

Toradex Linux C言語アプリケーション 開発マニュアル BSP5用

本マニュアルは岡本無線電機株式会社が独自作成したものでありメーカーが保証した内容ではありません。万が一本マニュアルに間違いがあり、事故が生じたとしても岡本無線電機株式会社は一切の責任を問われないものとさせていただきます。

本マニュアルについて

本マニュアルはトラデックスのCPUモジュール上で動作するC/C++言語アプリケーションを作成する手順を記述しています。

参考:

[http://developer.toradex.com/knowledge-base/board-support-package/openembedded-\(core\)](http://developer.toradex.com/knowledge-base/board-support-package/openembedded-(core))

<https://developer.toradex.com/knowledge-base/linux-sdks>

1. 実行環境

本マニュアルの実行環境は下記です。

仮想化ソフト: VMWARE Player v15.5.7

Host OS: Windows 10 21H2

Guest OS: Ubuntu Desktop 20.04LTS 64bit(英語版)

BSP: v5.7

CPUモジュール: Verdin iMX8M Plus Quad 4GB Wi-Fi / Bluetooth IT V1.1A

キャリアボード: Verdin開発ボード Rev 1.1C + アクセサリーキット

本マニュアルとは異なるモジュールや評価ボード以外のキャリアボードを使われても大雑把には同じ操作となります。

インターネット接続環境が必要になります。

2. 事前準備

本マニュアルはLinux OSイメージ開発マニュアルの内容をすべて終えた状態で進めています。

3. 前提知識

Linux OSイメージ開発マニュアルの内容、TEZIによるOS書き込みをご理解いただいた状態を前提としています。

4. 注意点

オープンソース系を利用した開発に共通することですがすべてを理解しようとするときりがなく開発効率を損ないます。必要なタイミングで必要な知識を身につけるというスタンスで理解することを推奨いたします。

開発環境と実行環境の違いをわかりやすくするためにコマンドの表記の前に下記をつけています。

開発環境(パソコン)上で入力するコマンド:[Ubuntu]\$

実行環境(モジュール)上で入力するコマンド(Linux) : [Module]#

実行環境(モジュール)上で入力するコマンド(U-Boot): [U-Boot]#

コピーについて

本マニュアル内のコマンドなどをコピーした場合、改行が入ったり「-」が抜けてしまうことがあるのでご注意ください。一度テキストエディタなどに張り付けてコピーした内容をご確認ください。

SDKの作成

C言語開発を行うためにOpen Embeddedでターゲットイメージ向けのSDKを作成します。

```
[Ubuntu]$ cd /work/oe-core
```

```
[Ubuntu]$ . export
```

```
[Ubuntu]$ bitbake -c populate_sdk tdx-reference-multimedia-image
```

/work/oe-core/build/deploy/sdk配下にSDKが出力されます。

実行してSDKを展開します。(モジュールやBSPのバージョンによってシェルの名前が変わります。)

```
[Ubuntu]$ /work/oe-core/build/deploy/sdk/tdx-xwayland-glibc-x86_64-Reference-Multimedia-Image-aarch64-verdin-imx8mp-toolchain-5.7.0.sh
```

下記のようにSDKのインストールディレクトリを問われるので入力します。

```
[Ubuntu]$ Enter target directory for SDK (default: /opt/tdx-xwayland/5.7.0):
```

本マニュアルではデフォルトの/opt/tdx-xwayland/5.7.0にインストールしますのでそのままEnterキーを押します。

下記のように問われますので

```
[Ubuntu]$ You are about to install the SDK to "/opt/tdx-xwayland/5.7.0". Proceed [Y/n]?
```

Yを入力してEnterキーを押します。

管理者権限で実行しますので途中でパスワードを問われます。パスワードを入力してください。

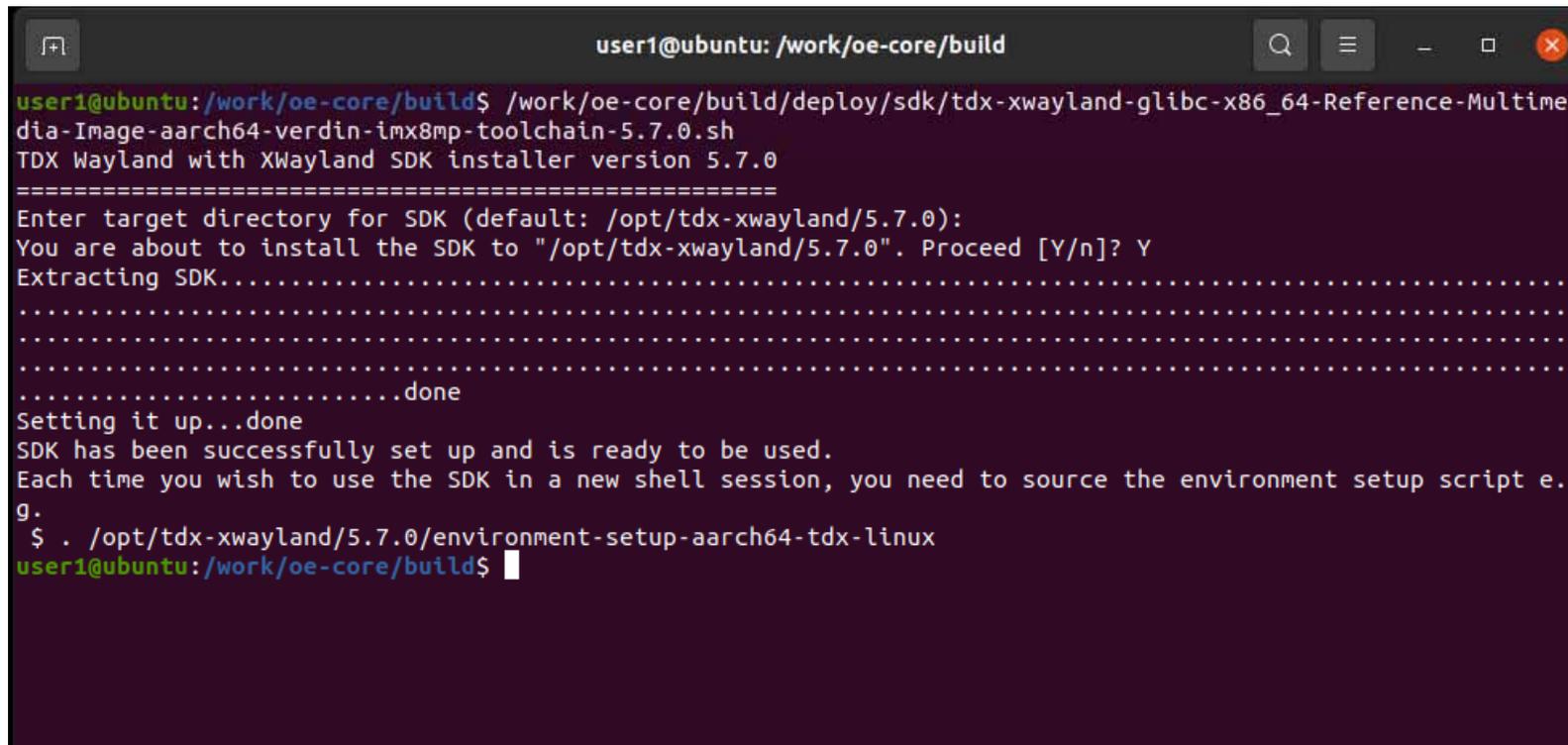
すでに存在していた場合は下記のように上書きするかどうかを問われますがその場合は一度シェルを停止してシェル実行前にディレクトリを削除しておいたほうが良いです。

```
If you continue, existing files will be overwritten! Proceed[y/N]?
```

```
[Ubuntu]$ sudo rm -rf /opt/tdx-xwayland/5.7.0
```

最後に下記のような環境変数設定シェルのパスの案内があります。このシェルは後の工程で使用します。

```
. /opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux
```



```
user1@ubuntu: /work/oe-core/build
user1@ubuntu:/work/oe-core/build$ /work/oe-core/build/deploy/sdk/tdx-xwayland-glibc-x86_64-Reference-Multimedia-Image-aarch64-verdin-imx8mp-toolchain-5.7.0.sh
TDX Wayland with XWayland SDK installer version 5.7.0
=====
Enter target directory for SDK (default: /opt/tdx-xwayland/5.7.0):
You are about to install the SDK to "/opt/tdx-xwayland/5.7.0". Proceed [Y/n]? Y
Extracting SDK.....
.....done
Setting it up...done
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
$ . /opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux
user1@ubuntu:/work/oe-core/build$
```

Eclipseのインストール

作業ディレクトリ作成

```
[Ubuntu]$ mkdir -p /work/app && cd /work/app
```

Eclipse起動にはJavaが必要です。Javaをインストールします。

```
[Ubuntu]$ sudo apt-get -y install openjdk-11-jre
```

Eclipse(202006バージョン)のサイトから入手します。(ブラウザなどで入手)

```
https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2020-06/R/eclipse-cpp-2020-06-R-linux-gtk-x86_64.tar.gz
```

app配下にコピーします。

展開します。

```
[Ubuntu]$ tar -xf ./eclipse-cpp-2020-06-R-linux-gtk-x86_64.tar.gz
```

Eclipse実行シェル作成

Eclipse起動前にSDK用環境変数を定義する必要があります。またコンパイルオプションも長くなるためSDK用環境変数設定やコンパイルオプションを定義してからEclipseを実行するシェルを作成します。

シェル内にはSDKが出力した環境変数設定シェルの実行も行っています。(長い赤線部分)

定義する内容は搭載するARMのアーキテクチャやBSPのバージョン、SDK出力ディレクトリなどによって変わります。

```
[Ubuntu]$ cd /work/app/  
[Ubuntu]$ gedit ./sdk_eclipse.sh
```

最適化オプションはCPUごとに指定を変える必要があります。下記に参考情報があります。

<https://gcc.gnu.org/onlinedocs/gcc-9.3.0/gcc/AArch64-Options.html#AArch64-Options>

<https://ja.wikipedia.org/wiki/ARM%E3%82%A2%E3%83%BC%E3%82%AD%E3%83%86%E3%82%AF%E3%83%81%E3%83%A3>

内容は下記です。

```
#!/bin/sh
```

```
./opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux
```

```
${SDK_ENV_SET}
```

```
export SDK_INCLUDE=${SDKTARGETSYSROOT}/usr/include/
```

```
export SDK_OPTIMIZATION="-march=armv8-a -mtune=cortex-a53"
```

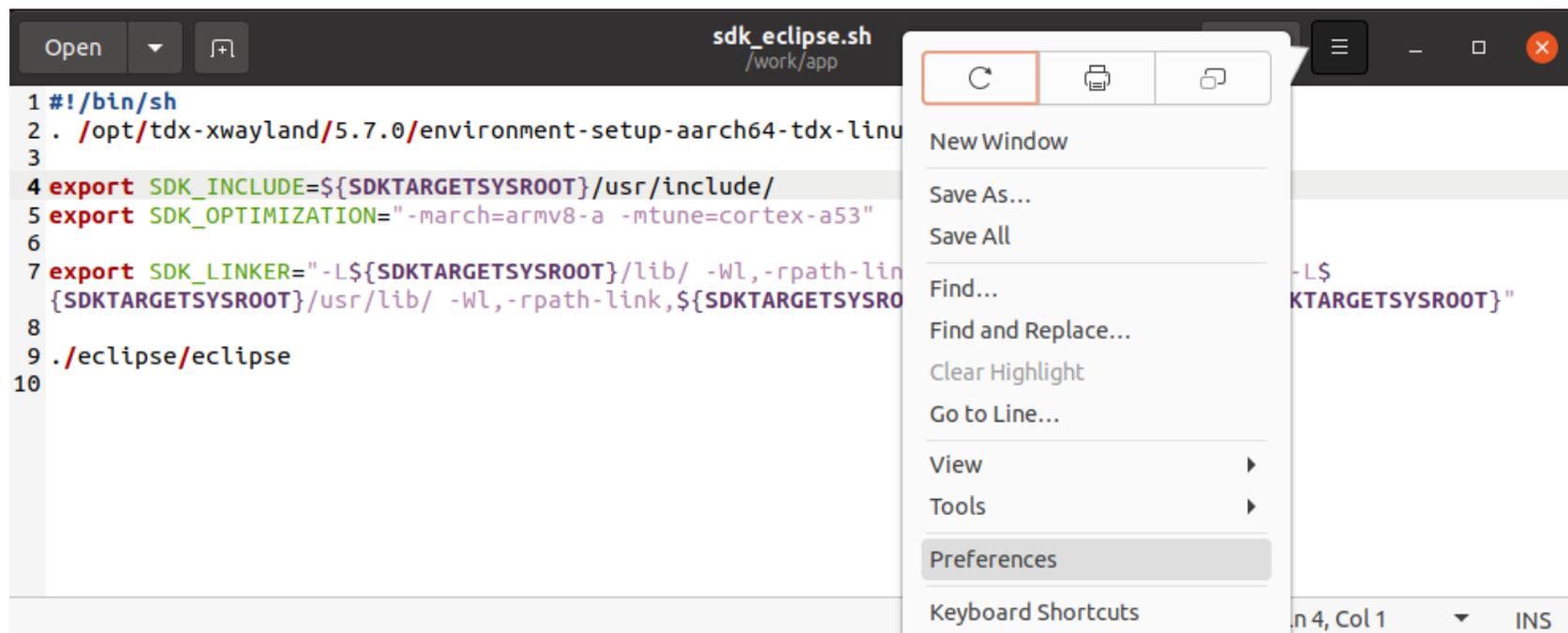
```
export SDK_LINKER="-L${SDKTARGETSYSROOT}/lib/
```

```
-Wl,-rpath-link,${SDKTARGETSYSROOT}/lib/ -L${SDKTARGETSYSROOT}/usr/lib/
```

```
-Wl,-rpath-link,${SDKTARGETSYSROOT}/usr/lib/ --sysroot=${SDKTARGETSYSROOT}"
```

```
./eclipse/eclipse
```

各々の設定が改行されていないかを確認してください。改行があるとうまくいきません。
geditで行番号を出す設定にするとわかりやすくなります。行番号を表示するにはgedit起動後画面右上のText EditorでPreferencesを開きます。

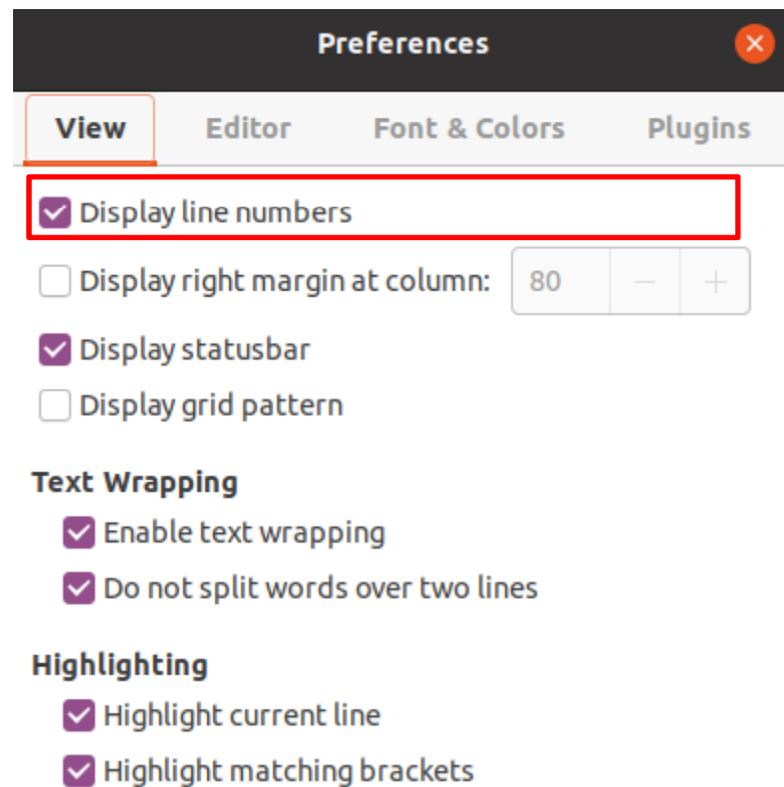


The image shows a terminal window with a dark theme. The title bar reads "sdk_eclipse.sh" and the current directory is "/work/app". The terminal content is as follows:

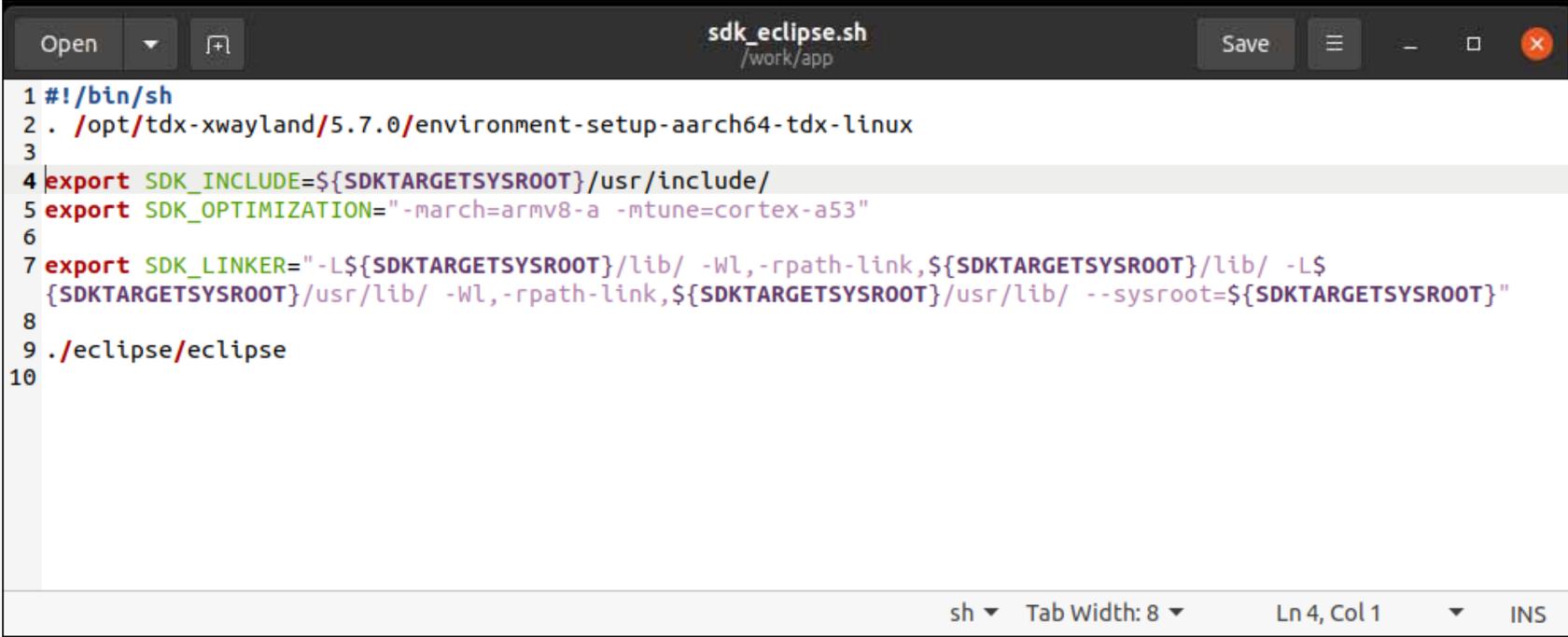
```
1 #!/bin/sh
2 . /opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux
3
4 export SDK_INCLUDE=${SDKTARGETSYSROOT}/usr/include/
5 export SDK_OPTIMIZATION="-march=armv8-a -mtune=cortex-a53"
6
7 export SDK_LINKER="-L${SDKTARGETSYSROOT}/lib/ -Wl,-rpath-link
  {SDKTARGETSYSROOT}/usr/lib/ -Wl,-rpath-link,${SDKTARGETSYSRO
8
9 ./eclipse/eclipse
10
```

A context menu is open over the terminal, listing various actions. The "Preferences" option is highlighted in grey. The menu items are: New Window, Save As..., Save All, Find..., Find and Replace..., Clear Highlight, Go to Line..., View, Tools, Preferences, and Keyboard Shortcuts. The status bar at the bottom right shows "n 4, Col 1" and "INS".

Display Line Numbersにチェックをいれて閉じます。



行番号が表示され各exportが一行で記述されているかどうか分かりやすくなります。



```
Open [ ] sdk_eclipse.sh /work/app Save [ ] [ ] [ ] [X]
1 #!/bin/sh
2 . /opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux
3
4 export SDK_INCLUDE=${SDKTARGETSYSROOT}/usr/include/
5 export SDK_OPTIMIZATION="-march=armv8-a -mtune=cortex-a53"
6
7 export SDK_LINKER="-L${SDKTARGETSYSROOT}/lib/ -Wl,-rpath-link,${SDKTARGETSYSROOT}/lib/ -L$
  {SDKTARGETSYSROOT}/usr/lib/ -Wl,-rpath-link,${SDKTARGETSYSROOT}/usr/lib/ --sysroot=${SDKTARGETSYSROOT}"
8
9 ./eclipse/eclipse
10
```

sh Tab Width: 8 Ln 4, Col 1 INS

作成したファイルに実行権限を付与します。

```
[Ubuntu]$ chmod +x ./sdk_eclipse.sh
```

ワークスペースディレクトリ作成

```
[Ubuntu]$ mkdir ./workspace
```

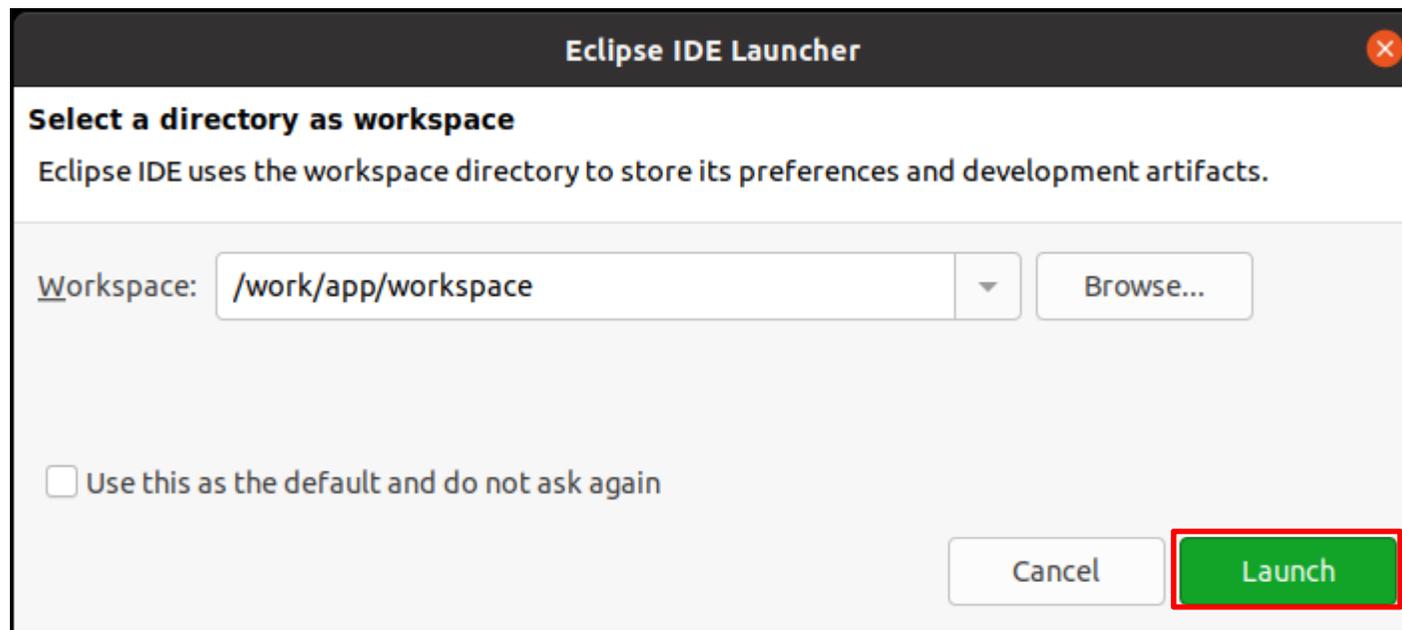
Eclipse実行

```
[Ubuntu]$ ./sdk_eclipse.sh
```

以後Eclipseを起動する場合はこのシェルを使ってください。

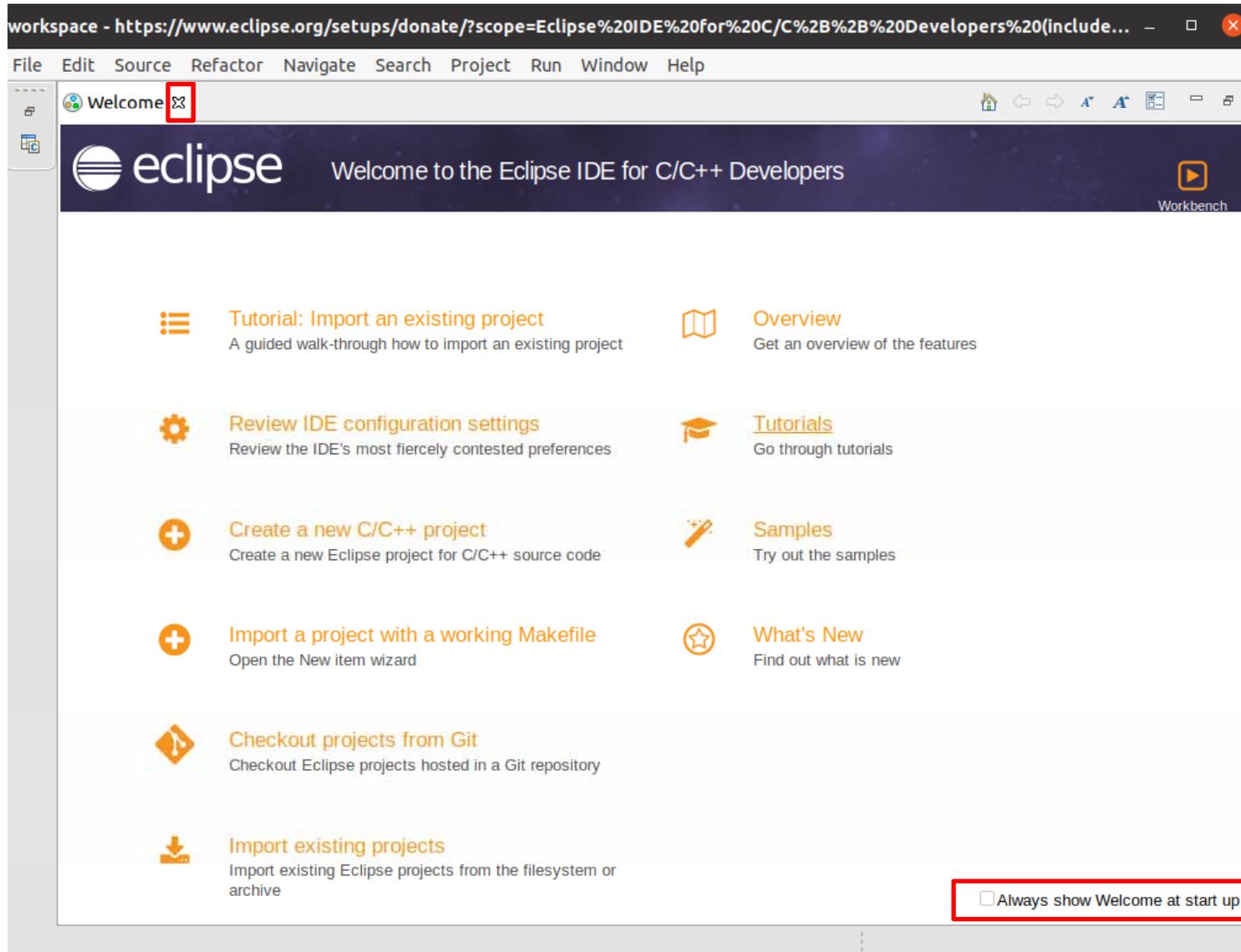
Eclipseを実行すると下記のようなワークスペースのディレクトリのパスを指定するウィンドウが表示されますのでワークスペースのパスを設定します。本マニュアルでは/work/app/workspaceにしています。

LaunchをクリックしてEclipseを起動します。



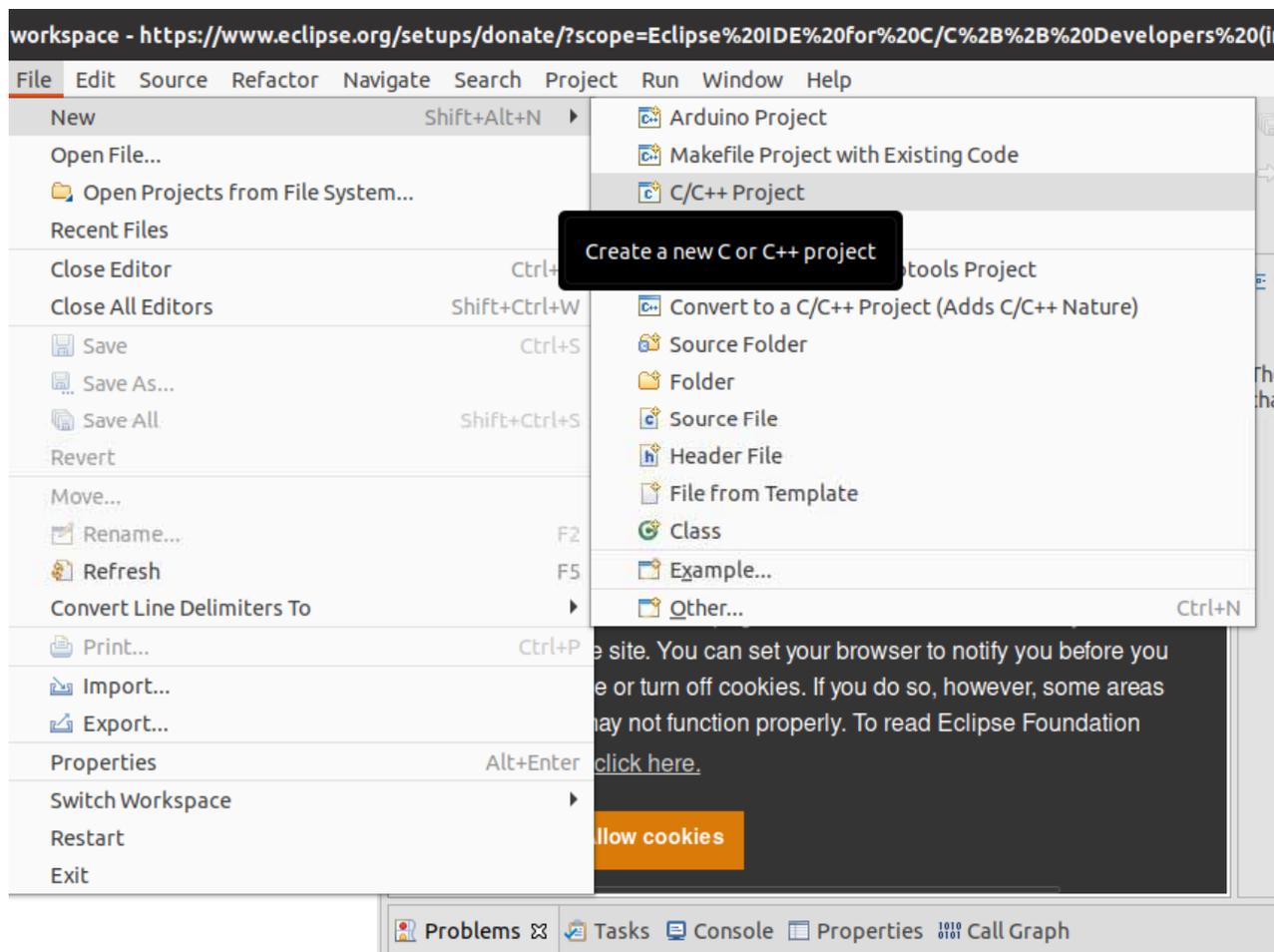
Welcomeと表示されますが不要なので × ボタンをクリックして消します。

右下のAlways show Welcome at start upのチェックをはずしておけば毎回起動時に表示されることがなくなります。

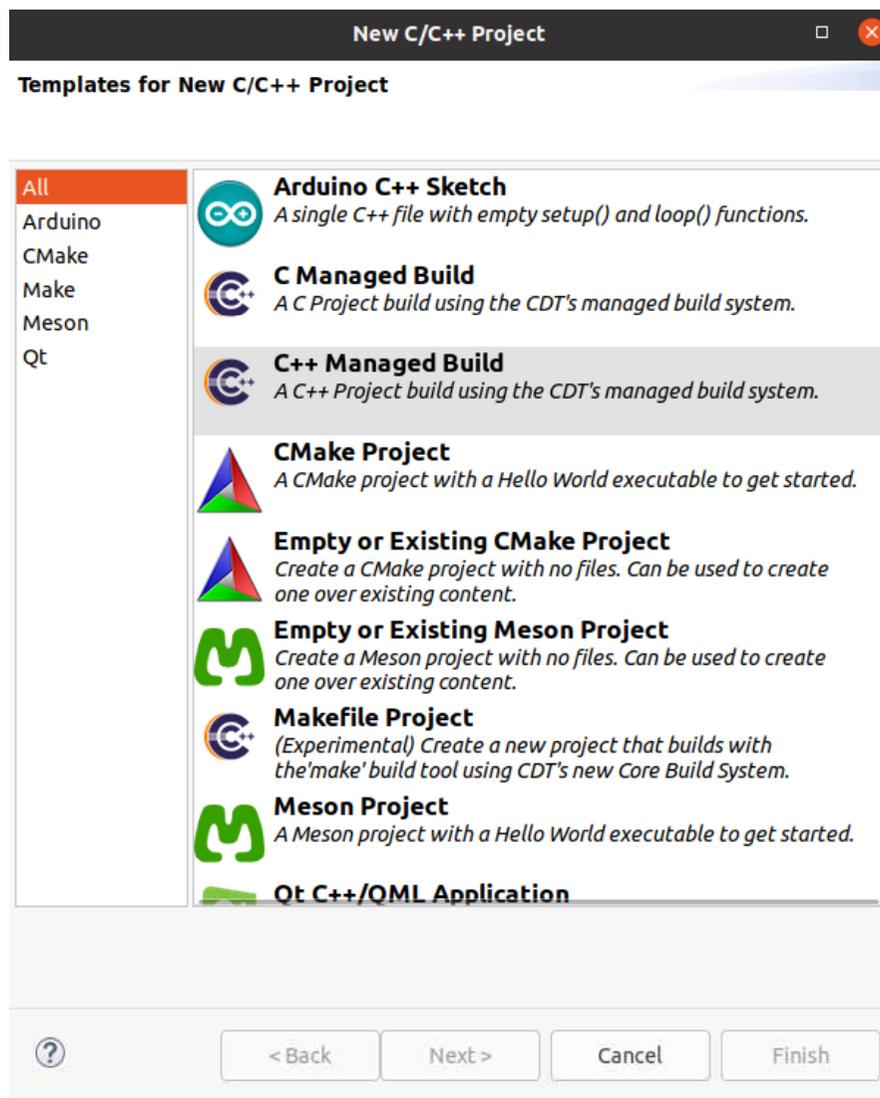


プロジェクトの作成&ビルド

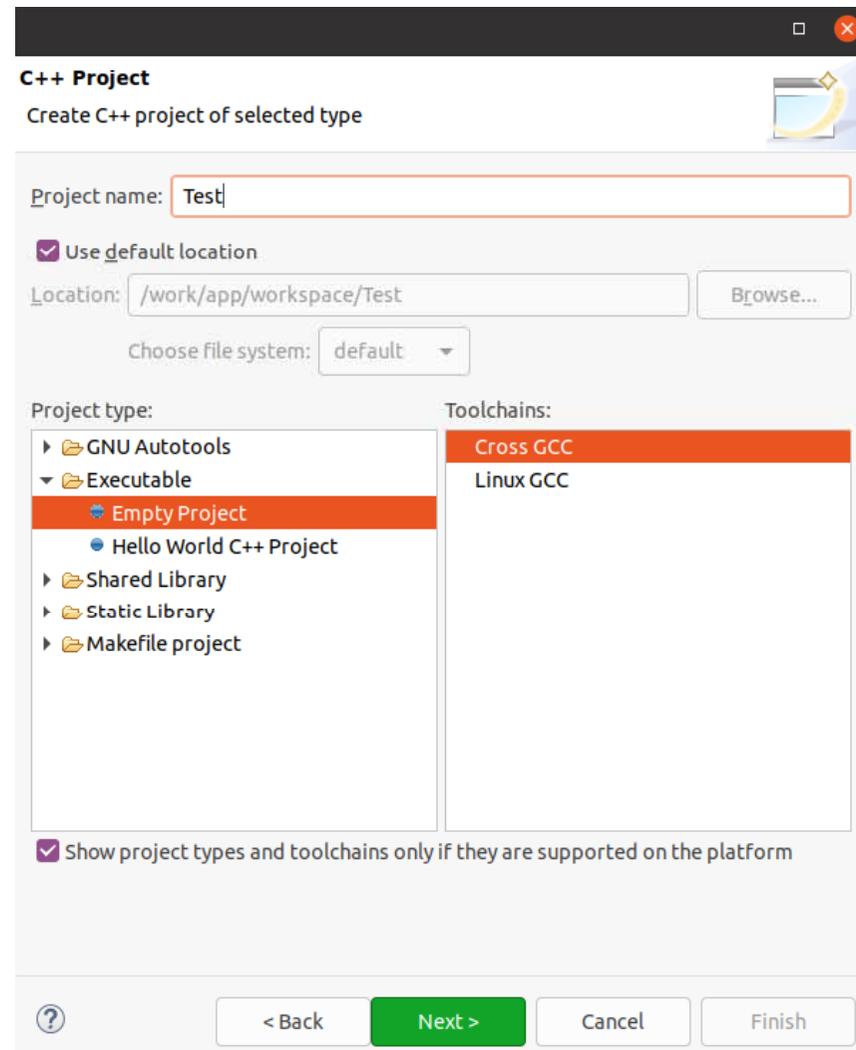
メニューからFile > New > C++Projectを選択します。



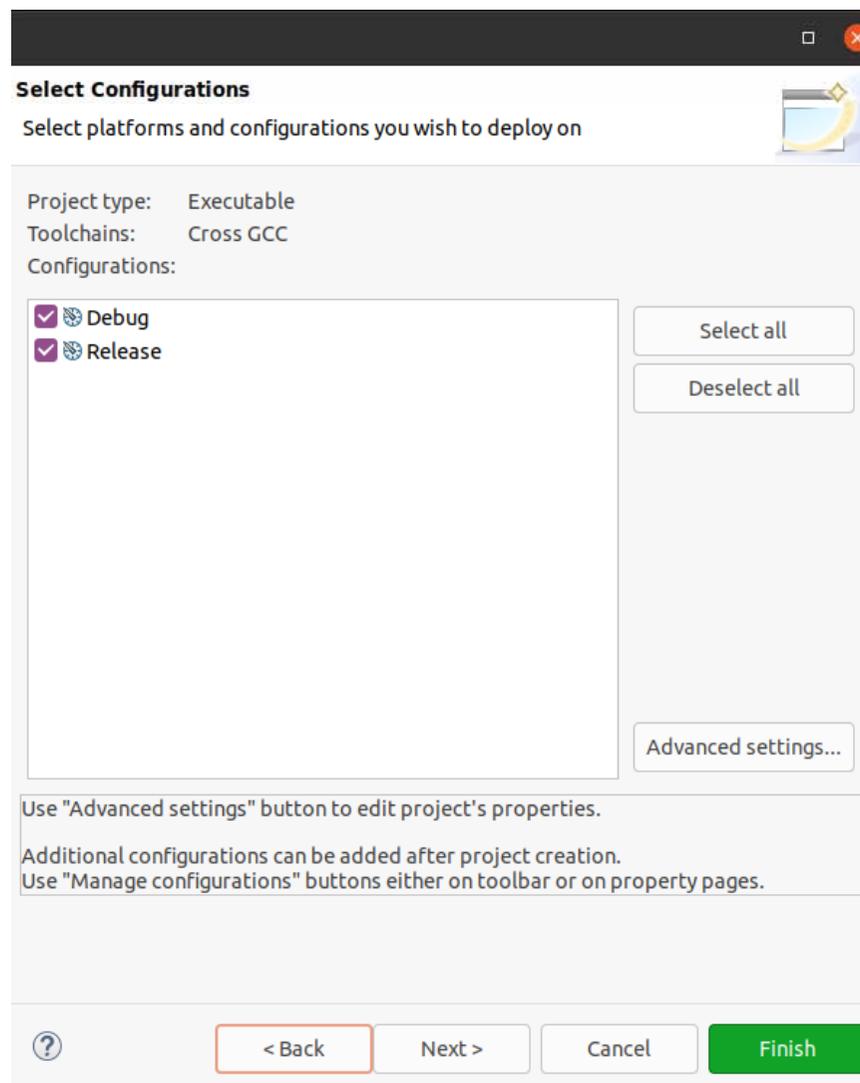
CもしくはC++言語のプログラムを作成するため「C++ Managed Build」を選択してNextをクリックします。



Project Nameを入力します。本マニュアルではTestにしています。
Project typeにEmpty Project、ToolchainsにCross GCCを選択します。
Nextをクリックします。



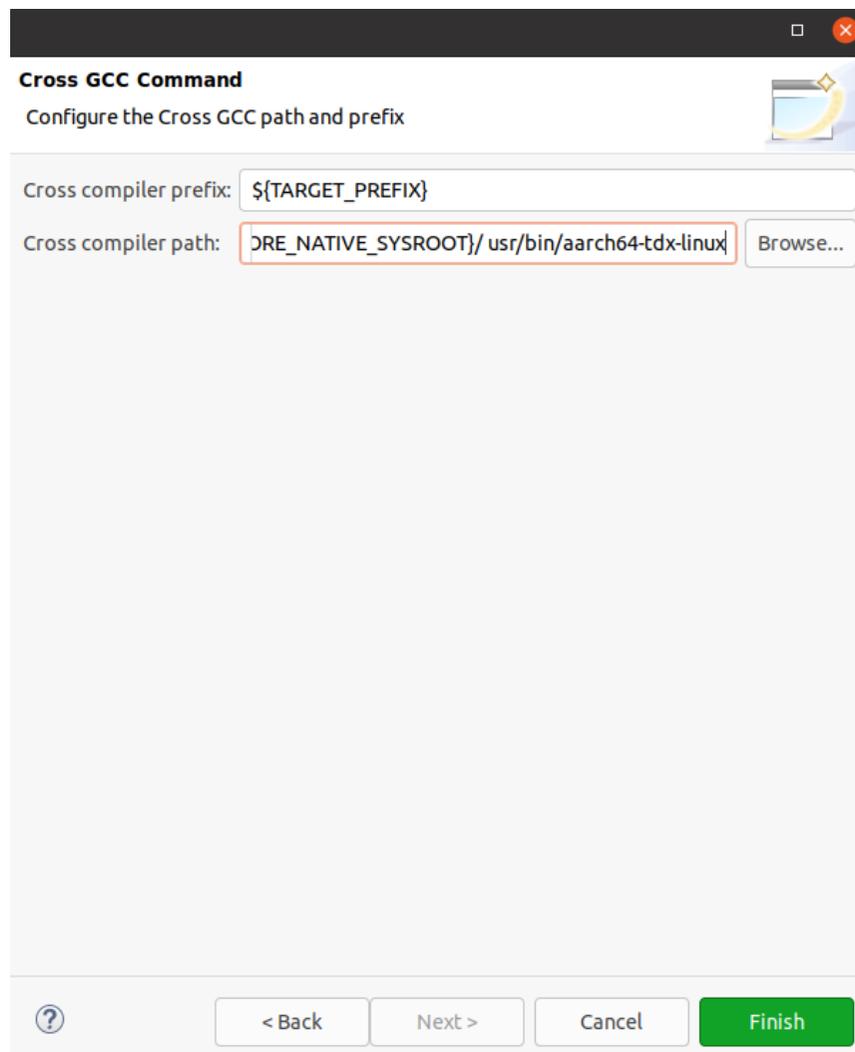
ここではデフォルト設定のままにします。DebugとRelease設定をもつProjectになります。Nextをクリックします。



Cross Compiler prefixに「`${TARGET_PREFIX}`」

Cross Compiler pathに「`${OECORE_NATIVE_SYSROOT}/usr/bin/aarch64-tdx-linux`」を入力してFinishをクリックします。

画面を大きくしないとCross Compiler pathの設定がすべて見えないので画面を最大化して確認してください。



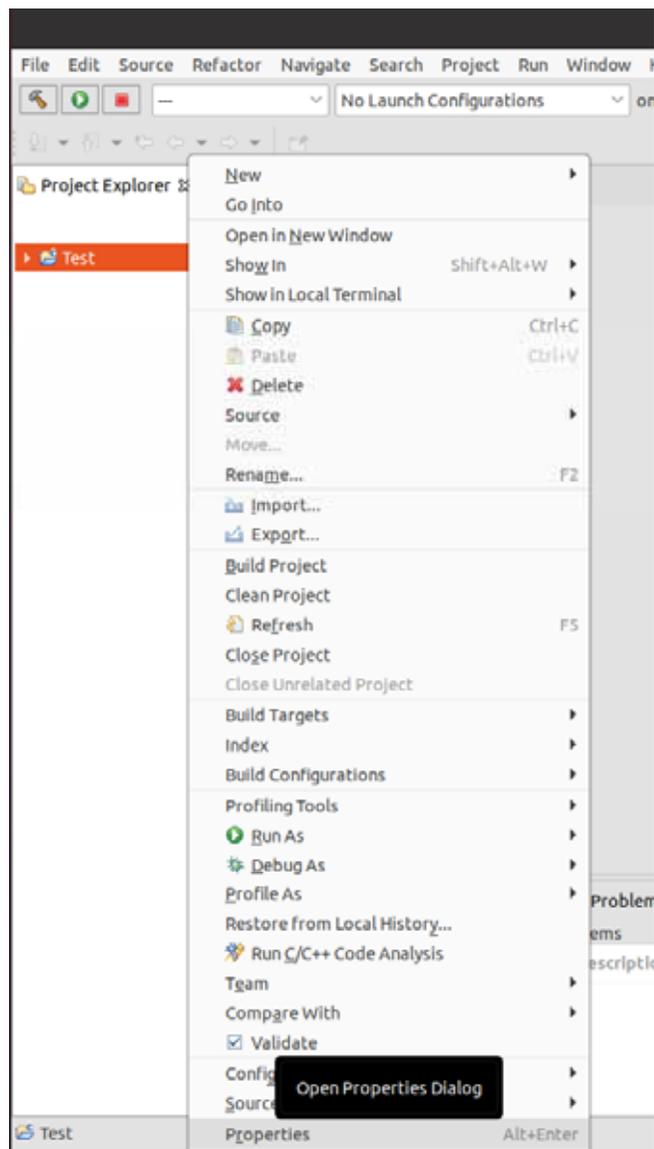
Cross GCC Command
Configure the Cross GCC path and prefix

Cross compiler prefix: `${TARGET_PREFIX}`

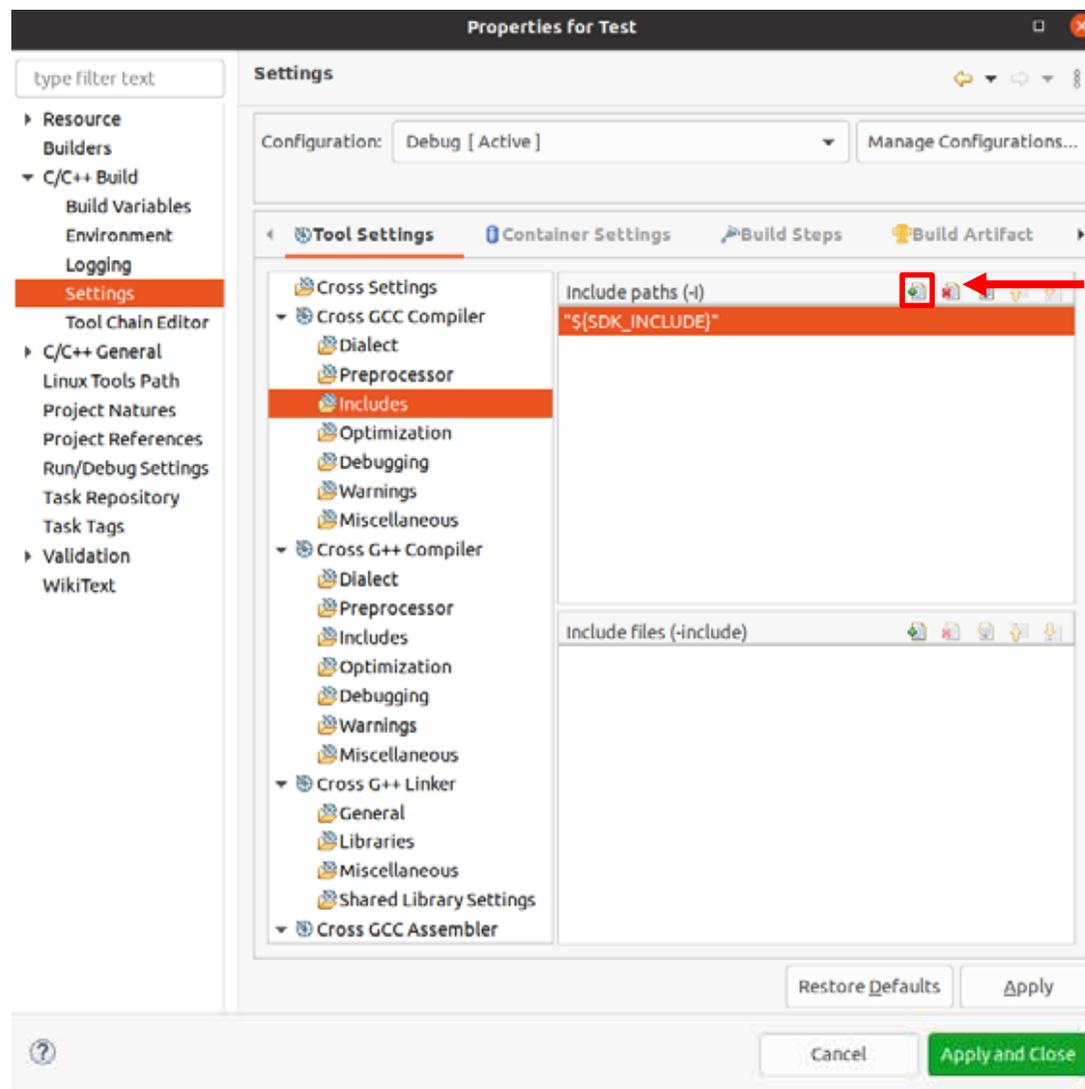
Cross compiler path: `${OECORE_NATIVE_SYSROOT}/usr/bin/aarch64-tdx-linux` Browse...

? < Back Next > Cancel Finish

作成したプロジェクトを右クリックしてPropertiesを開きます。

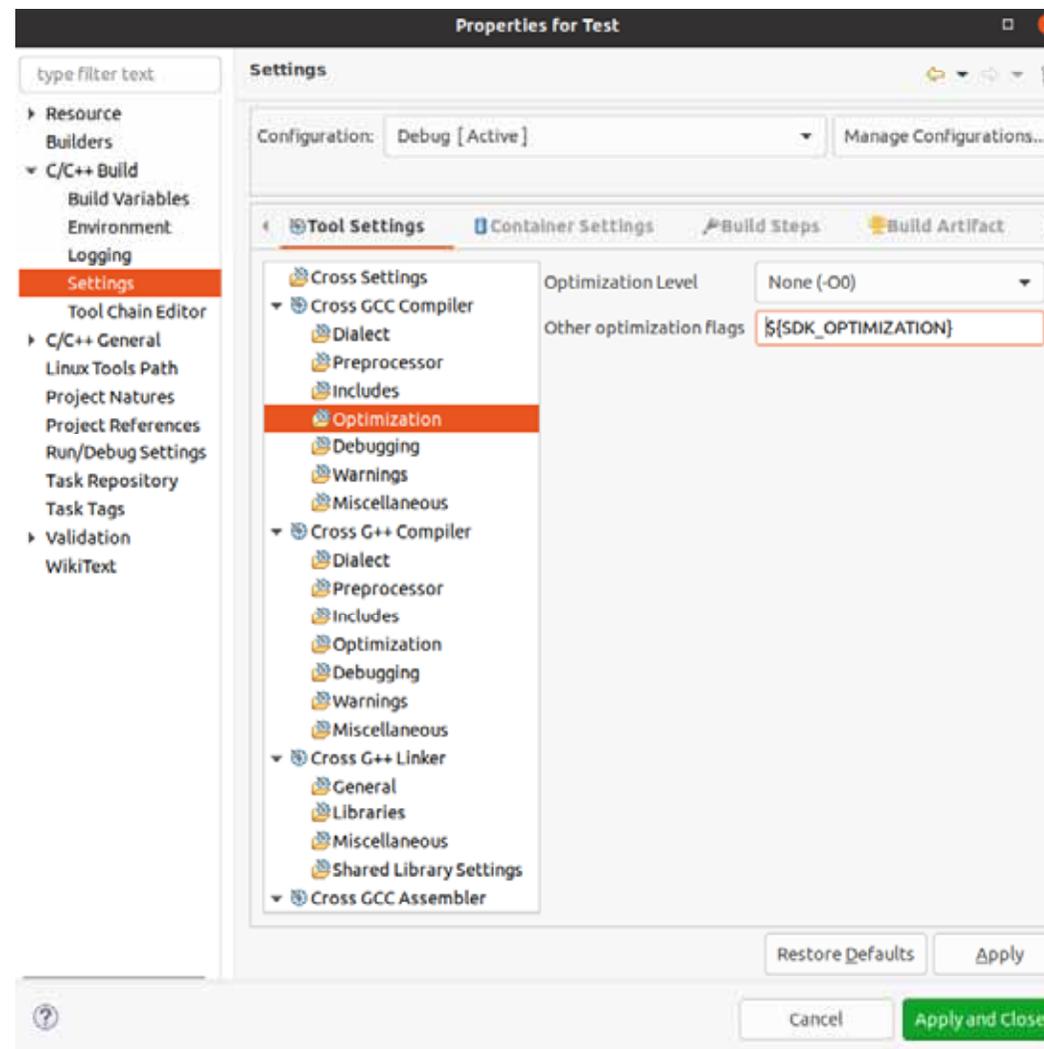


C/C++ Build > Settings > Cross GCC Compiler > Includesで+のアイコンをクリックして下記を設定します。
\${SDK_INCLUDE}

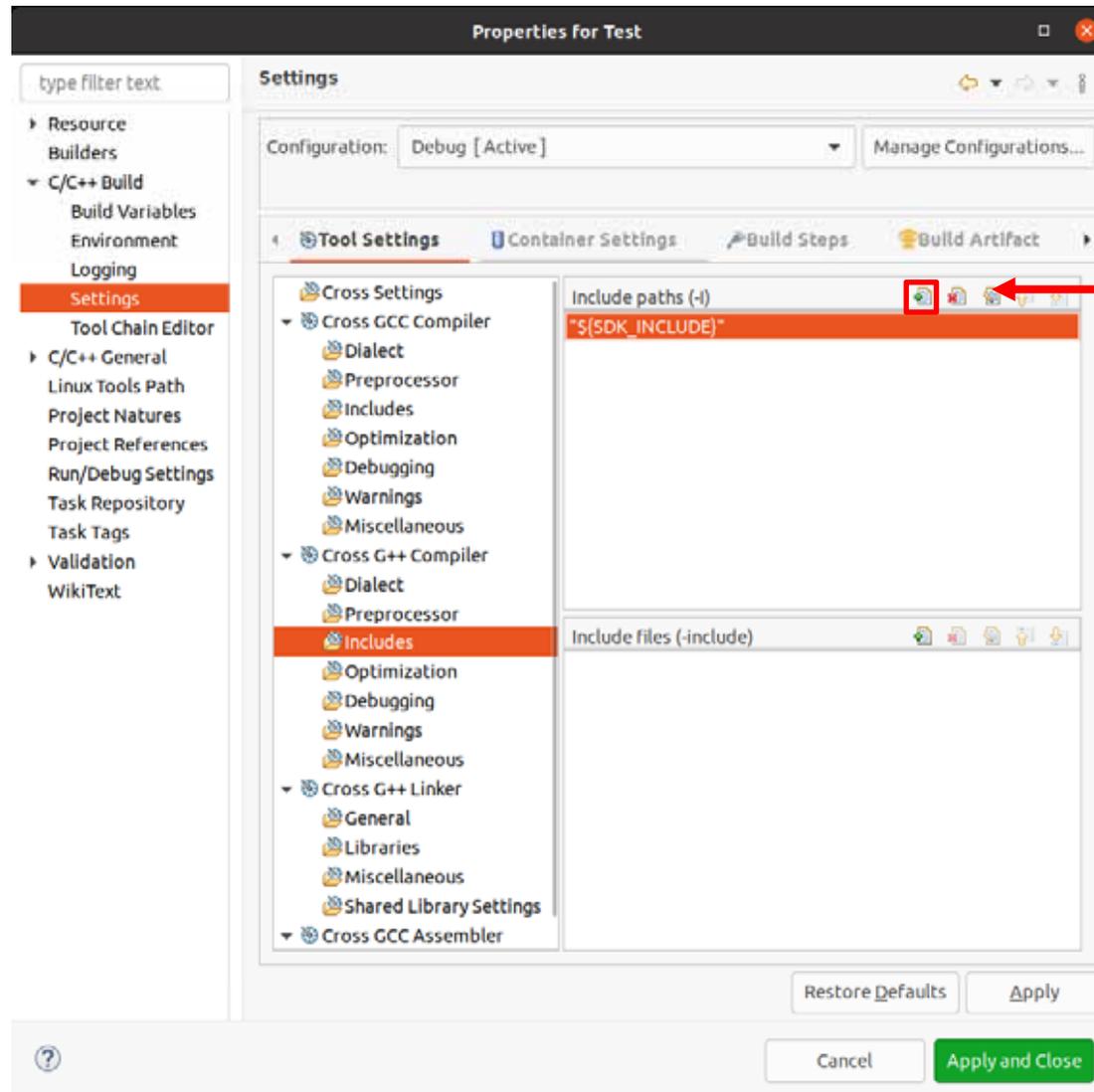


設定追加のアイコン

同画面のまま Optimization を選択し Other optimization flags に下記を入力します。
\${SDK_OPTIMIZATION}

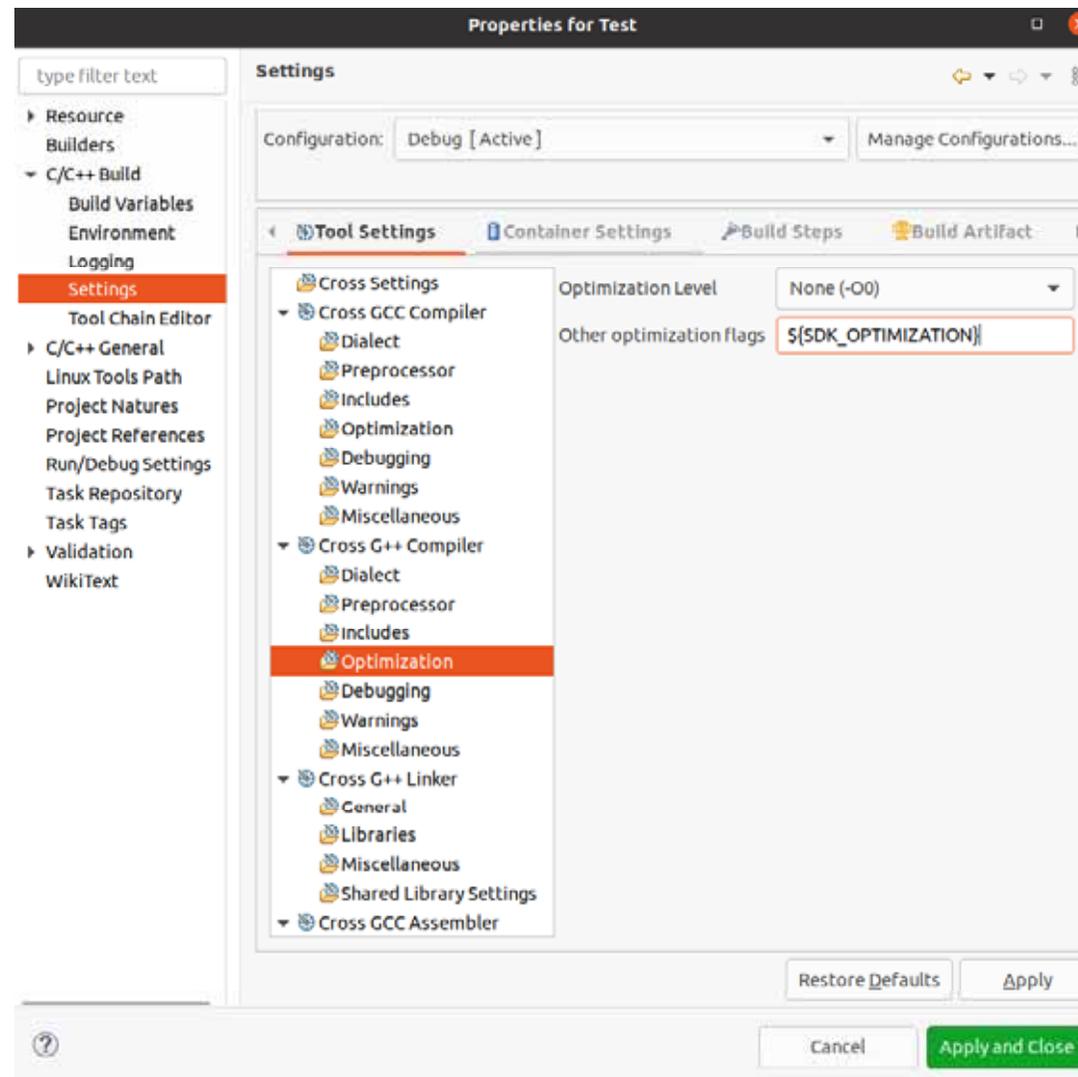


同画面のままCross G++ Compiler > Includesにも同様に下記を設定します。
設定を追加しただけで前回入力したものが入力されていますのでその場合は入力の必要はありません。
\${SDK_INCLUDE}

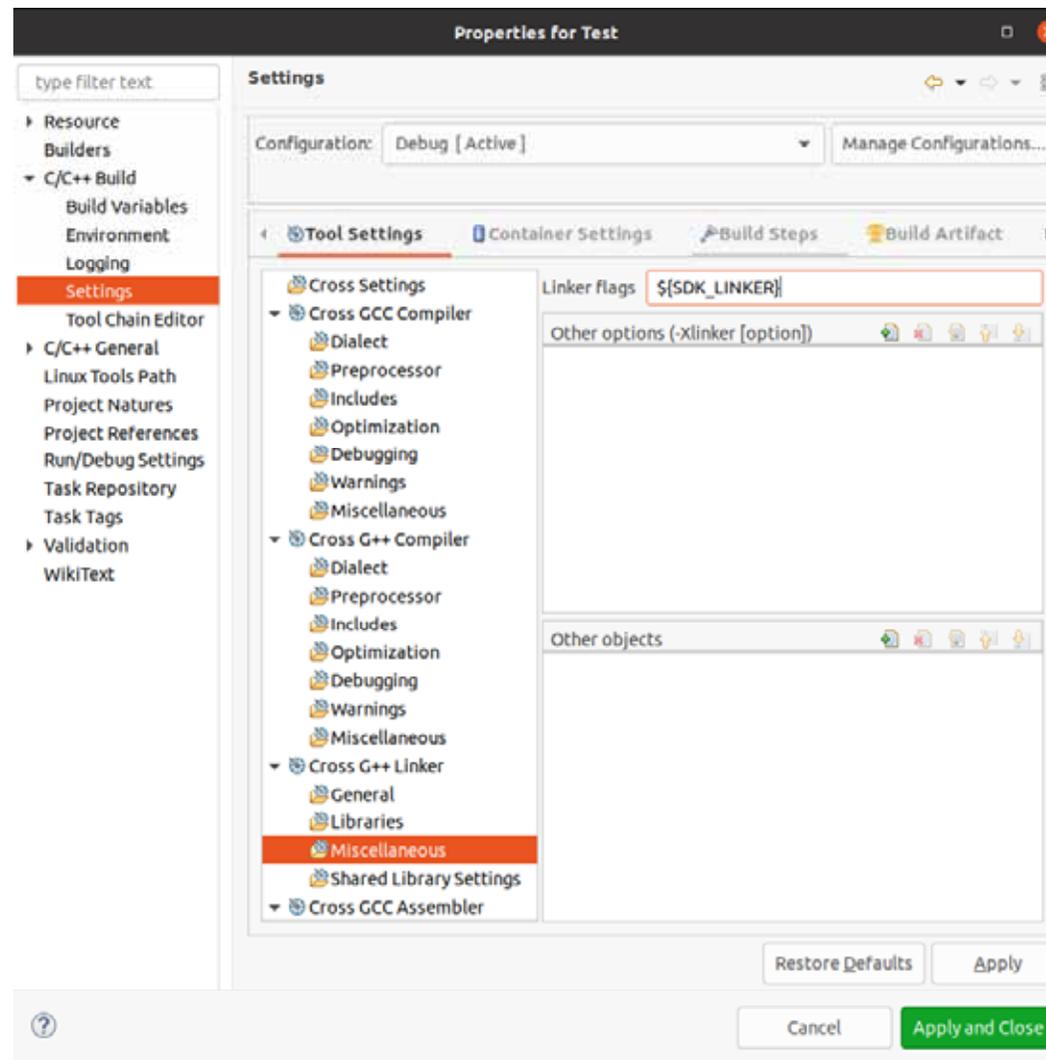


設定追加のアイコン

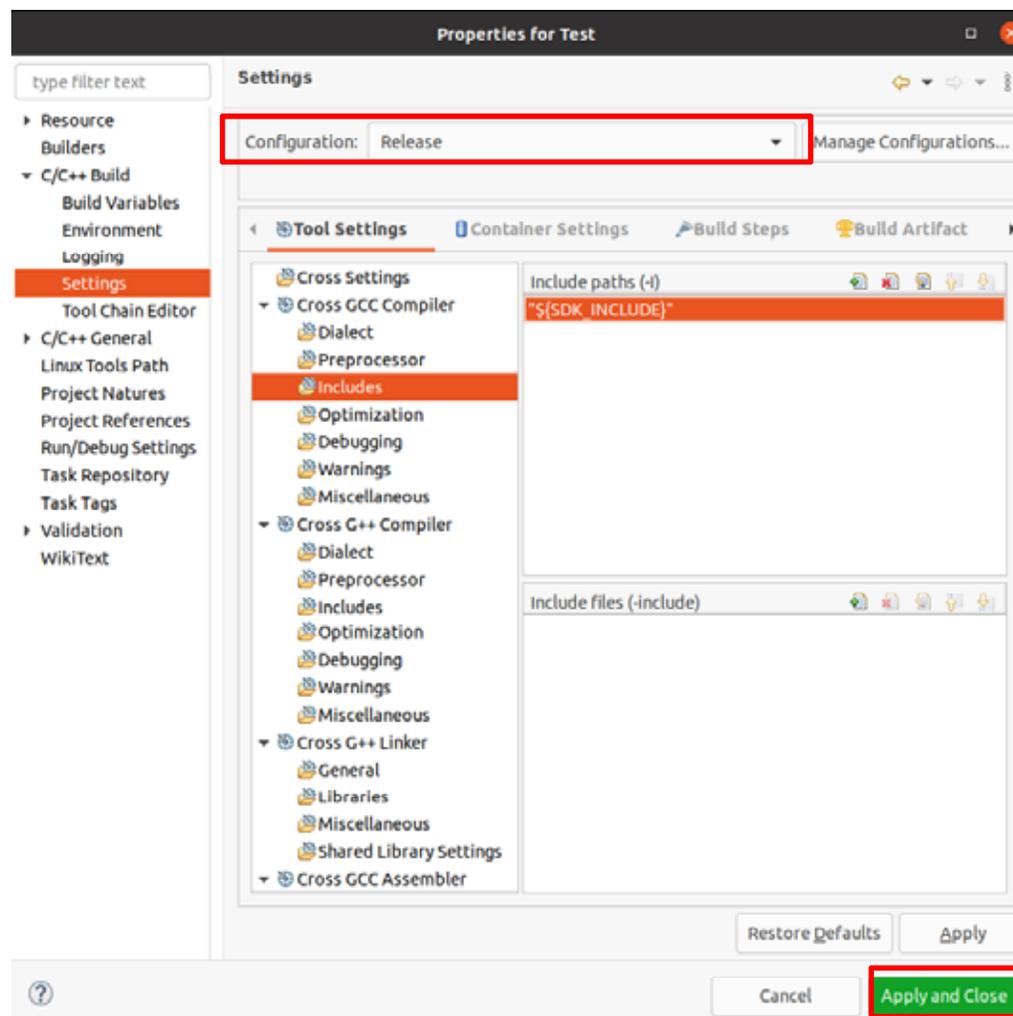
同画面のままOptimizationを選択しOther optimization flagsに下記を入力します。
\${SDK_OPTIMIZATION}



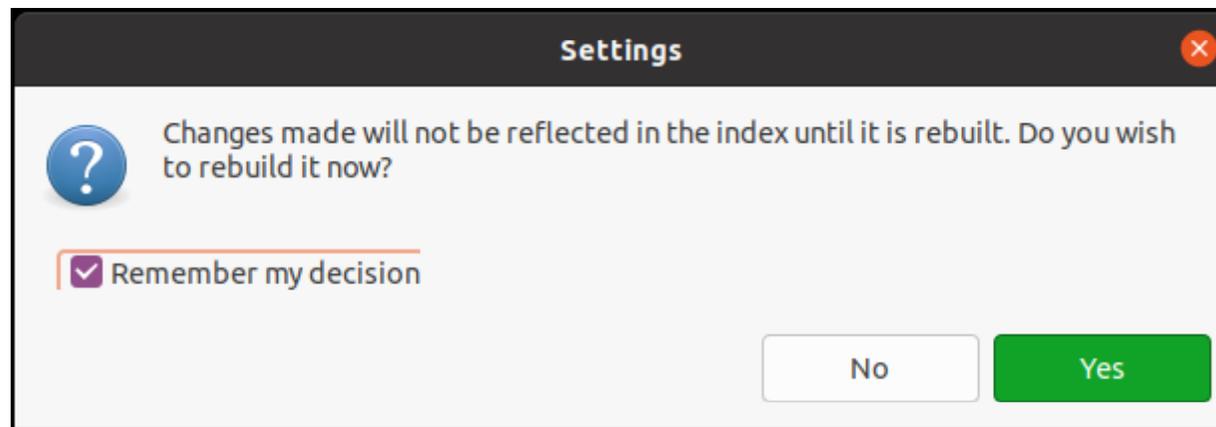
同画面のままCross G++ Linker > MiscellaneousのLinker flagsに下記を設定します。
\${SDK_LINKER}



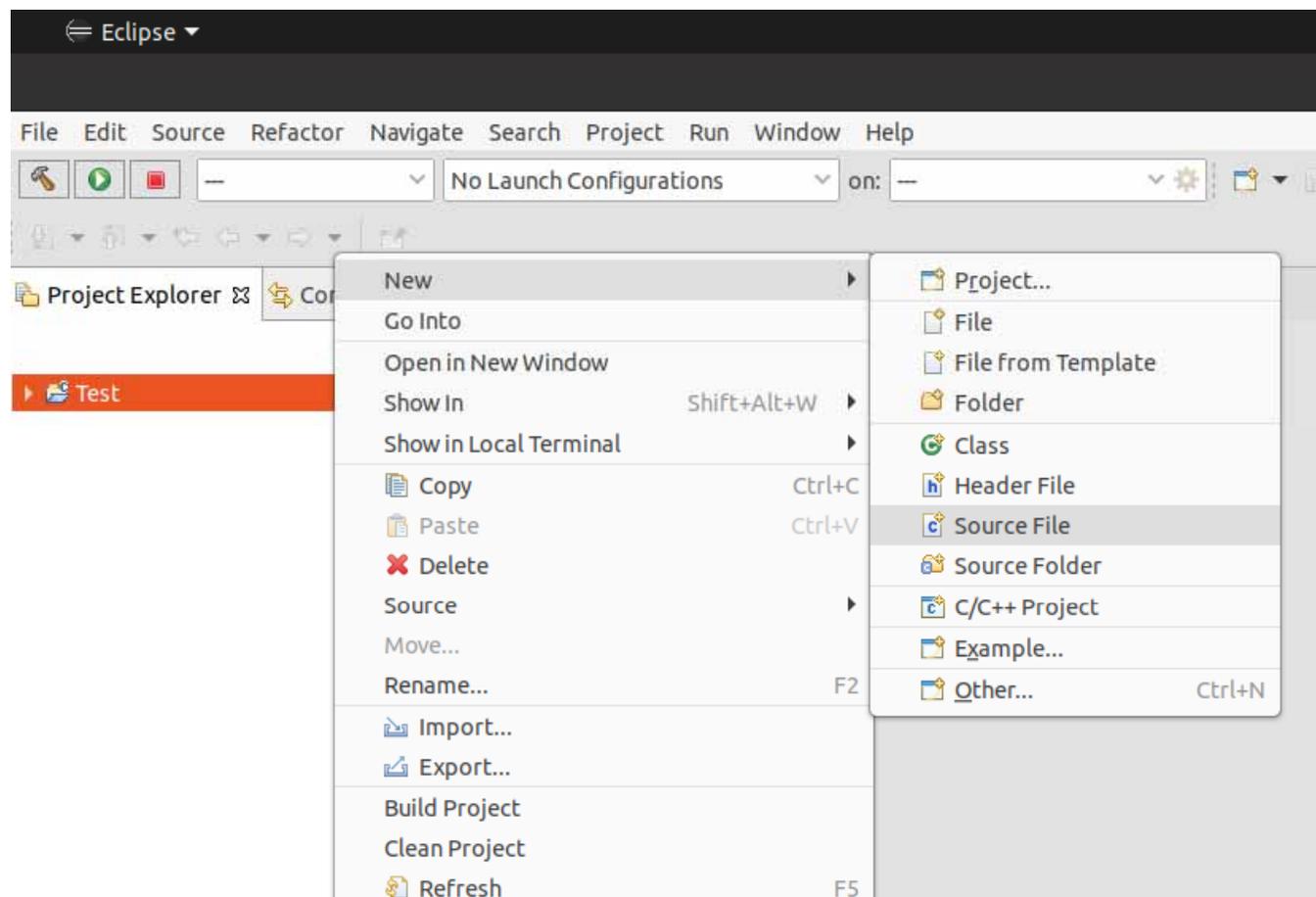
ConfigurationをReleaseにしてReleaseの設定にもIncludesやOptimization、Miscellaneousに同様のパラメータを入力します。すべての設定入力後Apply and Closeをクリックします。



下記のようなリビルドするまで変更が反映されないという警告が出ます。
基本的にプロジェクトの設定変更後はリビルドして反映させますのでRemember my decisionにチェックをいれてYesをクリックします。



プロジェクトを右クリックしNew > Source Fileをクリックしてソースファイルを追加します。



Source fileを入力します。拡張子を必ずつけてください。

Templateはソースコードに付加されるコメントですが本マニュアルではNoneとしています。Finishをクリックします。

New Source File

Source File
Create a new source file.

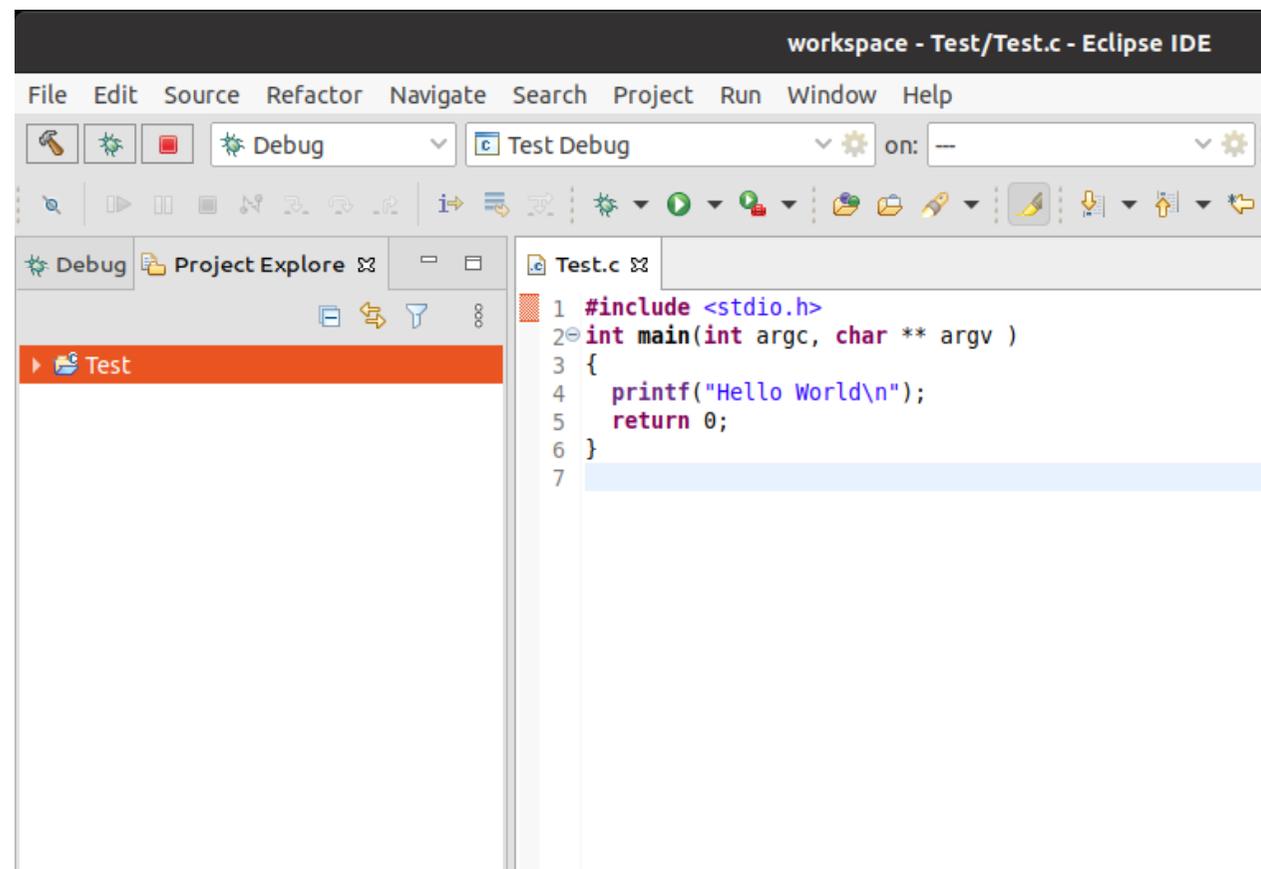
Source folder:

Source file:

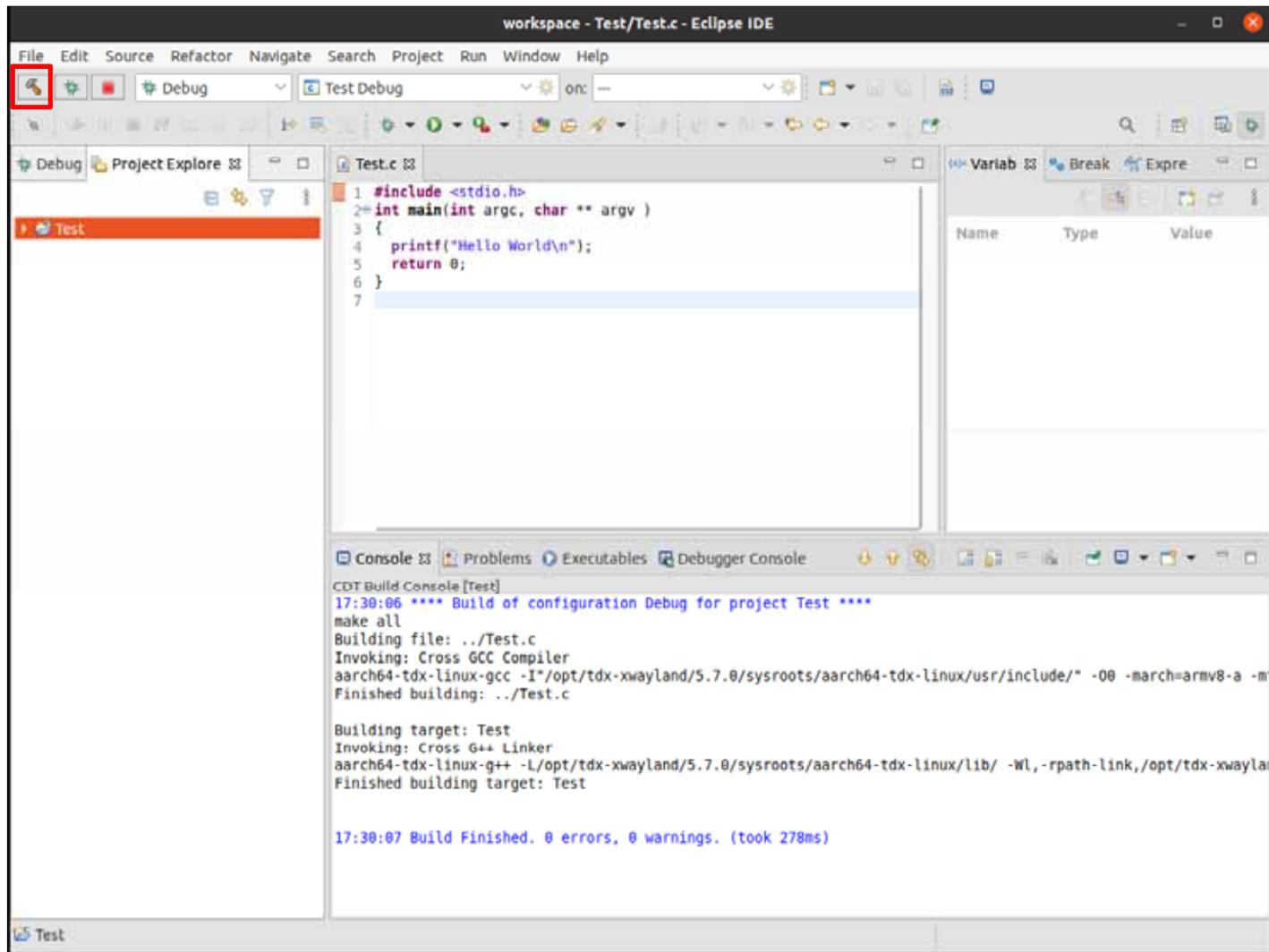
Template:

ソースコードを入力します。本マニュアルではHello Worldと出力するだけのソースコードを入力しています。

```
#include <stdio.h>
int main(int argc, char ** argv )
{
    printf("Hello World\n");
    return 0;
}
```



ビルドボタンを押してビルドを行います。Consoleにログが出力されます。正常に終わった場合、青字でログが出力されます。



ビルドログは下記のようになります。

もしエラーが出た場合にはなにかしらの設定ミスがあります。差異に注意してください。

```
17:30:06 **** Build of configuration Debug for project Test ****
```

```
make all
```

```
Building file: ../Test.c
```

```
Invoking: Cross GCC Compiler
```

```
aarch64-tdx-linux-gcc -I"/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/usr/include/" -O0 -march=armv8-a -mtune=cortex-a53 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"Test.d" -MT"Test.o" -o "Test.o" "../Test.c"
```

```
Finished building: ../Test.c
```

```
Building target: Test
```

```
Invoking: Cross G++ Linker
```

```
aarch64-tdx-linux-g++ -L/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/lib/ -Wl,-rpath-link,/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/lib/ -L/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/usr/lib/ -Wl,-rpath-link,/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux/usr/lib/ --sysroot=/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux -o "Test" ../Test.o
```

```
Finished building target: Test
```

```
17:30:07 Build Finished. 0 errors, 0 warnings. (took 278ms)
```

SSH接続設定作成

デバッグに使用するGDBはEthernetで接続してSSHプロトコルを利用してデバッグを行います。

デバッグを行うためにモジュールとSSHで接続できるようにしておく必要があります。

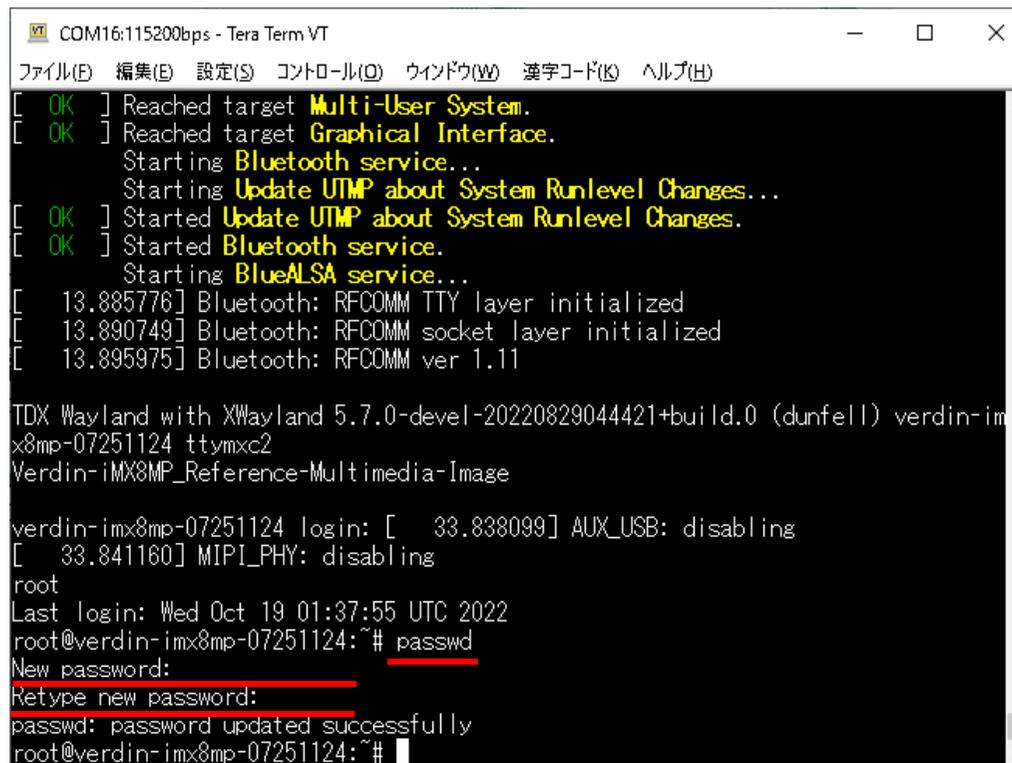
モジュール側の設定を行います。モジュールにEthernetケーブルを挿入し開発パソコンと同じサブネットに接続します。

デバッグコンソールを使ってコマンドを入力します。

Teraterm(Ubuntuの場合minicomなど)を起動してからモジュールを起動します。rootでログインしpasswdコマンドでパスワードを設定します。パスワードを2回入力するとパスワードが設定されます。(2回目は確認用)

本マニュアルではセキュリティを一切気にせず利便性のよいパスワード認証を使い、rootでSSHにログインできるようにします。あくまでデバッグ目的で設定するだけです。

```
[Module]# passwd
```



```
COM16:115200bps - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) 漢字コード(K) ヘルプ(H)
[ OK ] Reached target Multi-User System.
[ OK ] Reached target Graphical Interface.
Starting Bluetooth service...
Starting Update UTMP about System Runlevel Changes...
[ OK ] Started Update UTMP about System Runlevel Changes.
[ OK ] Started Bluetooth service.
Starting BlueALSA service...
[ 13.885776] Bluetooth: RFCOMM TTY layer initialized
[ 13.890749] Bluetooth: RFCOMM socket layer initialized
[ 13.895975] Bluetooth: RFCOMM ver 1.11

TDX Wayland with XWayland 5.7.0-devel-20220829044421+build.0 (dunfell) verdin-imx8mp-07251124 ttymxc2
Verdin-IMX8MP_Reference-Multimedia-Image

verdin-imx8mp-07251124 login: [ 33.838099] AUX_USB: disabling
[ 33.841160] MIPI_PHY: disabling
root
Last login: Wed Oct 19 01:37:55 UTC 2022
root@verdin-imx8mp-07251124:~# passwd
New password:
Retype new password:
passwd: password updated successfully
root@verdin-imx8mp-07251124:~#
```

次にモジュールのIPアドレスの設定を行います。何も設定していない場合はDHCPとなります。
設定ファイル/etc/systemd/network/wired.networkを新規作成します。

```
[Module]# vi /etc/systemd/network/wired.network
```

DHCPの場合

```
[Match]  
Name=eth0
```

```
[Network]  
DHCP=ipv4
```

eth0はインターフェイス名です。Verdin-iMX8M Plusはeth0とeth1があります。
モジュールやBSPのバージョンによって異なりますのでifconfigコマンドで調べてください。

固定IPの場合

```
[Match]  
Name=eth0
```

```
[Network]  
Address=192.168.179.92/24  
Gateway=192.168.179.1
```

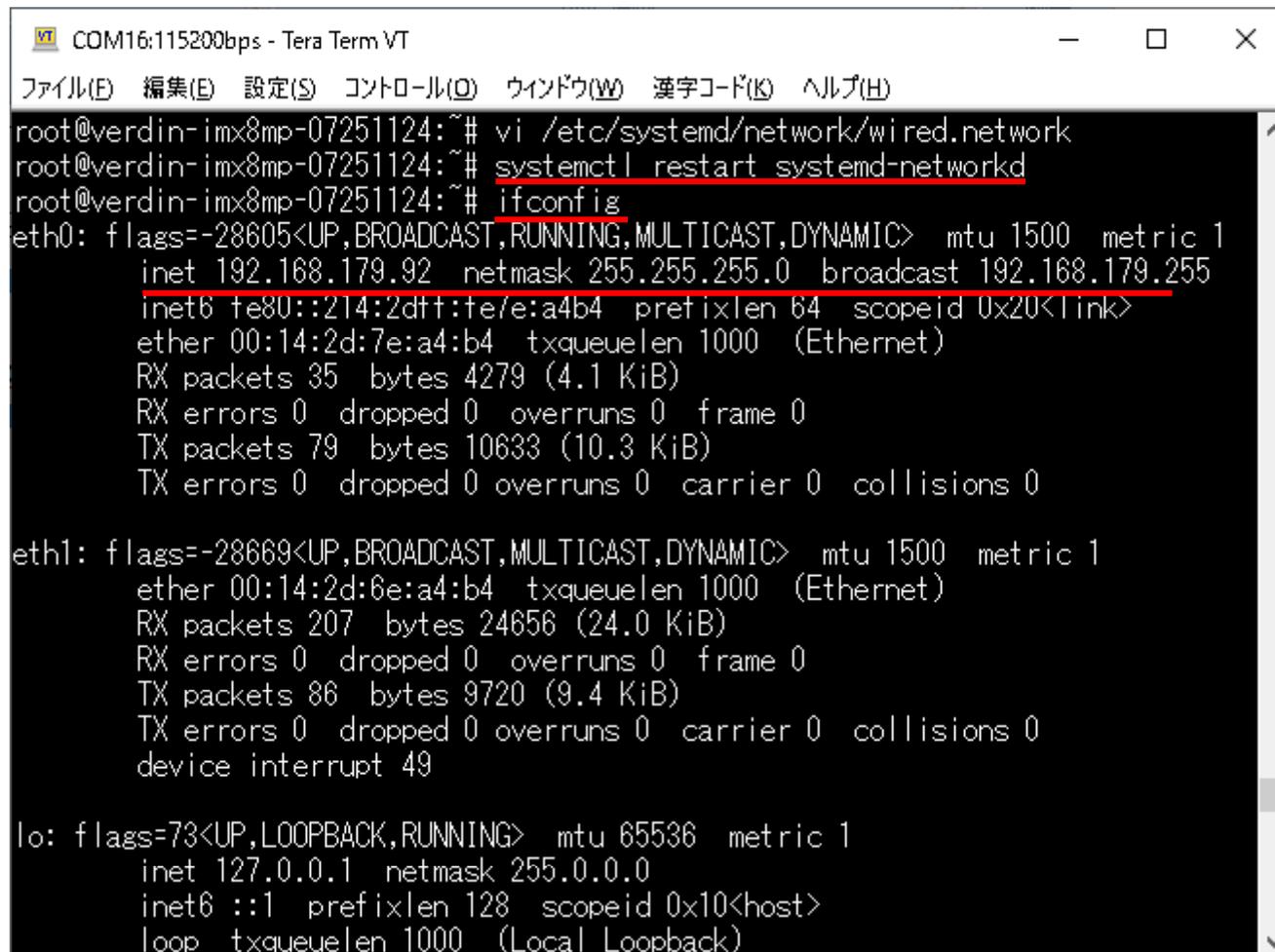
192.168.179.92/24はIPアドレス192.168.179.92 サブネットマスク255.255.255.0を意味します。
IPアドレスは環境に応じて設定してください。

下記コマンドでネットワークマネージャーを再起動します。

```
[Module]# systemctl restart systemd-networkd
```

ifconfigで設定が反映されているのを確かめます。

```
[Module]# ifconfig
```



```
COM16:115200bps - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) 漢字コード(K) ヘルプ(H)
root@verdin-ix8mp-07251124:~# vi /etc/systemd/network/wired.network
root@verdin-ix8mp-07251124:~# systemctl restart systemd-networkd
root@verdin-ix8mp-07251124:~# ifconfig
eth0: flags=-28605<UP,BROADCAST,RUNNING,MULTICAST,DYNAMIC> mtu 1500 metric 1
  inet 192.168.179.92 netmask 255.255.255.0 broadcast 192.168.179.255
  inet6 fe80::214:2dff:fe:e:a4b4 prefixlen 64 scopeid 0x20<link>
  ether 00:14:2d:7e:a4:b4 txqueuelen 1000 (Ethernet)
  RX packets 35 bytes 4279 (4.1 KiB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 79 bytes 10633 (10.3 KiB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth1: flags=-28669<UP,BROADCAST,MULTICAST,DYNAMIC> mtu 1500 metric 1
  ether 00:14:2d:6e:a4:b4 txqueuelen 1000 (Ethernet)
  RX packets 207 bytes 24656 (24.0 KiB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 86 bytes 9720 (9.4 KiB)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
  device interrupt 49

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536 metric 1
  inet 127.0.0.1 netmask 255.0.0.0
  inet6 ::1 prefixlen 128 scopeid 0x10<host>
  loop txqueuelen 1000 (Local Loopback)
```

SSH接続確認

ホストOSやゲストからpingで開発パソコンから接続できているか確認します。

VMwareの設定がデフォルトのNATになっている場合、ホストOSのWindows10で接続できていればUbuntu側で接続できます。接続できない場合は何かしらの設定が間違っている可能性があります。

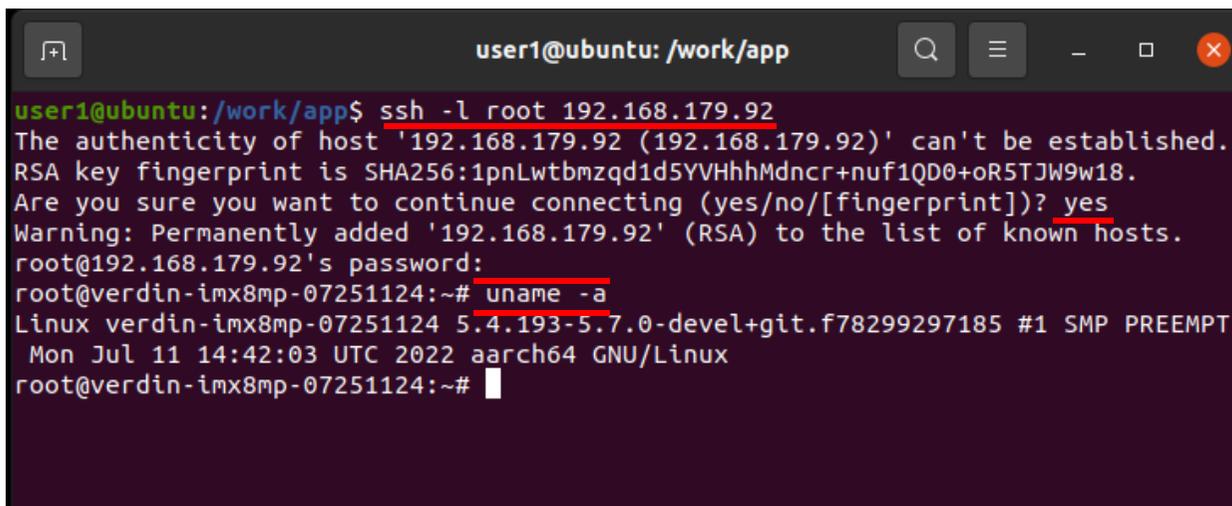
```
[Ubuntu]$ ping XXX.XXX.XXX.XXX
```

sshコマンドで接続できるか試します。

```
[Ubuntu]$ ssh -l root XXX.XXX.XXX.XXX
```

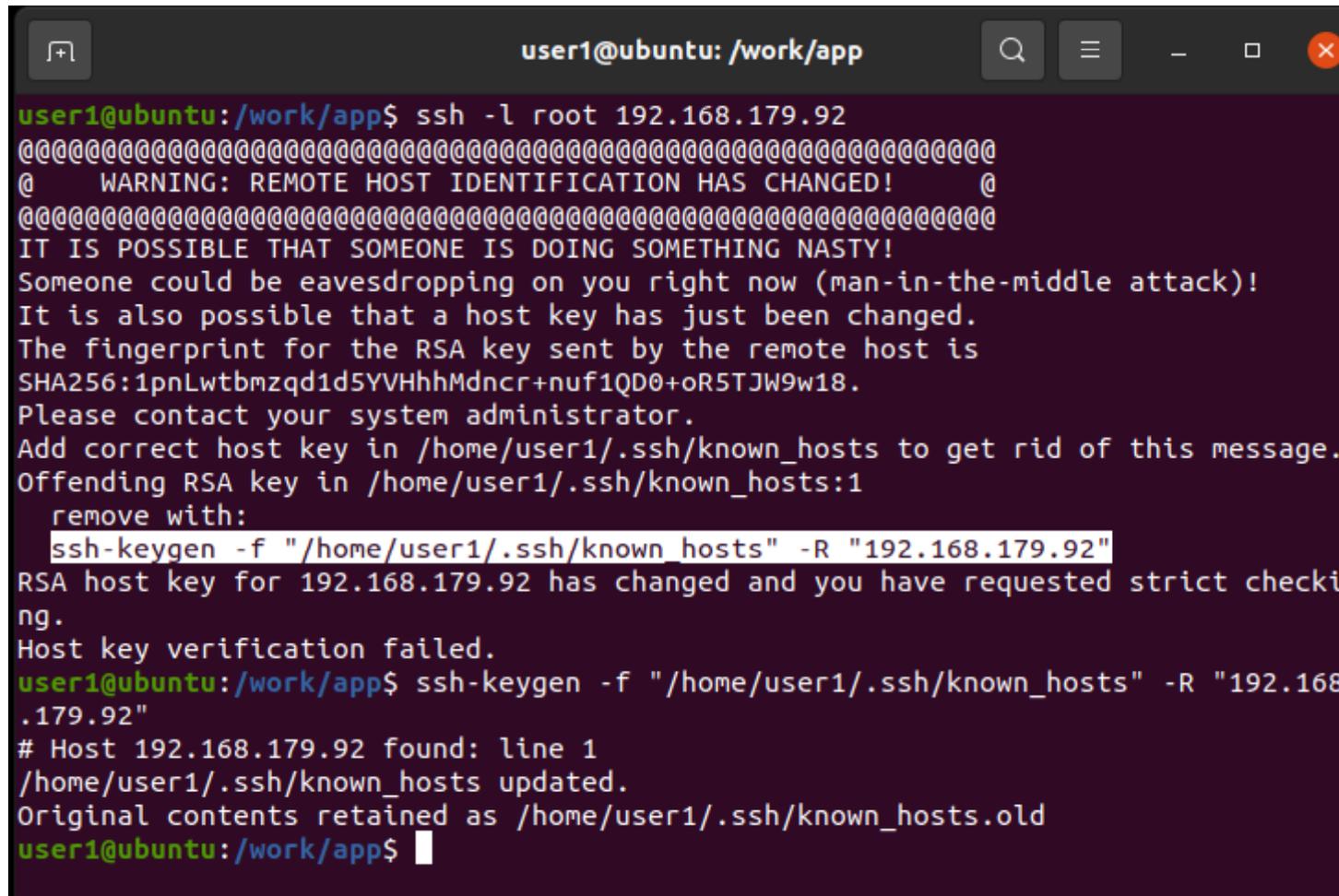
最後にunameコマンドでログインできているかを確認しています。

```
[Module]# uname -a
```

A terminal window titled 'user1@ubuntu: /work/app' showing the execution of an SSH command and a subsequent uname command. The SSH command is 'ssh -l root 192.168.179.92'. The output shows a warning about the host's authenticity, a confirmation prompt 'Are you sure you want to continue connecting (yes/no/[fingerprint])?' with 'yes' entered, and a password prompt. The terminal then shows the prompt 'root@verdin-imx8mp-07251124:~#' and the execution of 'uname -a', which outputs system information including kernel version '5.4.193-5.7.0-devel+git.f78299297185 #1 SMP PREEMPT' and architecture 'aarch64 GNU/Linux'.

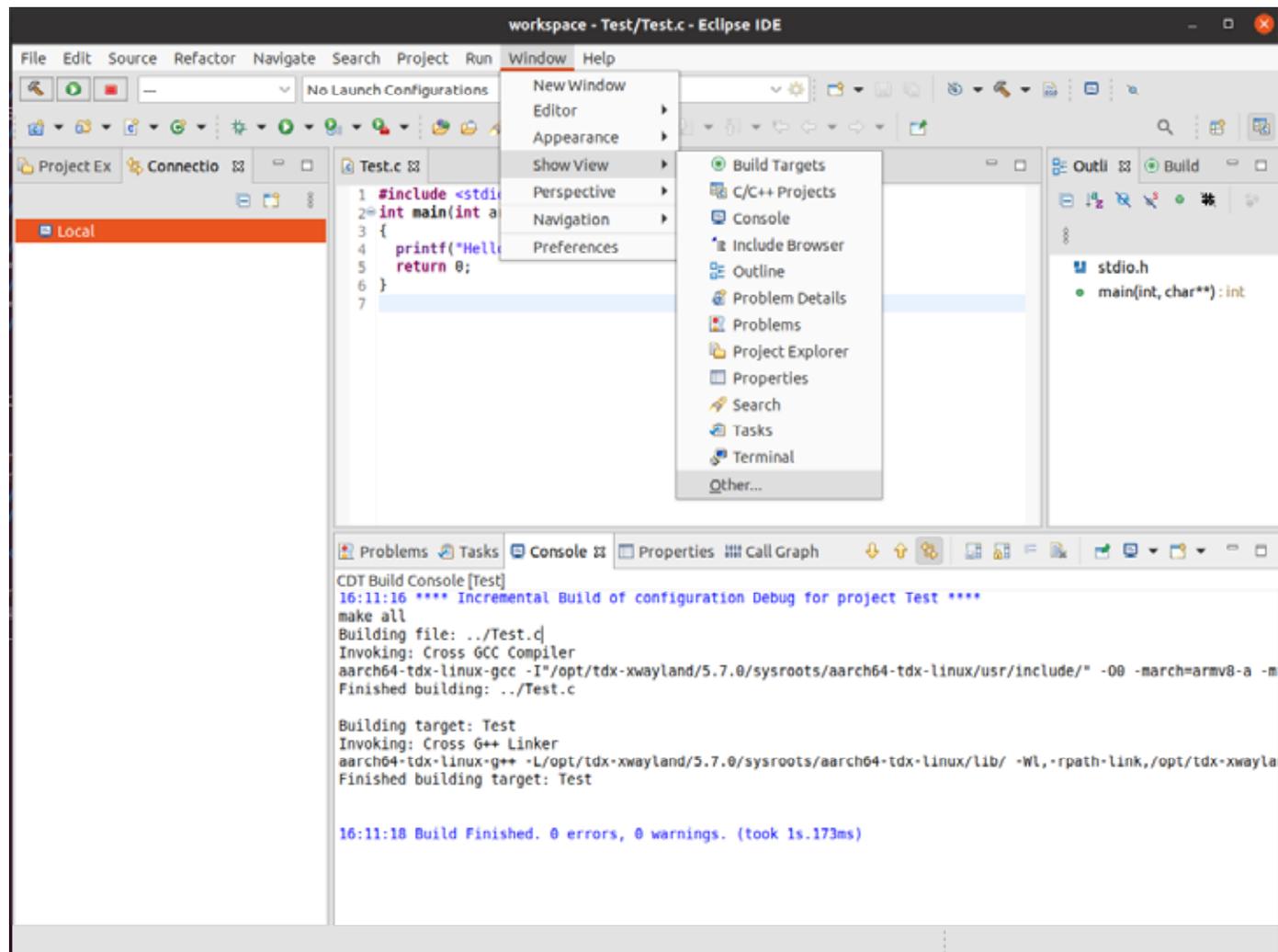
```
user1@ubuntu: /work/app$ ssh -l root 192.168.179.92
The authenticity of host '192.168.179.92 (192.168.179.92)' can't be established.
RSA key fingerprint is SHA256:1pnLwtbmzqd1d5YVHhhMdncr+nuf1QD0+oR5TJW9w18.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.179.92' (RSA) to the list of known hosts.
root@192.168.179.92's password:
root@verdin-imx8mp-07251124:~# uname -a
Linux verdin-imx8mp-07251124 5.4.193-5.7.0-devel+git.f78299297185 #1 SMP PREEMPT
Mon Jul 11 14:42:03 UTC 2022 aarch64 GNU/Linux
root@verdin-imx8mp-07251124:~#
```

sshコマンドで下記のようなエラーが出た場合
ssh-keygenでキーを一度削除してください。

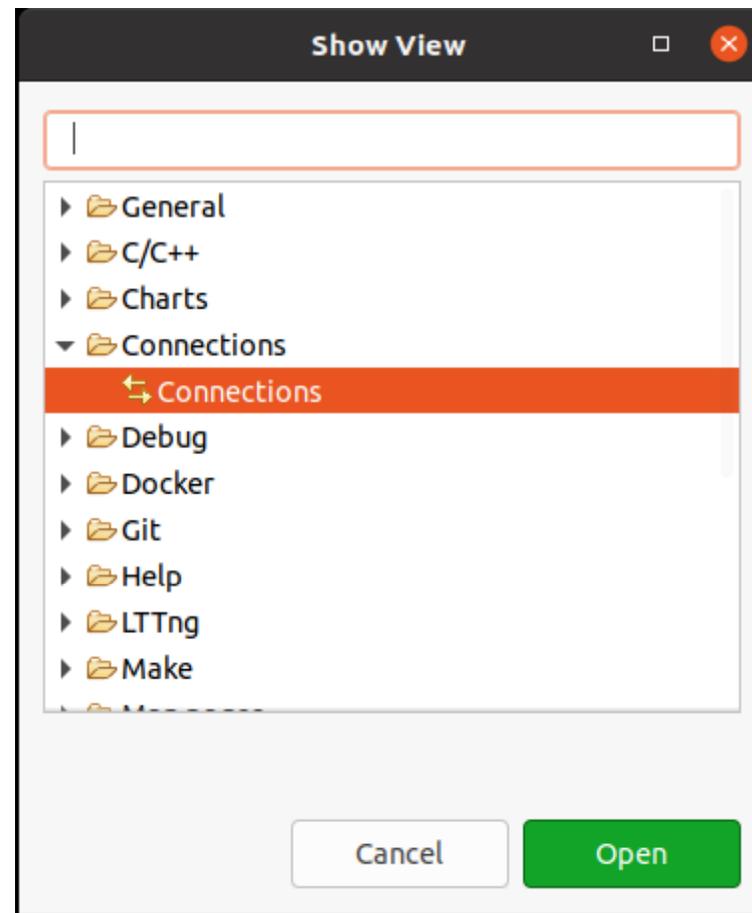


```
user1@ubuntu: /work/app
user1@ubuntu:/work/app$ ssh -l root 192.168.179.92
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:1pnLwtbmzqd1d5YVHhhMdnrc+nuf1QD0+oR5TJW9w18.
Please contact your system administrator.
Add correct host key in /home/user1/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/user1/.ssh/known_hosts:1
  remove with:
  ssh-keygen -f "/home/user1/.ssh/known_hosts" -R "192.168.179.92"
RSA host key for 192.168.179.92 has changed and you have requested strict checking.
Host key verification failed.
user1@ubuntu:/work/app$ ssh-keygen -f "/home/user1/.ssh/known_hosts" -R "192.168.179.92"
# Host 192.168.179.92 found: line 1
/home/user1/.ssh/known_hosts updated.
Original contents retained as /home/user1/.ssh/known_hosts.old
user1@ubuntu:/work/app$
```

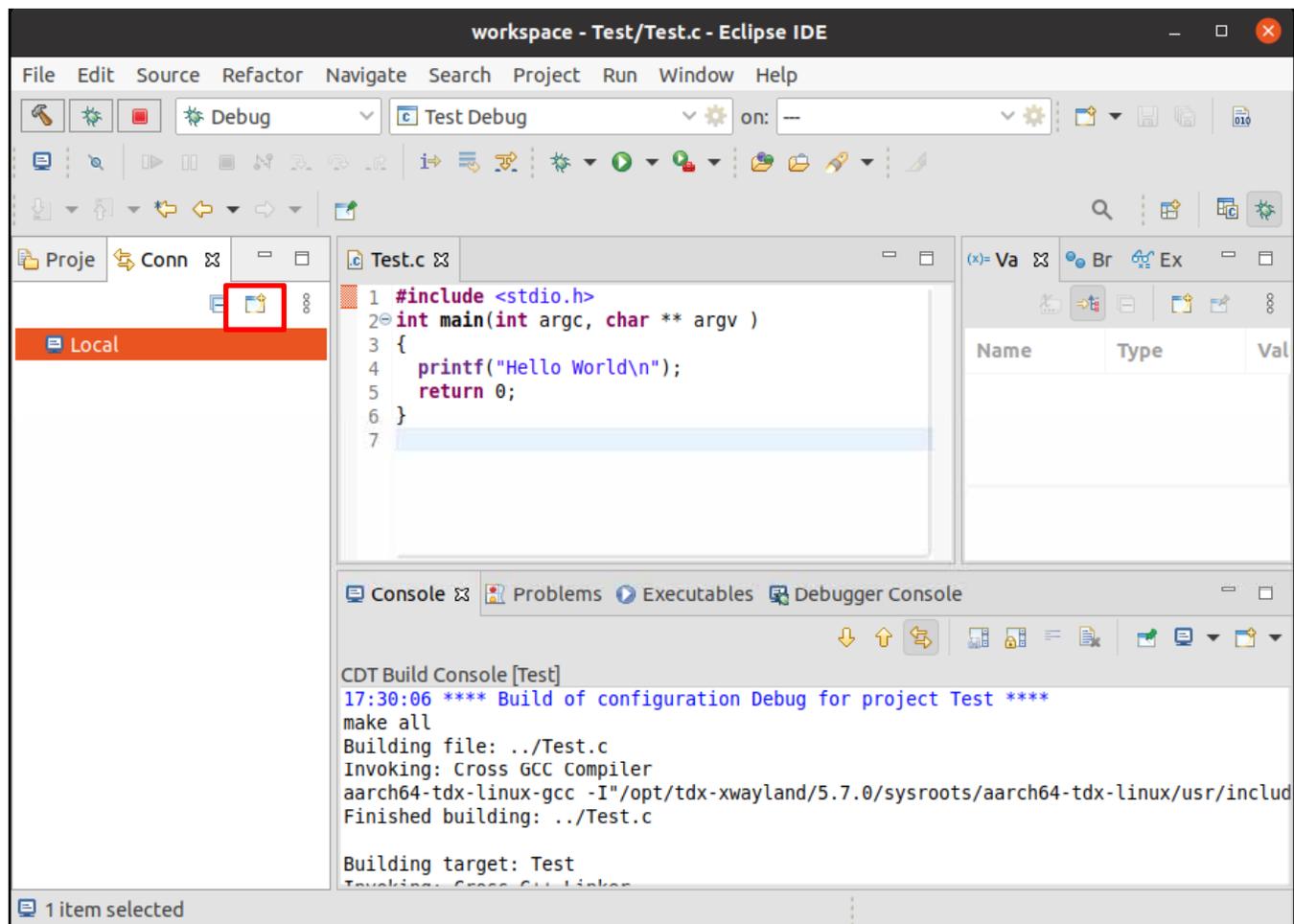
Eclipseに戻ります。EclipseのメニューのWindow > Show View > Otherを選択します。



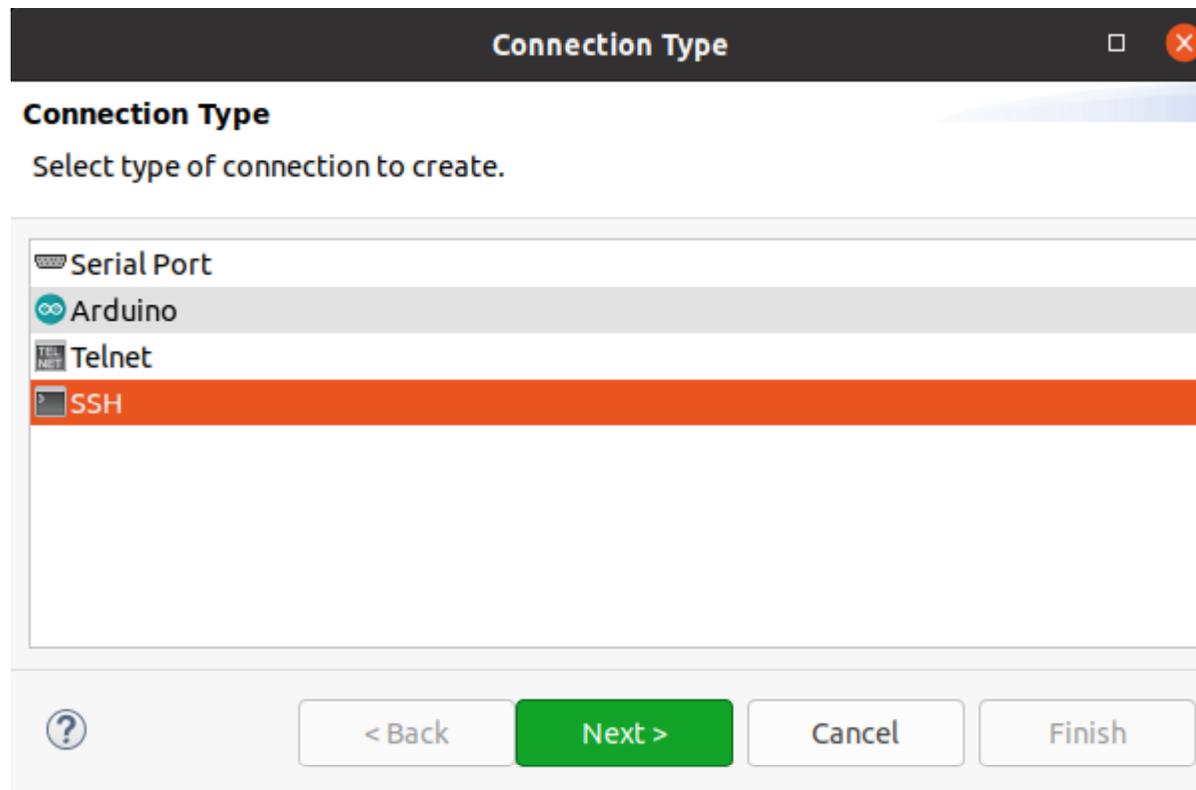
Connectionsを選択してOpenをクリックします。



+ マーク(赤枠)のアイコンをクリックしてConnection設定を追加します。



SSHを選択してNextをクリックします。



Connection nameにわかりやすい名前を付けます。

Host, User, PasswordにそれぞれモジュールのIPアドレス、ユーザー名、パスワードを入力してFinishをクリックします。

New Connection
Specify properties of a new connection

Connection name: Verdin-iMX8MP

Host information

Host: 192.168.179.92

User: root

Public key based authentication Keys are set at [Network Connections, SSH2](#)

Passphrase:

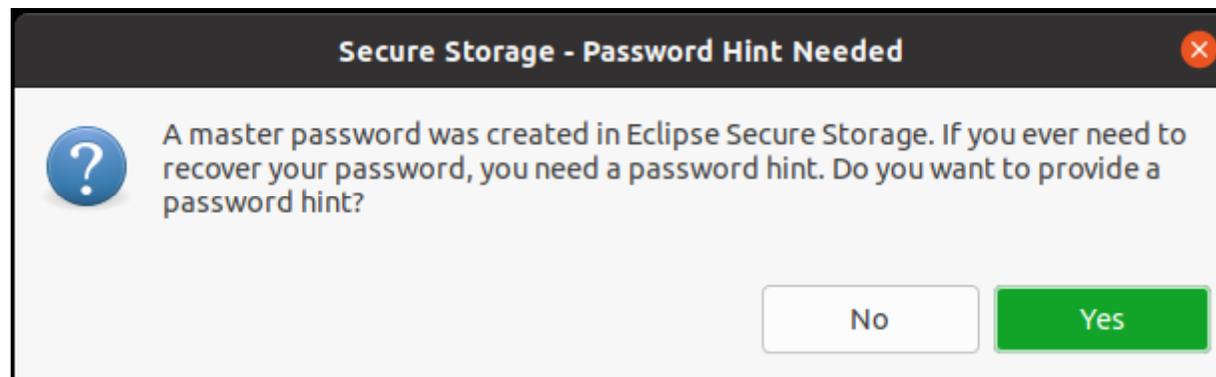
Password based authentication

Password:

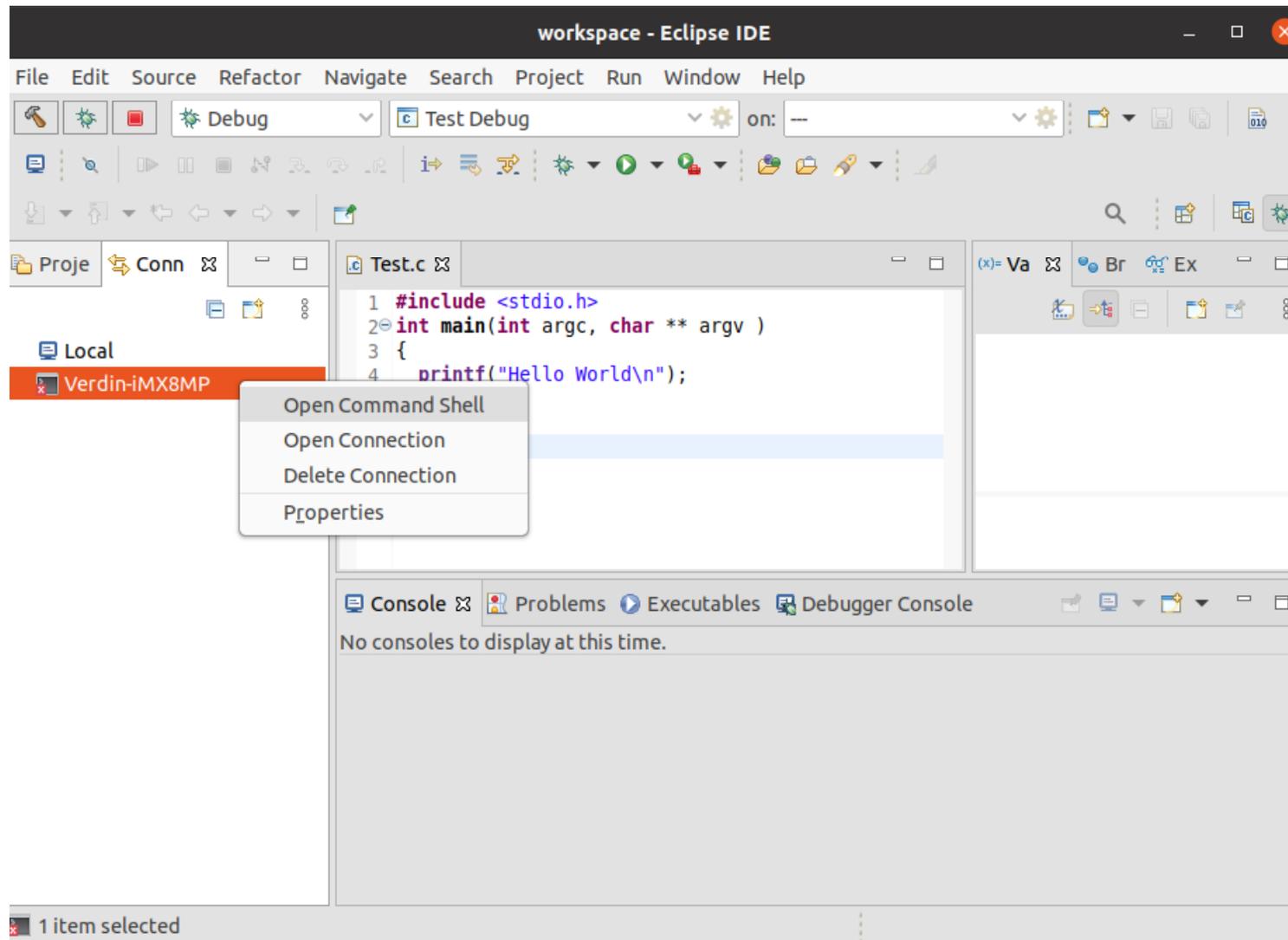
▸ Advanced

? < Back Next > Cancel Finish

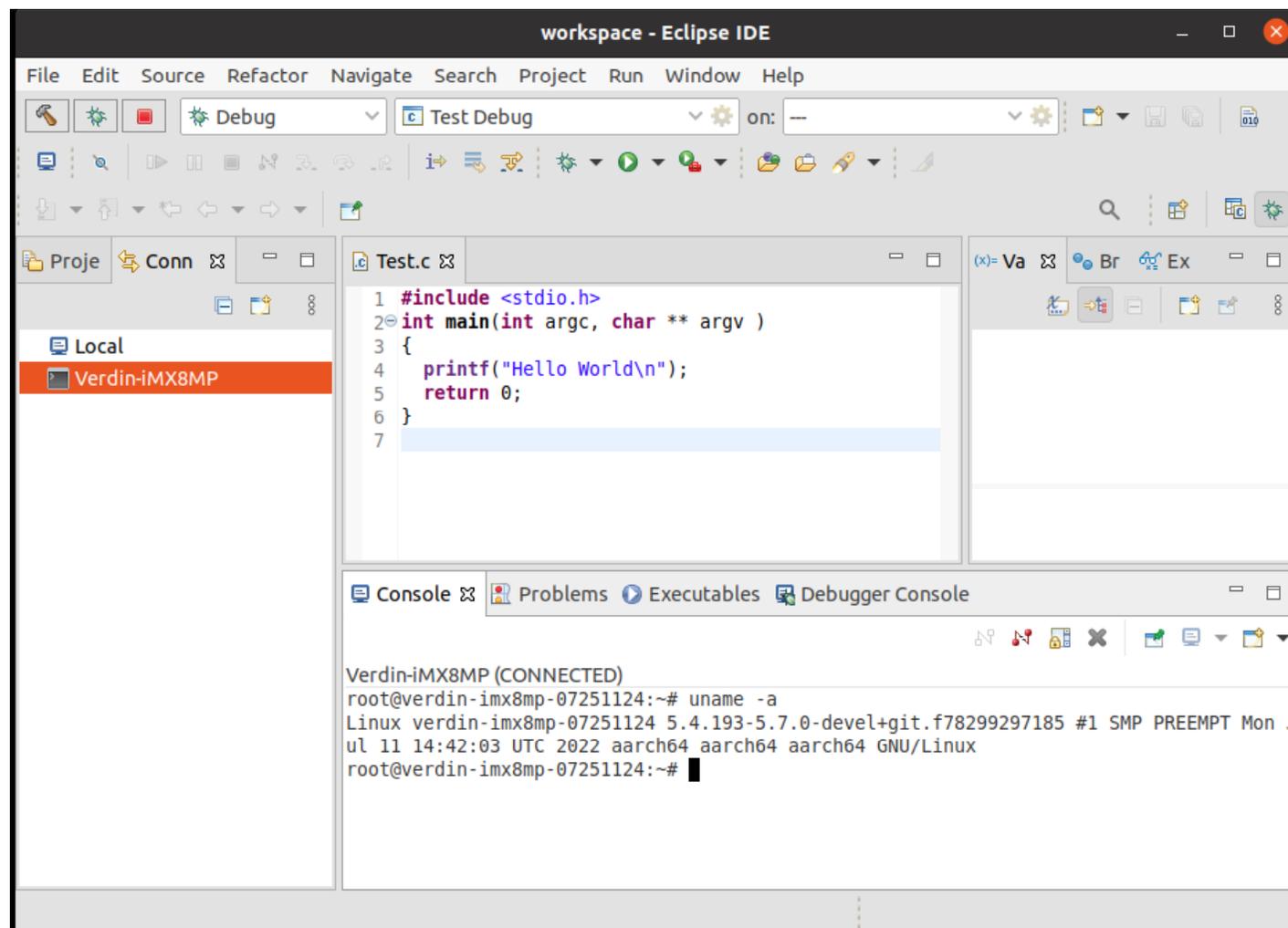
パスワードの保存を行った場合、パスワード復元用のヒントを作るかどうかを問われます。本マニュアルでは使用しないのでNoをクリックします。



作成したConnection設定を右クリックしてOpen Command Shellを選択します。



モジュールと接続できるとConsoleでコマンドが打てるようになります。下記ではunameコマンドを実行しています。



デバッグ設定作成

GDBの初期処理を記述するgdbinitファイルを作成します。本マニュアルでは/work/app/gdbinit に作成します。

```
[Ubuntu]$ gedit /work/app/gdbinit
```

内容は下記です。(環境変数は使用できません。)

```
set sysroot /opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux
```

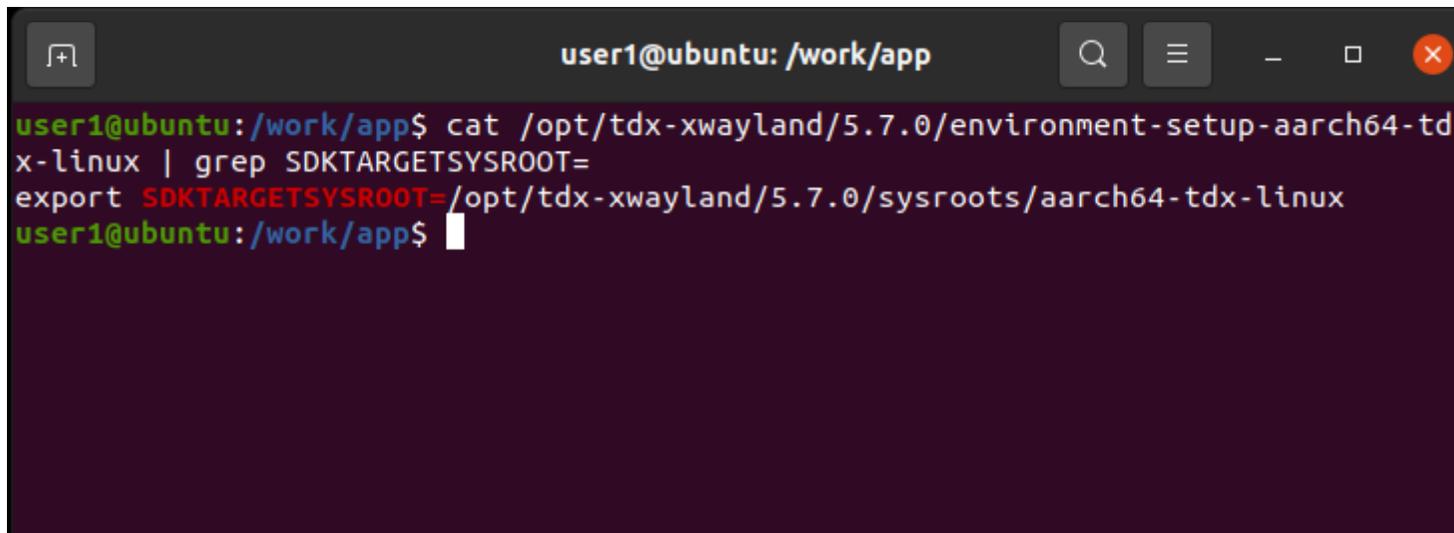
```
set auto-load safe-path /opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux
```

上記2つのコマンドで指定するパスはSDKが出力した環境変数設定シェル内の下記の内容になります。

```
SDKTARGETSYSROOT
```

下記コマンドで見ることができます。

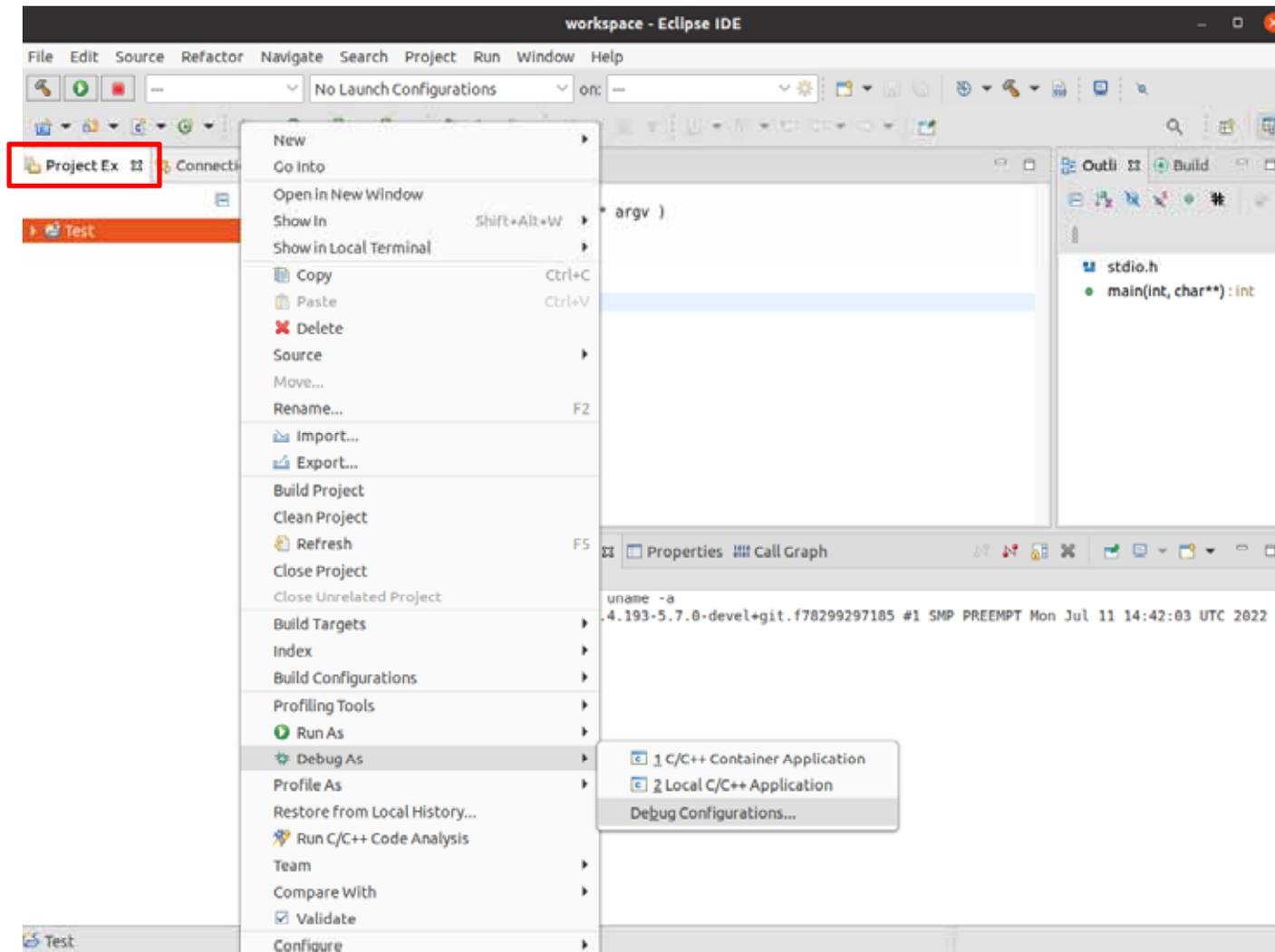
```
[Ubuntu]$ cat /opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux | grep SDKTARGETSYSROOT=
```



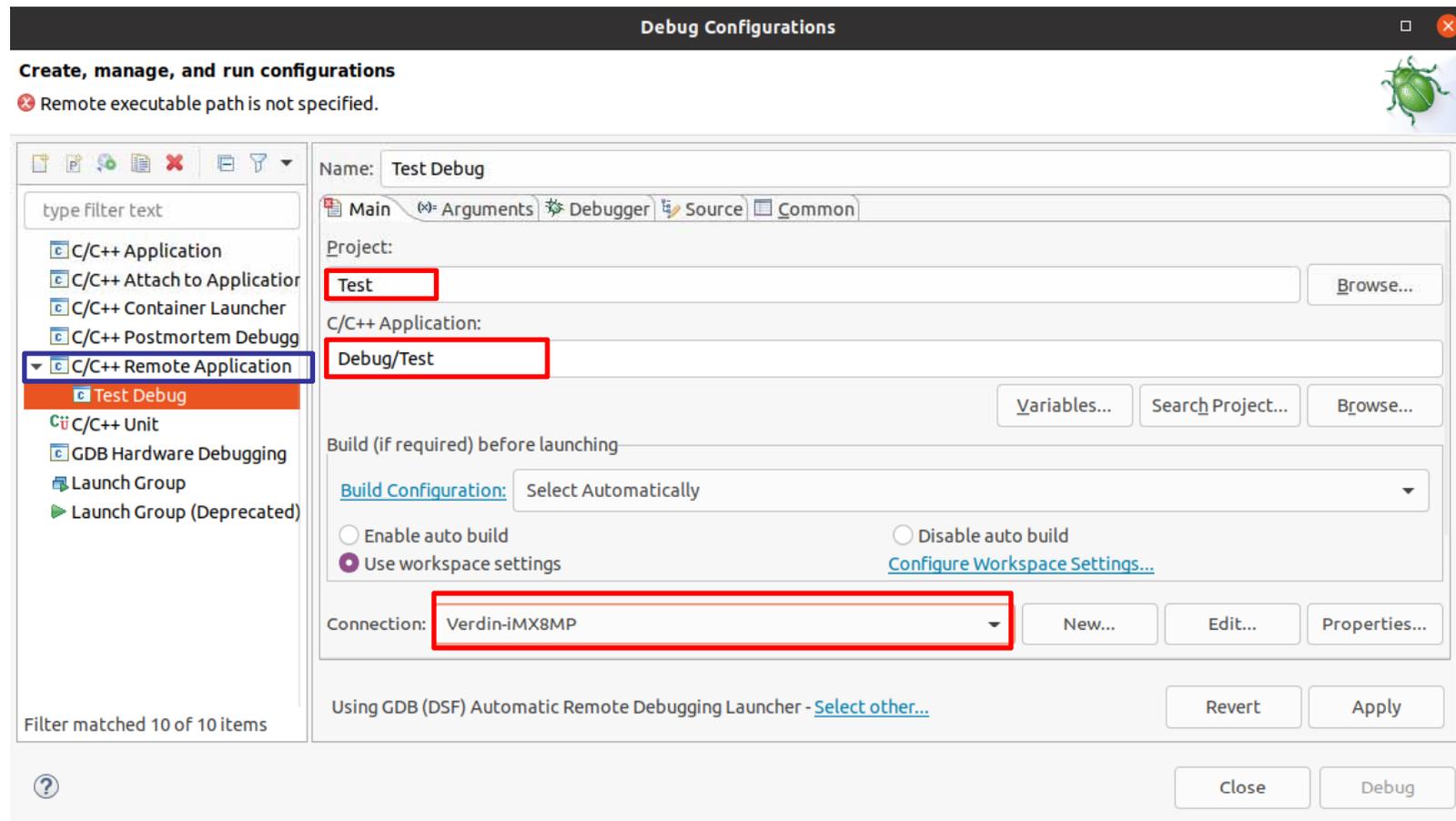
```
user1@ubuntu: /work/app
user1@ubuntu:/work/app$ cat /opt/tdx-xwayland/5.7.0/environment-setup-aarch64-tdx-linux | grep SDKTARGETSYSROOT=
export SDKTARGETSYSROOT=/opt/tdx-xwayland/5.7.0/sysroots/aarch64-tdx-linux
user1@ubuntu:/work/app$
```

デバッグ

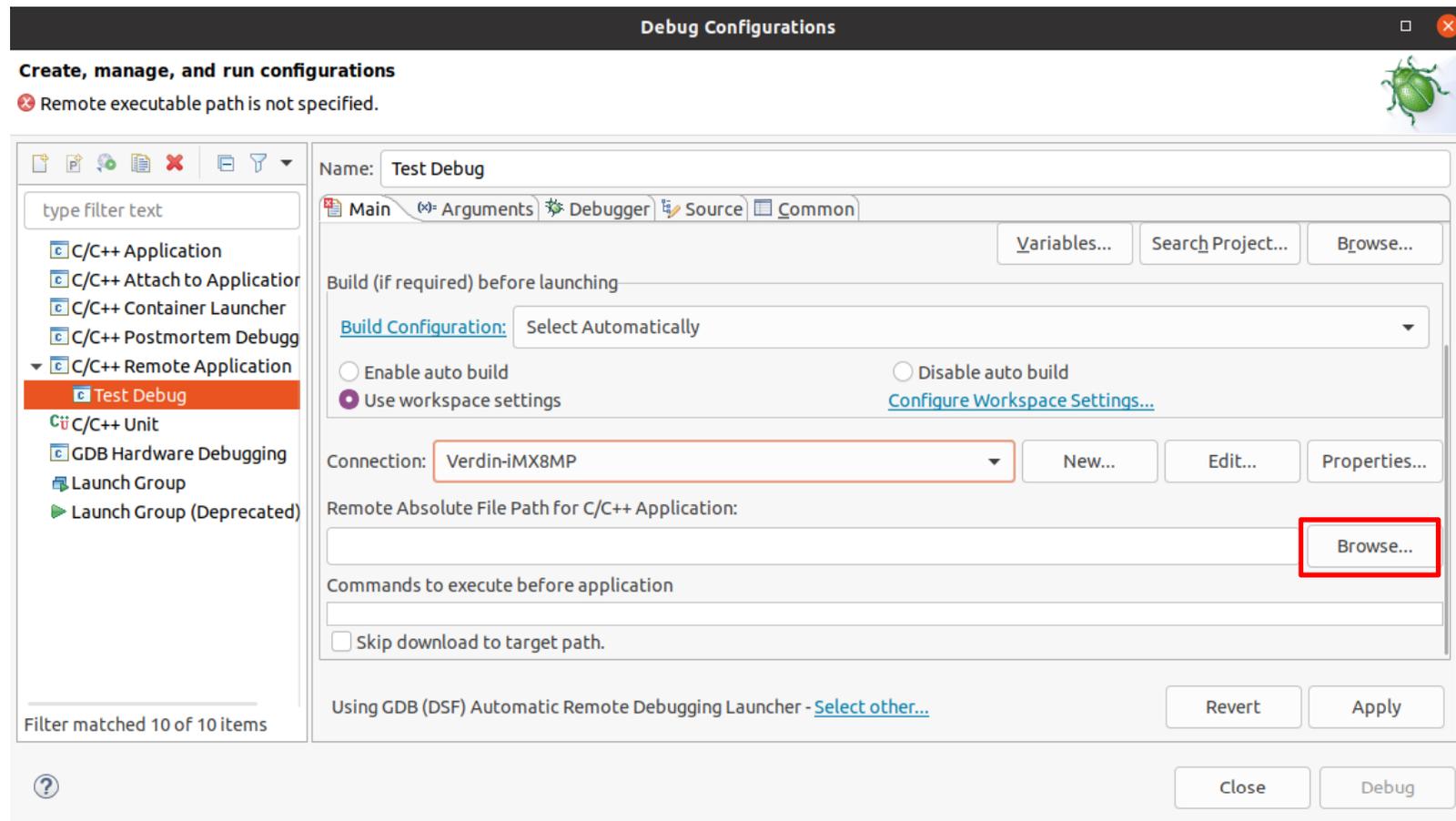
プロジェクトエクスプローラに戻り(赤枠をクリック)、プロジェクトを右クリックしDebug As > Debug Configurationsを選択します。



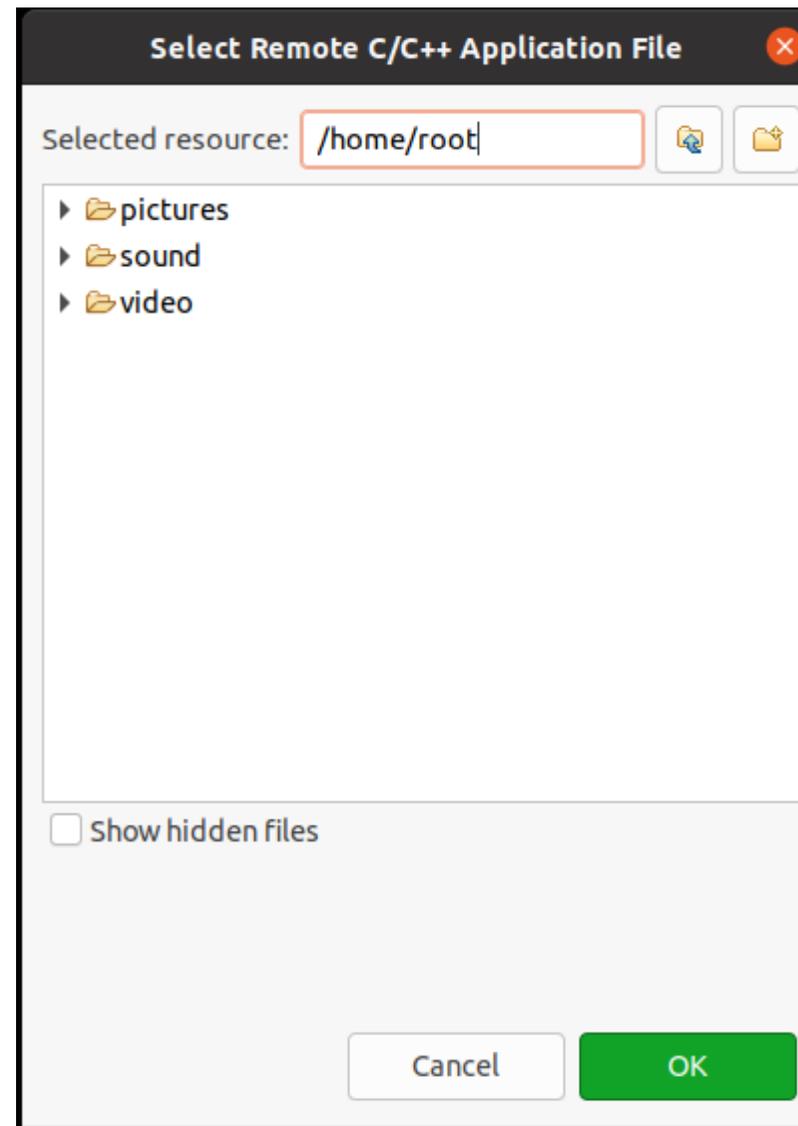
C/C++ Remote Application(青枠)をダブルクリックしデバッグ設定を追加します。赤枠内の設定を行います。
Projectと C/C++ Applicationは自動で設定されるままで問題ありません。
Connectionは先ほど作成した設定を選択します。



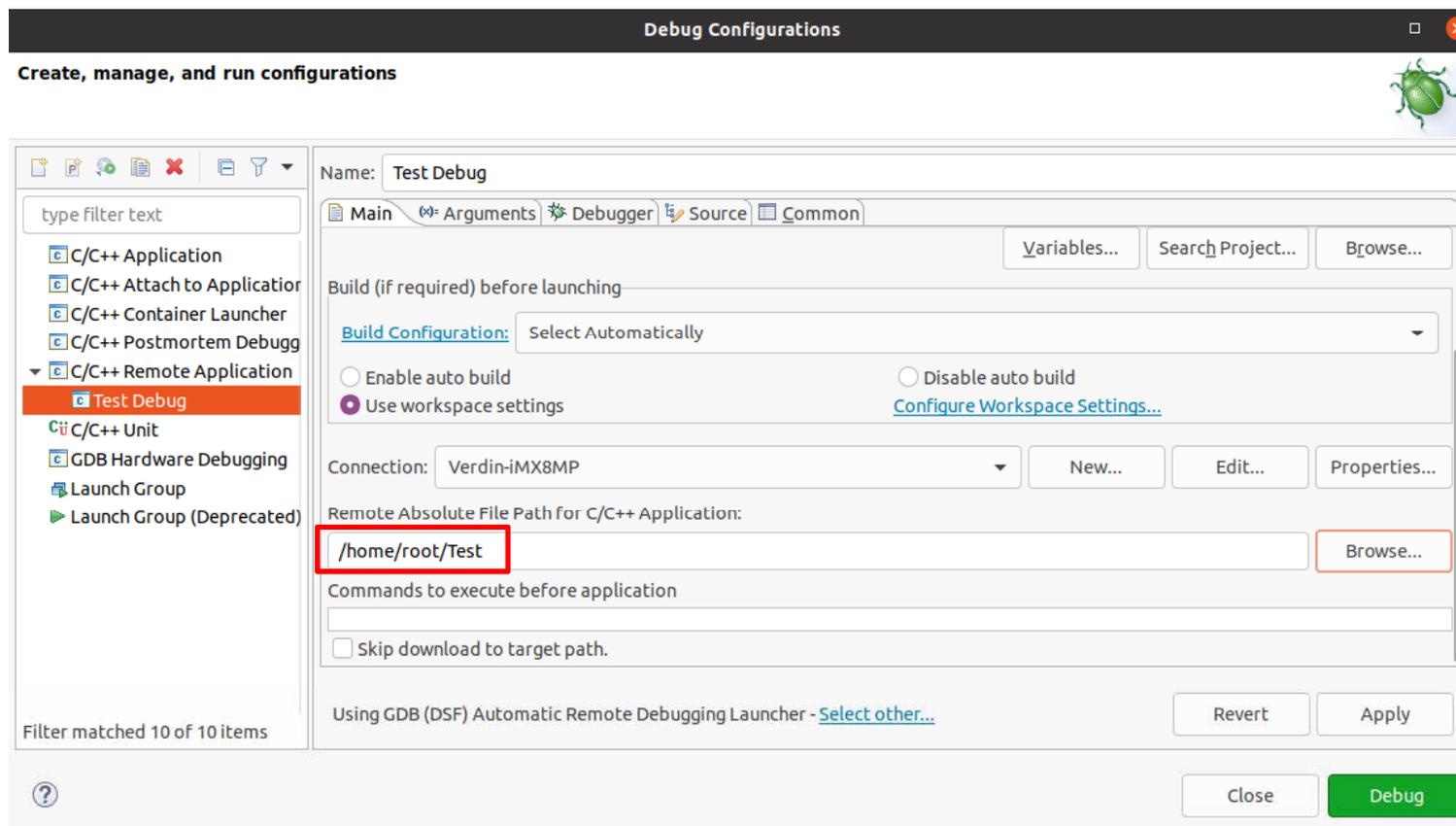
Browseをクリックします。



接続ができていればモジュール内のホームディレクトリが見えます。そのままOKをクリックします。



自動的にRemote Absolute File Path for C/C++ Applicationに実行パスが設定されます。



gdbのパスを確認します。

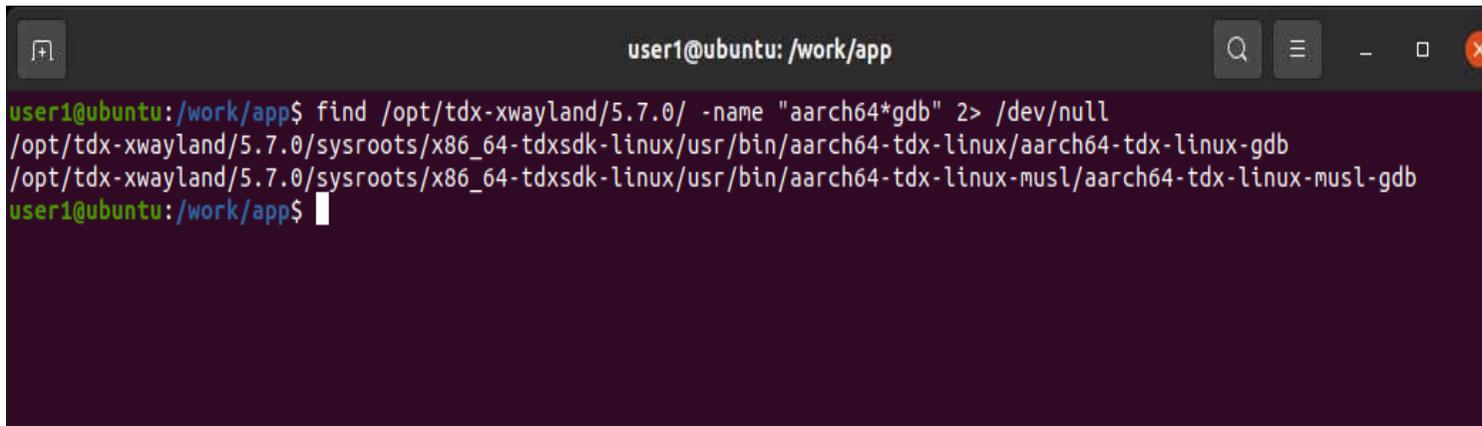
上記で指定するパスはSDK配下をaarch64用のgdbを探します。

```
find /opt/tdx-xwayland/5.7.0/ -name "aarch64*gdb" 2> /dev/null
```

使用するのはmuslがついていないほうです。

本マニュアルでは下記になります。

```
/opt/tdx-xwayland/5.7.0/sysroots/x86_64-tdxsdk-linux/usr/bin/aarch64-tdx-linux/aarch64-tdx-linux-gdb
```



```
user1@ubuntu: /work/app
user1@ubuntu:/work/app$ find /opt/tdx-xwayland/5.7.0/ -name "aarch64*gdb" 2> /dev/null
/opt/tdx-xwayland/5.7.0/sysroots/x86_64-tdxsdk-linux/usr/bin/aarch64-tdx-linux/aarch64-tdx-linux-gdb
/opt/tdx-xwayland/5.7.0/sysroots/x86_64-tdxsdk-linux/usr/bin/aarch64-tdx-linux-musl/aarch64-tdx-linux-musl-gdb
user1@ubuntu:/work/app$
```

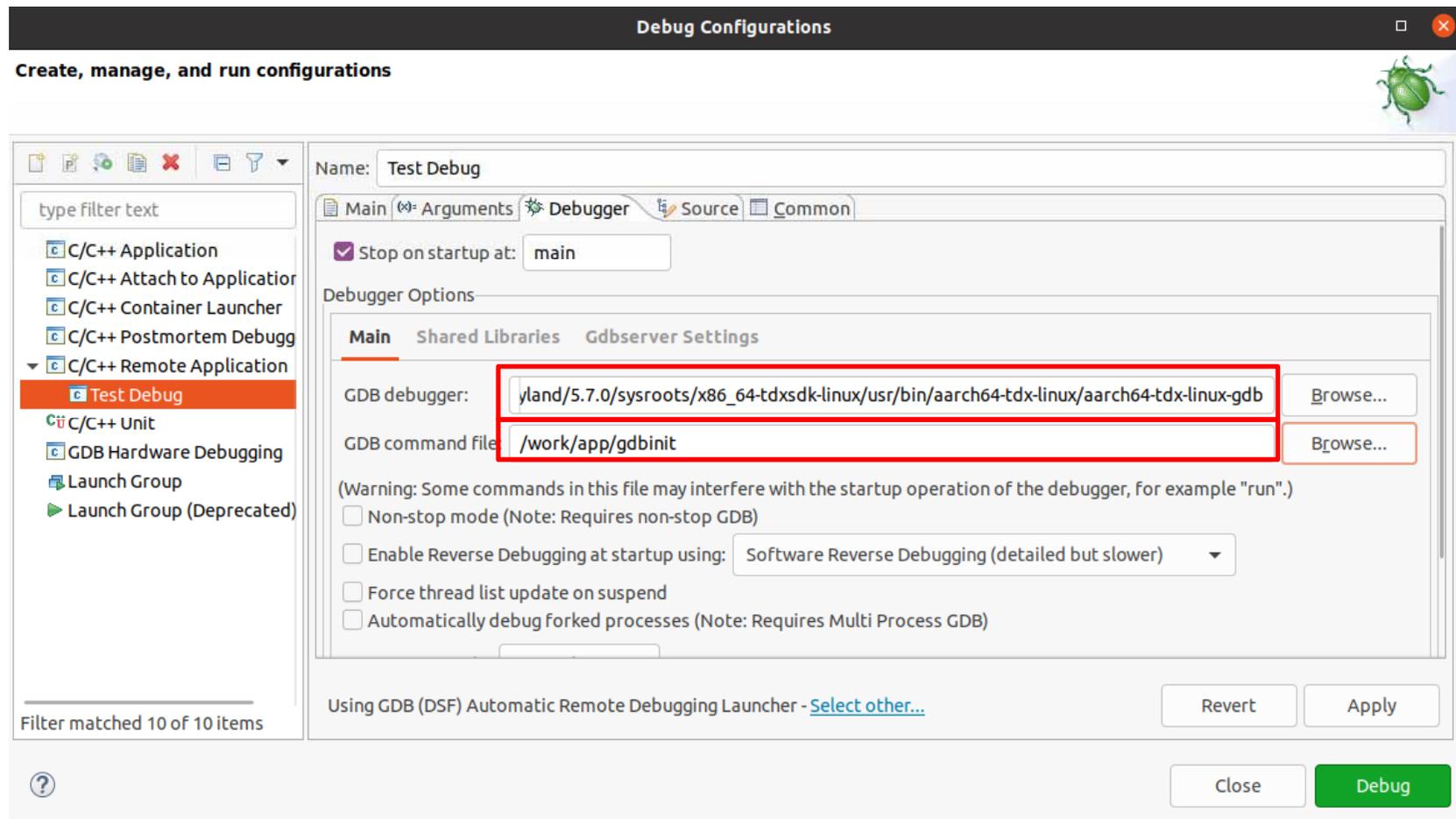
Debuggerのタブを開きます。GDB debuggerに下記を指定します。

/opt/tdx-xwayland/5.7.0/sysroots/x86_64-tdxsdk-linux/usr/bin/aarch64-tdx-linux/aarch64-tdx-linux-gdb

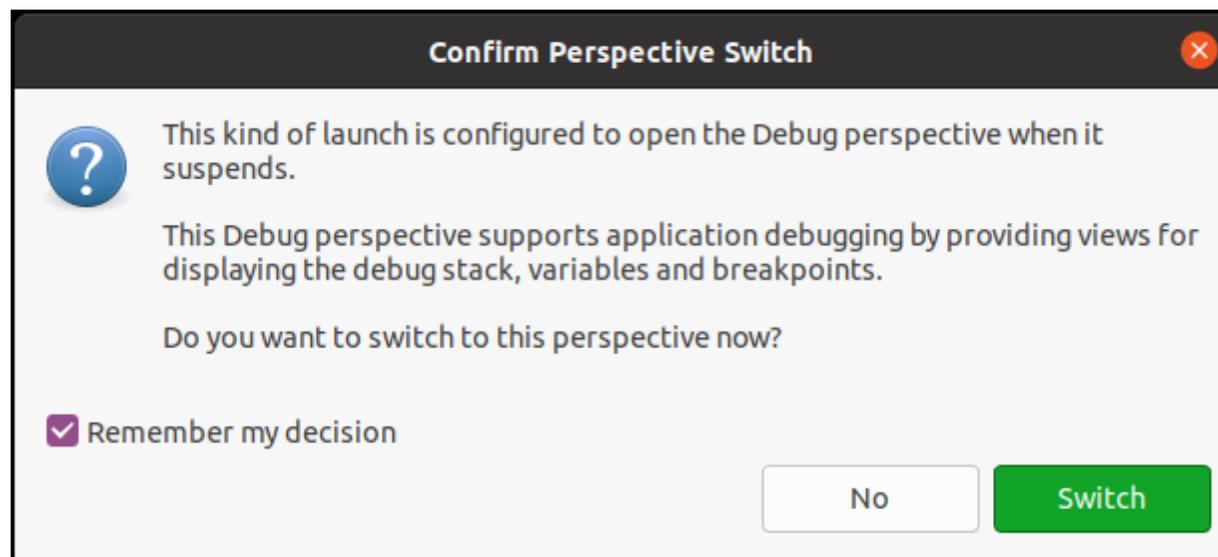
GDB command fileには先ほど作成したgdbinitのパスを指定します。本マニュアルでは下記になります。

/work/app/gdbinit

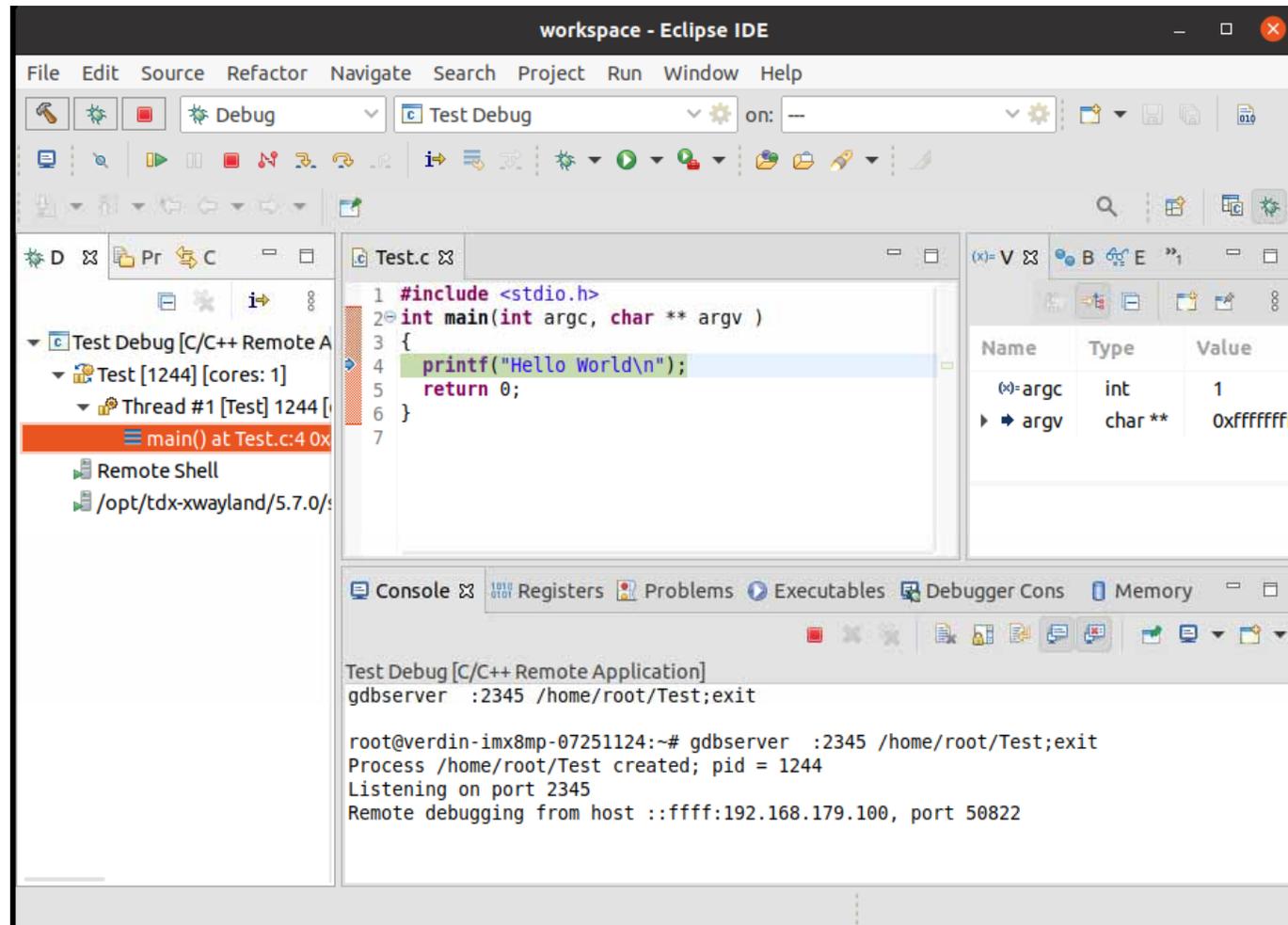
Applyをクリック後Debugをクリックします。



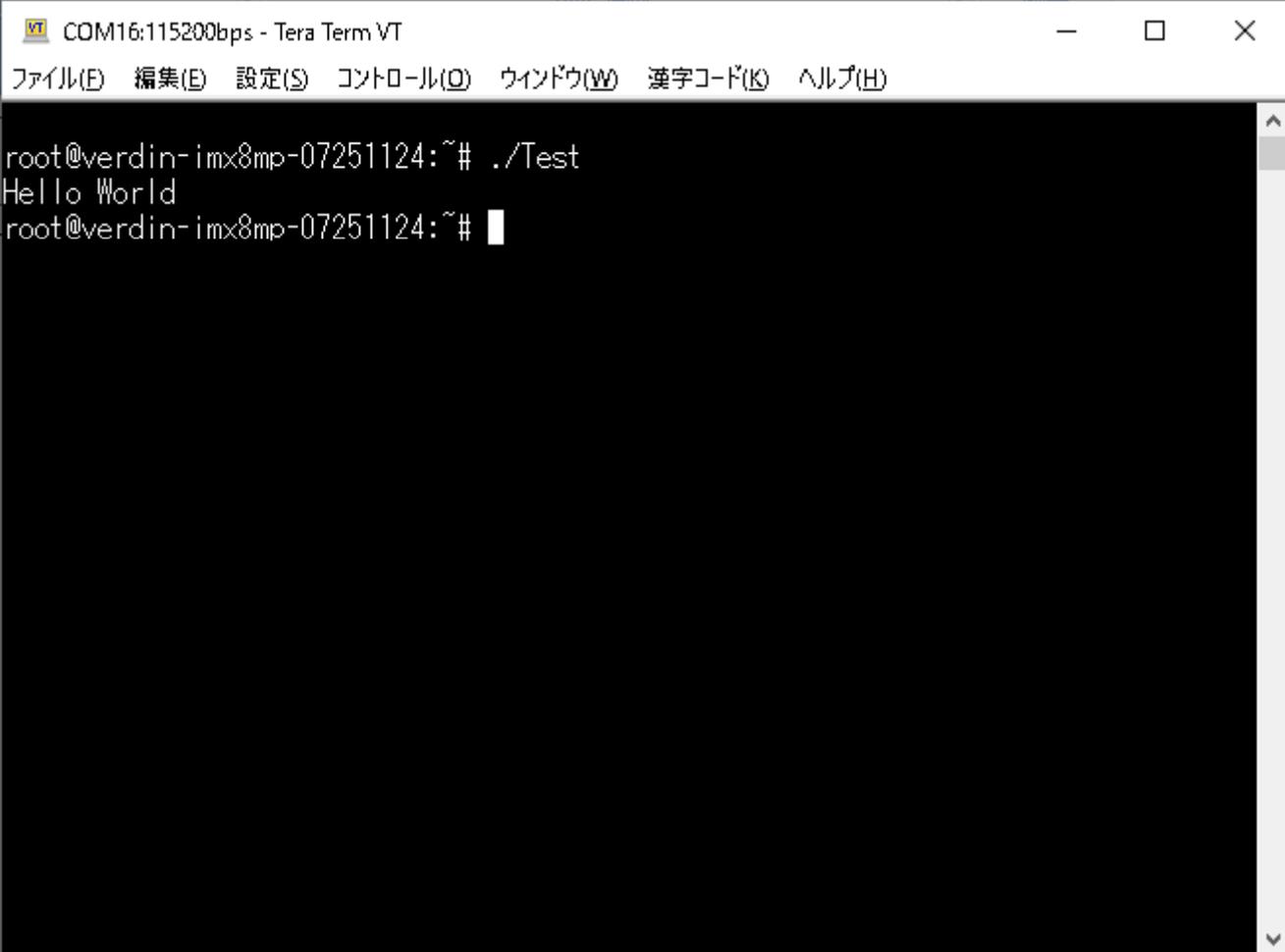
Debugが始まると下記のようなDebugウィンドウを開くかどうか問われますのでSwitchをクリックします。
この操作を覚える場合はRemember my decisionにチェックを入れておきます。



デバッグウィンドウが表示されデバッグを行うことができます。



デバッグを行うと自動的に実行環境の/root/homeに実行ファイルがコピーされています。実行可能です。



```
COM16:115200bps - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) 漢字コード(K) ヘルプ(H)
root@verdin-imx8mp-07251124:~# ./Test
Hello World
root@verdin-imx8mp-07251124:~# █
```

Releaseビルドで実行ファイルを作成

デバッグを行いプログラムを完成させた後はリリースビルドで最適化を行ったバージョンの実行ファイルを作成します。プロジェクトを右クリック Build Configurations > Set Active > Releaseを選択してReleaseに変更後ビルドボタンを押しビルドを行います。ビルドした実行ファイルは/work/app/workspace/Test/Releaseに格納されています。この実行ファイルをSDカードなどでモジュールに移動し実行することができます。

以上がC/C++言語アプリケーションの開発の環境構築からデバッグ、実行ファイル作成までの手順です。

