snickerdoodle

DEVELOPMENT GETTING STARTED GUIDE (WINDOWS)

DECEMBER 30, 2015

How to Read this Document

This document makes extensive use of links, references and notices in the page margins to detail additional information that can be useful while following the guide.

WARNING A warning notice indicates a potential hazard. If care is not taken to adhere to the safety precautions, damage may be done to snickerdoodle.

Warnings and cautions will be clearly visible in either the body of the text or in the margin and must be paid close attention while following the guided steps.

CAUTION A caution indication denotes a process that requires special attention. If the caution is not exercised and the process not adhered to, failure may result and/or potential damage to snickerdoodle.

Warning, caution and informational notices, such as this one, may also be found in the margin.

Keywords

Keywords and important terms are shown in *italicized* type. Additional important information can be found in the margins of text with superscript notation[1].

[1] Margin notes, such as this one, reference the body content and highlight technical details or references for further information.

Navigation of menus and directories are shown using ***bold italicized*** type. Any hierarchical navigation is shown using an arrow to denote a ***Parent*** → ***child*** relationship.

`Teletype` text is used to highlight inputs, variables and system files within the host environment.

Introduction

This guide assumes that you have already installed the Xilinx SDK which can be done by following the snickerdoodle "Development Environment Setup" guide. This guide uses the Xilinx SDK 2015.2 installed on Windows 10.

The SDK has an extensive built-in user guide which provides assistance on many common tasks such as creating and building projects and using the system debugger. The user guide can be found by navigating to ***Help*** → ***Help Contents*** from the menubar. This will launch the user guide in a web browser for navigation.

> *Because the SDK is Eclipse-based, any features and customizations available to common Eclipse distributions can be used in the IDE. Additionally, resources and knowledge bases for Eclipse can be leveraged for information on navigation and settings of the IDE.*

Launching the SDK

If the parent directory for the SDK has been added to the `$PATH` variable, then the SDK can be run by exectuting the `xsdk` command from a terminal:

```
user@ubuntu:~$ xsdk
```

After starting the SDK environment, you will be prompted to select a workspace in which to store application projects and files. This dialog can be bypassed in the future by selecting the "Use this as the default..." checkbox.

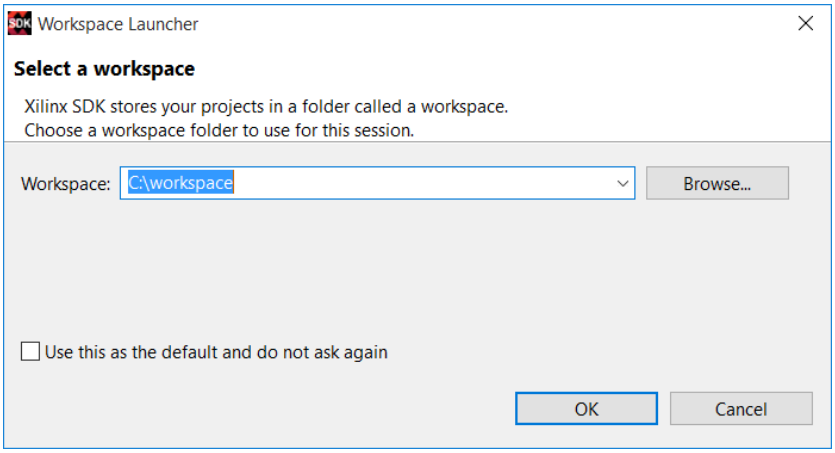

Figure 1: Selecting an SDK Workspace Path from the Workspace Launcher Dialog

Creating a Project

After selecting a workspace, the SDK environment will open and you will be able to create new projects. To create a new Linux project, start by navigating

to ***File → New → Application Project*** from the menubar or ***New → Application Project*** from the toolbar as shown in Figure 2 .

From the "New Project" dialog, a project type can be selected. This will open the wizard for the selected project type, in this case an *Application Project*. "Application Project" can be selected from the "Xilinx" folder. This will create a pre-configured project, ready to be cross-compiled using the Xilinx compiler toolchain and managed by the SDK.

From within the *Application Project Wizard*, the project can be named and a project type selected. For a Linux application, the OS Platform should be changed to "linux." The "Linux System Root" and "Linux Toolchain" do not need to be changed to build Linux applications for snickerdoodle.

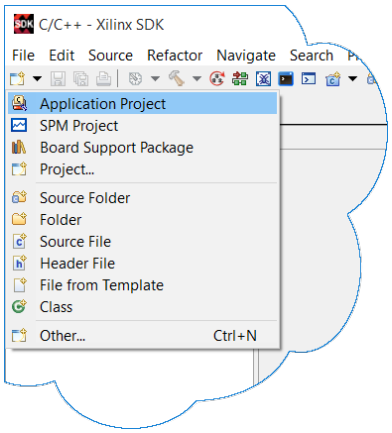

Figure 2: Starting a New Project from the Toolbar

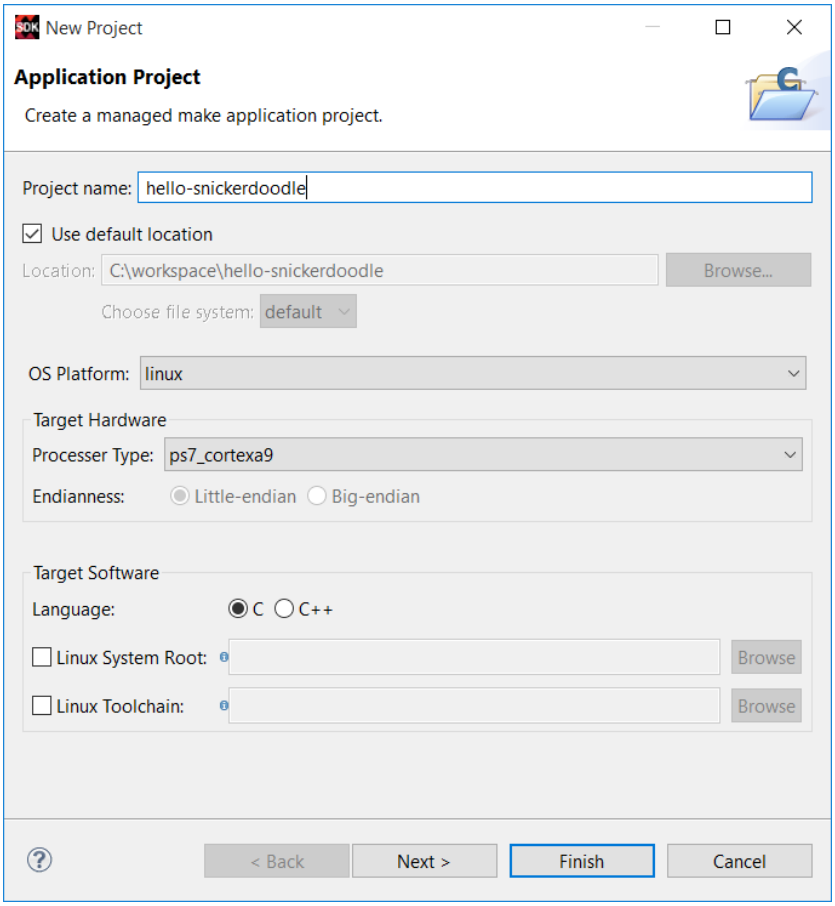

Figure 3: Creating a Linux Application in the Application Project Wizard

To target a Linux system booting on the Cortex-A9 processor, select `ps7_cortexa9` from the *Processor Type* drop-down menu. Other options include `microblaze` for systems running on microblaze processors that have been synthesized in

programmable logic.

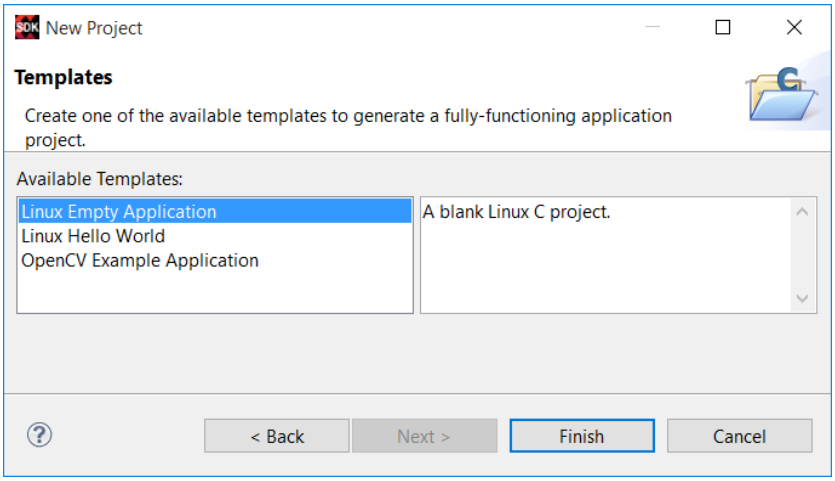

Figure 4: Finishing Application Project Generation by Selecting a Project Template

After selecting ***Next*** from the *Application Project Wizard*, a template for the application can be selected. For this example, the application project will be left empty by selecting the "Linux Empty Application" template. This will create a C project without any source files.

Clicking ***Finish*** will generate the application project and make it available in the SDK environment's project explorer.

Create Project Source Files

After the project is created, it can be populated with project files. The simplest case is a single file program. In this example, a single file named `main.c` is created and populated with the program's entry point function, `main()`.

Creation of new files can be done by right-clicking on the desired parent directory or selecting ***File*** from the menubar, and selecting ***New*** → ***Source File***. Source files can also be created, as shown in Figure 5 , by selecting ***New*** → ***Source File*** from the toolbar.

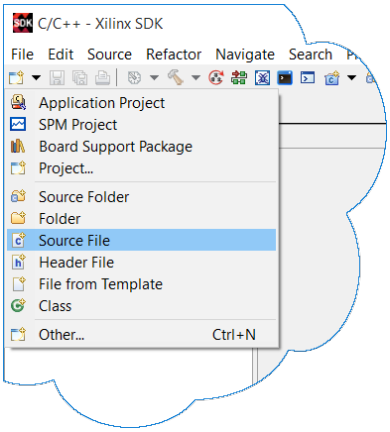

Figure 5: Creating a New Source File from the Toolbar

> *Additional information on configuring and customizing code templates can be found at* `http://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Freference%2Fcdt_u_c_code_templates_pref.htm`

From the ***New File*** dialog, the file name (including extension), parent directory and source file template can be selected. Templates can be configured and customized to include common file elements such as headers and comments.

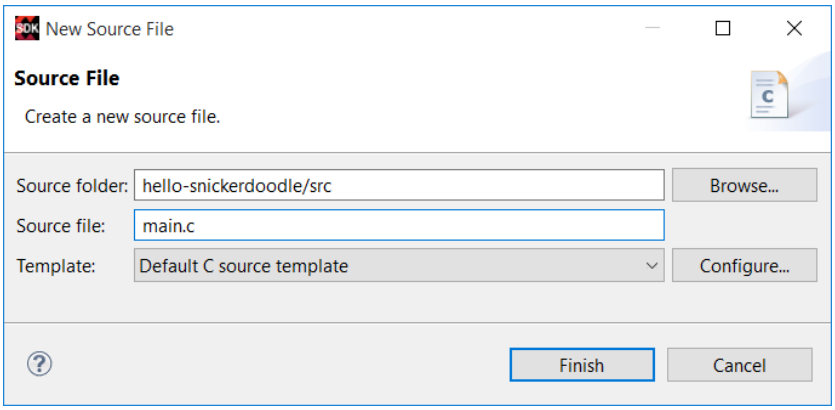

Figure 6: Selecting the File Name and File Template

Writing Source Files

The structure of applications for snickerdoodle takes the familiar form of any C/C++ Linux application. Project management and file structure will be recognizable to any user with some experience with C/C++ programming in an Eclipse environment. For this example, a simple "hello world" style program is generated by populating the `main.c` file with the following code:

```
/*
 *  File:  main.c
 */

#include <stdio.h>
#include <stdlib.h>

int main( int argc, char * argv[] )
{
   printf( "Hello snickerdoodle!\n" );

   return EXIT_SUCCESS;
}
```

Building the Application

Cleaning and building the application can be done by selecting ***Project*** → ***Clean..*** which will as shown in Figure 7 .

The invocation of the compiler toolchain can be seen in the console of the IDE. After a successful build, the console should show messages similar to the

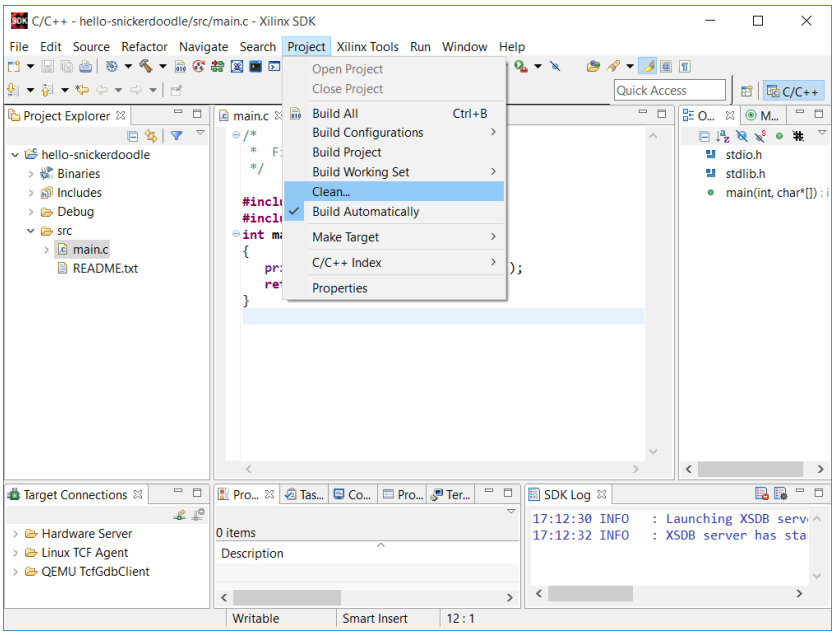

Figure 7: Clean and Build the Project from the Menubar

following:

```
17:33:01 **** Build of configuration Debug for project hello-snickerdoodle ****
make all
'Building file: ../src/main.c'
'Invoking: ARM Linux gcc compiler'
arm-xilinx-linux-gnueabi-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/main.o" -MMD -MP
    -MF"src/main.d" -MT"src/main.d" -o "src/main.o" "../src/main.c"
'Finished building: ../src/main.c'
' '
'Building target: hello-snickerdoodle.elf'
'Invoking: ARM Linux gcc linker'
arm-xilinx-linux-gnueabi-gcc -o "hello-snickerdoodle.elf" ./src/main.o
'Finished building target: hello-snickerdoodle.elf'
' '
'Invoking: ARM Linux Print Size'
arm-xilinx-linux-gnueabi-size hello-snickerdoodle.elf |tee "hello-snickerdoodle.elf.size"
   text    data     bss     dec     hex filename
   1254     292       4    1550     60e hello-snickerdoodle.elf
'Finished building: hello-snickerdoodle.elf.size'
' '

17:33:02 Build Finished (took 605ms)
```

At this point, the Linux application has been compiled and the executable can be copied to *ROOTFS*. The SD card can now be ejected and mounted on snickerdoodle where the system will be booted. The application can be executed from the command line of your booted snickerdoodle. Execution of the program

should look like the following:

```
user@snickerdoodle~$ hello-snickerdoodle
Hello snickerdoodle!
```