

Callyn Villanueva

Task 1: Integrating a Makefile into Code Composer Studio

Objective

The objective is to create and integrate a Makefile into a Code Composer Studio (CCS) project for efficient build management of a C/C++ application targeting the TM4C1294NCPDT microcontroller (i'm using the TM4C1294XL development board). I couldn't find anything online & decided to create this guideline for anyone who is using this particular board.

Steps Taken

1. **Project Setup:**
 - Created a new project in Code Composer Studio tailored for the TM4C1294NCPDT microcontroller. This setup included defining the necessary source files and project configuration.
2. **Makefile Creation:**
 - Developed a Makefile to automate the build process. The Makefile was structured to include:
 - **Variables:** Defined compiler settings, such as `CC` for the compiler and `CFLAGS` for the compiler flags.
 - **Targets and Dependencies:** Specified the target executable and the object files required for its creation, detailing how each target depends on its corresponding object files.

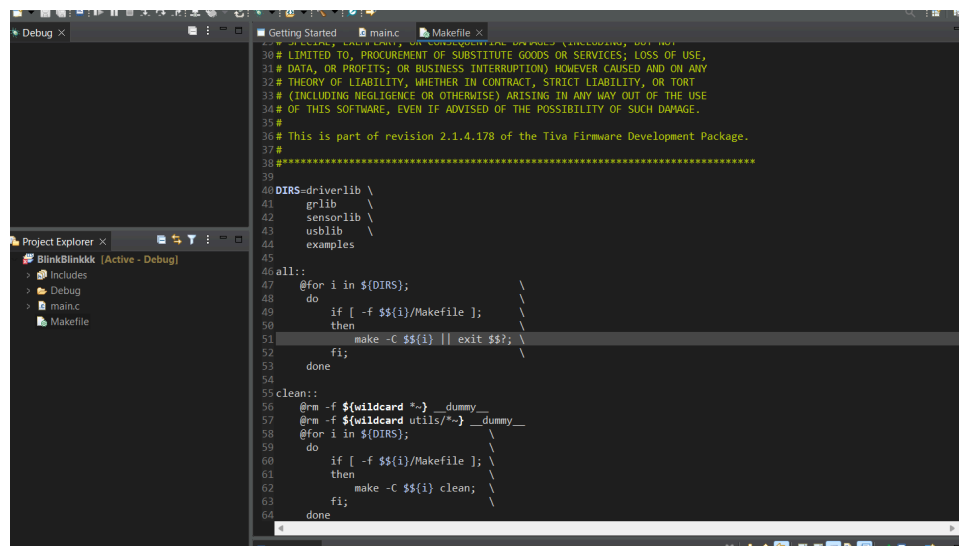
- **Build Rules:** Established rules for compiling object files from source files, enabling modular compilation.
- **Clean Rule:** Implemented a clean command to facilitate the removal of compiled artifacts, ensuring a clean build environment.

CC: Specifies the compiler to use. In this case, it's set to **gcc**. For CCS projects targeting ARM (like the TM4C1294NCPDT), you may want to change this to something like **arm-none-eabi-gcc**.

CFLAGS: Compiler flags. **-Wall** enables all warnings, and **-g** includes debugging information.

OBJ: Lists the object files that will be generated from your source files. Make sure these correspond to your actual source files.

TARGET: The name of the final executable that will be created.



```

30# LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
31# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
32# THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
33# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
34# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
35#
36# This is part of revision 2.1.4.178 of the Tiva Firmware Development Package.
37#
38#
39#
40DIRS=driverlib \
41    grlib \
42    sensorlib \
43    usblib \
44    examples
45
46all:
47    @for i in $(DIRS); \
48    do \
49        if [ -f $$i/Makefile ]; \
50        then \
51            make -C $$i || exit $$?; \
52        fi; \
53    done
54
55clean:
56    @rm -f ${wildcard *}__dummy__
57    @rm -f ${wildcard utils/*}__dummy__
58    @for i in $(DIRS); \
59    do \
60        if [ -f $$i/Makefile ]; \
61        then \
62            make -C $$i clean; \
63        fi; \
64    done
  
```

3. Importing the Makefile into CCS:

- Created a new file named **Makefile** within the CCS project directory.
- Copied the contents from the previously created Makefile text file and pasted them into the new **Makefile** in CCS.
- Saved the changes, ensuring that the Makefile was correctly formatted and free of syntax errors.

4. Project Configuration in CCS:

- Accessed the project properties by right-clicking on the project in the Project Explorer.
- Navigated to **Build > Builder** and configured CCS to use the Makefile instead of the default build system:

- Unchecked the default build option and selected **Make** as the build method.
- Set the build command to `make` and defined the clean command as `make clean`.

- Ensured that any necessary include paths and library references were correctly specified within the Makefile.

5. **Building the Project:**

- Initiated the build process by selecting **Project > Build Project** in the CCS menu.
- Verified that CCS utilized the Makefile correctly, leading to successful compilation of the project.

6. **Testing and Debugging:**

- After building, proceeded to test and debug the application using the standard CCS debugging tools, confirming the successful integration of the Makefile.