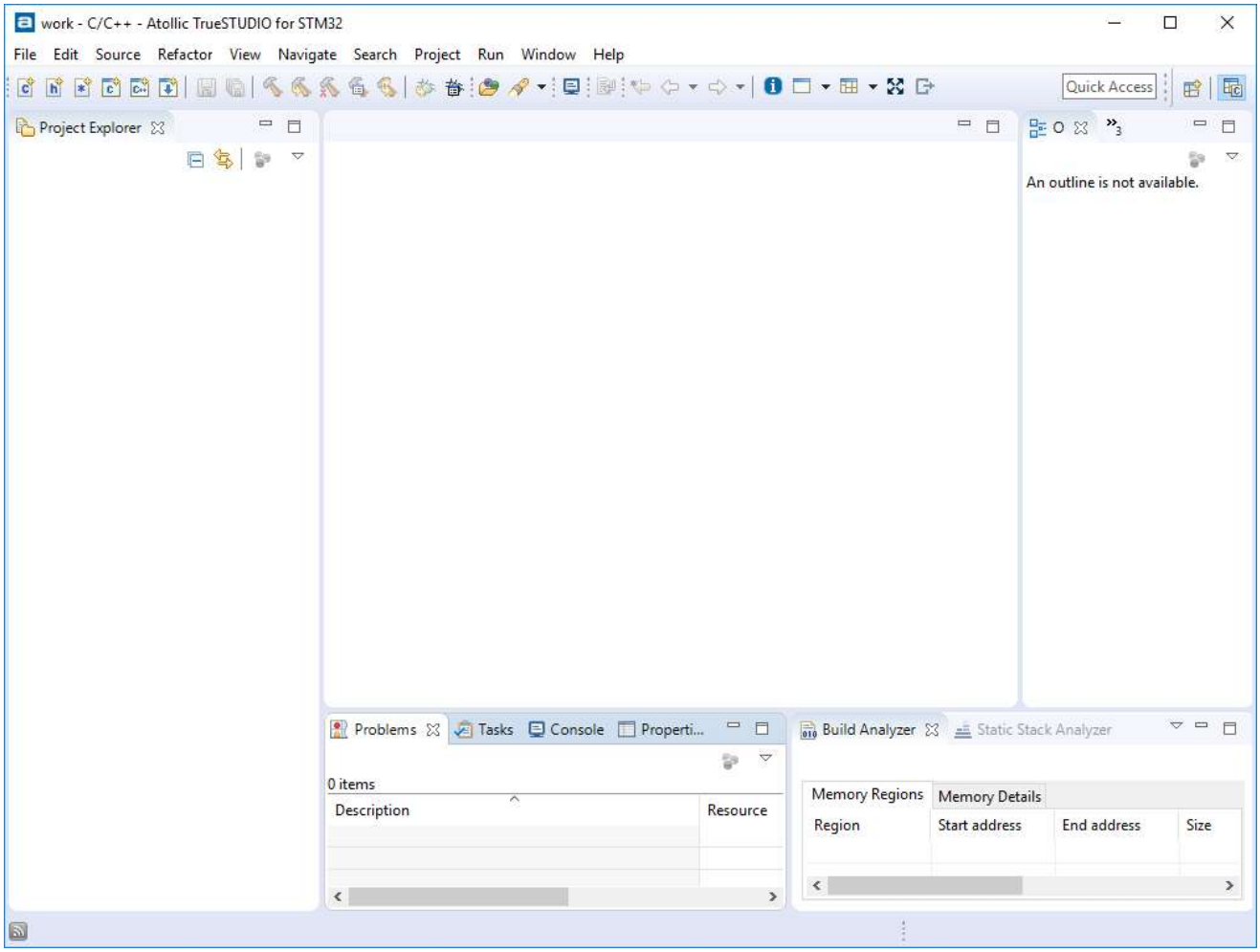
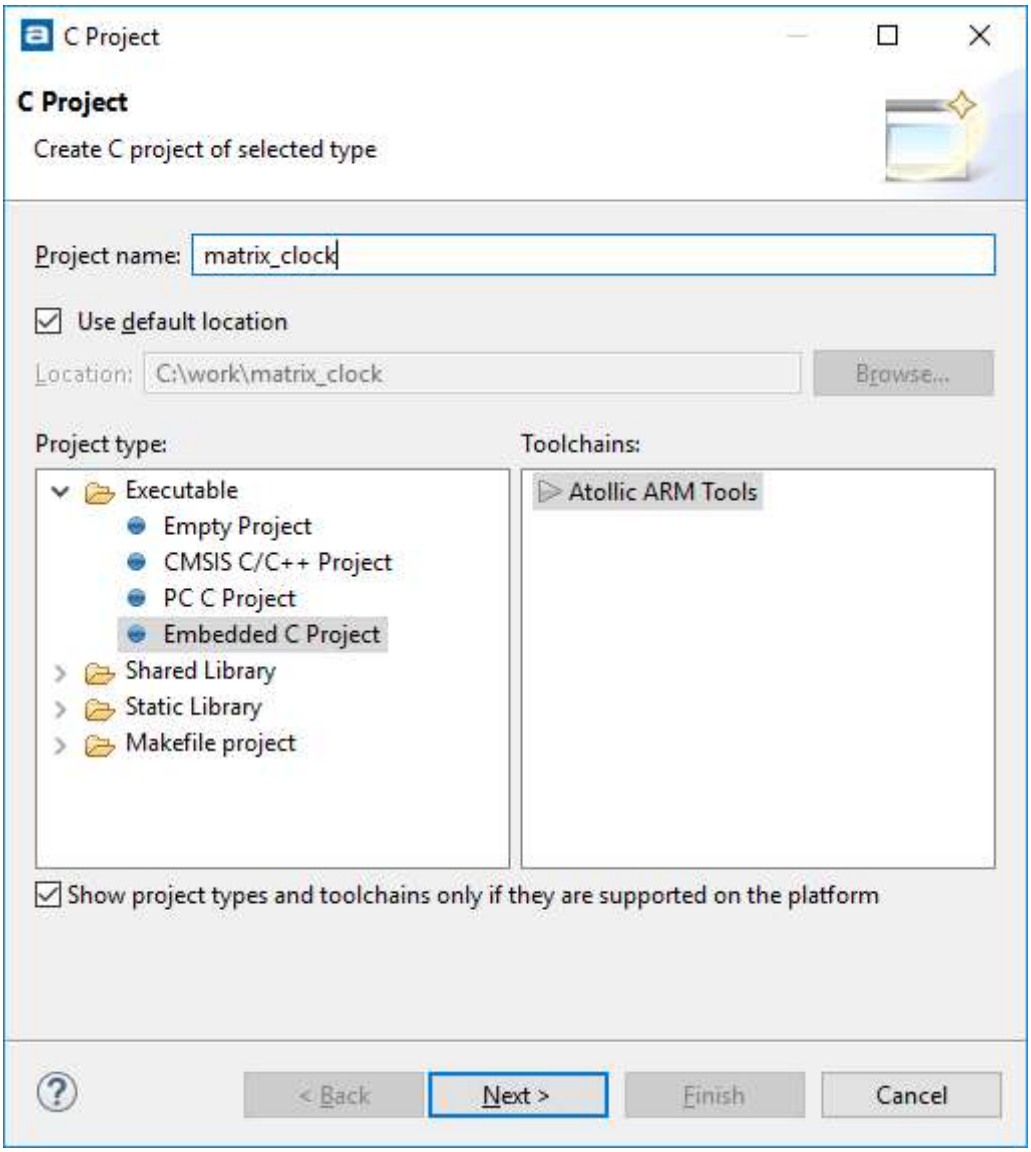
Atollic TrueStudio

> До недавнего времени у компании ST не было собственной IDE, однако в конце 2017 года она приобрела Atollic TrueStudio. Среда построена на основе Eclipse, и предлагает довольно много возможностей.
>
> Так как это официальная среда для разработки, в курсе мы сделаем упор на неё.

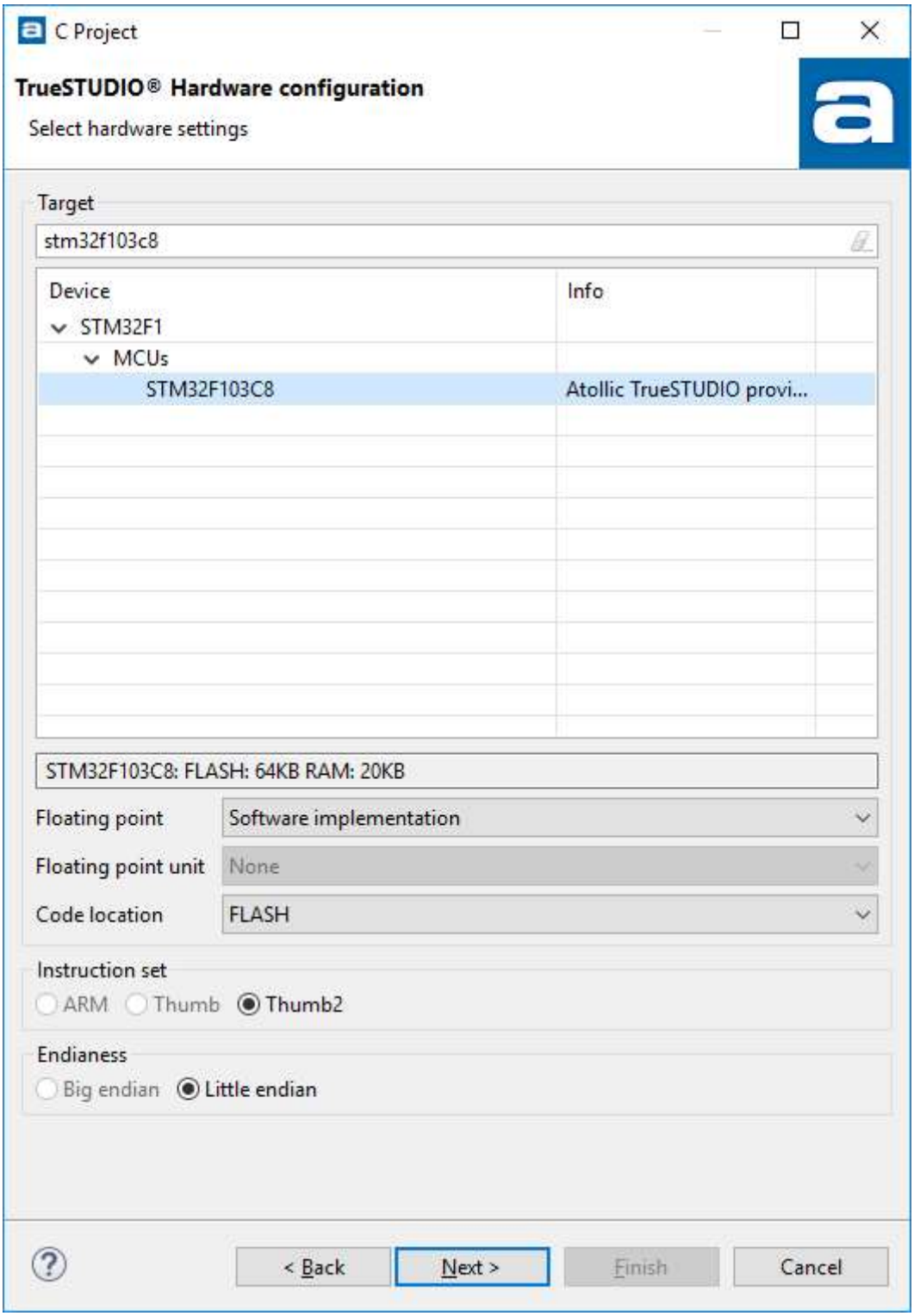
При запуске среды Atollic TrueStudio предложет выбрать рабочее окружение (англ. workplace). Согласитесь со стандартным расположением, в прочем потом вы можете создать другую и переключаться между ними. Рабочее окружение позволяет группировать ваши проекты. Пока у нас нет никаких проектов, давайте создадим один.



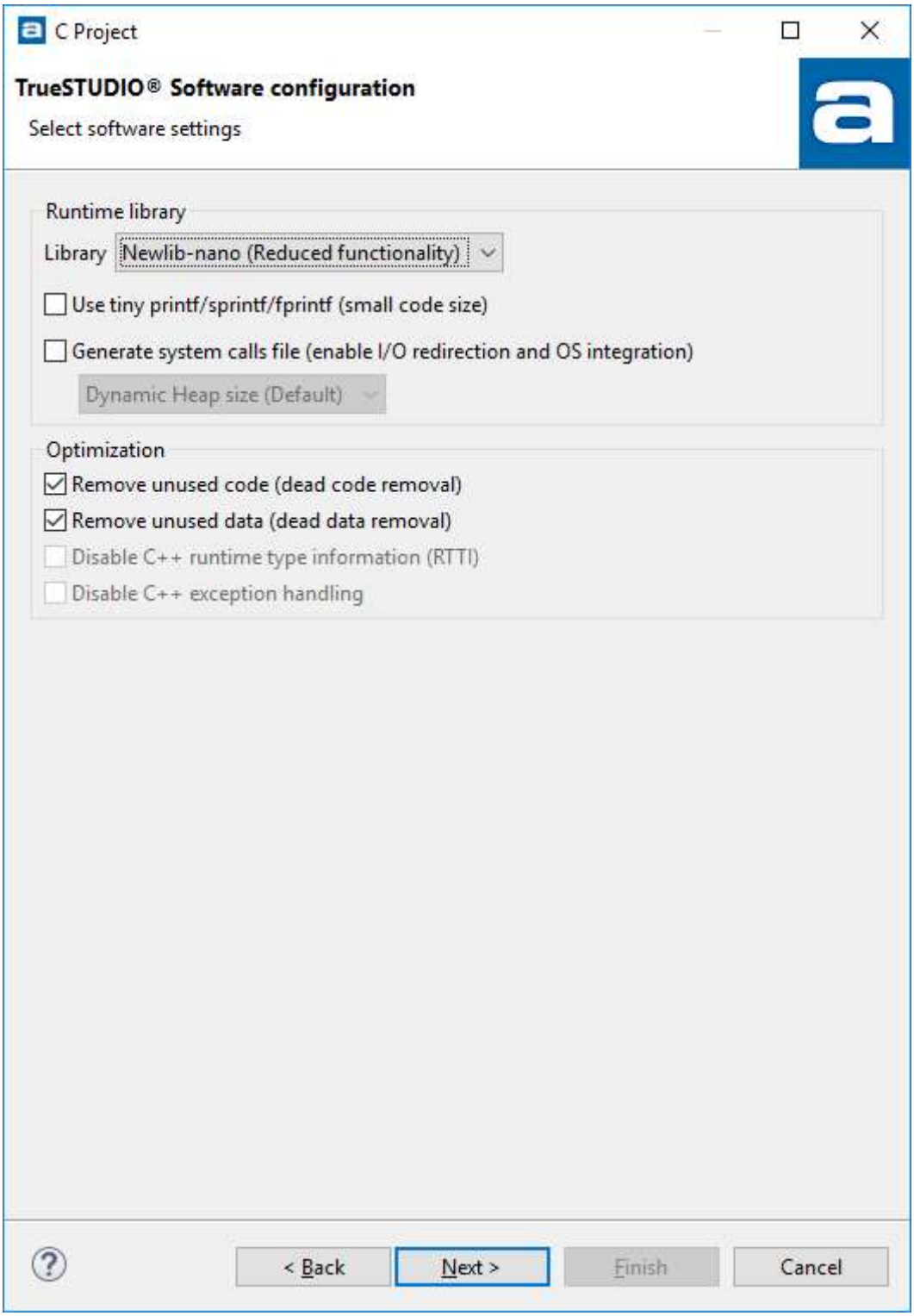
Зайдите в меню **File** ⇒ **New** ⇒ **C Project**. Выбирите **Embedded C Project** и назовите проект в поле **Project Name**, скажем `matrix_clock` . Нажмите **Next**.



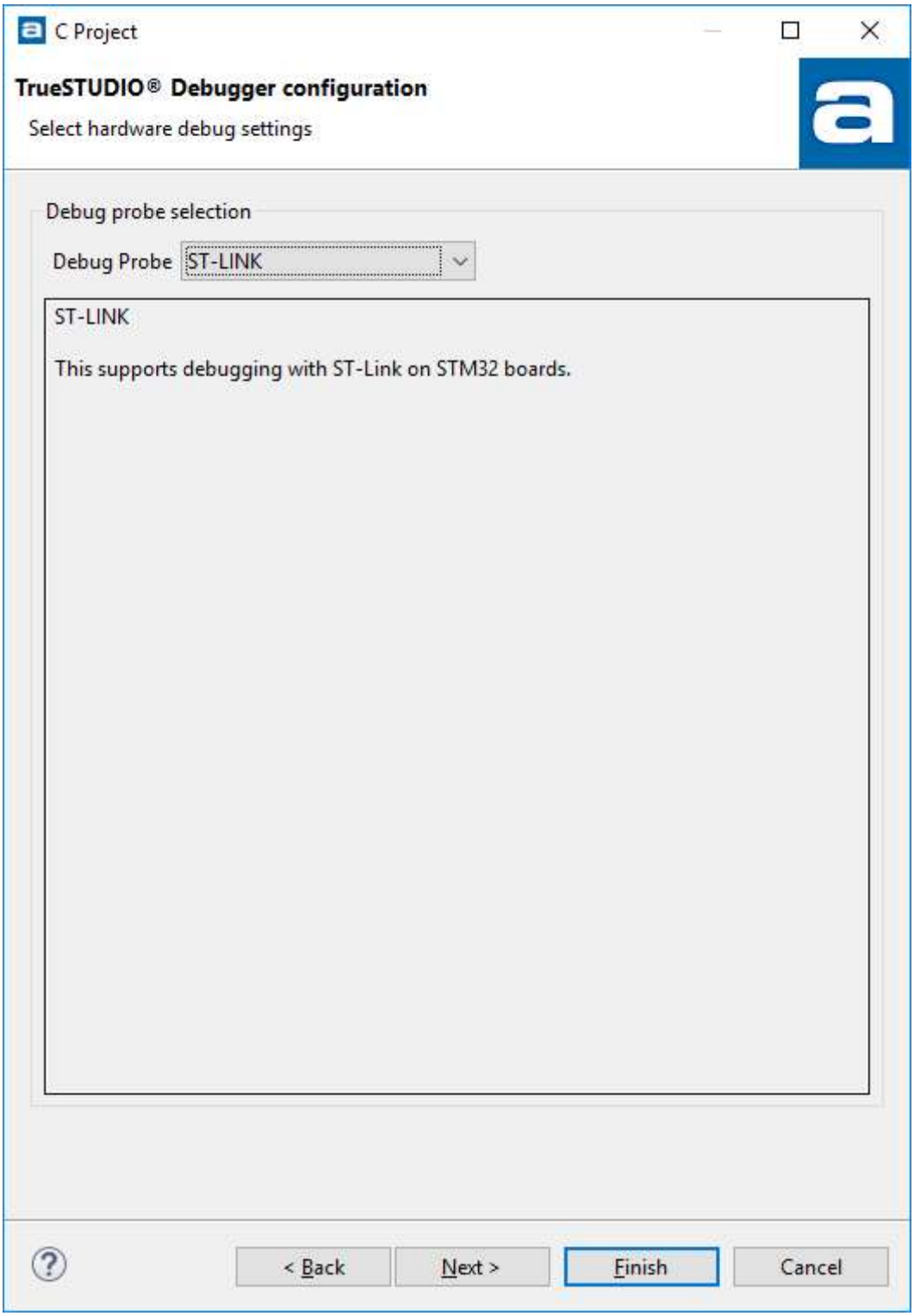
Далее следует выбрать целевой микроконтроллер. Дело в том, что каждый имеет определённый набор периферии, различный объём памяти и т.д. Библиотеке, линкеру и компилятору нужно знать эти памаметры, чтобы ваш проект успешно собирался и его можно было заливать на устройство. Мы используем **stm32f103c8**. Нажмите **Next**.



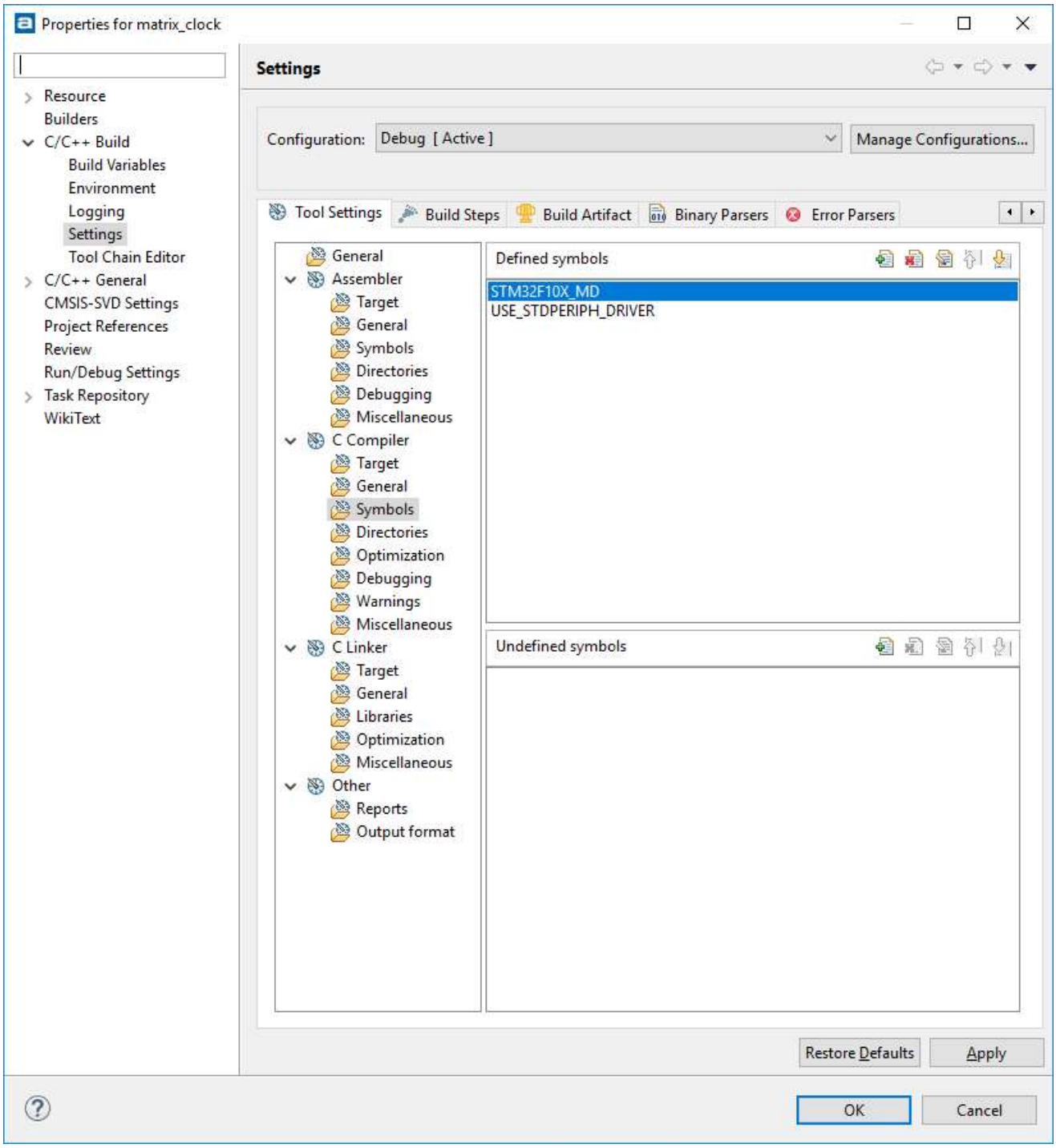
Уберите галочку с **Use tiny printf/sprintf/fprint (small code size)**, мы не будем пользоваться этими функциями. Нажмите **Next**.



В качестве отладчика нужно выбрать **ST-Link**. Нажмите **Finish**.



Проект создан, но давайте зайдём в настройки и кое-что там посмотрим и поменяем. Открыть настройки проекта можно либо кликнув правой кнопкой в **Project Explorer** и выбрав **Properties**, либо через меню **Project** ⇒ **Properties**. Откройте **C\C++ Builder** ⇒ **Settings**, во вкладке **Tool Settings** перейдите в **C Compiler** ⇒ **Symbols**. Здесь определяются макросы-макркеры. В нашем случае указано, что используется стандартная библиотека периферии (`USE_STDPERIPH_DRIVER`) и микронконтроллер F1-серии Middle Destiny (`STM32F10X_MD`). Менять здесь ничего не нужно, но вы можете добавлять сюда свои макросы-маркеры.

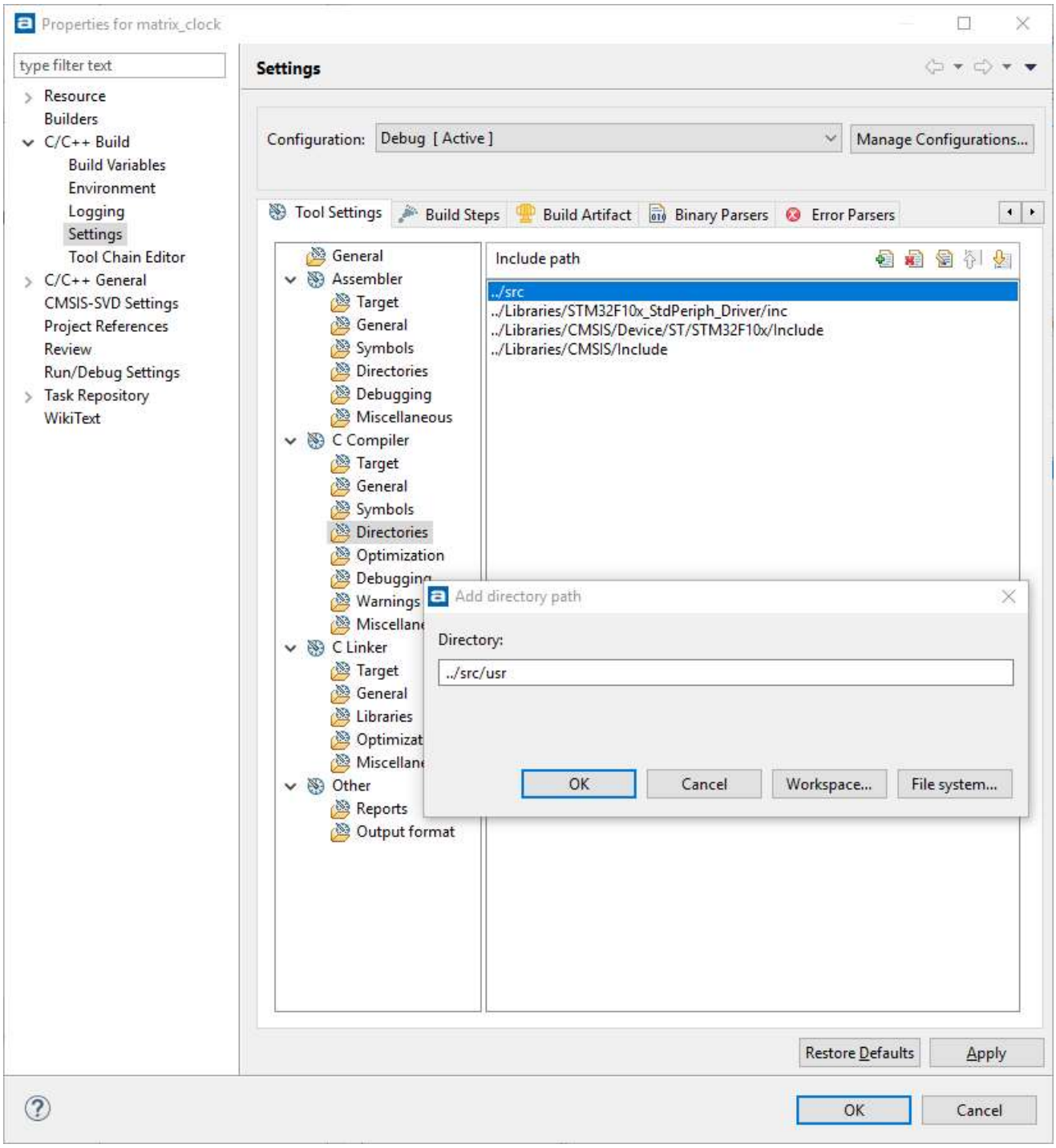


Перейдите в подменю **C Compiler** ⇒ **Directories**. Здесь находится список директорий, в которых среда будет искать подключаемые файлы. Конечно все файлы можно расположить в папке `src`, но это не будет удобно при написании прошивки. Скачайте из репозитория на GitHub папку `template` и скопируйте из неё папку `usr` в директорию вашего проекта, в папку `src`. После этого все файлы можно будет подключать к проекту так:

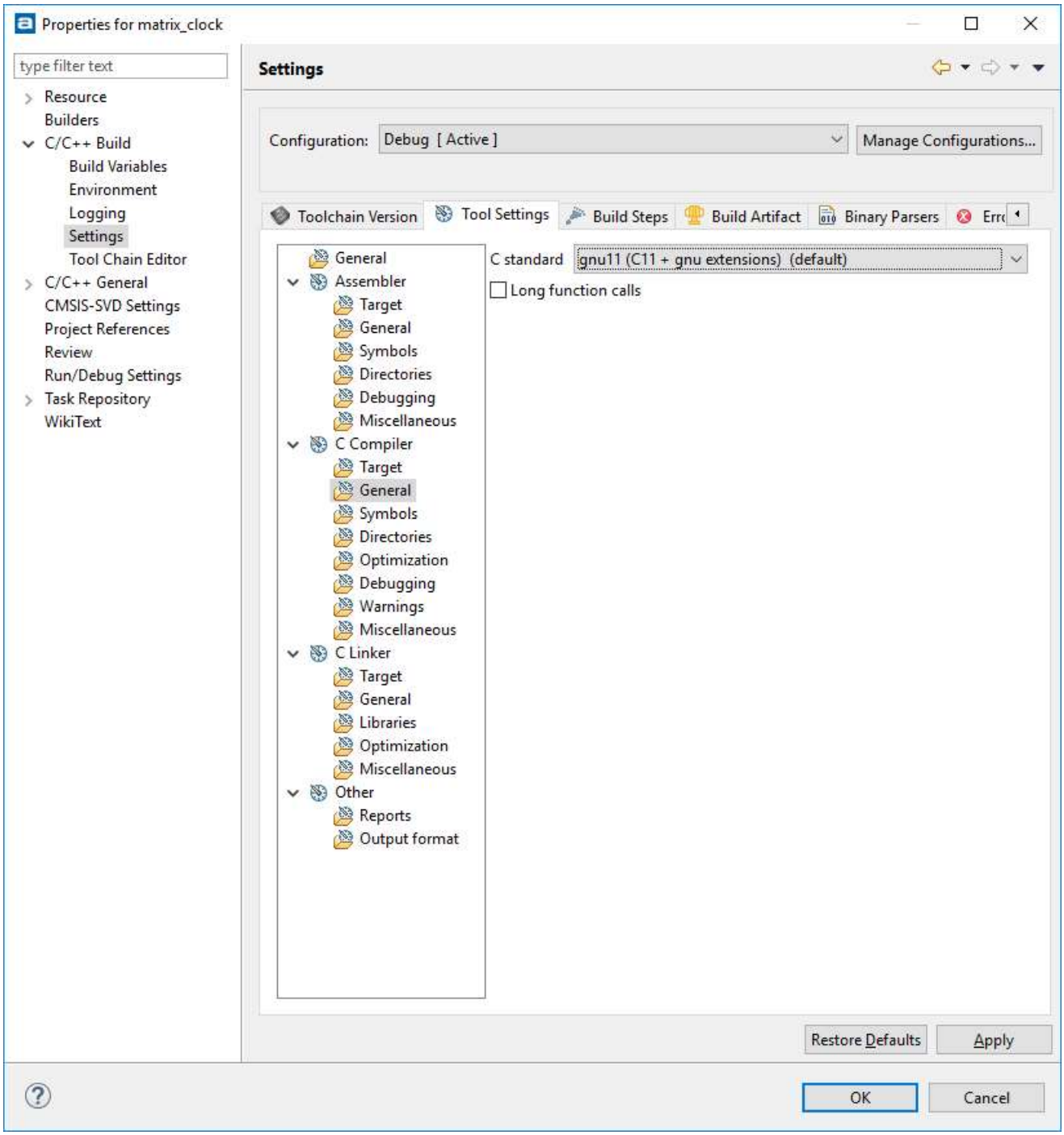
```
#include "usr/button.h"
```

Что бы избавиться от необходимости указывать поддиректорию, добавьте `usr` в список папок среды. После этого все файлы можно будет подключать так:

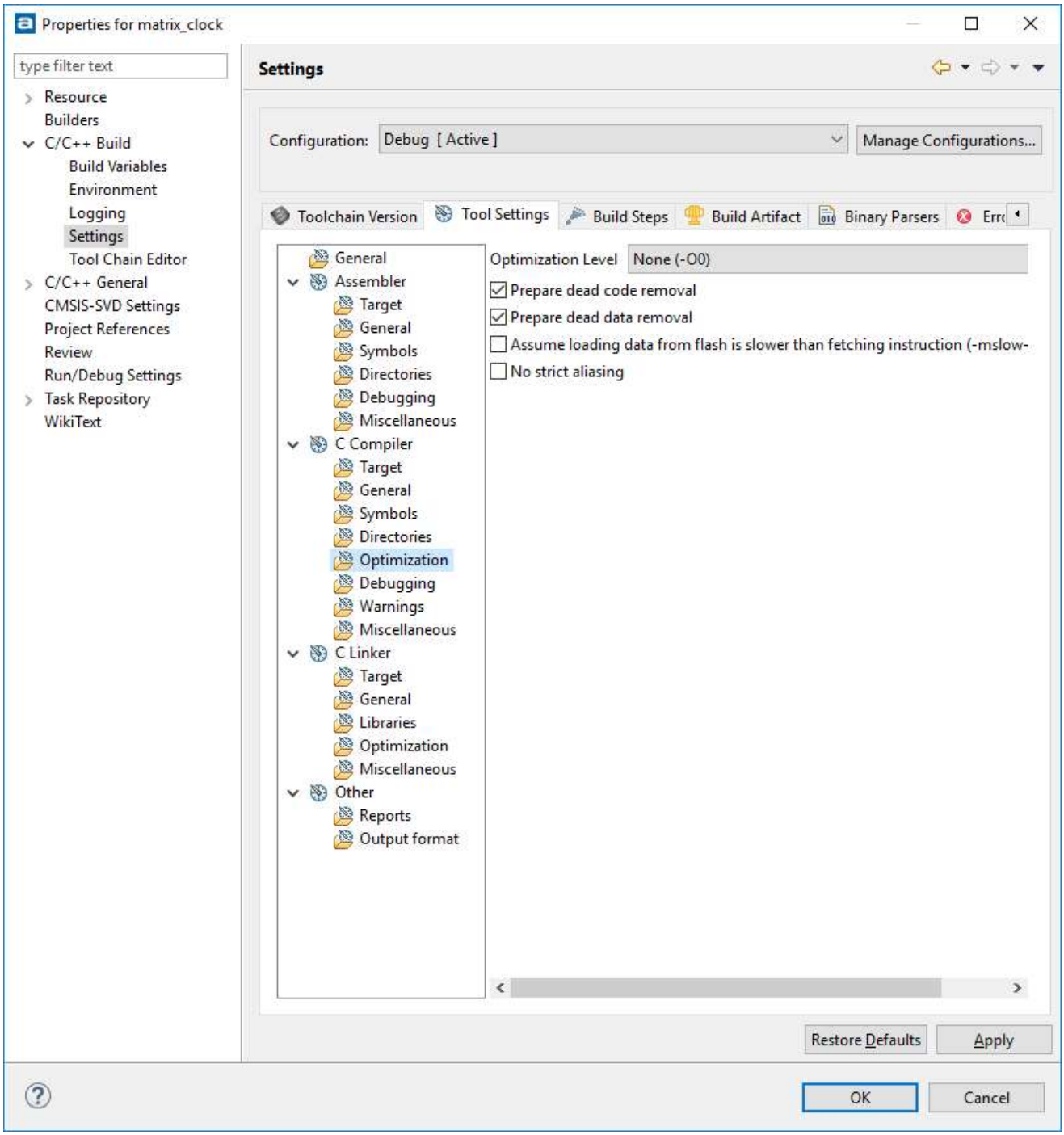
```
#include "button.h"
```



У языка Си существует несколько стандартов. Мы будем использовать последний `c11` (`gnu11`). Если вам нужно поменять стандарт, сделать это можно через **C Compiler** ⇒ **General**.



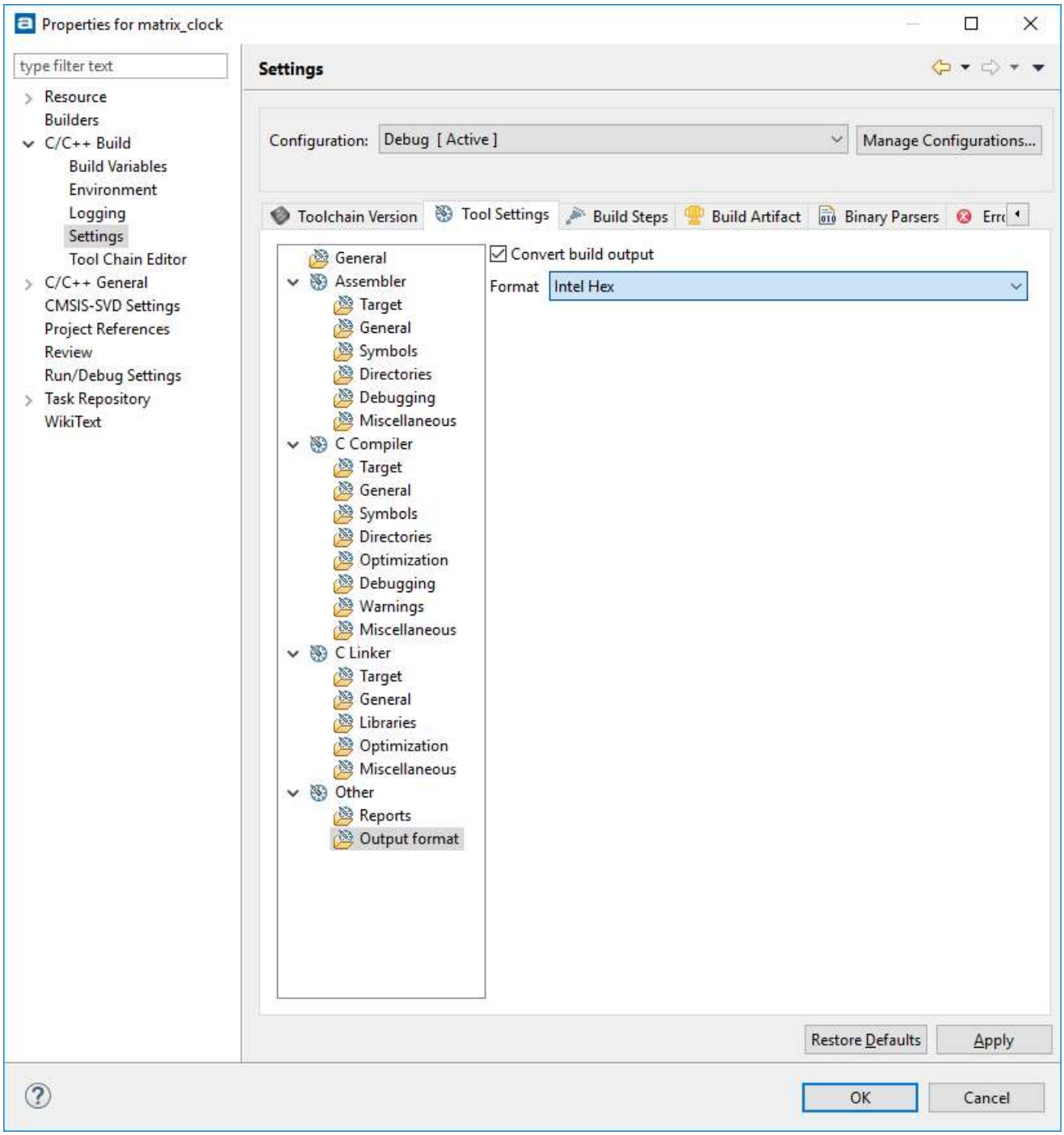
Мы уже говорили, что компилятор может оптимизировать ваш код. По-умолчанию оптимизация отключена `-O0`. Для MVP-прошивки включать её не стоит, это лишь осложнит процесс отладки. А вот если функциональность вашего устройство многократно усложняется и памяти не хватает, то имеет смысл включить оптимизацию по размеру `-Os`. Получить доступ к этой опции можно через **C Compiler** ⇒ **Optimization**. Сейчас здесь ничего менять не нужно.



Последнее о чём нужно сказать, это то, как сгенерировать файл прошивки, который будет понятен утилите для прошивки. Что бы включить генерацию hex-файла, зайдите **Other** ⇒ **Output format** и поставьте галочку **Convert build output**, выбрав **Intel Hex**.

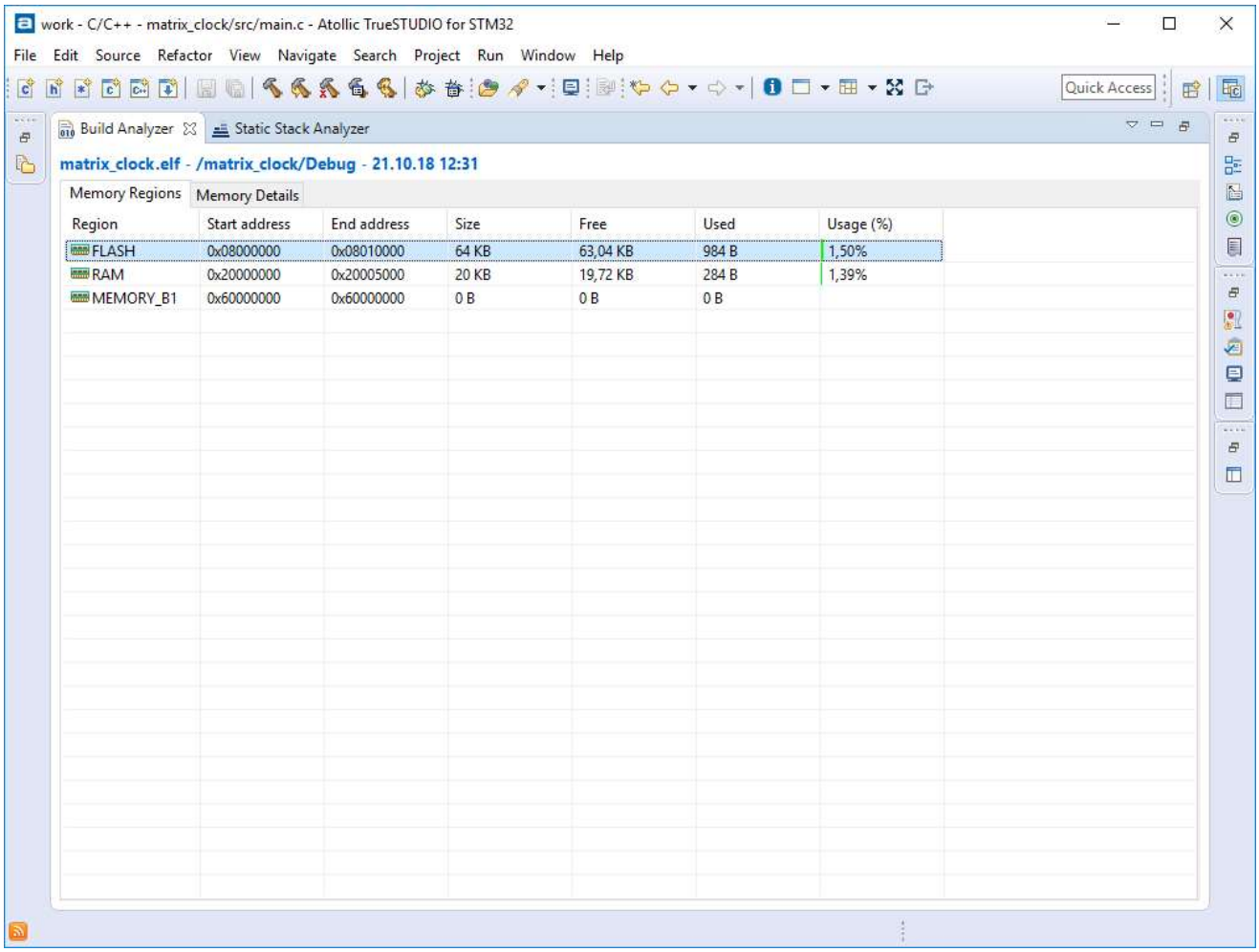
После сборки проекта hex-файл появится в директории:

```
%PROJECT%\Debug\
```



Говоря о размере прошивки, в Atollic TrueStudio есть специальное окошко **Build Analyzer**, которое по-умолчанию расположено в нижнем правом углу. В нём отображается получившийся размер прошивки, а так же объём занимаемой оперативной памяти.

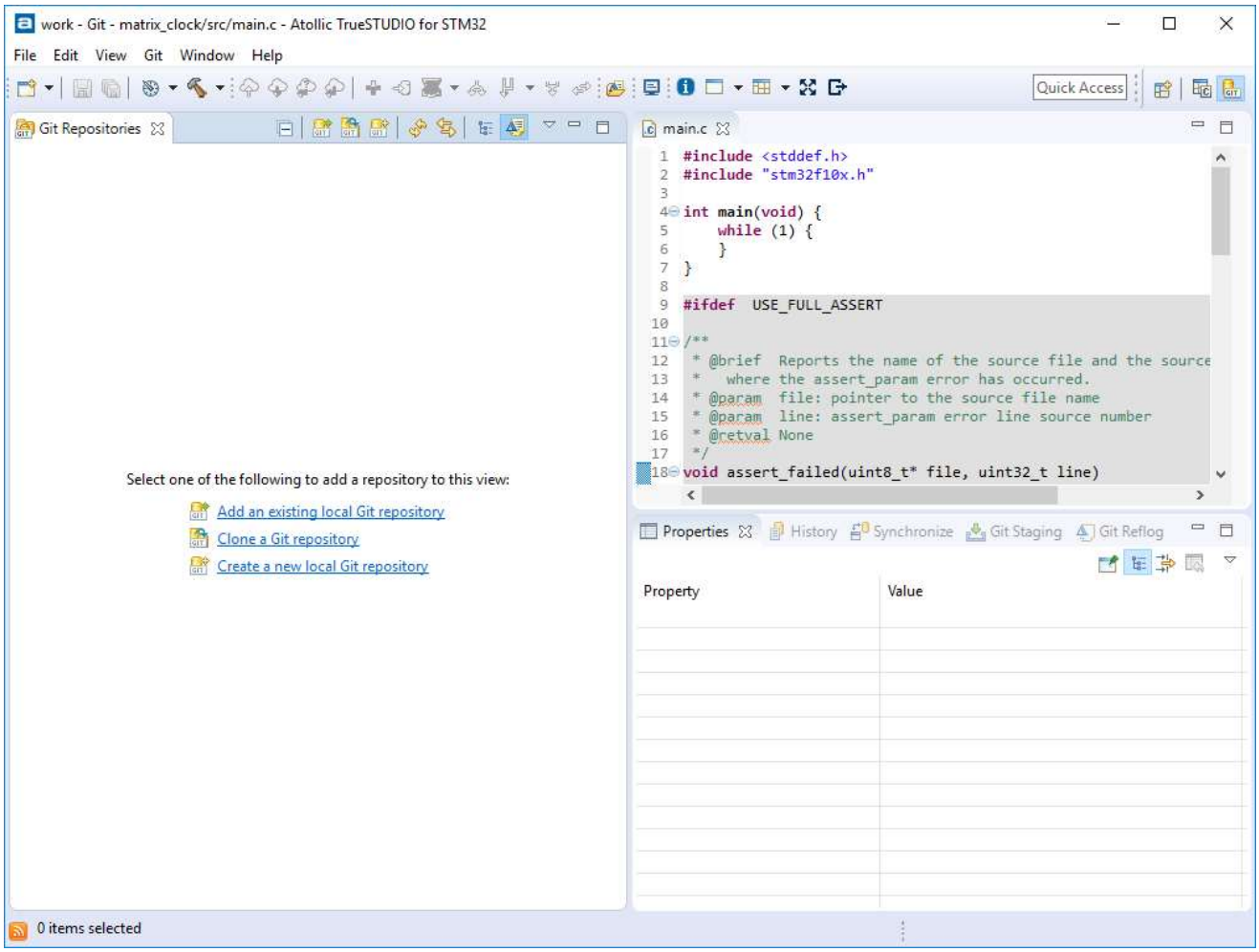
Любое окно внутри среды можно развернуть на полный экран, просто два раза щёлкнув по нему. Вернуть всё в исходный вид можно таким же образом.



В Eclipse есть такое понятие как перспектива (англ. perspective). Они созданы для удобстра разработки и позволяют сгруппировать необходимые окна на экране. По-умолчанию включены только два: **C\C++** и **Debug**. Первое вы видели после создания проекта, в нём удобно редактировать код, компилировать его и загружать на устройство. Второй **Debug**, мы рассмотрим чуть ниже.

Сейчас сложно себе представить разработку без системы контроля версий. В Atollic TrueStudio присутствует специальная перспектива для работы в графическом режиме со средой Git. Активировтаь её можно перейдя в меню **View** ⇒ **Open Perspective** ⇒ **Git**. Здесь вы можете создавать новые репозитории, комиттить, пушить и т.д.

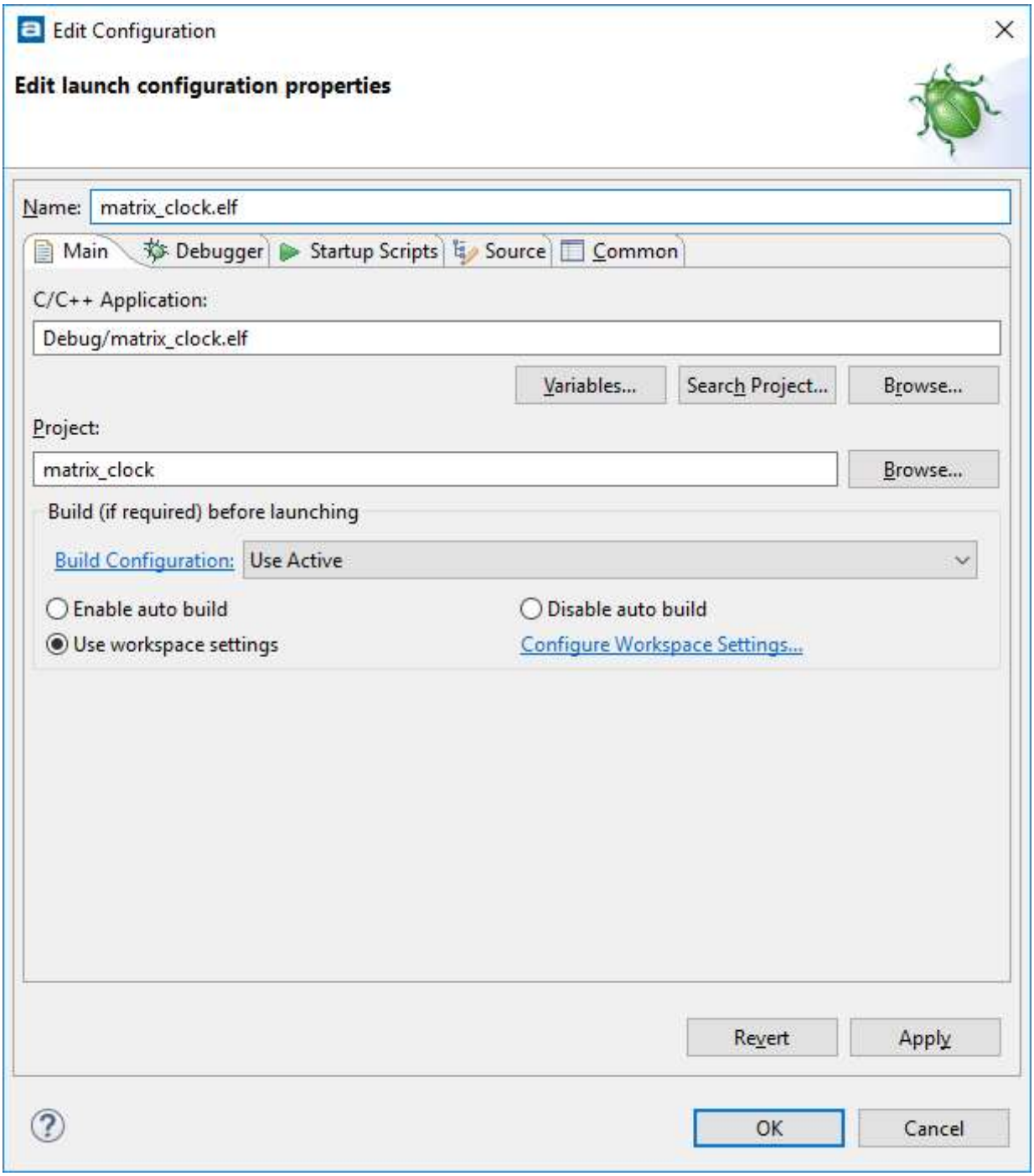
Если вы не знакомы с `git`, посетите его официальный сайт, там имеетс яцелая книга на русском языке посвящённая его работе, а так же обоснование зачем он нужен.



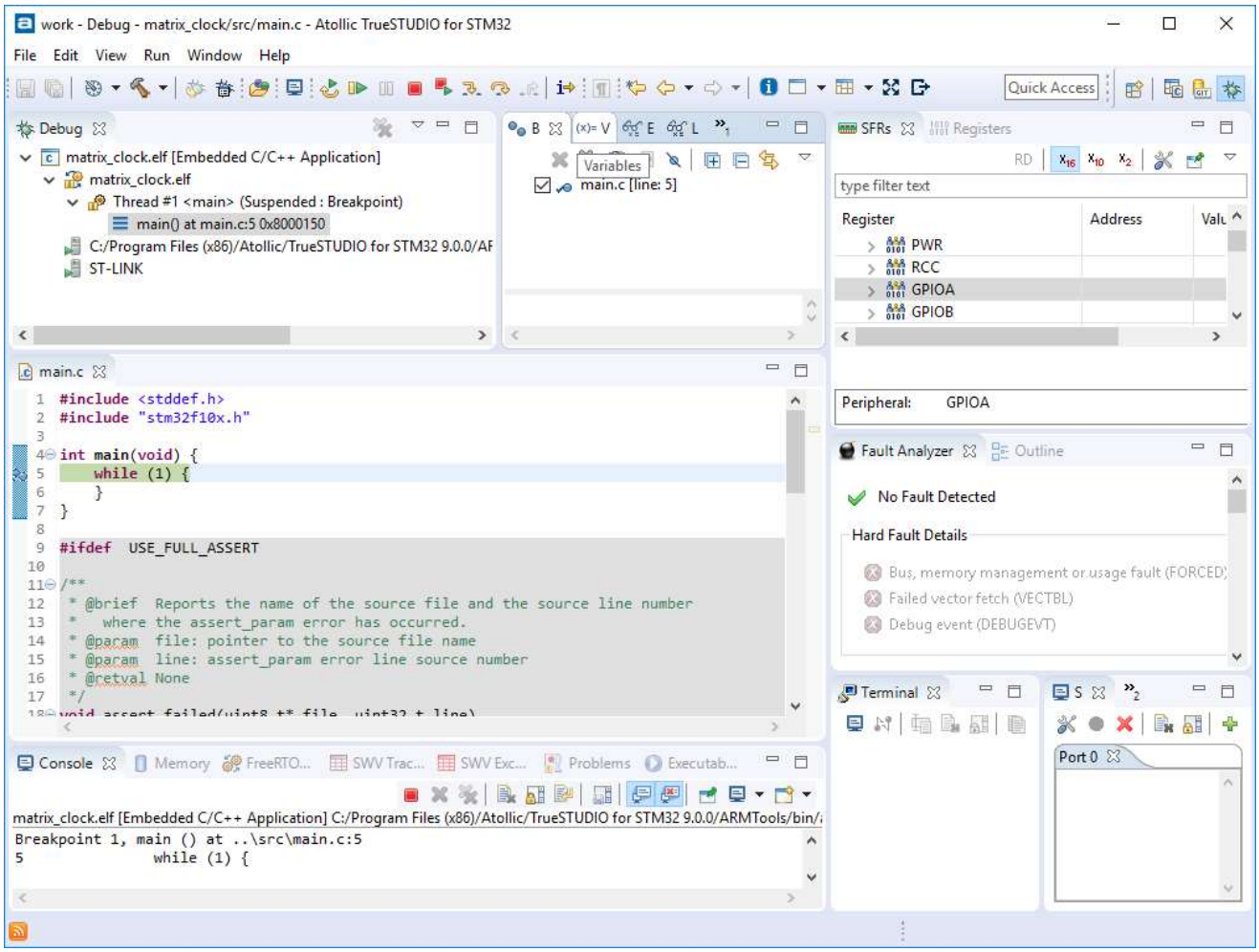
В отличии от простого блокнота, IDE обычно позволяет работать в режиме отладки (англ. debug). Вы можете выполнять программу построчно, попутно изучая состояние регистров микроконтроллера и видя значение переменных.

Для быстрого доступа к определённой точке в коде нужно поставить точку остановки (англ. breakpoint). В среде это делается довольно просто: совершите двойной клик левой кнопкой мыши в области с номером строки и сбоку появится синий кружок.

При первом запуске программы среда спросит вас сконфигурировать процесс загрузки прошивки. Здесь ничего менять не нужно. Нажмите **OK**.



Далее среда автоматически переключит вас в режим отладки, со всеми необходимыми инструментами. В верхнем инструментальном меню (англ. toolbar) появятся дополнительные кнопки, позволяющие приостанавливать работу программы, пыполнять пошаговое выполнение и т.д. Красный квадрат (стоп), позволит вам выйти из режима отладки и среда автоматически переключится в режим написания кода (**C\C++**).



Отлаживать программу добавляя в код `printf()` из стандартной библиотеки очень плохая практика, особенно для встраиваемых систем. Функция достаточно тяжёлая и приводит к раздутию кода и замедлению работы программы. Однако, отладчик позволяет создавать динамические `printf()`, которые не встраиваются непосредственно в программу. В Atollic вы можете добавить его щёлкнув правой кнопкой напротив желаемой строки и выбрать пункт **Add dynamic printf...** и передать ей желаемый аргумент, напримпер так:

```
"values is %d", value
```

Данная среда, в отличии от IAR System Workbench доступна под Linux. В последнй версии Ubuntu совершён переход на GTK3, по этой причине программа будет запускаться не корректно, в частности верхнее инструментальное меню не будет видно. Чтобы исправить это, перед запуском следует отключить GTK3.

```
export SWT_GTK3=0
```

Запускать среду из консоли не очень удобно, поэтому лучше отредактировать ярлык запуска.

```
#!/usr/bin/env xdg-open

[Desktop Entry]
Version=1.0
Type=Application
Terminal=false
Exec=sh -c "export SWT_GTK3=0 && /opt/Atollic_TrueSTUDIO_for_STM32_x86_64_9.0.1/ide/TrueSTUDIO"
Name=Atollic TrueStudio
Comment=Start Atollic IDE
Icon=/opt/Atollic_TrueSTUDIO_for_STM32_x86_64_9.0.1/ide/TrueSTUDIO.ico
Name[ru]=Atollic TrueStudio
```

Назад | Оглавление | Дальше