

# ELEC-240 Lab1

## Introduction to the ST-Microelectronics STM32F429 Nucleo-144 Development Board

### 1 Introduction

The purpose of this lab is to introduce you to the ST-Microelectronics Cortex-M4 processor using the ARM Keil MDK toolkit featuring the  $\mu$ Vision IDE. The hardware you will be using in this lab session is the STM32F429 Nucleo-144 which incorporates the STM32F429 micro-controller. It also has an on-board ST-Link V2 debugger with Serial Wire Viewer (SWV) which we will be using to program and debug the board.

Please see Table 2 for access to the relevant support documentation for the Nucleo-144 development boards.

- ❖ By the end of this lab exercise you should be able to
- ✓ Install and run the Keil  $\mu$ Vision IDE on a Windows PC and install the necessary packs, legacy support drivers and hardware (USB) drivers.
- ✓ Create a  $\mu$ Vision project, edit and compile code and download it to the micro-controller.
- ✓ Open and manage  $\mu$ Vision project files.
- ✓ Use the debug function to monitor variables, pause and single step through code execution.

If you get stuck with any part of the exercise, **ASK for help!** otherwise you'll fall behind.

### 2 Keil Software Installation

Keil projects contain multiple files.

- ❖ **First rule of Keil project file management**
- **Keep all project files inside their folder, do NOT separate them!**

When copying or moving Keil projects make absolutely sure you copy/move the entire folder so that all the associated files are kept together with the project file. Keep this in mind for later tasks.

- If you already have Keil  $\mu$ Vision installed on your PC then skip to Section 3.

1. You can acquire the installer for Keil v5.21 from the lab network storage at

**`\\10.208.130.20\StudentShare\elec230\Installer`**

Copy the **MDK521a.exe** file to your computer and run it, do not try to execute it directly from the network drive.

2. Install the USB drivers

Assuming you've installed to the default directory go to

**`C:\Keil_v5\ARM\STLink\USBDriver`**

and run the ***dpinst amd64.exe*** if you're using a 64bit OS or ***dpinst x86.exe*** if you're running a 32bit OS.

### 3 Opening a Keil Project

The ***.uvprojx*** extension is the project file which will launch Keil and open the project.

#### 3.1 Installing Legacy Support

If you're working with a new install of Keil you may see this message on start (if not then skip to Section 4).

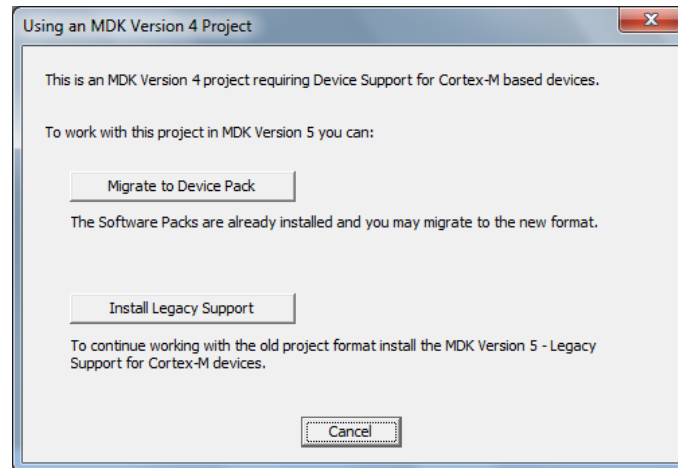


Figure 1: Legacy Support Drivers Missing Message

If this is the case then click the ***Install Legacy Support*** (bottom button). This will take you to here where you will see the following screen (Figure 2).

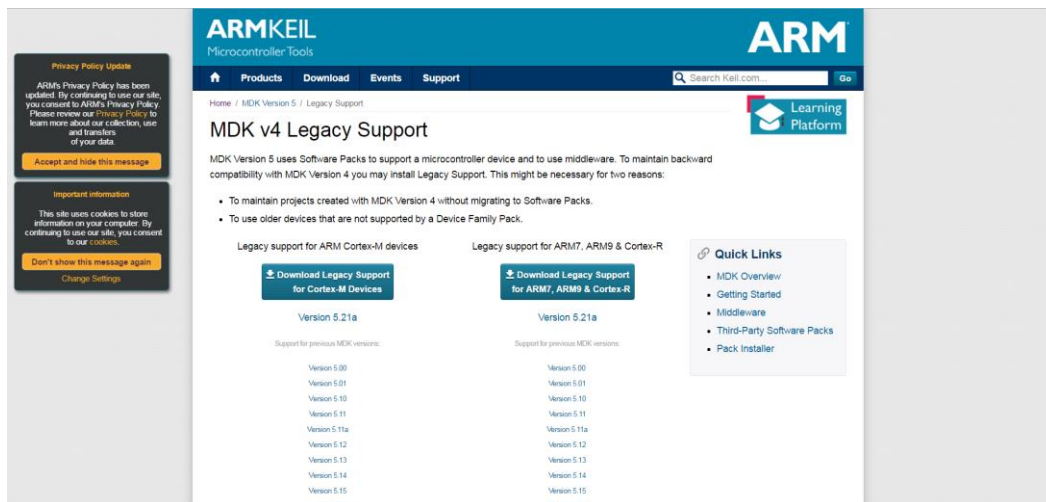


Figure 2: ST Legacy Support Page

Click ***Download Legacy support for Cortex-M devices*** (left-hand button). This will download the ***MDKCM5xx.exe*** file. Run this file and follow the instructions to install the legacy drivers.

**Note:** On the screen which asks you to backup old configurations, un-tick the box as this is not necessary.

## 3.2 Installing Packs



In Keil, click on the icon on the top bar, this window will open

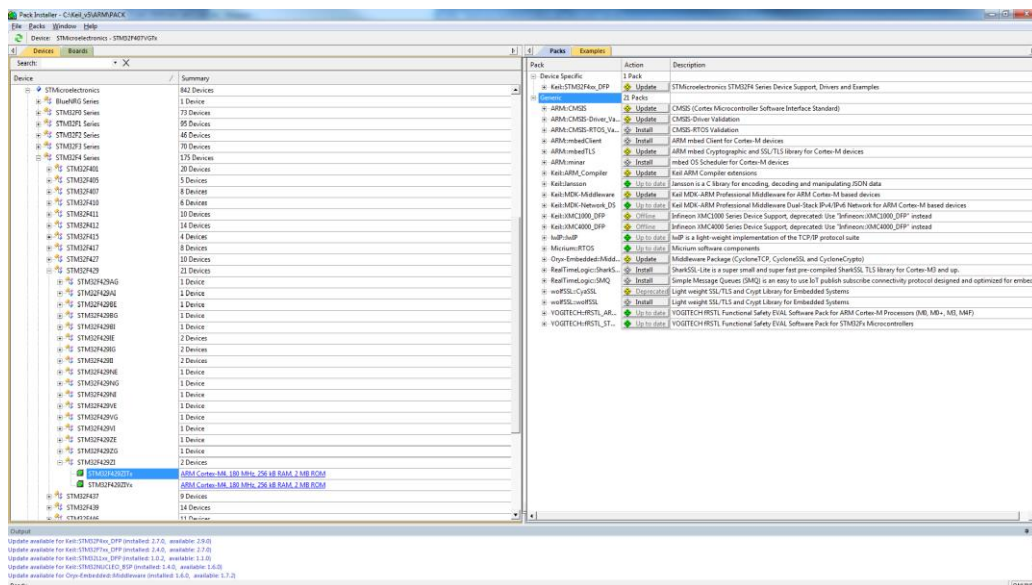



Figure 3: Keil Pack Installer Window

If the list does not appear as shown in the left-hand pane of Figure 3 then click the refresh  icon in the top-left corner. Progress of any operation is shown in the bottom-right corner as shown in Figure 4.

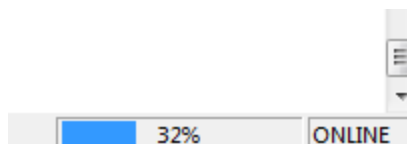


Figure 4: Pack Installer Progress Indicator

Once refreshed the left-hand pane should show a list of available devices, navigate to the correct device as shown in Figure 3 left-hand pane.

In the right-hand pane, for any devices that aren't showing as up to date, click Update/Install to bring them up to date (ignore Offline or Depreciated attributes).

Again the progress of the install is indicated in the bottom-right corner as shown in Figure 4.

Once complete they should all display Up To Date as shown in Figure 5.

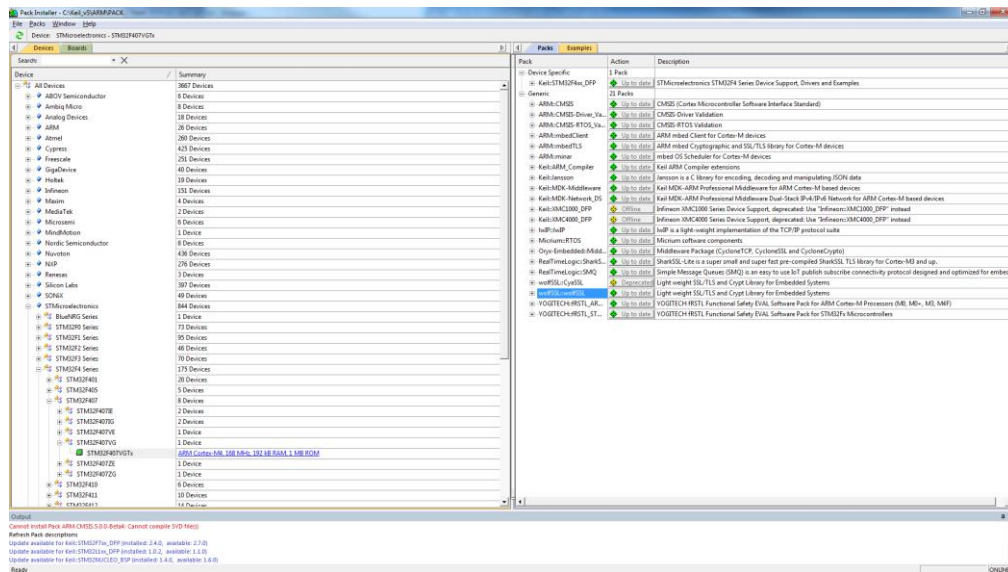


Figure 5: Pack Installer Window showing all attributes Up To Date

Once complete close the pack installer window. If Keil has been running while the packs were being installed it will now display a pack reload request, shown in Figure 6, click **Yes** and you should then be presented with the screen shown in Figure 7.

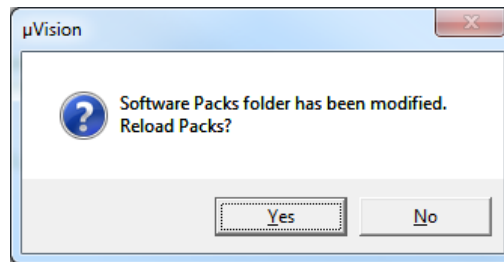


Figure 6: Reload Packs Message

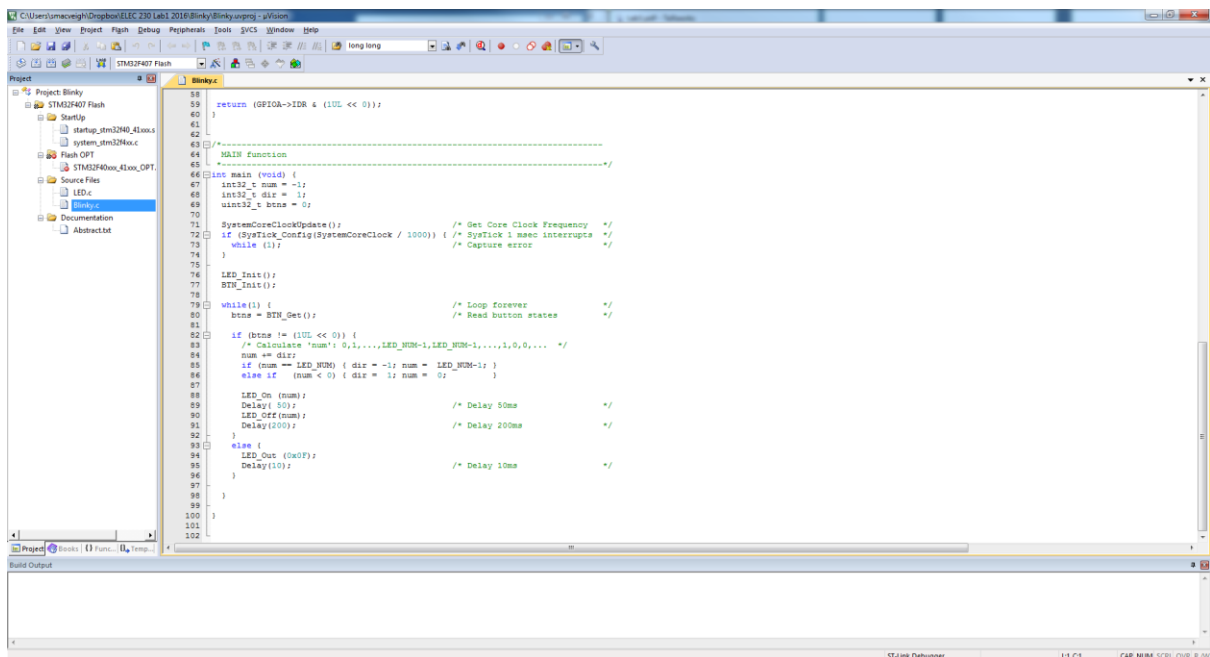


Figure 7: Keil Main Screen

The pane down the left-hand side shows all the files associated with the project. The main section in the centre shows the code.


#### 4 Task 1: LED Flash

Make sure the board is plugged in to the computer and that the USB drivers are all installed (refer to Section 2).

Download the example project from the DLE (under “Lab 1” section) to a local directory on your PC such as the Desktop and extract the files from the ZIP folder.

Inside the extracted folder ‘STM32F429 Nucleo-144 test code’ you will find a variety of files.

Open the file labelled **test.uvprojx**. See Section 3 for details if necessary.

- Click the  button to open the settings window (shown in Figure 8a).

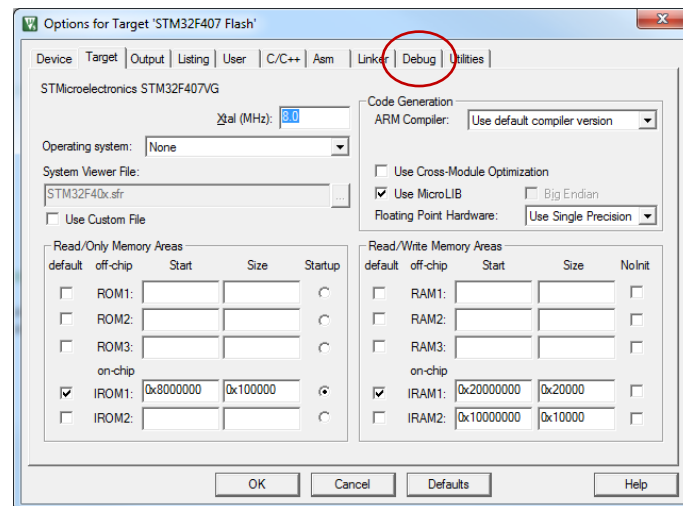


Figure 8a Settings Window

- Under the **Debug** tab select the ST-Link debugger from the list (Figure 8b) and then click **Settings**.

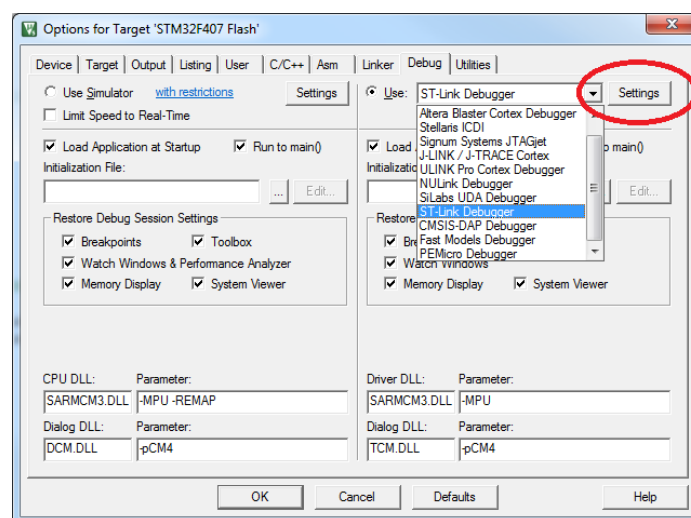


Figure 8b Debugger Selection Window

- In the **Debug Adaptor** section set the Port to SW communications protocol (Figure 8c circled in Blue).

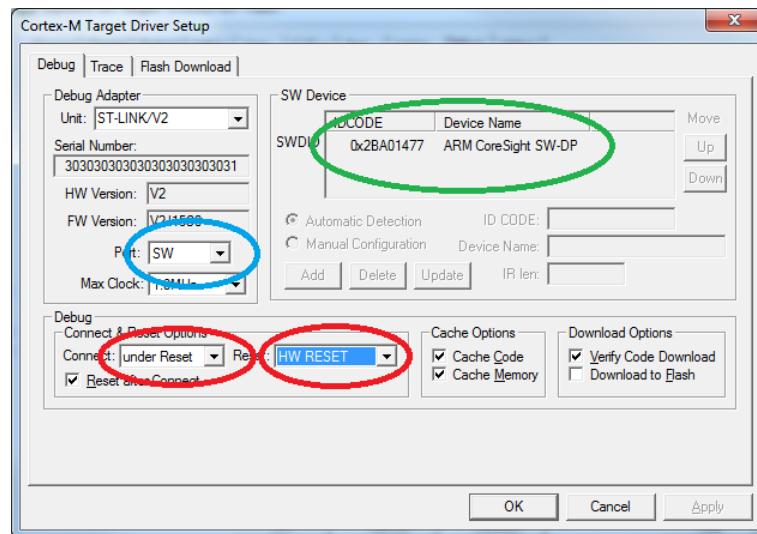


Figure 8c Debugger Settings Window

- Set the **Connect & Reset** Options so they match that shown in Figure 8c circled in Red.
- If this is done correctly the board ID should be displayed in the **SW Device** section as shown in Figure 8c circled in Green.
- Under the **Flash Download** Tab ensure that the **Reset and Run** box is ticked (shown in Fig 8d circled in Red).

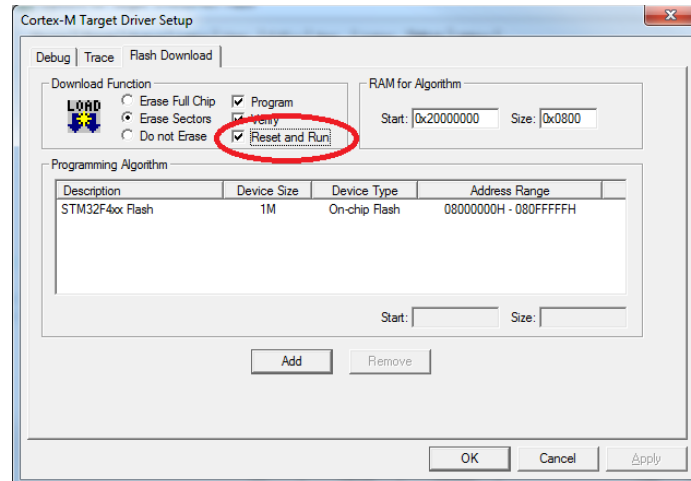



Figure 8d Flash Download Settings Window

#### 4.1 Compiling the Code

Click the Build  button or press F7 on the keyboard to compile the code.


The compile output is shown in the bottom pane.

If all was successful it should end showing 0 Errors and 0 Warnings.

Warnings are displayed when undesirable commands are detected such as unused variables, mathematical errors etc.

Errors are displayed when illegal commands are detected. Most common causes of illegal commands are typographical errors when the code is being written. Misspelling a variable or function name will cause the compiler to complain that an undefined function is being called or an undefined variable is being used. Also forgetting to close a bracket at the end of a statement or function will produce an Error. In any event if Warnings or Errors have been detected you can scroll back up the output window to see what the cause of each was and double-clicking on it will take you to the offending line in the code.

#### 4.2 Downloading the Code

Click the Load  button or press F8 on the keyboard to download the code to the board. The code can only be downloaded to the board if the compile was successful (**no Errors**).

#### 4.3 Editing the Code

Define a variable labelled **value** as shown in Figure 9a.

```
1  #include <stm32f4xx.h>    //INCLUDE THE HEADER FILE F
2                               //this file contains th
3  unsigned char value=0;
4  //100ms BLOCKING_DELAY
5  void delay_100ms_blocking(void)
6  {
7      unsigned int i;        //increment a variable to was
8      for(i=0; i<3000000; i++); //incrementing a variab
9  }
10
11 int main(void)
12 {
13     //initialise GPIO pins
```

Figure 9a Defining a Variable

Insert a simple piece of code within the main program to use this variable as shown in Figure 9b.

```
37
38     GPIOB->BSRR=(    //set GPIOB pins low (write to higher 16 bits of GPI
39         (1u<<(0+16))    //PB0 (cleared by writing to bit 16)
40         |(1u<<(7+16))   //PB7 (cleared by writing to bit 23)
41         |(1u<<(14+16))  //PB14 (cleared by writing to bit 30)
42     );
43
44     delay_100ms_blocking(); //call the delay subroutine (defined above
45
46     value++;
47     if(value > 0x10)
48     {
49         value=0;
50     }
51
52
53 }
```

Figure 9b Using a Variable


In Figure 9a the variable is defined as a **char** which is short for 'Character' and is 8bits wide. Other available word lengths are 'short' which is 16bits wide and 'long' which is 32bits wide. The ARM micro-controller series have a 32bit wide bus architecture meaning that the default 'int' length is 32bit same as a 'long'. Table 1 shows the full range of available word lengths and the values they can hold.


Word type	length (bits)	Min value	Max value
unsigned char	8	0	255
signed char	8	-128	127
unsigned short	16	0	65,535
signed short	16	-32,768	32,767
unsigned long/int	32	0	4,294,967,295
signed long/int	32	-2,147,483,648	2,147,483,647
unsigned long long	64	0	18,446,744,073,709,551,615
signed long long	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807


Table 1: Integer Word Lengths and the Values they can hold

#### 4.4 Debugging

Compile and download the code to the board. Click to enter/exit debug mode. The debug screen is shown in Figure 10.

Click  button to run the code.

Click  button to pause the code.

Click  button to reset the code.

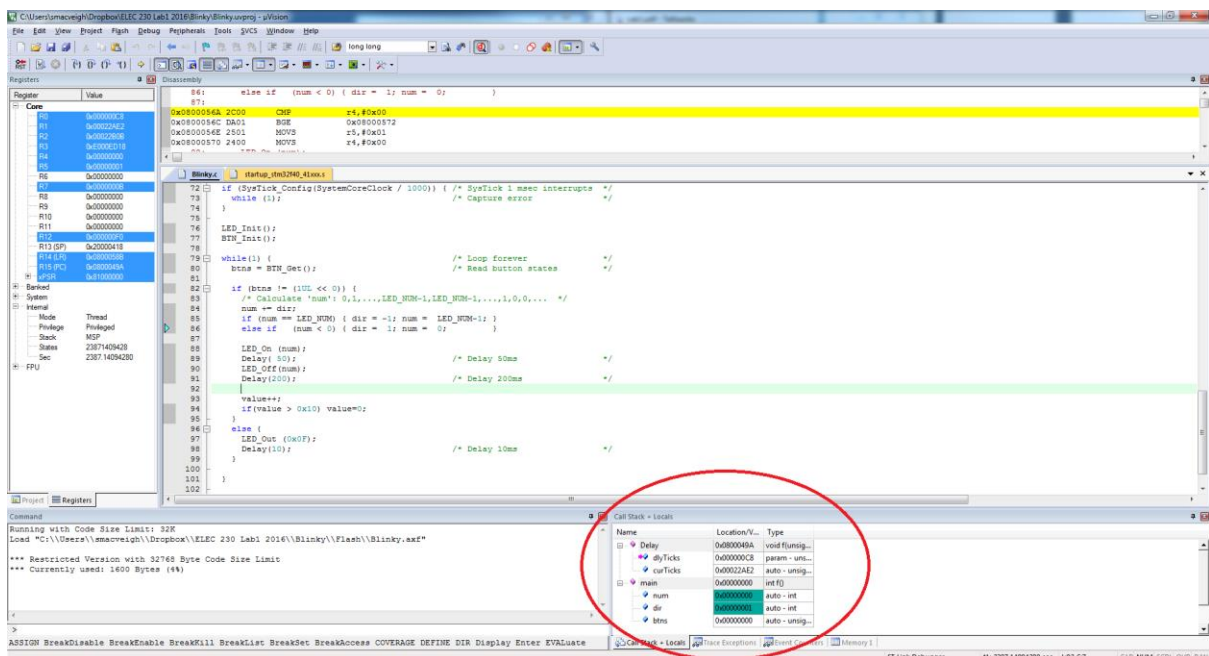


Figure 10: Debug Mode Screen

- The contents of local variables can be monitored in the **Call Stack + Locals** window (Figure 10 circled in Red).



- The contents of global variables can be monitored in the **Watch** windows. To add a global variable to the Watch window, right-click and add as shown in Figure 11.
- ❖ Details on Local vs Global variables are given in Section 6.

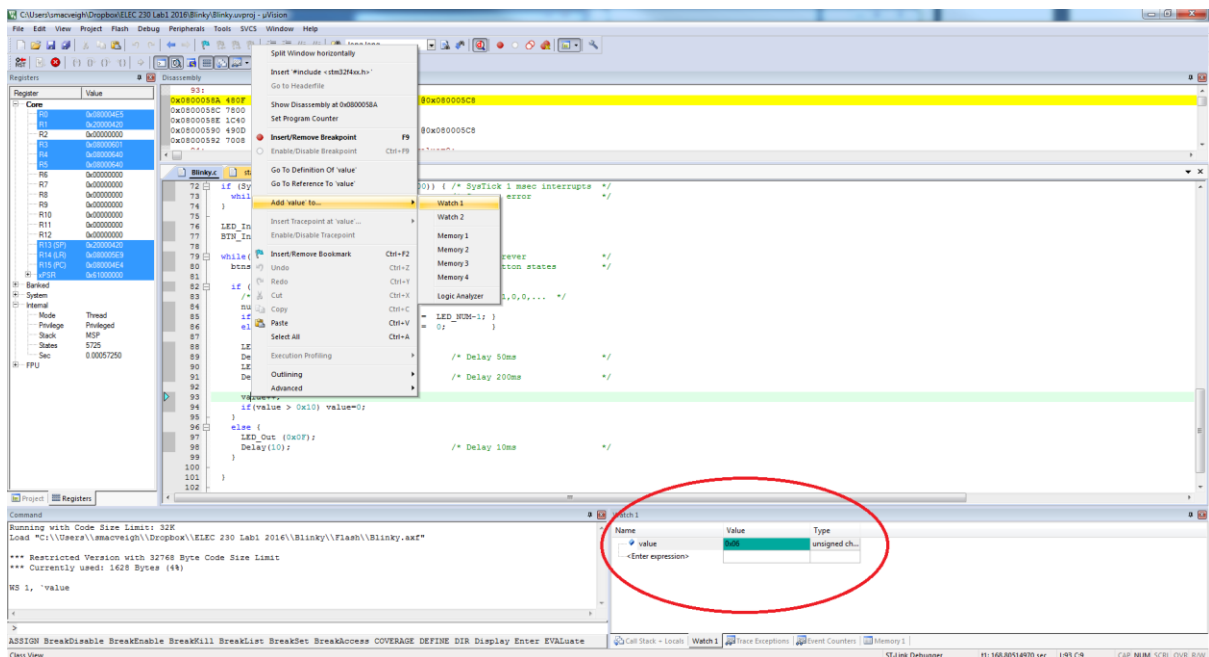



Figure 11: Adding Global Variables to the Watch Windows

#### 4.5 Break Points

Clicking on the side pane next to a line of code will create a break point. This will pause code execution when this line of code is reached. Clicking again will remove the break point. See

Figure 12. To remove all break points click  button.

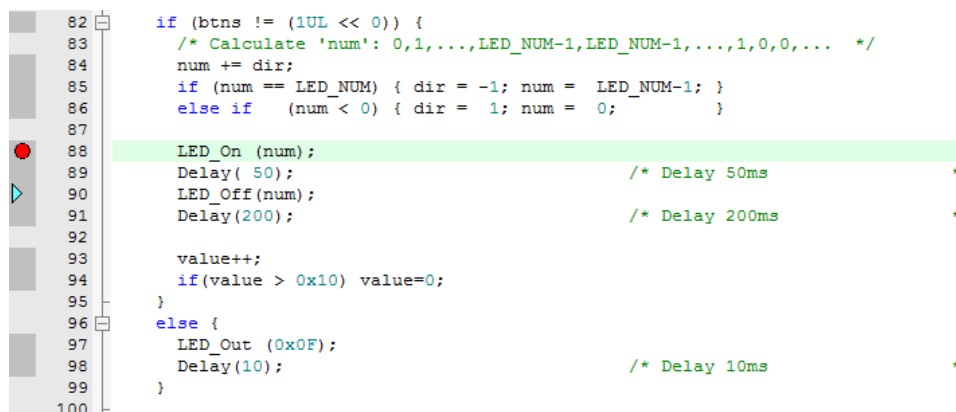
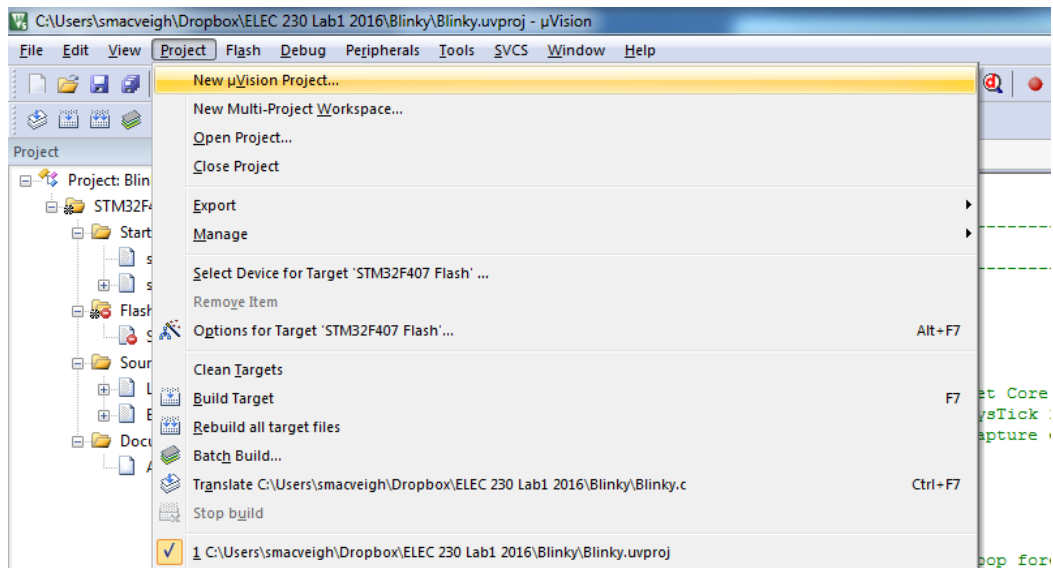


Figure 12: Setting a Break Point in Debug Mode

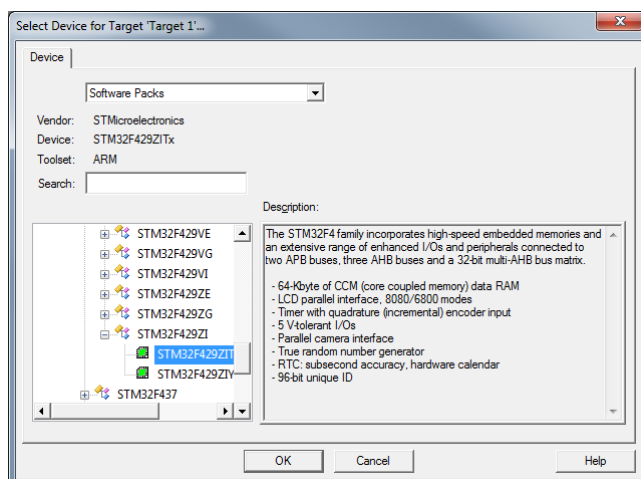
### 5 Task 2: Creating a Keil uVision Project

1. In Keil go to **Project -> New uVision Project**

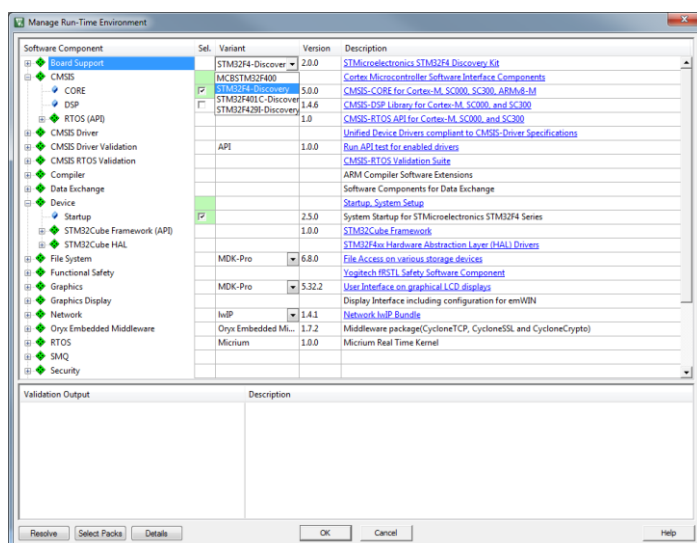


2. Create a new folder for this project to be stored into. (**ELEC240Lab1**)

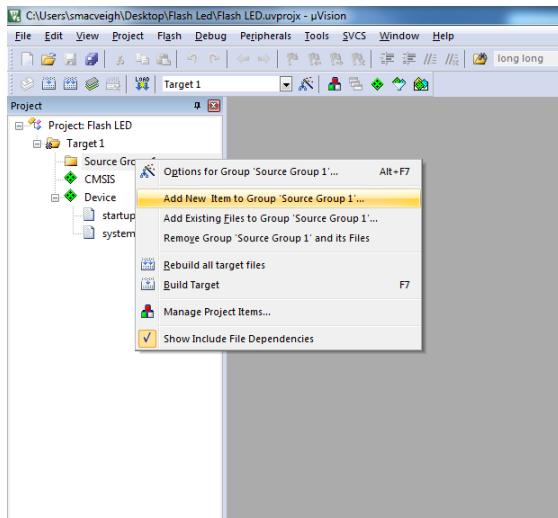
3. Select the micro-controller version (**STM32F429ZIT**)



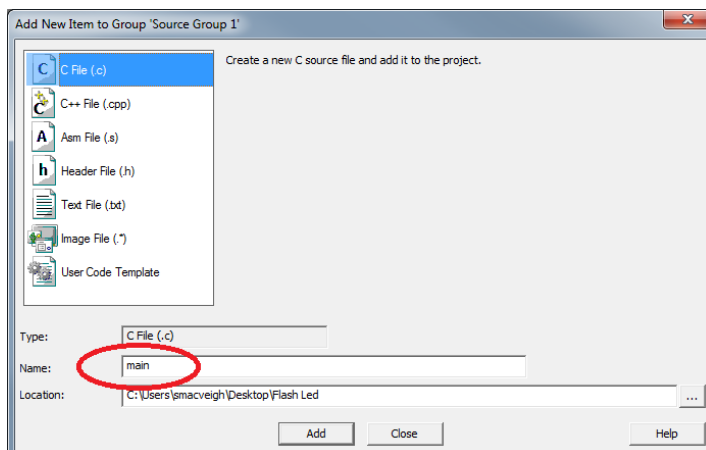
4. Include the relevant support files. Select **CMSIS->CORE** and **Device -> Startup**



## 5. Add a new source file to the project



## 6. Select the file type as .c and give it a name as shown (**main** is a good choice).



This file is where the code will be written/edited and compiled as covered in Section 4.

7. If a source file already exists simply right click the 'Source Group 1' folder, select 'Add Existing Files to Source Group 1' and import the necessary files.

## 6 Local Vs Global Variables

When a variable is defined within a function that variable becomes a **local** variable.

When the program counter enters the function containing this variable it is allocated a slot in a temporary block of memory called the stack.

This variable is deleted when the program counter leaves the function. Because of this the contents of the local variable can only be monitored when the program counter is paused inside the function.

When a variable is defined outside of a function or in a header file it becomes a **global** variable and is visible and usable to any functions below it in the page.

A global variable is allocated a static address in SRAM when the code is compiled and always maintains its value except for when the MCU is reset or powered off.

## 7 Support Documentation

The document names are hyperlinks, please click on them to access the documents and familiarise yourself with the information in the bullet points. **You will need this for future reference.**

### [UM1974 User manual](#)

- Pin identification and the supported special functions
- Circuit schematics
- Jumper and component identification
- Header pinouts

### [RM0090 Reference manual](#)

- MCU memory and peripherals architecture
- Peripheral control registers, addresses and bit-fields