**Creating a Xilinx SDK 2017.4 C Project**

This document will explain the process for creating a new project in the Xilinx Software Development Kit, version 2017.4. The steps to complete this process will be thoroughly explained, including the choices for hardware files, etc.

[1. Acronyms and Background Information 2](#_Toc3394961)

[1.1 Acronyms 2](#_Toc3394962)

[1.2 Background Information 3](#_Toc3394963)

[2. Preparation For Project Creation 4](#_Toc3394964)

[2.1 Software to Download 4](#_Toc3394965)

[2.2 Download SDK 2017.4 4](#_Toc3394966)

[3. Creating a Project 6](#_Toc3394967)

[4. Appendix 15](#_Toc3394968)

# Acronyms and Background Information

## Acronyms

SDK - Xilinx Software Development Kit, version 2017.4

BSP - board support package

Bitstream - the hardware files (.bit and others) which are used by SDK to generate the BSP

SoC - System on a chip integrated circuit

MicroZed - AVNET development board with a Zynq XCZ7020 SoC

RMD Digital Board - Custom digital board with the Zynq XCZ7020 SoC

XC - Commercial grade Zynq 7000 series microprocessor SoC

XQ - Military grade Zynq 7000 series microprocessor SoC

Workspace - the area (folder) on the computer where SDK is opened when first started by the user. This area must not have spaces in the file path.

OS - operating system

XILFFS - Xilinx Fat File System; the SD card support and implementation libraries

XILRSA - Xilinx RSA Library; provides RSA encryption and decryption algorithms and SHA 2 hash generation algorithm.

SFN/LFN - Short file name/long file name

JTAG - Programming cable for programming the FPGA through SDK or Vivado Lab tools

FSBL - First Stage Boot Loader; allows the system (XC or XQ Zynq processor chip) to boot in standalone mode

EEPROM - Flash memory on the board

mcs file - boot file

elf file - boot file

BOOT Files - the files necessary to allow a mircoZed or RMD digital board to boot from the EEPROM and run. These files are: BOOT.mcs and “project\_name”.elf

## Background Information

By following these instructions, the user will be able to create a working project in SDK 2017.4, but the instructions can be applied generally to create a new, blank C/C++ project which can be modified further. Existing code will be used for this example and is located in the following GitHub repository:

<https://github.com/RMDInc>

Depending on the project that you are working with RMD on (or something else entirely!) these repositories may be interesting to you.

When creating a workspace in SDK, it is vitally important to note that it has no support for spaces “ “ in any file paths. Take this into consideration when opening a project or linking libraries to an existing project.

The code we will be using as an example will allow us to run a board in debug mode or in release by flashing the EEPROM.

# Preparation For Project Creation

## Software to Download

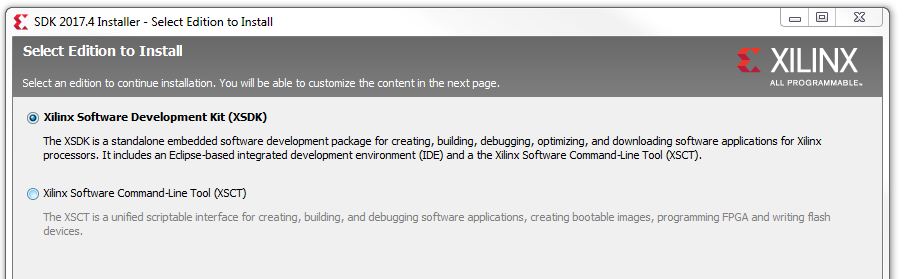
* Xilinx SDK 2017.4
* Source code for a C/C++ project
* Library code for supporting the project
* Bitstream which supports the hardware being targeted with SDK

The important item in this list is just SDK. The others will depend on what the user has in mind for their project. If starting from scratch, building your own flight software, or using a project from a repository (like the RMD GitHub above), then skip the second item on the list. SDK has a number of libraries built into it which support all of the functionality available with the microZed board and the RMD digital board. Finally, RMD provides the hardware files (bitstreams) which are located in their respective repositories. If you wish to extend these files, you will need to use Vivado or talk with someone at RMD. This is beyond the scope of this document.

## Download SDK 2017.4

This set of instructions requires the use of the Xilinx Software Development Kit software, version 2017.4. [This link](https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-design-tools/2017-4.html) will take you to the Xilinx website and the Embedded Development software downloads page. This software is free, though it does require the user to register on the Xilinx.com website.

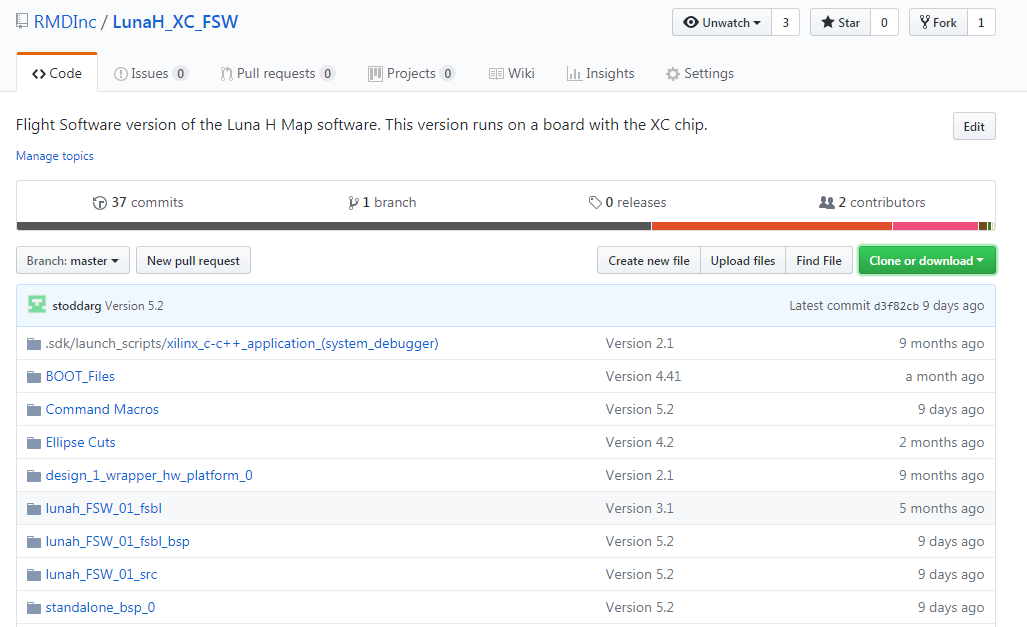
1. Click the link above to go to the Xilinx website
2. Select “SDK 2017.4 WebInstall for …” for your appropriate OS on the computer you’ll be using
3. After clicking on the download (Windows or Linux), the site will ask the user to log in. After doing so, information about the user is needed, enter it then press “Next” at the bottom. This will start the download.
4. An executable file will be downloaded.
5. Run the executable when it has finished downloading.
6. Press continue when the installer prompts the user to get the latest version
7. Press next on the Welcome screen
   * You may have to log in using the credentials that were used to download the software, then press next
8. Click “I Agree” for each box then press next
9. Make sure that the following boxes are selected for your installation, then press next:



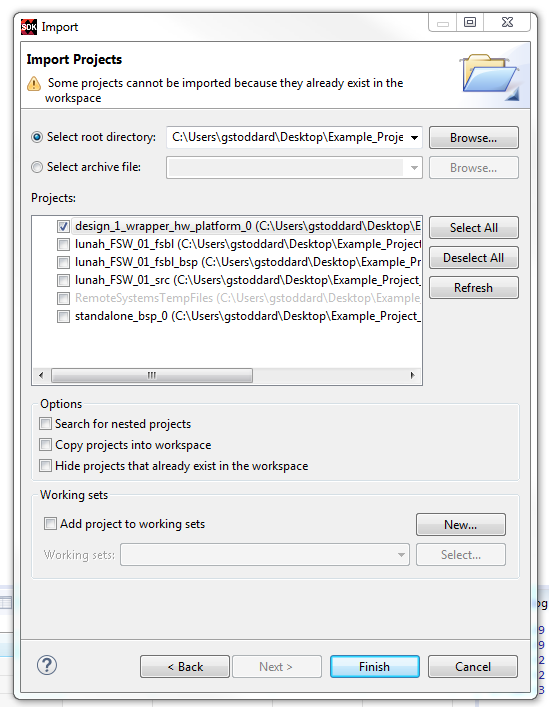
1. You may check all the boxes when prompted to customize your installation. This will give access to multiple different compiler tool chains for working with other processors in the future. For a smaller installation, we don’t need the following:
   * ARM Corex-A53
   * ARM Cortex R5
   * Enable WebTalk for SDK
2. On the Select Destination Directory screen, make sure that the program will install in an appropriate location, then press next
3. The Installation summary will show the options which were selected, press Install. The install should take ~1-3 minutes
4. Near the end of the install, a box will pop up telling the user to disconnect all Xilinx cables. If you have a JTAG or programming cable plugged in, unplug it from your computer.
5. When the install finishes, a box will pop up telling the user that the install completed successfully, press OK. SDK is now installed.

# Creating a Project

1. Go to the RMD GitHub and download the LunaH\_XC\_FSW repository.
   1. Note: This set of instructions was developed for Version 5.2 of the LunaH\_XC\_FSW. If there are further commits, these instructions might not work, in which case, contact Graham or Erik at RMD.
2. Click “Clone or Download”
3. Click “Download Zip”

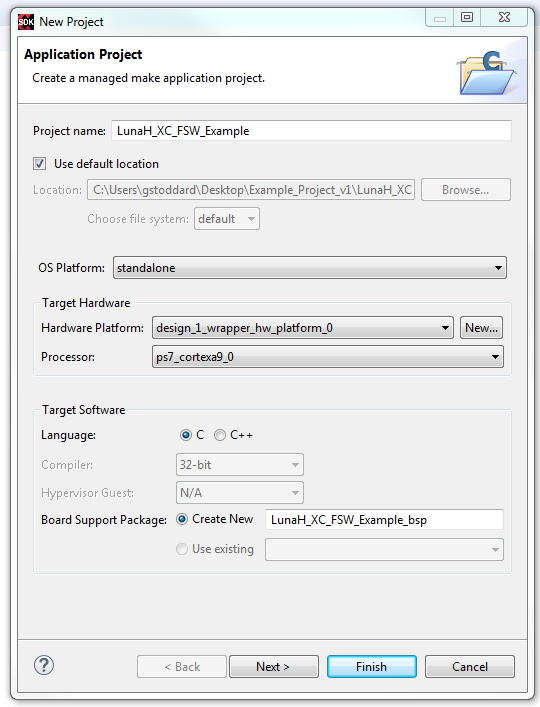


1. Unzip the files to a workspace with no spaces in the path
2. Open SDK 2017.4 and browse to the workspace where we unzipped the FSW code
   1. Close the Welcome screen.
   2. On the left of the window there is a tab called the “Project Explorer”. That is where any files and folders will display when they are created and/or added to the project. Currently, it should be blank.
3. Import the bitstream into the workspace
   1. File > Import…
   2. General > Existing Projects Into Workspace
   3. Click Next >
   4. Next to “Select root directory:” click the Browse button and navigate to the workspace, then hit OK.
   5. The “Import” window will pop up. The box under the Projects heading will populate with possible projects to import into SDK. Select the bitstream (in this case named “design\_1\_wrapper\_hw\_platform\_0”) from the list and de-select any other projects by ticking or un-ticking the boxes next to the project names; we don’t want any of the other code yet.



* 1. Press the finish button

1. Create an application project
   1. File > New > Application Project
   2. Give the project a descriptive name like “LunaH\_XC\_FSW\_Example”. This step will create a new board support package for the project based on the bitstream we imported in the previous steps.



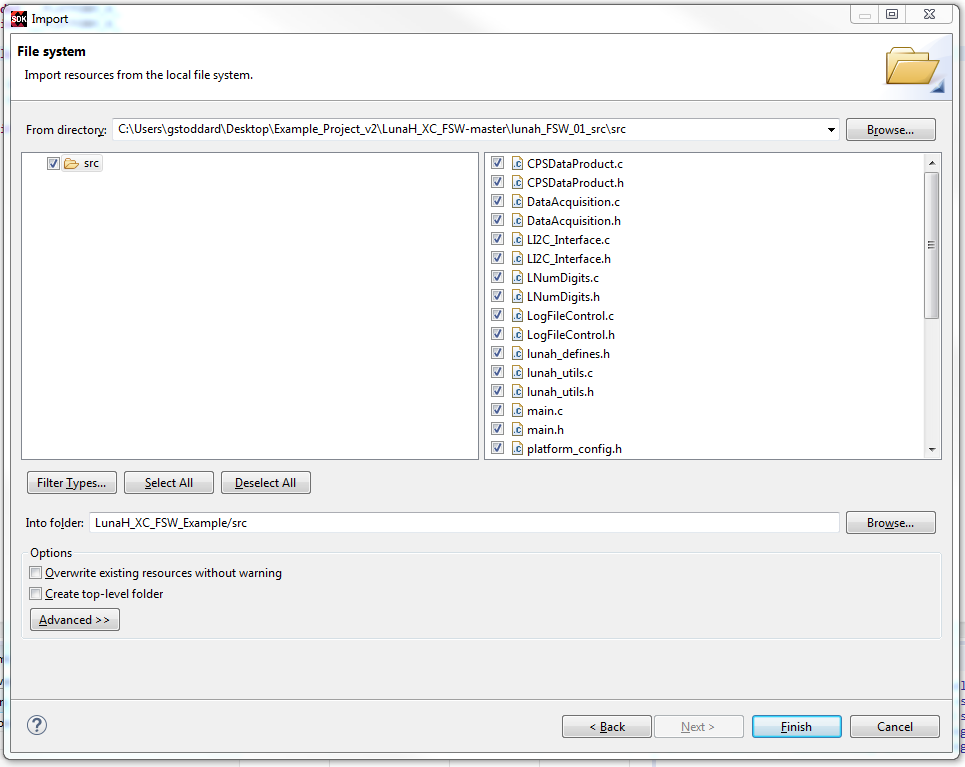
* 1. Press Next
  2. Select “Hello World” from the available templates, then press finish.
  3. It may be a minute while SDK builds the project.

Depending on the settings implemented by the source code being used, this could be the last step for project creation before building the project and running on a board. The LunaH project that was downloaded for the example project needs to be modified further, which will give the user a deeper glimpse into the workings of SDK and how to modify a project.

1. By opening up the Application Project folder we just created, there are four folders which SDK has created, with src being the one we’re interested in. Open the folder and we can see that there are a few automatically generated files in the src folder.
   1. Delete helloworld.c by right clicking and clicking OK when asked if we are sure.
   2. Ignore the errors generated, we’ll bring in the source code now
   3. Right click on the src folder > Import
   4. General > File System
   5. Press the Next button to go to the Import window.
   6. Click the Browse button and select the source code folder from the files we unzipped. The folder should be:

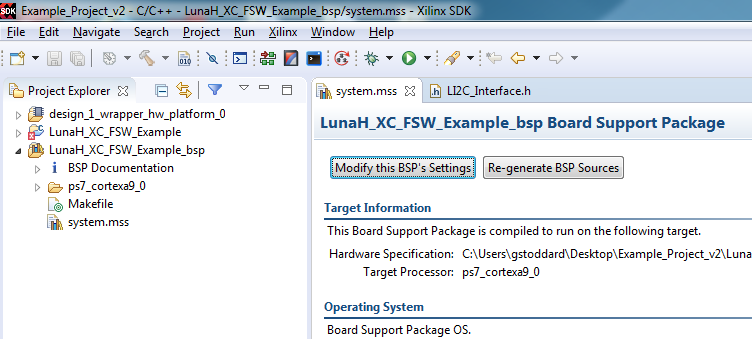
Workspace\Example\_Project\_v2\LunaH\_XC\_FSW-master\lunah\_FSW\_01\_src\src

* 1. The import screen will now have both boxes populated with the src folder on the left and the files on the right. Tick the box next to the src folder in the left box.

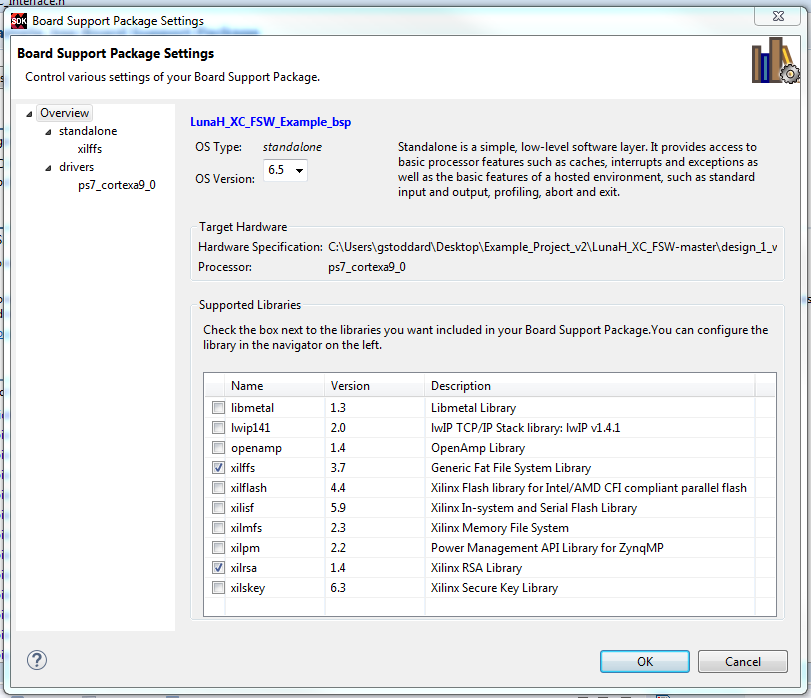


* 1. De-select the following files from the box on the right:
     1. platform\_config.h
     2. platform.c
     3. platform.h
     4. ps7\_init.c
     5. ps7\_init.h
     6. Xilinx.spec
  2. Verify that the folder we’re importing the files into is the src folder for the example project by looking at the “Into folder: “ heading. Press Finish when you are satisfied.

1. SDK will copy the files we selected into the src directory and will start re-building the project, but will generate an error as it is missing the ps7\_init files we did not bring in. Since these are specific to the bitstream, we want to grab them from the hw files each time we use a new bitstream. In this case, that means copying them into the src folder.
   1. In the Project Explorer tab, open the “design\_1\_wrapper\_hw\_platform\_0” folder
   2. Select ps7\_init.c and copy it to the src folder
   3. Select ps7\_init.c and copy it to the src folder
2. SDK will rebuild again each time you copy a file into src and will do so here. There will be another error, this time with SDK not finding ff.h, a library file which is found in the BSP.
   1. In the Project Explorer tab, open the “LunaH\_XC\_FSW\_Example\_bsp” folder
   2. Open the file “system.mss” to access the bsp settings
   3. Click on the button “Modify this BSP’s Settings”

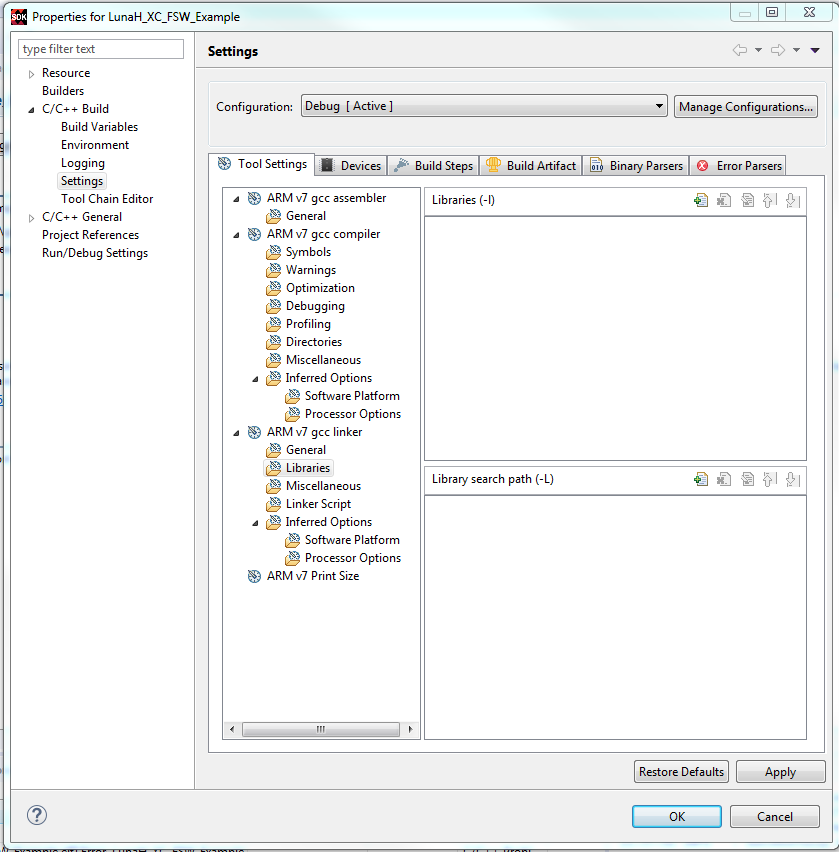


* 1. Tick the boxes next to XILFFS and XILRSA



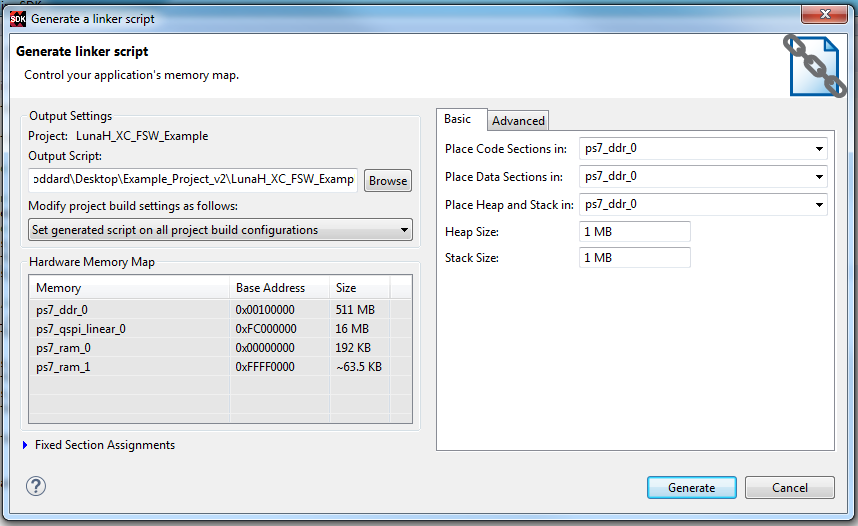
* 1. In the top left corner, under Overview > standalone, click on xilffs to access the settings for the Xilinx Fat File System (the SD card libraries)
  2. Change the setting for the “use\_lfn” configuration parameter from false to true.
  3. Press OK and SDK will begin rebuilding the BSP. This may take a minute.

1. There will be three errors, calling out that we have undefined references to the ‘floor’ math function. To include this in the project, we will tell SDK to compile the math library into our project.
   1. Right click on the “LunaH\_XC\_FSW\_Example” folder > C/C++ Build settings
   2. In the Tool Settings tab, under the “ARM v7 gcc linker” heading, click on Libraries.
   3. In top right corner of the “Libraries” box, press the Add… button



* 1. Type “m” into the line then press OK
  2. SDK will rebuild and we are left with a project with 6 warnings, but no errors.

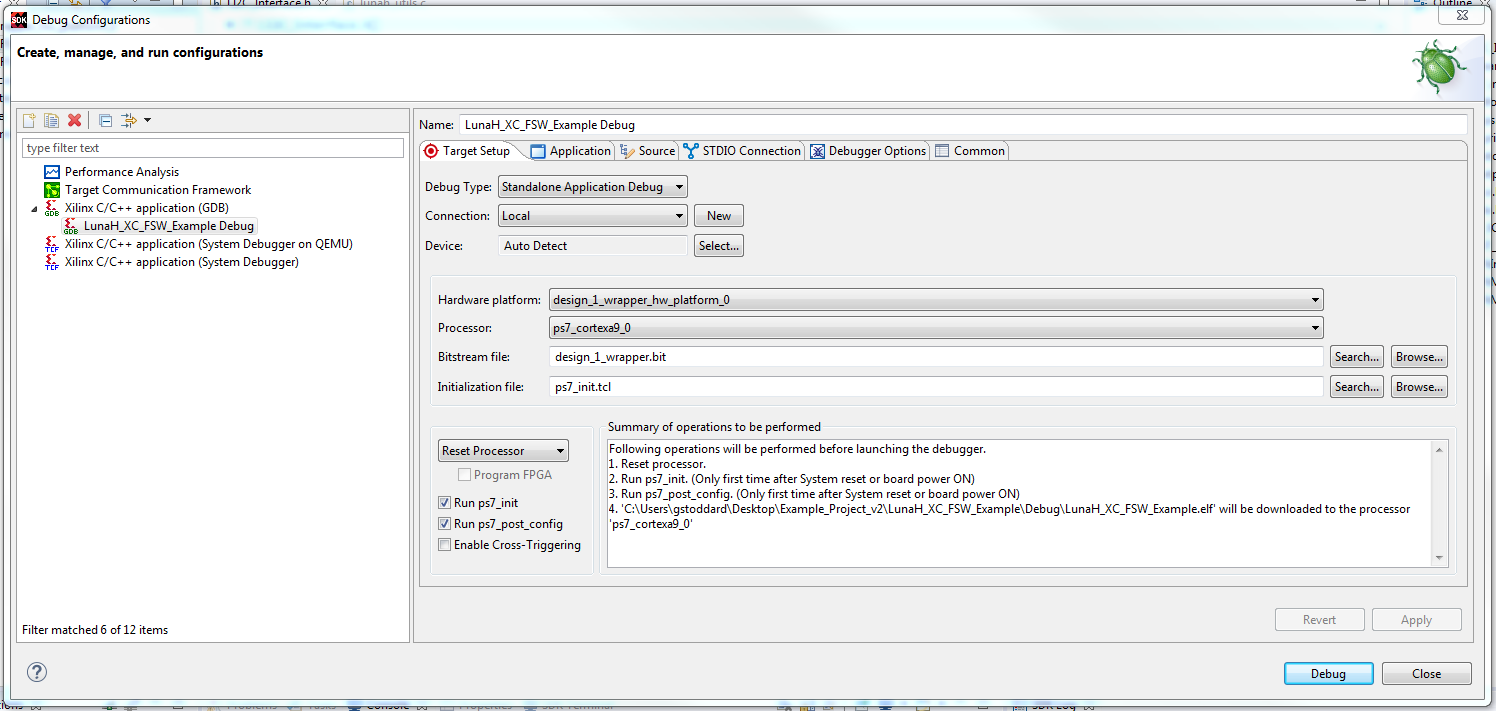
1. For the LunaH Example code, we now need in increase the size of the stack and heap available to the program when it’s running.
   1. Right click on the “LunaH\_XC\_FSW\_Example” folder > Generate Linker Script



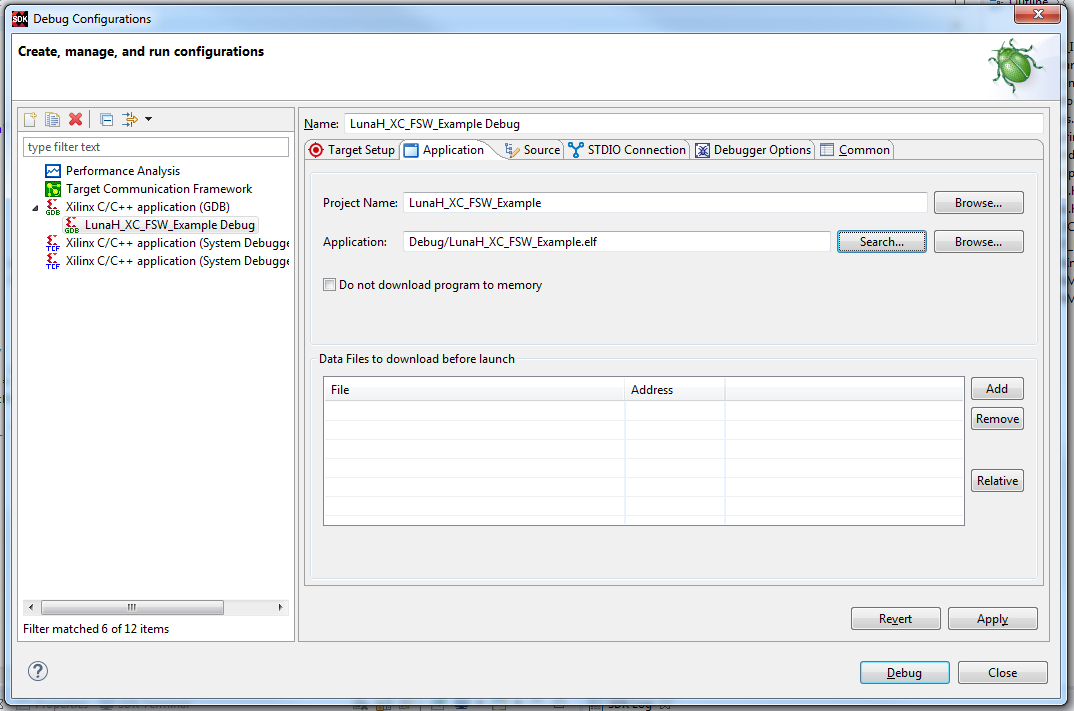
* 1. On the right side of the window is the allocation for the Stack and Heap. The default is 1024 (1 kB), change that number to 1048576 (1 MB) and press Generate.

The software is now ready for running on a board. The following steps describe how to program the FPGA with the bitstream, create a GDB debug configuration, and run a debug session.

1. Check your physical setup around the board you’ll be running.
   1. Plug in the JTAG or Xilinx programming box to the computer and then to the board. If this is the first time that you’ve connected them to your computer, you may see Windows start installing drivers. Let this finish before proceeding.
   2. For a microZed board, make sure that the jumpers are set up for Cascaded boot mode.
   3. Hook up the serial connection between the computer and the board. For the microZed, this is a microUSB cable, for the RMD digital board, this means wiring up header J4 to a RS-422 to USB converter (or other setup).
   4. Power on the board. Depending on the previous configuration of the board, the “done” LED may or may not be illuminated. For the microZed, this is a blue LED, for the RMD digital board this is a green LED.
2. Build the project in debug configuration.
   1. Click on the “LunaH\_XC\_FSW\_Example” folder to select it
   2. Project > Build Configurations > Set Active > Debug
   3. Click the “Hammer” icon on the toolbar to build the project in debug
3. Program the FPGA
   1. Xilinx > Program FPGA
   2. The defaults should be fine; hit Program at the bottom.
   3. The “done” LED on the RMD digital board will turn off while the board is being programmed. It will re-illuminate when the process is finished. On the microZed board, the same thing will happen.
4. Create a debug configuration
   1. Run > Debug Configurations…
   2. Double click “Xilinx C/C++ application (GDB)”
      1. Sometimes the configuration will auto-populate the necessary fields, but we will go over them here.



* 1. Next to “Bitstream file:” hit the Search button and you should be presented with one option (the only bitstream in the project). Choose it and press OK.
  2. Do the same with the “Initialization file: “
  3. Set the pull down box under the “Initialization file: “ header to Reset Processor
  4. Tick the boxes next to:
     1. Run ps7\_init
     2. Run ps7\_post\_config
  5. In the “Application” tab click Browse next to “Project Name:” and choose the project name.
  6. Next to the “Application: “ header, hit the Search button and click on the file “LunaH\_XC\_FSW\_Example.elf” to select it. Then press OK.
     1. If nothing is shown in these boxes, return to step 13 to generate the debug files.



* 1. Click Debug to launch the session.

1. SDK will prompt you if you want to open the Debug perspective. I suggest doing so as it provides useful information about the debug session. Press yes to open it.
2. Open TeraTerm (or a serial terminal) and connect to the correct port.
   1. Ensure that the correct baud rate is set within your terminal session.
3. When the debug session is loaded and ready to be launched, the “Resume” button will turn green and the “Terminate” button will turn red. To begin, press the green button.
4. Output should display on your serial terminal.
   1. If the output is garbled, check the baud rate of the connection.
5. At any point within a debug session, pressing the yellow “Suspend” button will pause execution, allowing you to inspect the current variables or to step through a process.
6. Breakpoints can also be set by double clicking along the left hand side of the editor or by right clicking the same area and choosing Set Breakpoint. This is also how to turn on line numbering within the editor.
7. Pressing the Terminate button will stop execution of the debug session.

If, instead, we want to flash the EEPROM of a board, then a First Stage Boot Loader is necessary and there are different steps we will take to create a project which we will build as a Release verison. This will be handled in a later document.

# Appendix