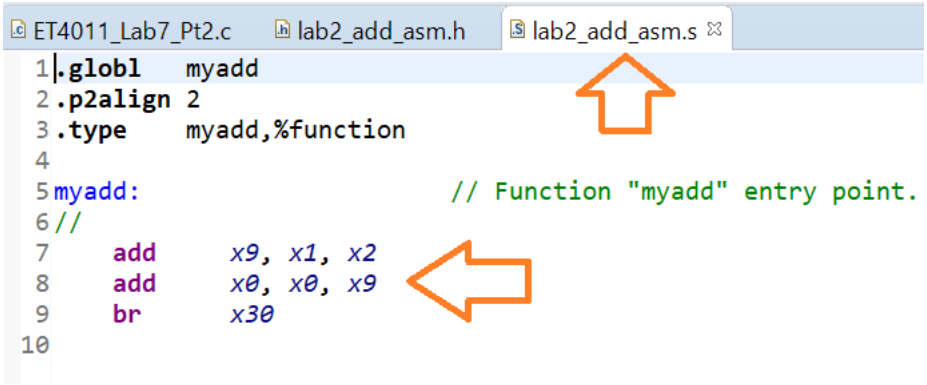
2. When you have successfully imported the project you should see the following project in the Project Explorer Window.



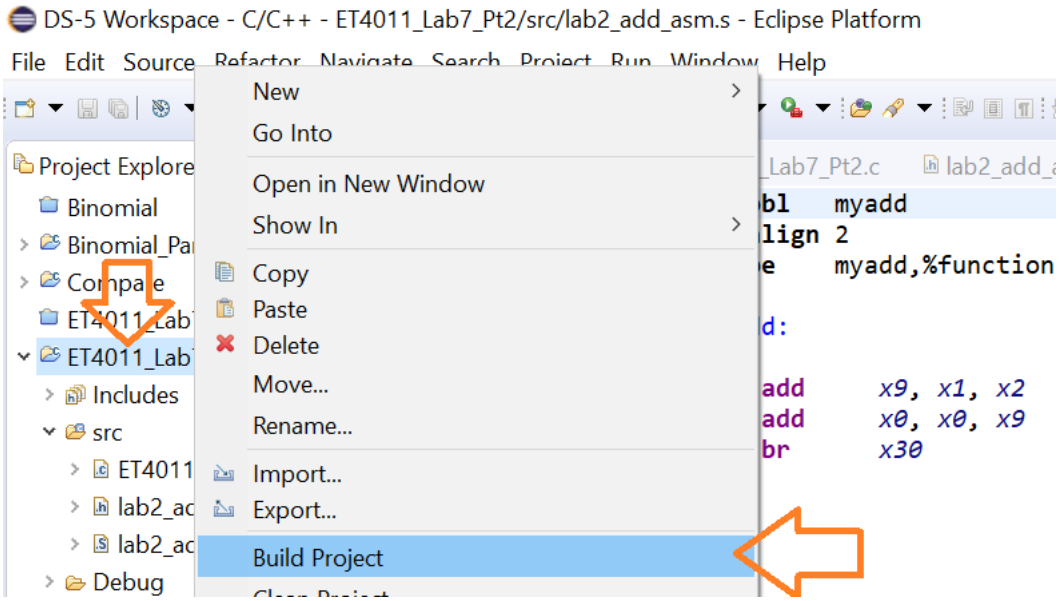
3. If the Source (src) files are not displayed, click on each one, so that it appears the in Editor. Here, ET4011_Lab7_Pt2.c is shown, with the other files visible if you click the tab identifying them.



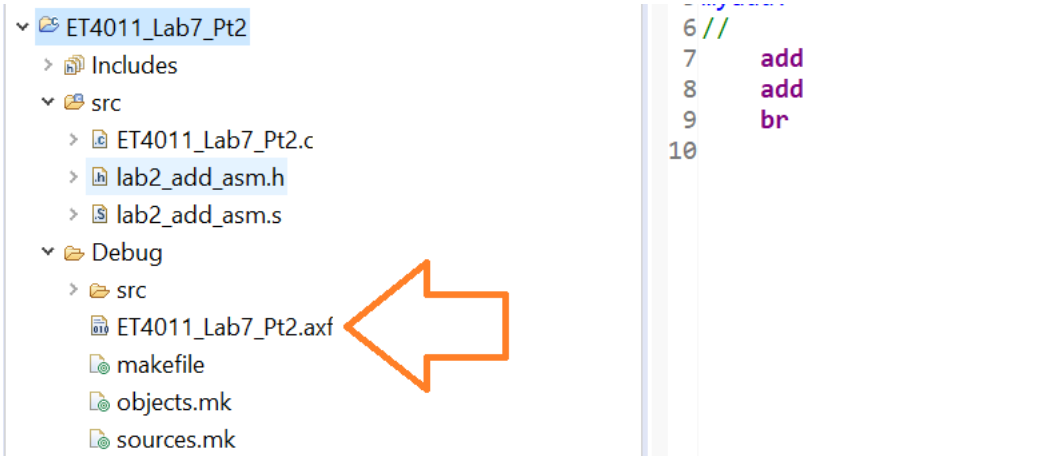
4. This program simply adds three integers, 4, 10 and 7. But the addition operations are done in an ARMv8 assembly language program called myadd, and this is available in the file 'lab2_add_asm.s' as seen below.



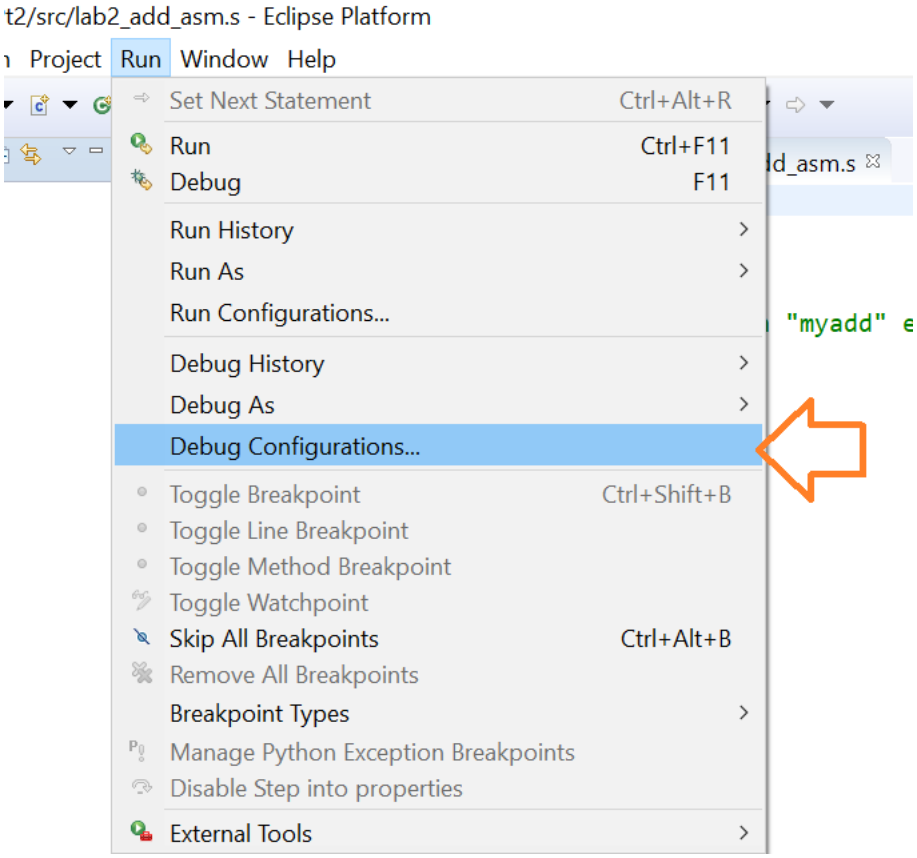
5. Just as you did in Lab 7 Part 1 Build this program using the 'Build Project' command.



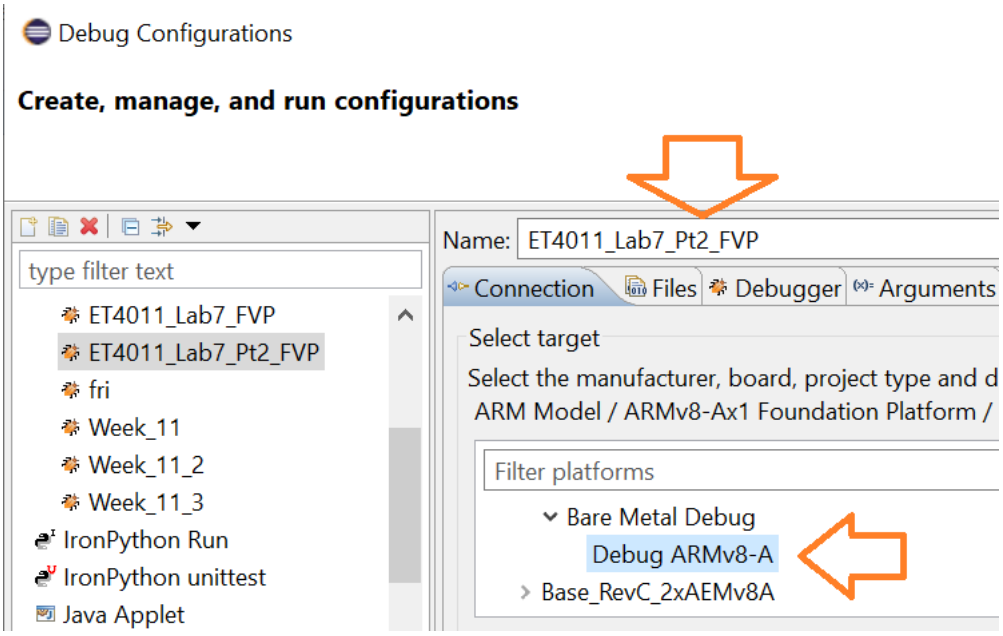
6. If this is successful, you will see a file called 'ET4011_Lab7_Pt2.axf' in the Debug folder.



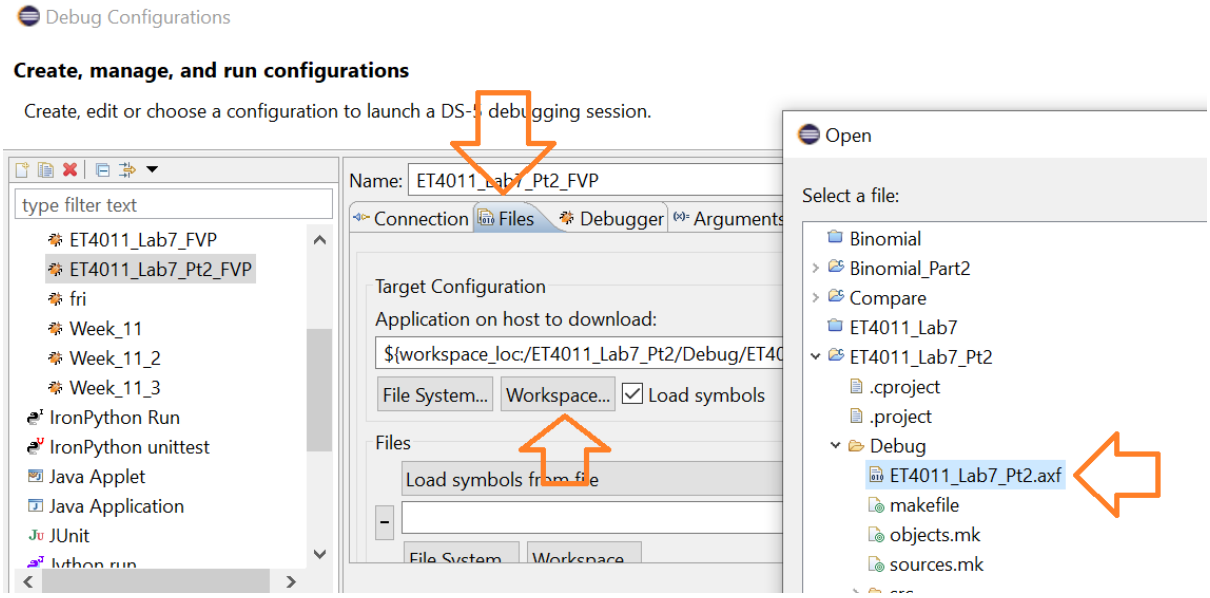
7. Again, using the Run->Debug Configurations Drop Down Menu, create a DS-5 Debug Configurations to run the program in the DS-5 Simulation/Debug tool.



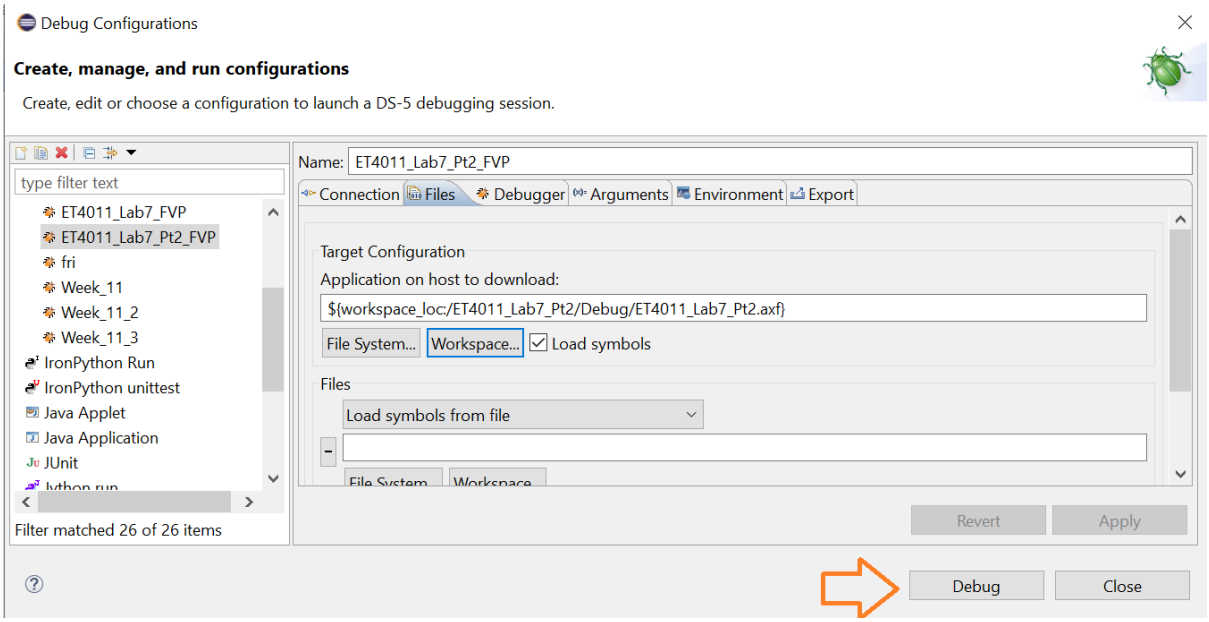
8. Create a new debug configuration. Call it 'ET4011_Lab7_Pt2_FVP' and select ARMv8 bare Metal Debug as the target.



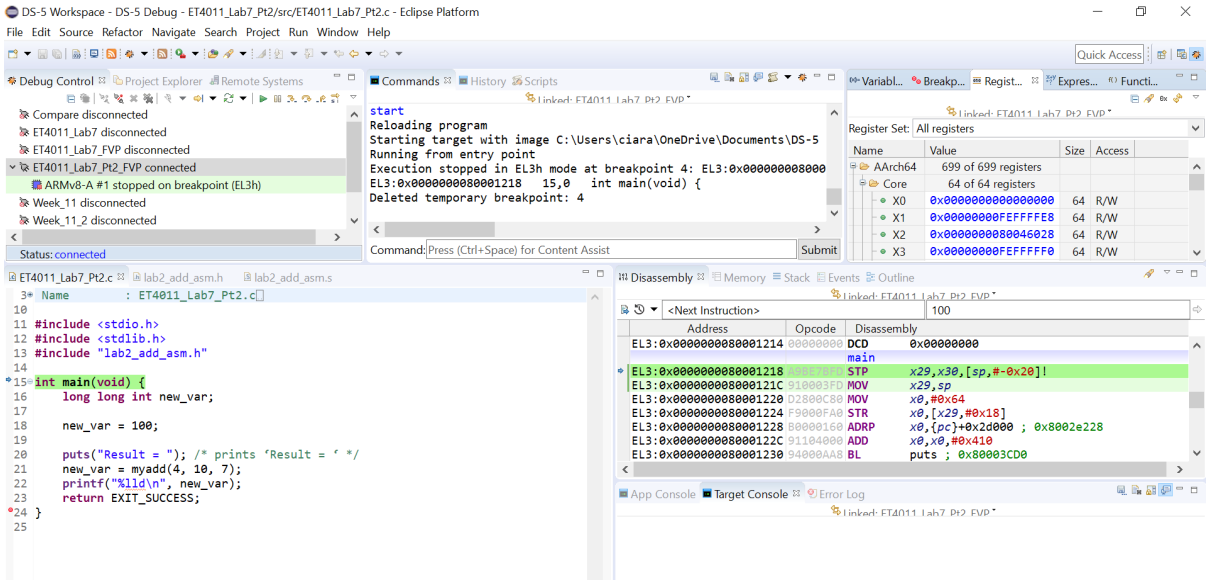
9. Select the Files Tab and Browse to the 'ET4011_Lab7_Pt2' Workspace. Select 'ET401_Lab7_Pt2.axf' as the file to debug. Make sure that the 'Load Symbols' box is ticked. Then Press 'OK'.

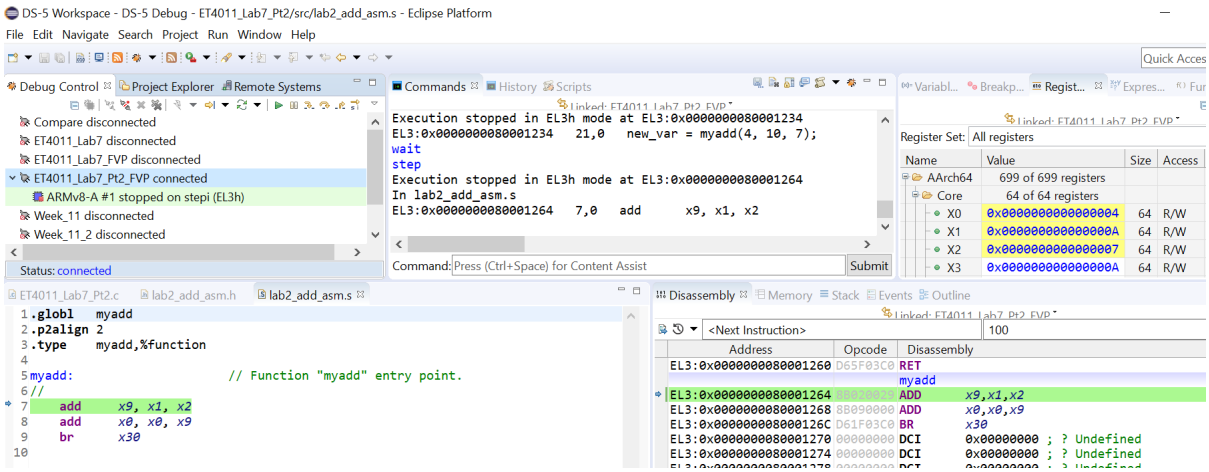


10. Press 'Debug' in the next screen, and allow the debugger to open, when prompted.



11. This sequence of steps opens the DS-5 debugger.





14. Answer the following questions (note that the register contents may be in hexadecimal):
    a. What data is in register X0?
    b. What data is in register X1?
    c. What data is in register X2?
    d. How do the register contents relate to the passed-in parameters?

a	X0:
b	X1:
c	X2:
d	Answer:

15. Step the next instruction (add x9, x1, x2). Fill in the table again.
    a. What data is in register X0?
    b. What data is in register X1?
    c. What data is in register X2?
    d. What data is in register X9?

a	X0:
b	X1:
c	X2:
d	X9:

16. Step the next instruction (add x0, x0, x9). Fill in the table again.
    a. What data is in register X0?
    b. What data is in register X1?
    c. What data is in register X2?
    d. What data is in register X9?
    e. What data is in register x30?

a	X0:
b	X1:
c	X2:
d	X9:
e	X30:

17. Step the next instruction (br x30). Explain what has just occurred. Also, what data is register X0?

Explanation	
X0:	

18. Fill in the answers in the Word Version of this Lab Sheet and upload your answers to Sulis.

Lab 8: More complex ARM Assembly Language program in DS-5 with Linaro

In this lab session, we are going to modify a C language program that includes an ARMv8 Assembly Language module. You will need to write the assembly language code for this module. It will be a short program to test an input in an **if …else** structure and add or subtract some numbers using some of the ARM's internal registers. The assembly language program will be called by a C program as in Lab 7 Part 2 and it will return a result to be printed using the calling C program's *printf* function.

We will compile the program with its two modules and use the DS-5 Debugger to see how the assembly program is executed and how it uses the CPU registers for arithmetic operations.

Exercise 1

Repeat the steps 1 to 8 in the lab sheet for Lab Week 7 Part 2, but this time Import the file ET4011_Lab8.zip.

In this case the C file calls the assembly code myadd2 with four parameters, x, y , z and control. It's given below:

```
#include <stdio.h>
#include <stdlib.h>
#include "lab_2_add_asm_v3.h"

int main(void) {
	long long int new_var, control_var;

	new_var = 100;
	control_var = 1;

	puts("Result = "); /* prints 'Result = ' */
	new_var = myadd2(5, 10, 6, control_var);
	printf("%lld\n", new_var);
	return EXIT_SUCCESS;
}
```

The header file lab_2_add_asm.h is:

```
#ifndef LAB_2_ADD_ASM_V3_H_
#define LAB_2_ADD_ASM_V3_H_

extern long long int myadd2(long long int x, long long int y, long long int z, long long int ctrl);

#endif /* LAB_2_ADD_ASM_V3_H_ */
```

Next write the assembly file. It should implement the following:

if (ctrl == 0)

  f = x + (y + z);

else

  f = x + (y-z);

return f;

Note that this is pseudocode: you must implement this code in ARMv8 Assembly language.

Use X9 as a temporary value but don't try to save it on the stack. The shell of the assembly file is:

```
.globl   myadd2
.p2align 2
.type    myadd2,%function

myadd2:                     // Function "myadd2" entry point.
//
//     Put your code next to implement the desired operation


       br       x30   // Return by branching to the address in the link register.
```

Note also that the returned data should be returned in register X0.

Outcomes:

Please upload your .c, .s and .h files to Sulis.

Exercise 2

Modify your program to save X9 on the stack before using it and restore its value before returning.

Use the LDR and STR instructions to load and store registers.

Use the LDR and STR instructions to load and store registers.

Note that you will need to allow 16 addresses on the ARMv8 stack for X9. This restriction arises from the way that ARMv8 does its memory alignment.

Outcomes:

Please upload your .c, .s and .h files to Sulis.