

**DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING**  
**UNIVERSITY OF BRITISH COLUMBIA**  
**CPEN 391 – Computer Systems Design Studio**  
**2019/2020 Term 2**

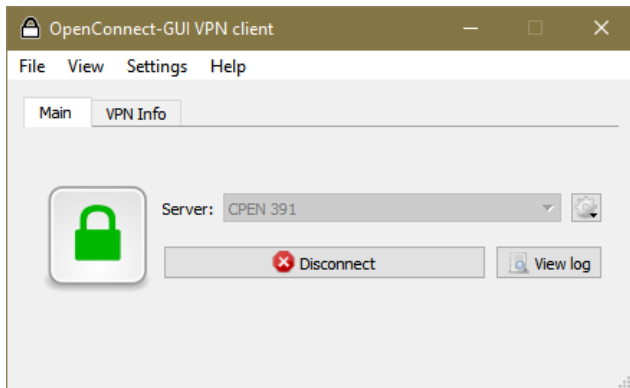
**Tutorial 1.5 – Using ARM DS5 Development Studio**

ARM produce their own Integrated C/C++ development studio based on Eclipse; it's called DS5 and we downloaded it in tutorial 1.0a as part of the Altera SoC EDS. Maybe you have used eclipse in another course as it is the open source IDE used by many manufacturers.

In this tutorial we'll use DS5 to create, compile, download and single step C/C++ "Bare Metal" programs intended to run on your Altera ARM/HPS based DE1 board. In Microcomputer terminology, a "Bare Metal" program is one that runs directly on the microcontroller without any supporting operating system. This means all IO operations on hardware devices have to be done directly using pointers, there are no OS drivers to simplify the task.

### **1.0 - Login to OpenConnect VPN Client**

Before you start using DS5 and especially if working "off Campus", it's important to login to the **OpenConnect VPN client** you downloaded in **tutorial 1.0a** (see tutorial 1.0 and 1.5 for the download so that a connection to **ECE's** license server can be made to verify the DS5 licence. If you log in, you should get a dialog box such as this – this one shows success.

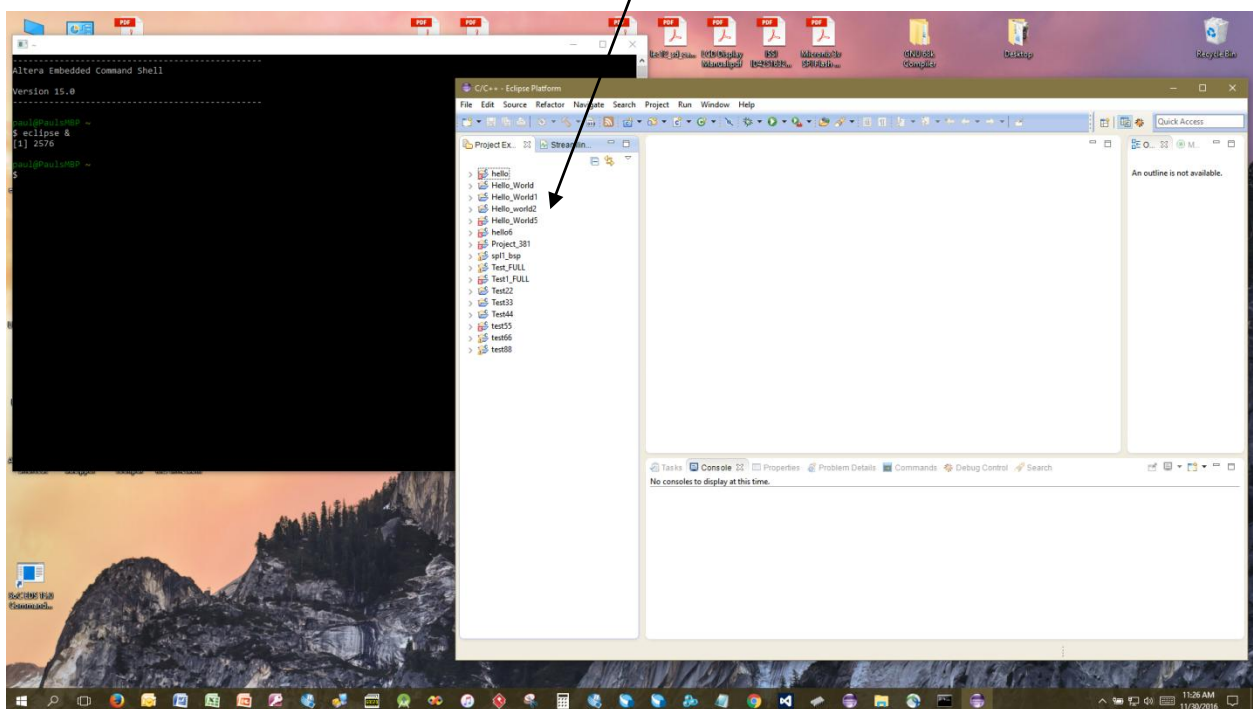


### **2.0 - Starting DS5**

Altera recommends that you do **not** start DS5 directly from the windows **Start menu** nor from any shortcut you might have placed for DS5 on the desktop. Instead it is recommended we first open the **Altera Embedded Design Command Shell** from the **shortcut** we placed on our desktop in **tutorial 1.0a**. Right click on that shortcut and select "**Run as Administrator**" – this is **important**.

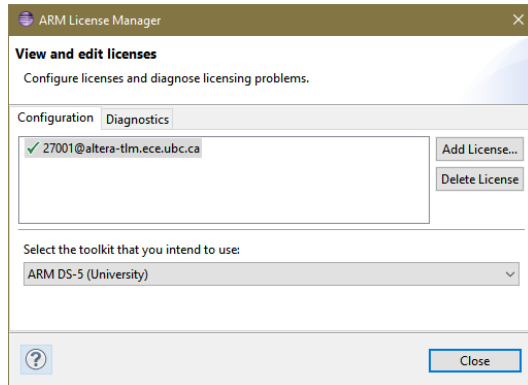


The Altera Embedded Command Shell window will appear as shown below. You can type “**eclipse &**” into it to start **eclipse** (i.e. the ARM DS5 IDE) as a background task. You can see eclipse and the various projects I have created in the past in the left hand pane.

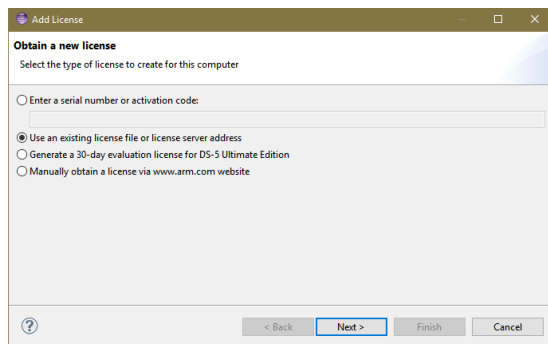


## Point Eclipse to ECE License Server

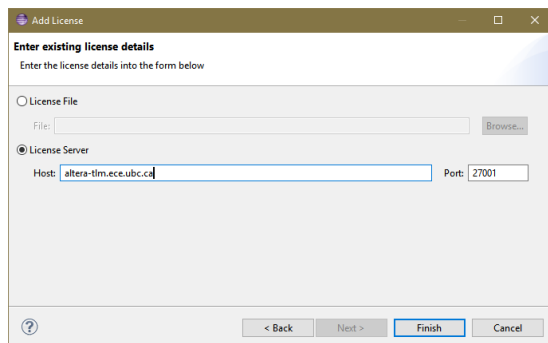
Before we create any projects, we need to activate the DS5 environment with a license. You may remember doing this in **Tutorial 1.0a**, but as a reminder it is repeated below. Click on the **Help Menu** in DS5 and select the option related to **ARM License manager**. You can see below the dialog that pops up and shows an active license (green tick) for the ECE server and a toolkit to use for the license.



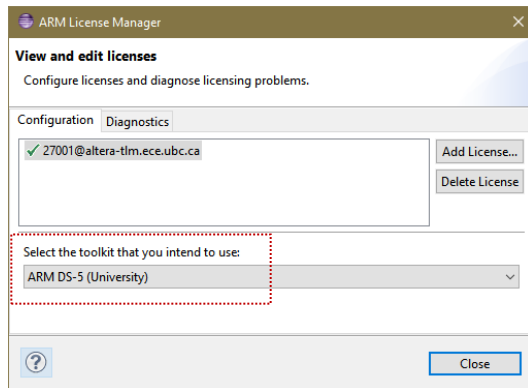
To add our own license, click the “**Add License**” button, this dialog will pop up, select the 2<sup>nd</sup> option as shown below, then click **Next**



This dialog will then pop up. Enter the data (**altera-tlm.ece.ubc.ca**) and port **27001** as shown in the box below and click **Finish**. **Note** if this should fail, try (**teaching-lm2.ece.ubc.ca**) and port **27001**

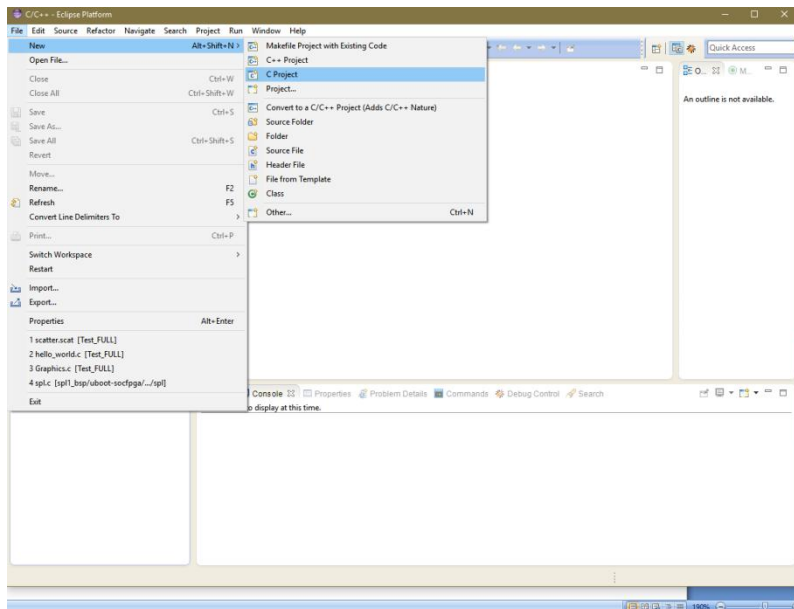


You will be returned to the main dialog for the License manager. Make sure you tick the correct tool kit if that option is available to you, as shown below.

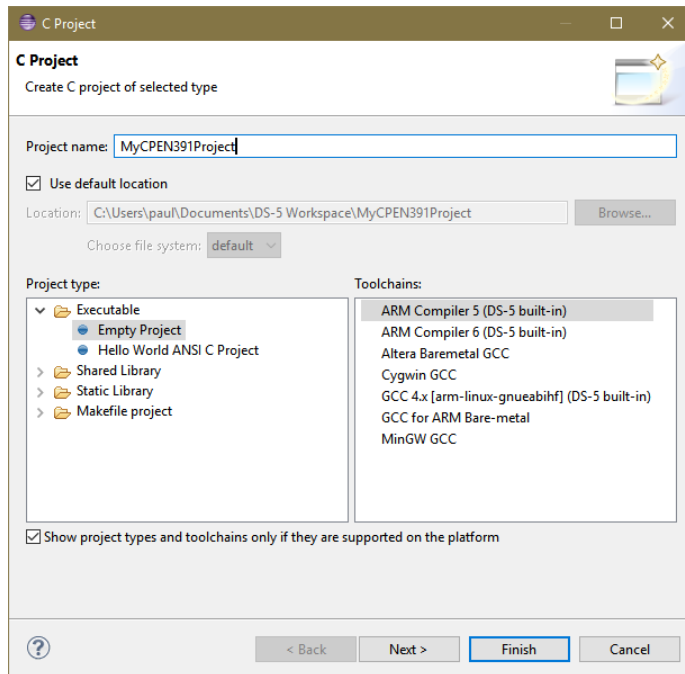


### 3.0 Making a new Project

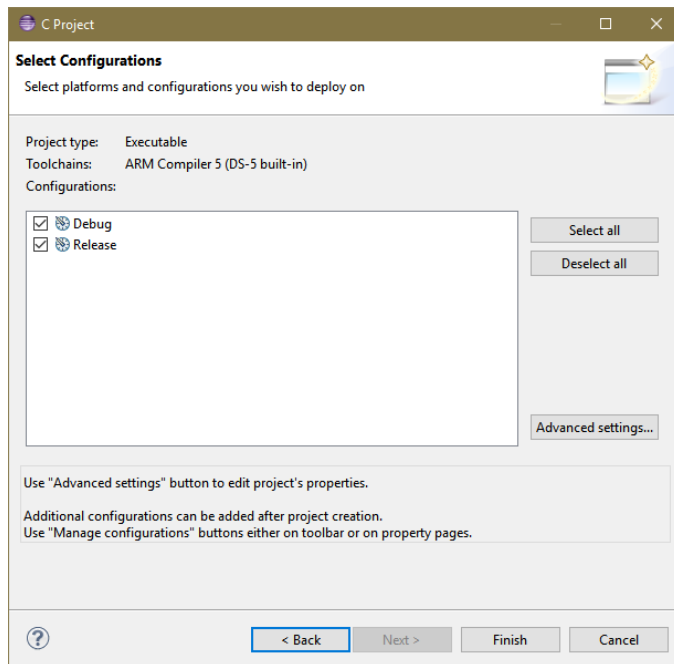
Create a new project like this by clicking menu “File->New->C Project”



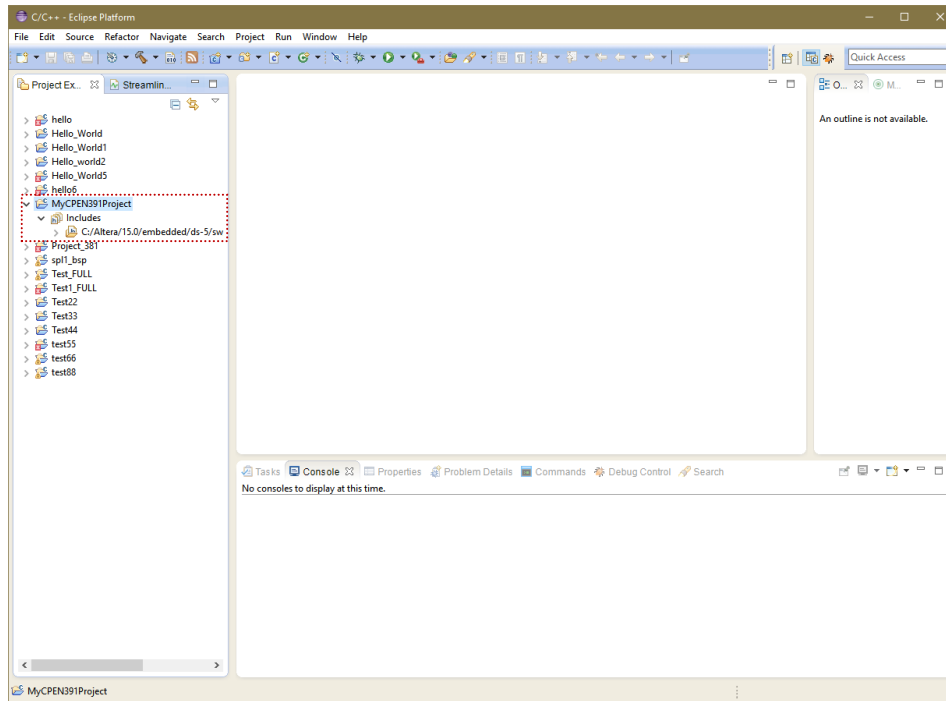
Give the project a suitable **name** as shown below e.g. **MyCPEN391Project** and choose an **Empty Project** based on the **ARM Compiler 5** that is built into DS5. Note you might not see these choices if you have forgotten to login in with Cisco VPN – that’s why it important to do that before starting DS5.



Click **Next**, this will appear

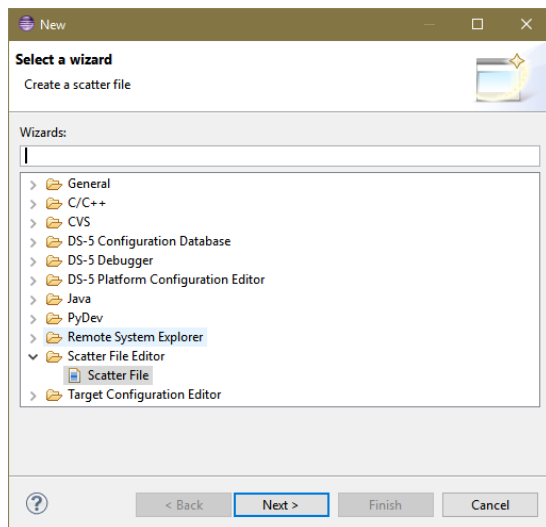


Click **Finish**. The project will be created and you can expand it as shown below

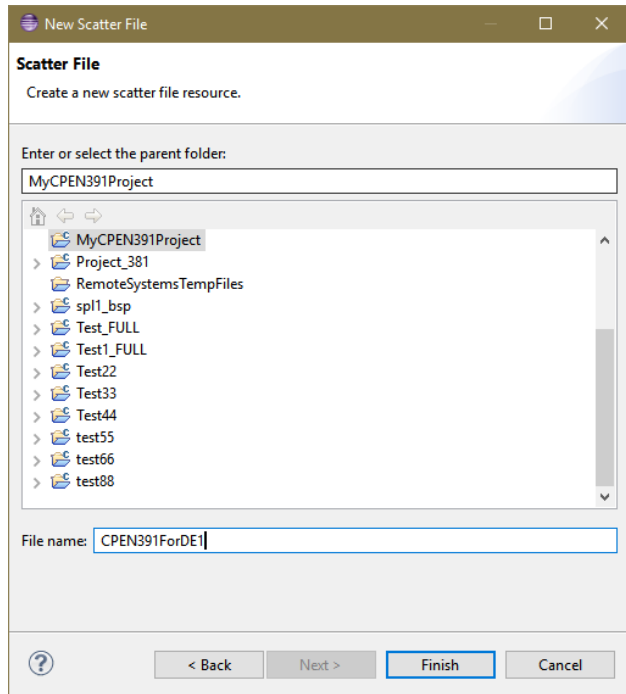


#### 4.0 - Creating a “Scatter File” for the Linker

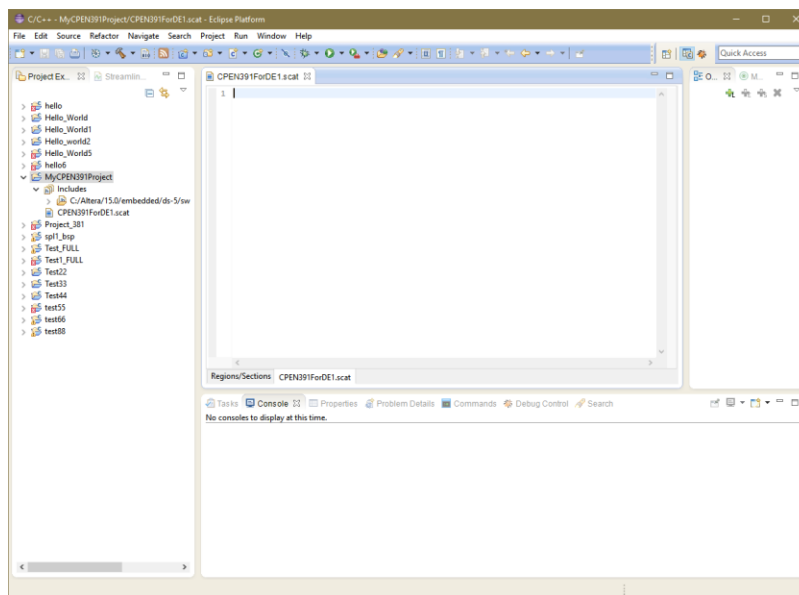
A scatter file is needed by the ‘C’ Compiler’s **Linker** so that it knows where in memory the program should be downloaded to. We have to **create** that scatter file and **add** it to the project and tell the linker to use it. It’s pretty easy. Start by clicking menu “**File->New->Other**”. Select **Scatter File**



Click **Next**. Select the **project** you want the scatter file to be associated with (see below)



Enter a suitable **name** for the scatter file (as above) and click **Finish**. You'll see the scatter file is now part of the project.



The scatter file is opened and is initially empty. Type the following into the text window

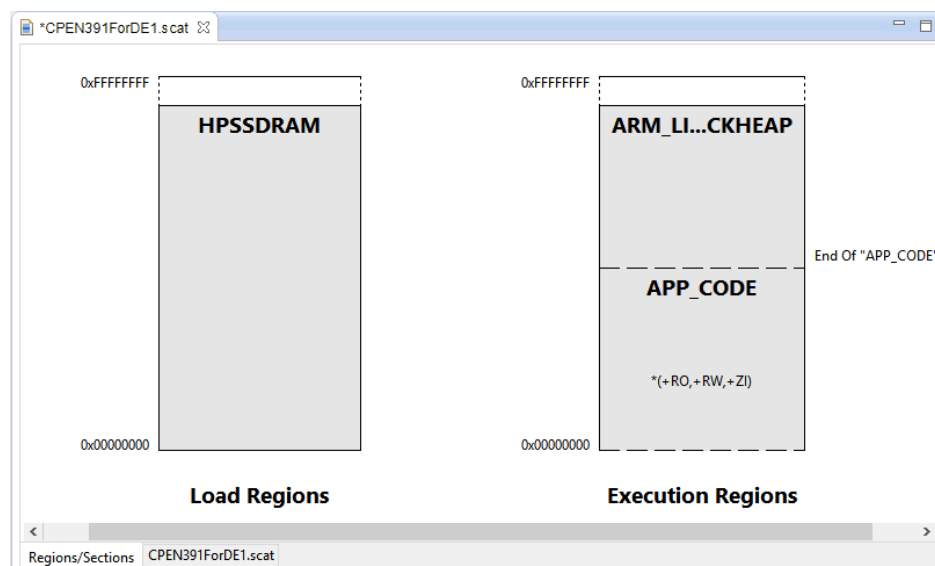
```

1 HPSSDRAM 0x00000000 0x40000000 ; A memory block name followed by a start address and a size in hex
2 {
3     APP_CODE +0 ; code loaded at base + 0
4     {
5         * (+RO, +RW, +ZI) ; read only, read write and zero initialised
6     }
7
8 ARM_LIB_STACKHEAP +0 EMPTY 0x10000 ; A size for the Stack and Heap (64kbytes) at top of memory
9 { }
10 }

```

A Scatter file allows us to define **regions** in the memory of our target system (the DE1) and dictates where programs can be loaded, where their **stack, heap, code and variables** will go. In the minimalist example above we defined a region of memory called **HPSSDRAM** starting at **address 0** and with a size of **1GByte** (that's the value 0x40000000 above). This is where our code and variables will live.

The stack and heap will be placed above the code (where exactly depends upon size of code) and in this example is **64kbytes (0x10000)** in size. If you click on the regions tab (at the bottom in the above image), you will see this displayed graphically (see below).



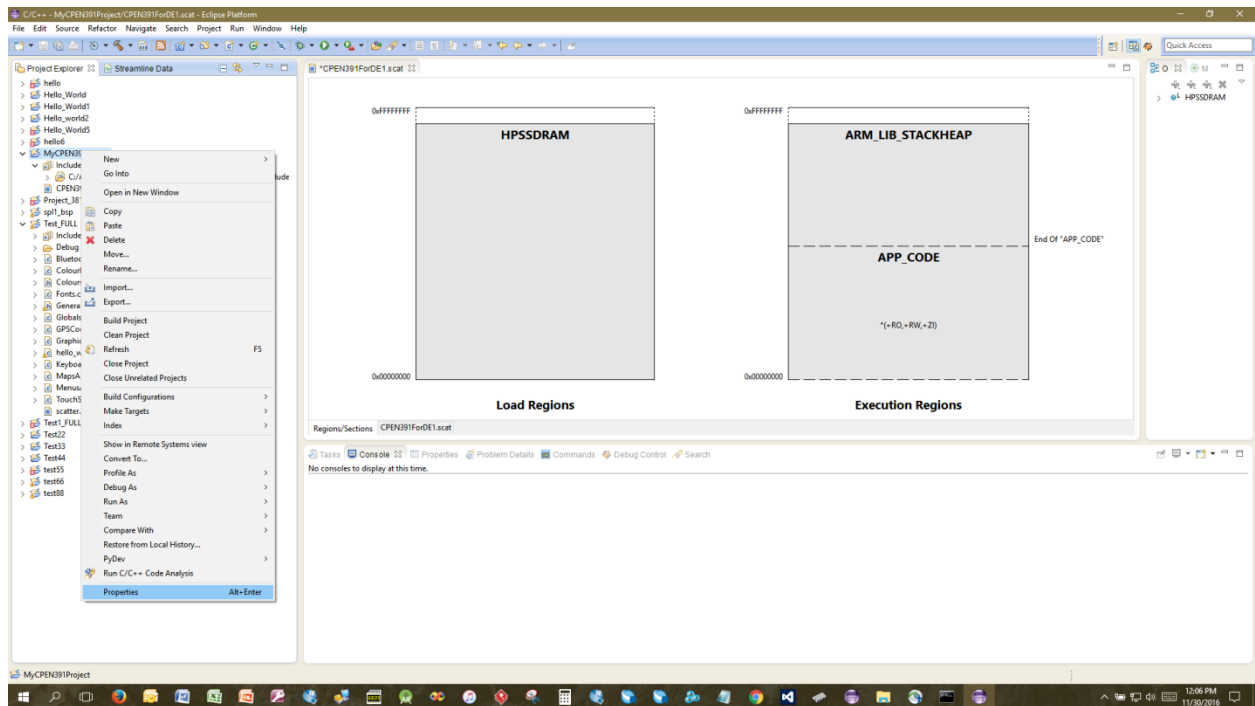
For more info about scatter files, you can read this

[http://infocenter.arm.com/help/topic/com.arm.doc.dui0474m/DUI0474M\\_armlink\\_user\\_guide.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.dui0474m/DUI0474M_armlink_user_guide.pdf)

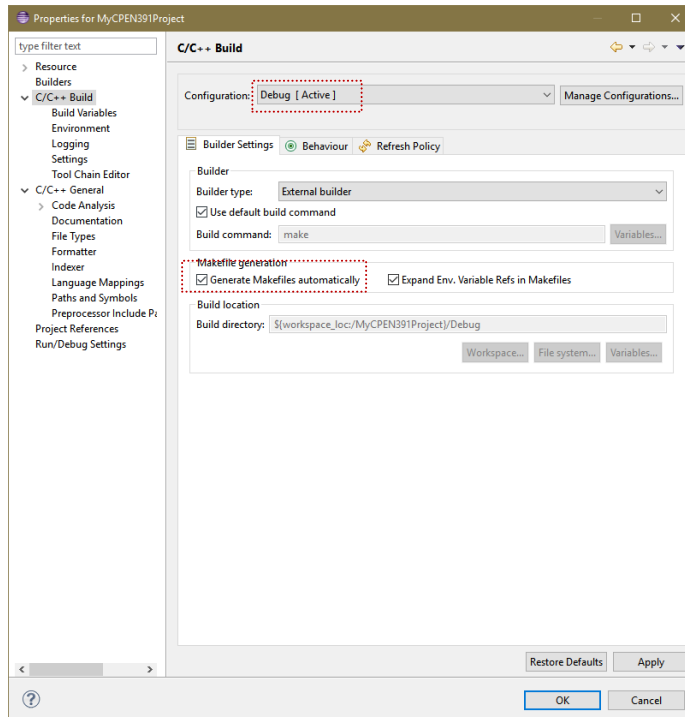


## 5.0 - Setting the Compiler Options and selecting the Scatter File

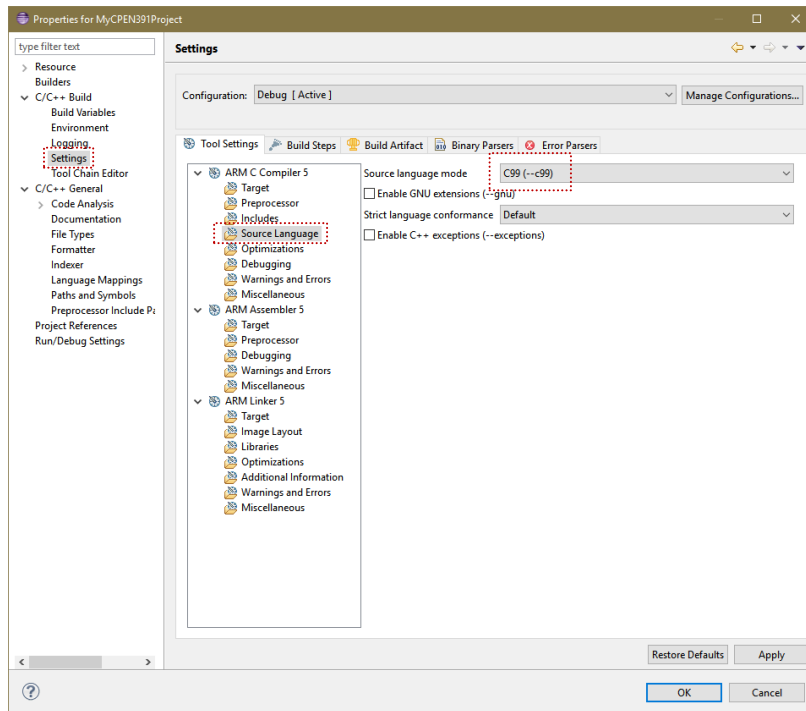
Right mouse click on the new project and select Properties (see below)



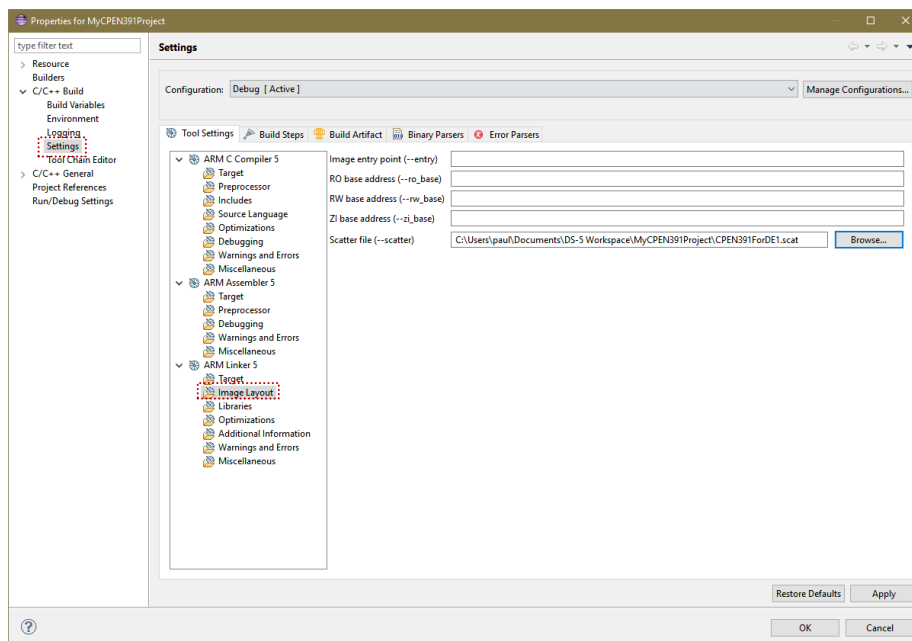
This window appears – you'll see the default configuration is **debug** (so we can single step etc) and it **generates** the **makefiles** for our project.



There are only a few settings to change for the project since the default ones work pretty well. Here we are going to set the 'C' Compiler source language to adhere to the **1999** 'C' standard as shown below. We do this as I've noticed that some DS5 tutorials and projects "on-line" imply this and will produce compile time errors unless this setting is selected.



In the **ARM Linker 5 ->Image Layout**, select the scatter file we generated above (see below)

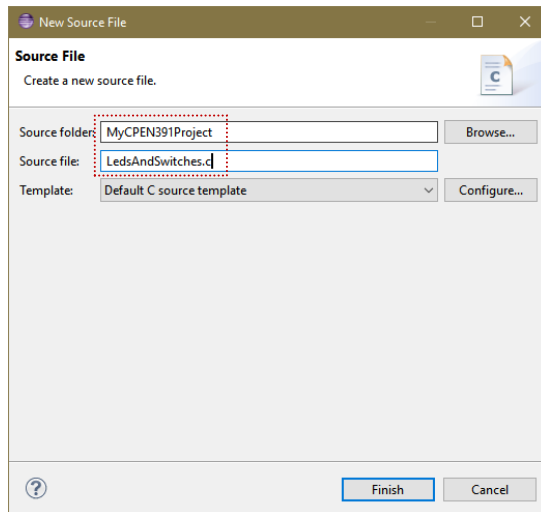


Click **OK**

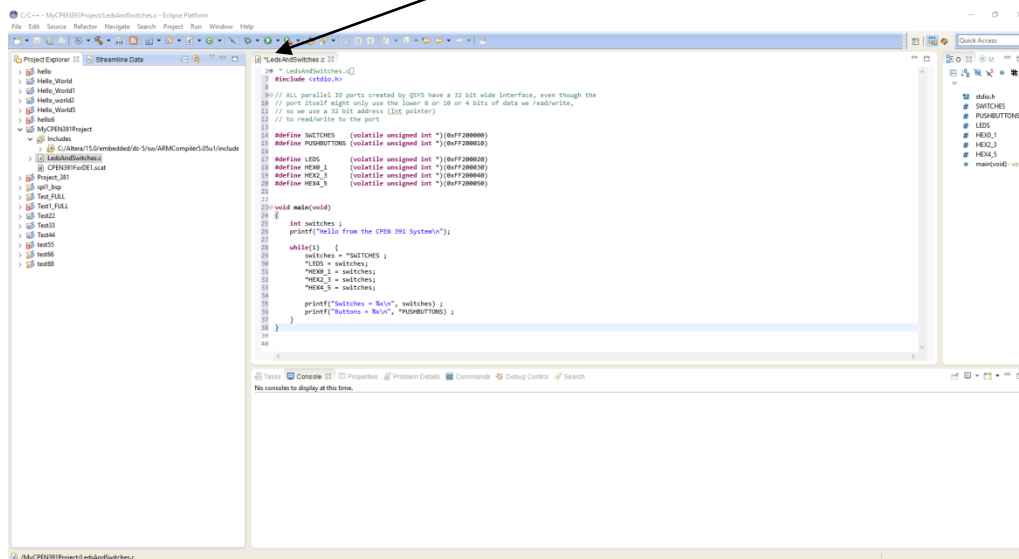
## 6.0 - Creating a new Source file for the project

We can add as many source files to DS5 as we like, to build up a multi-source file project and it will generate the **make files** etc. to recompile the files that change in the project. First let's create a single new source file. As a simple test, let's use a program that we already know works, the **LedsAndSwitches.c** program we wrote earlier (*it's on Canvas*).

In DS5 click menu "**File->New->Source File**". In the pop up window give a **name** like this one and associate it with our **project/source folder**. Note you have to physically type the **".c"** onto the file name **yourself**, it doesn't add one by default.

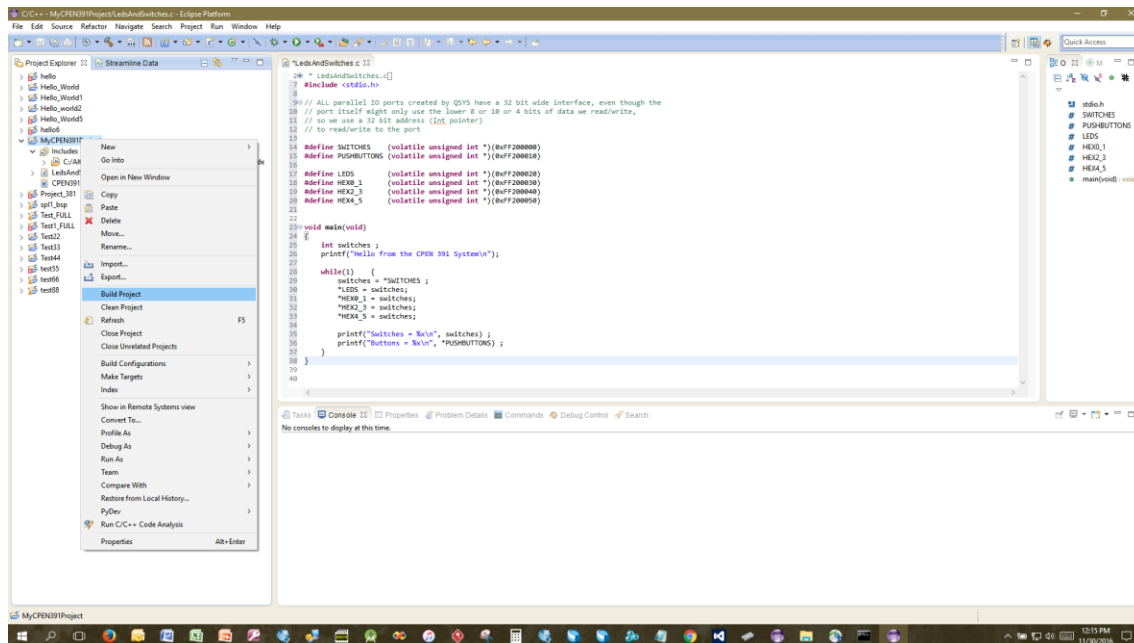


Click **Finish**. Now copy/paste that previous C file from **tutorial 1.3** into the new **C** file (*see below*)  
**Important:** Make sure you actually save the file (DS5 doesn't automatically save a file just because you chose to build the project). You can see the small **"\*"** next to the name of the file here showing it has not yet been saved.

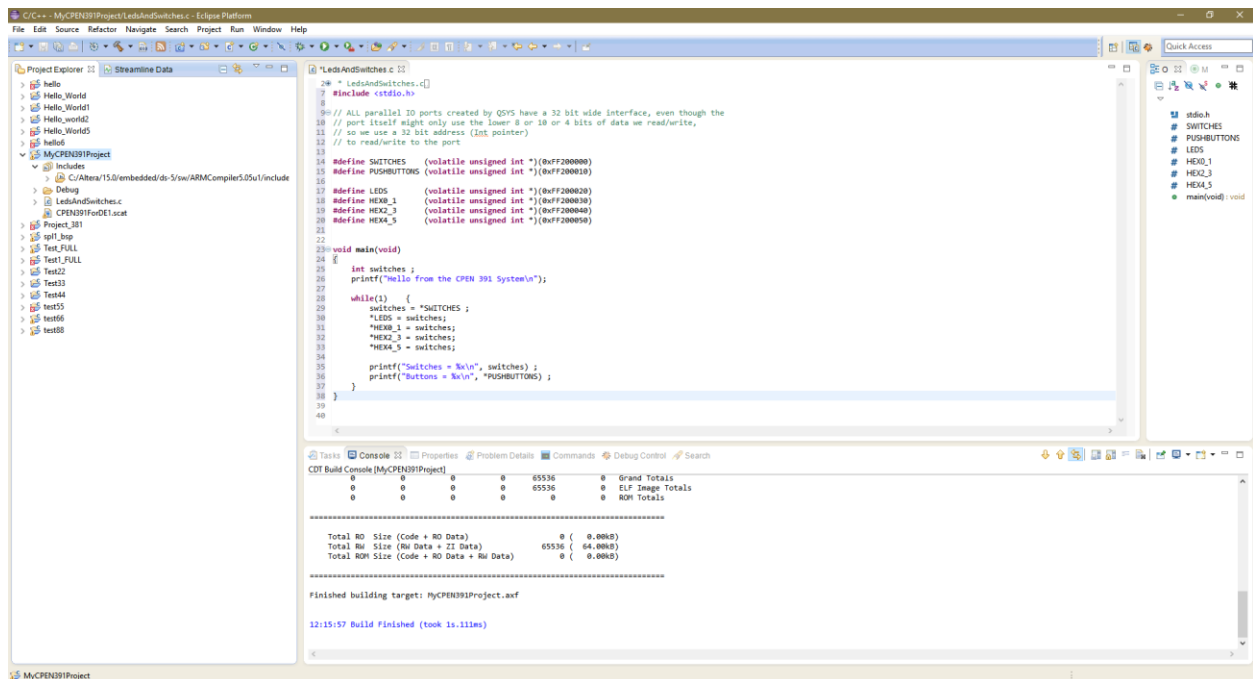


## 7.0 - Building the project

Right mouse click on the **project** and select **Build Project**

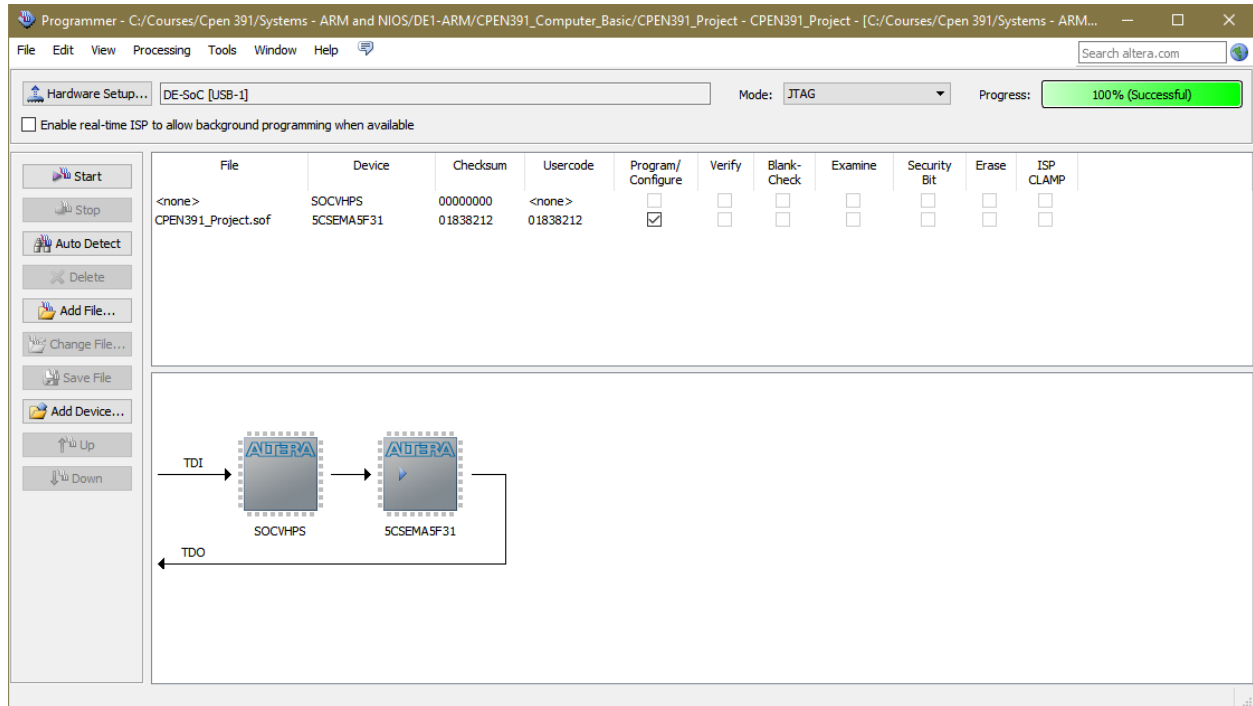


If all goes well the project will **build** and the **executable** will be **created** (see below). If you get an error message about the compiler **NOT BEING LICENSED**, check that you have logged in via **VPN** (see earlier) and if that doesn't work, **quit** DS5 and restart it.



## 8.0 - Downloading our Compiled Code

First make sure you have downloaded the hardware “.sof” file produced by Quartus when you compiled your hardware design. Use the **Quartus** Programmer to download the design to the DE1. See below



## 8.1 - Downloading the Pre-loader.

Remember that the pre-loader is a program that is run before the application to initialize all the HPS system components and bring the system to life before the application is run.

In the **Altera Embedded Command Shell** Window (the one we still have open from the time we started eclipse), type in this command to download a pre-loader to the DE1 ARM cores to execute. It's a bit long so you might want to keep this somewhere useful, such as in a **simple text file on the desk top** where you can open it quickly in **Notepad** and copy/paste it to the **Command Shell window**. Anyway, here is the command for Quartus **V15.0**

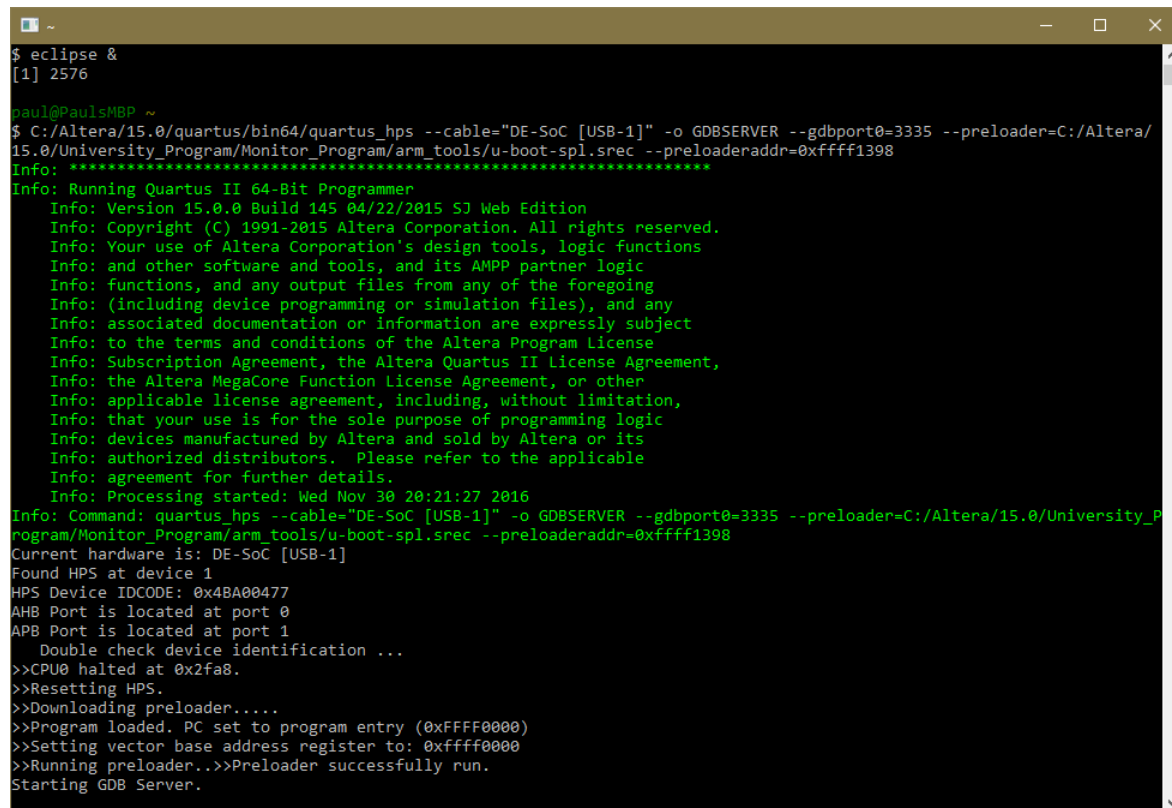
```
C:/Altera/15.0/quartus/bin64/quartus_hps --cable="DE-SoC [USB-1]" -o GDBSERVER --gdbport0=3335 --preloader=C:/Altera/15.0/University_Program/Monitor_Program/arm_tools/u-boot-spl.srec --preloaderaddr=0xffff1398
```

This is the one that seems to work for **V18.1**. You can see this command being issued by the Altera Monitor program from Tutorial 1.3 when it starts to download the system.

```
C:/intelFPGA_lite/18.1/quartus/bin64/quartus_hps --cable="DE-SoC [USB-1]" -o GDBSERVER --gdbport0=3335 --preloader=C:/intelFPGA_lite/18.1/University_Program/Monitor_Program/arm_tools/u-boot-spl.de1-soc.srec --preloaderaddr=0xffff13a0
```

You will see a pre-written “pre-loader” program downloaded to memory at address **0xFFFF1398** (for **V15.0**) or **0xFFFF13A0** (for **V18.1**) which is on chip Ram within the ARM cores. The pre-loader came with the University Program (UP) files we installed in tutorial 1.0. We could write our own, but it’s not a trivial task, so we’ll use the UP version.

When we used the **Altera Debug Monitor** earlier, this process was automated for us. Now we are using DS5, we have to do it our self, since DS5 is not specific to the DE1 board or Altera for that matter and does not know about these things. Here’s the output from that command. You can see it was executed **successfully**.



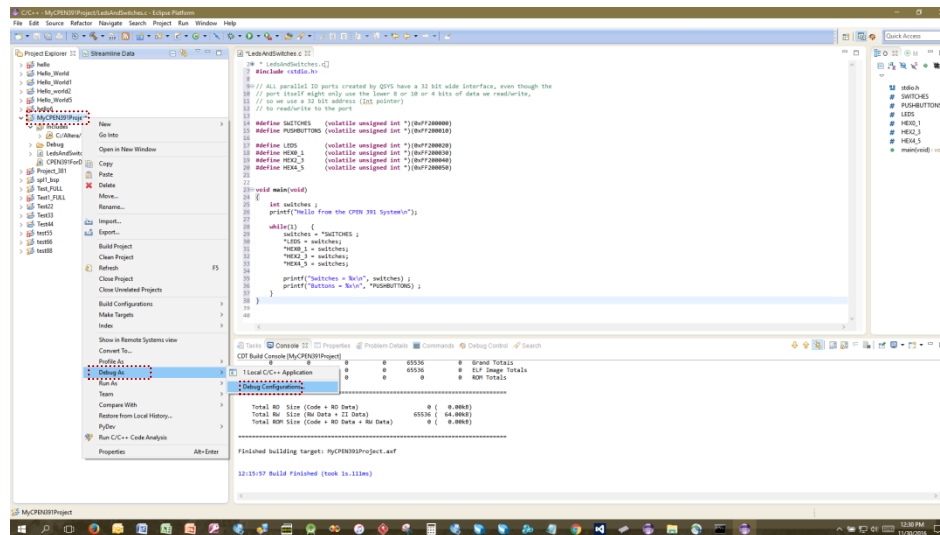
```
$ eclipse &
[1] 2576

paul@PaulsMBP ~
$ C:/Altera/15.0/quartus/bin64/quartus_hps --cable="DE-SoC [USB-1]" -o GDBSERVER --gdbport=3335 --preloader=C:/Altera/15.0/University_Program/Monitor_Program/arm_tools/u-boot-spl.srec --preloaderaddr=0xffff1398
Info: *****
Info: Running Quartus II 64-Bit Programmer
Info: Version 15.0.0 Build 145 04/22/2015 SJ Web Edition
Info: Copyright (C) 1991-2015 Altera Corporation. All rights reserved.
Info: Your use of Altera Corporation's design tools, logic functions
Info: and other software and tools, and its AMPP partner logic
Info: functions, and any output files from any of the foregoing
Info: (including device programming or simulation files), and any
Info: associated documentation or information are expressly subject
Info: to the terms and conditions of the Altera Program License
Info: Subscription Agreement, the Altera Quartus II License Agreement,
Info: the Altera MegaCore Function License Agreement, or other
Info: applicable license agreement, including, without limitation,
Info: that your use is for the sole purpose of programming logic
Info: devices manufactured by Altera and sold by Altera or its
Info: authorized distributors. Please refer to the applicable
Info: agreement for further details.
Info: Processing started: Wed Nov 30 20:21:27 2016
Info: Command: quartus_hps --cable="DE-SoC [USB-1]" -o GDBSERVER --gdbport=3335 --preloader=C:/Altera/15.0/University_Program/Monitor_Program/arm_tools/u-boot-spl.srec --preloaderaddr=0xffff1398
Current hardware is: DE-SoC [USB-1]
Found HPS at device 1
HPS Device IDCODE: 0x4BA00477
AHB Port is located at port 0
APB Port is located at port 1
Double check device identification ...
>>CPU0 halted at 0x2fa8.
>>Resetting HPS.
>>Downloading preloader....
>>Program loaded. PC set to program entry (0xFFFF0000)
>>Setting vector base address register to: 0xffff0000
>>Running preloader...>>Preloader successfully run.
Starting GDB Server.
```

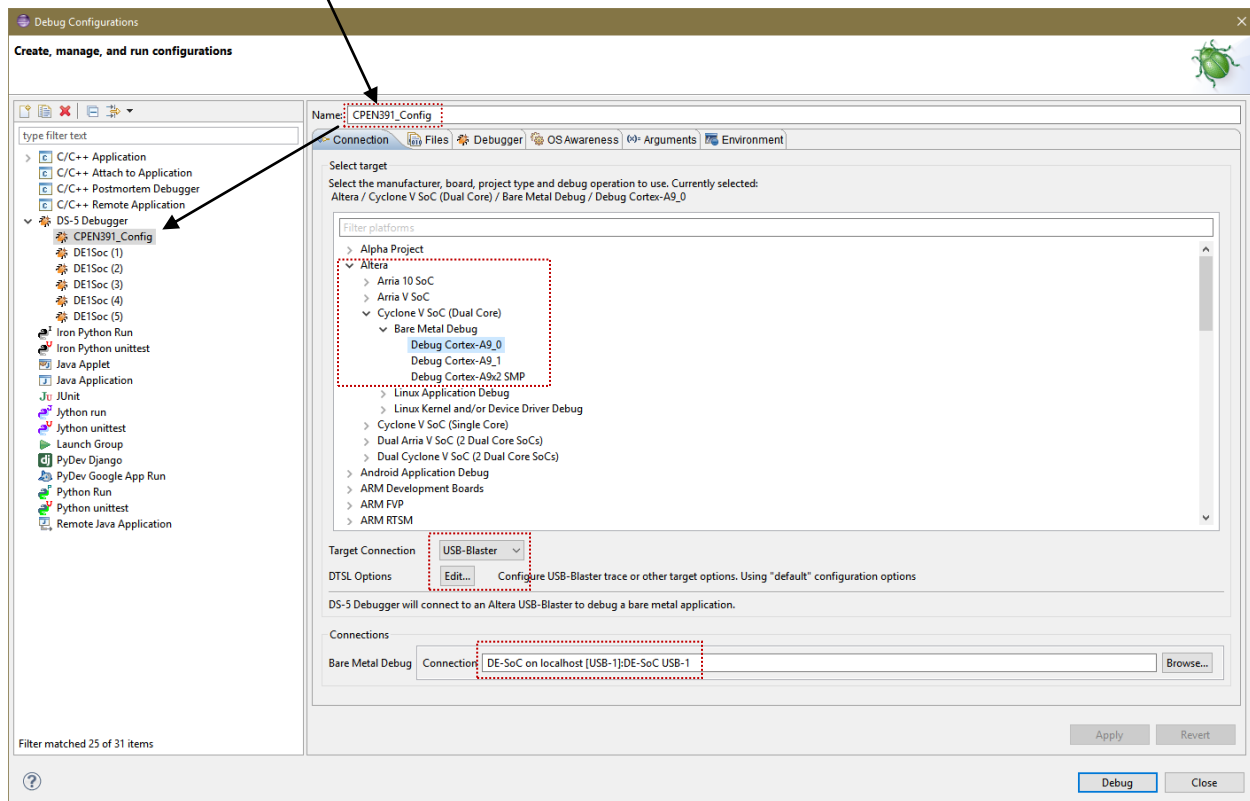
At this point a **GDB server is running** creating a debug session on the DE1. We actually want to **quit** that, so press **^C (Control-C)** to stop the server and get the ‘\$’ prompt back, we use **DS5** to set up a **debug** session instead.

## 8.2 - Using DS5 to download our Application – Creating a Debug Configuration

Right mouse click on the project and select **DebugAs->Debug Configurations**.

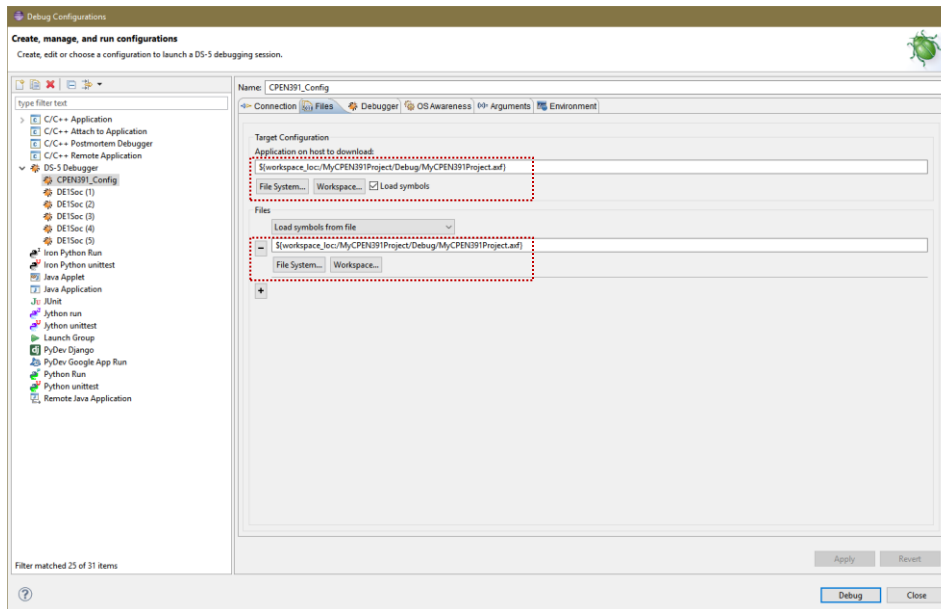


This Window appears – click on **“new”** to create a new configuration and give it a suitable name such as **CPEN391\_Configuration**. Make sure you select the other options/choices shown below (e/g Cyclone V->Bare Metal Debug-> Debug Cortex-A9\_0 (i.e. core 0 on Arm A9 processor integrated into a Cyclone V FPGA running Bare Metal application without an OS). **Click Apply**

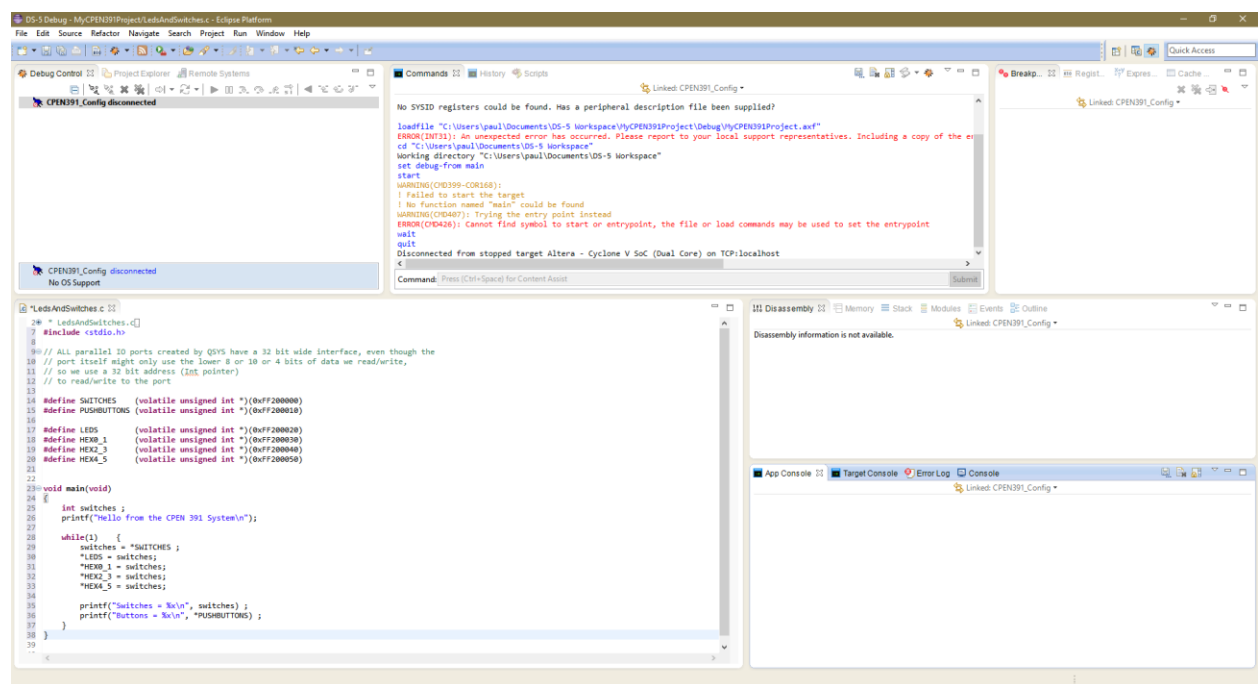


In the **Files** tab, select the **“.axf”** file for the **executable** program and **symbol files** for the program code we just built (*loading the symbols means we can single step/debug at the C source code level*) – see below. Click **Apply** the **Debug**. In the example below you will see some other configurations that I personally have made before which are not relevant here. Focus only on the one you create.



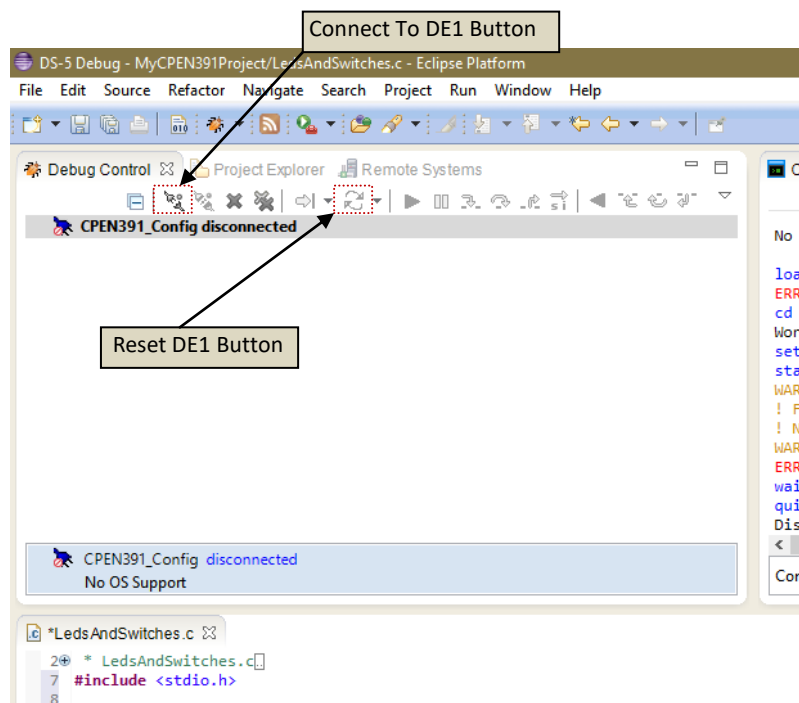


The debugger will start and the eclipse environment will switch to a new **Debug perspective**. Our source code to be run is shown **bottom left** along with the status of the ARM cores (top left).

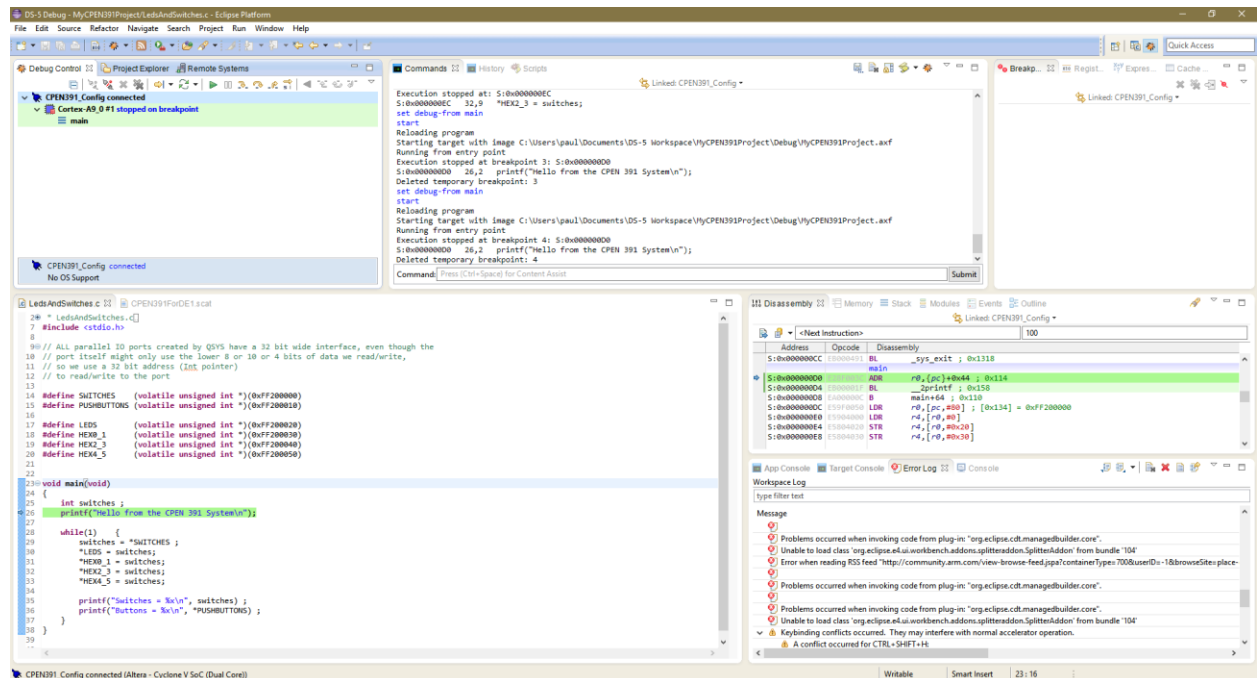


At this moment we are **disconnected** from the ARM cores, so click the **Connect** button shown below. During the connection, the program will **actually be downloaded** to memory on the DE1 board.

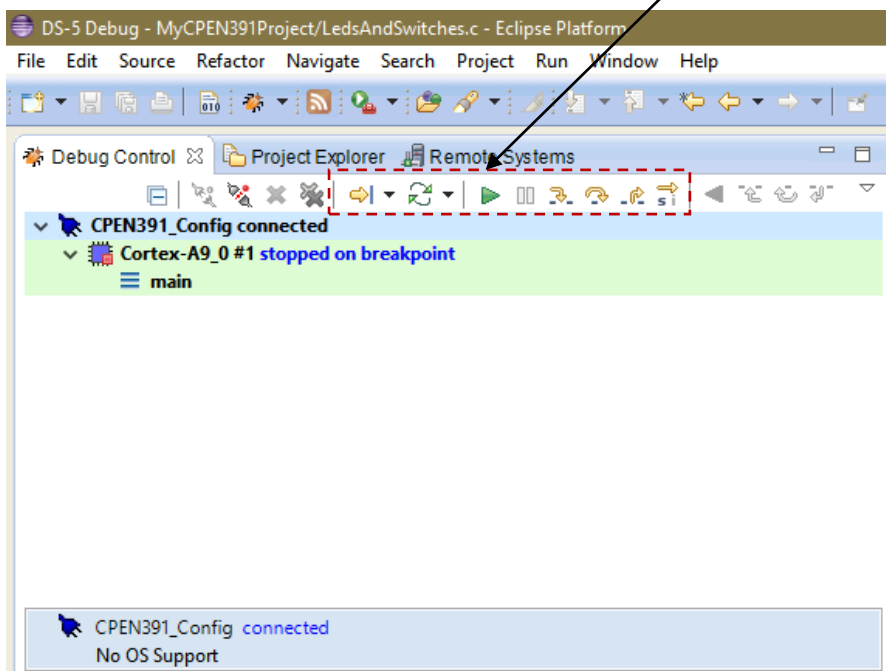
**Important:** The download will **fail** unless you have downloaded the “**pre-loader**” as per **Section 8.1** above. It will also fail if you have not VPN’ed to gain access to the ECE license server.



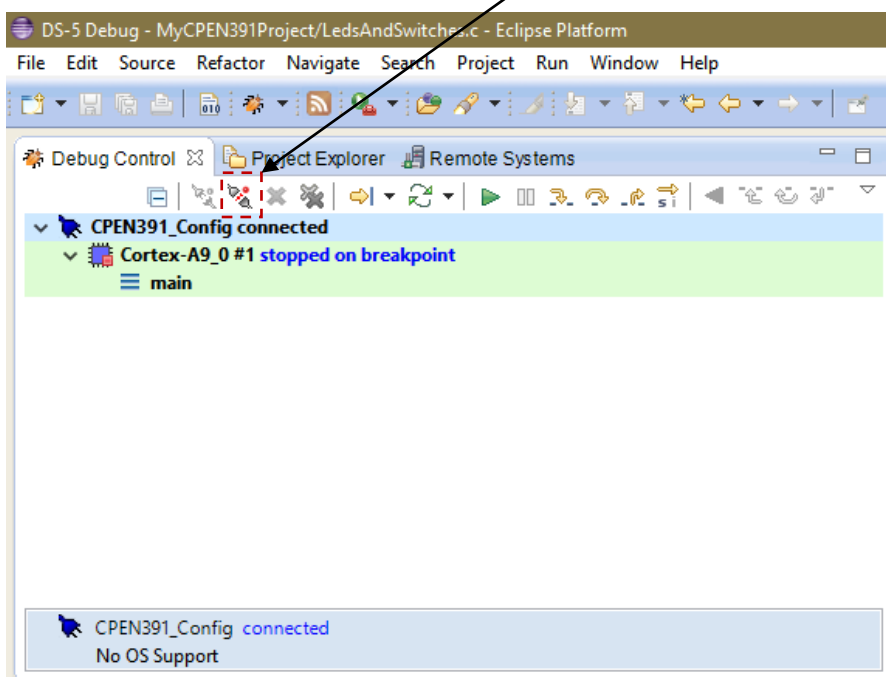
We can **disconnect** using the button to the **red** of it at any time. **Note** that if you **reset** the HPS at any time (either using the **cold reset button** on the DE1 or via the **reset button in the window above**), you will have to **download the Pre-loader again**. For the moment use the **connect/disconnect buttons**, not **reset**. Once you have connected, this should appear.



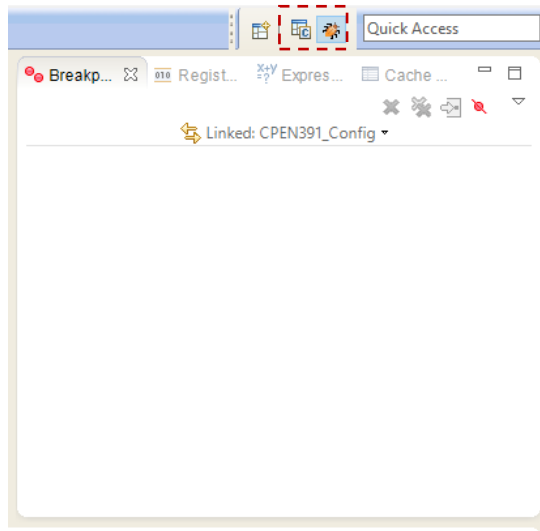
You can use the buttons below to **single step**, **start debugging from main()** again, or just **run**.



When you are done debugging, click the **Disconnect** button shown below



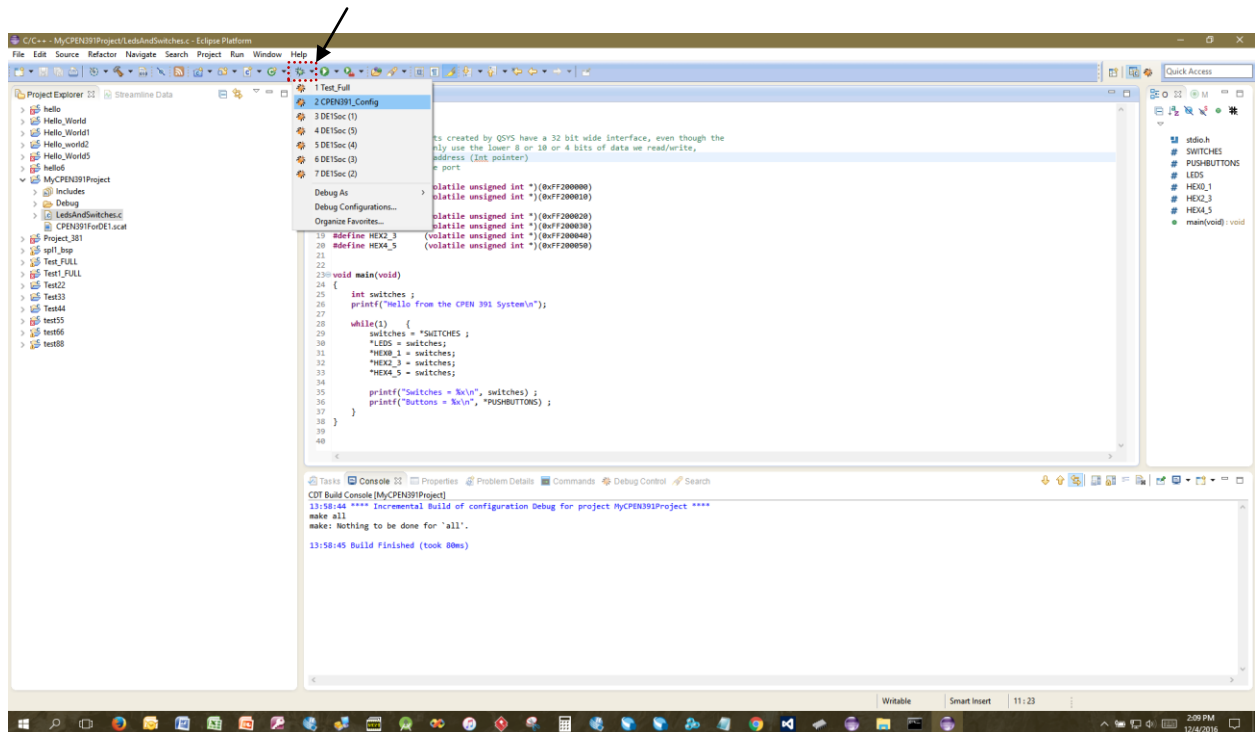
You can continue code development by returning to the **C perspective** shown here. In fact you can switch between the two perspectives using the two buttons shown **below**.



That's pretty much all you need to know. The rest you can pick up and experiment with.

### 8.3 - Shortcut to Debugging

It's worth pointing out a **fast shortcut** to building/downloading and starting a new debug session, as it does it all with one click of the button. In DS5 click on this button and select the previously created debug session, your project will be rebuilt, the perspective will change to debug, the code will be downloaded to the DE1 and a new debug session will start.



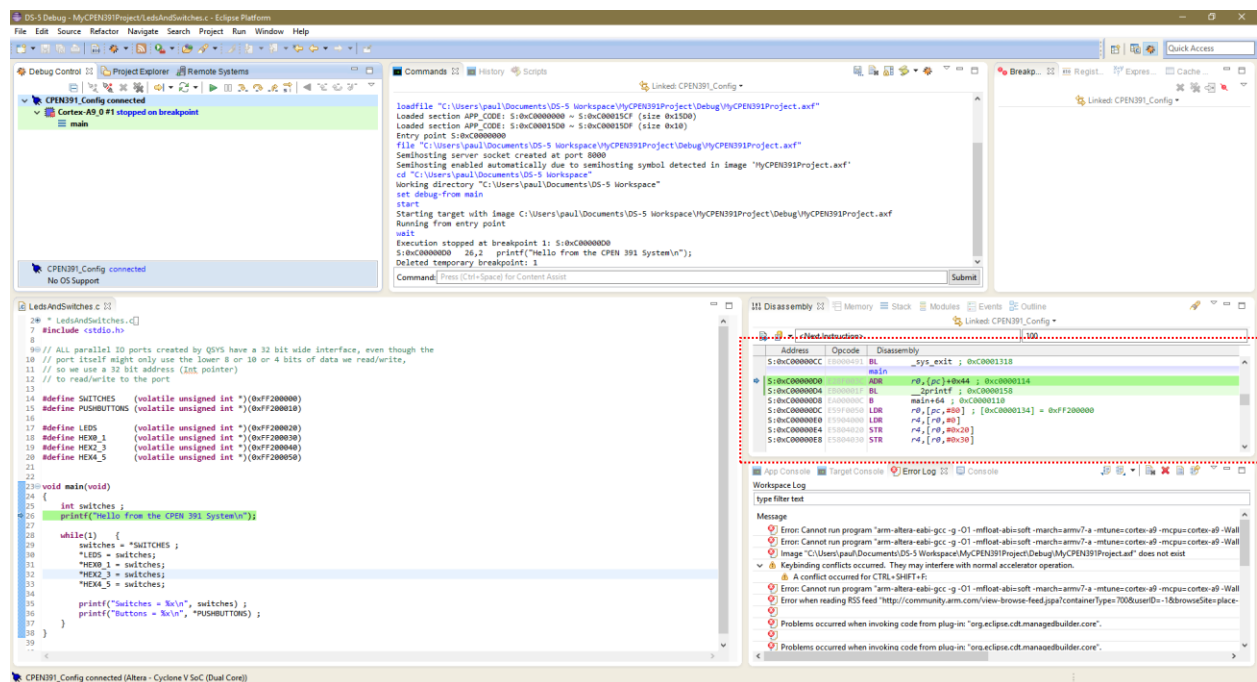
## 8.4 - Testing the Program on the FPGA side 64Mbyte SDRAM

One last thing, let's verify that the SDRAM on the FPGA side of the DE1 board works. The Memory (64Mbytes of it), lives at location **0xC0000000** in the ARM/HPS memory map, so let's (temporarily – because we really want to use the HPS side 1Gbyte memory chips) change our scatter file to make the program load to the FPGA side Dram.

Here are the changes to the **scatter file** – just the **top line** really, the **base address** and **size** of the memory block – makes these changes, “**clean**” then rebuild the project in DS5.

```
1 HPSSDRAM 0xC0000000 0x04000000 ; A memory block name followed by a start address (C0000000) and a size in hex (64Mbytes)
2 {
3     APP_CODE +0                ; code loaded at base + 0
4     {
5         * (+RO, +RW, +ZI)      ; read only, read write and zero initialised
6     }
7
8     ARM_LIB_STACKHEAP +0 EMPTY 0x10000 ; A size for the Stack and Heap (64kbytes) at top of memory
9     { }
10 }
```

When you debug the program, you will see this. You can see the program is loaded at hex **C0000000** and it will run – if not, you did something wrong in Qsys when you added the SDRAM to the project so go back and fix it.



Ok now we have verified the SDRAM on the FPGA side, **go back** and **return** the scatter file back to the original memory map/region.