

Intro to Embedded Linux Part 6 - Add Custom Application to Yocto Build

[By ShawnHymel](#)

In [the previous tutorial](#), we showed you how to enable an I2C port using the Yocto Project. This time, we will use that I2C port to communicate with a TMP102 temperature sensor to print the temperature reading to the console. We will use bitbake to build the application into our deployed Linux image. Note that you have the option of creating a simple "Hello, world!" application instead if you'd like to see how this process works without connecting a sensor.

See [here](#) if you would like to view this tutorial in video format:

Required Hardware

I will try to explain what is happening at each step in these tutorials so that you can generalize the instructions to almost any single board computer (assuming the board is supported by the build system). However, for the demonstration, I will be using the following platform:

[STM32MP157D-DK1](#)

You will also need an SD card. The STM32MP157D-DK1 kit should come with an SD card. In addition, you will need a USB-C power supply capable of supplying 5V, 3A.

For this I2C demo, you will also need a [TMP102 temperature sensor breakout board](#).

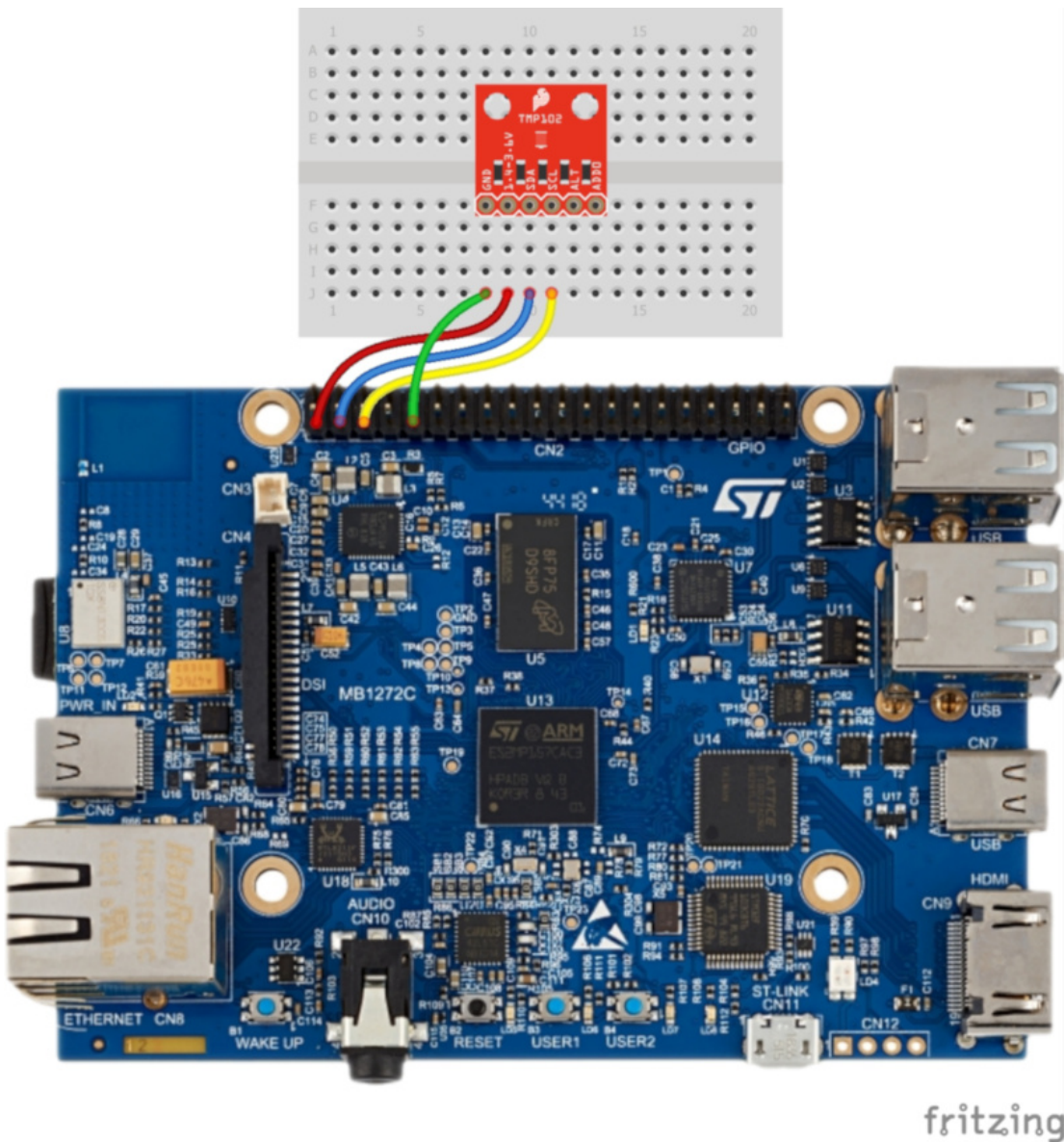
Required Software

You will need Linux for this project, as all of the tools we are using must be run in Linux (on the host computer). I will show steps that work in Ubuntu and Linux Mint (and likely other flavors of Debian), but you can probably get almost any Linux distro to work. LiveCD, dual-booting, Windows Subsystem for Linux (WSL), and pre-made Docker images will also likely work.

I also recommend using a fairly modern computer with at least 4GB of RAM. While you can probably build a Linux image in a Raspberry Pi, expect it to take a very long time.

Hardware Connections

As from the previous tutorial, make sure you have a TMP102 temperature sensor connected to your STM32MP157D-DK1 as follows:



Create Application

Create a C source code file in your custom layer (that we created in [Part 4](#)). Note that you can develop an application on another Linux system with similar I2C settings. You will likely need various debugging tools enabled (that we don't have on our custom image) to help you create the application.

Create a *recipes-apps/* directory in your custom layer and store the source code in *mytemp/files/src* directory:

Copy Code

```
cd ~/Projects/yocto/meta-custom
mkdir -p recipes-apps/mytemp/files/src
cd recipes-apps/mytemp
vi files/src/gettemp.c
```

Copy in the following code:

Copy Code

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
```

```

#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <fcntl.h>

int main()
{
    // Settings
    const unsigned char tmp102_addr = 0x48; // I2C address of the TMP102
    const unsigned char reg_temp = 0x00;    // Address of temperature register
    const char filename[] = "/dev/i2c-1";    // Location of I2C device file

    int file;
    char buf[5];
    int16_t temp_buf;
    float temp_c;

    // Open the device file for read/write
    if ((file = open(filename, O_RDWR)) < 0)
    {
        printf("Failed to open the bus.\n");
        exit(1);
    }

    // Change to I2C address of TMP102
    if (ioctl(file, I2C_SLAVE, tmp102_addr) < 0) {
        printf("Failed to acquire bus access or talk to device.\n");
        exit(1);
    }

    // Start read by writing location of temperature register
    buf[0] = 0x00;
    if (write(file, buf, 1) != 1)
    {
        printf("Could not write to I2C device.\n");
        exit(1);
    }

    // Read temperature
    if (read(file, buf, 2) != 2)
    {
        printf("Could not read from I2C device.\n");
        exit(1);
    }

    // Combine received bytes to single 16-bit value
    temp_buf = (buf[0] << 4) | (buf[1] >> 4);

    // If value is negative (2s complement), pad empty 4 bits with 1s
    if (temp_buf > 0x7FFF)
    {
        temp_buf |= 0xFFFF;
    }

    // Convert sensor reading to temperature (Celsius)
    temp_c = temp_buf * 0.0625;

    // Print results
    printf("%.2f deg C\n", temp_c);

    return 0;
}

```

Alternatively, you could try this with a simple "Hello, world!" program if you don't want to mess with I2C and other hardware. Either will get built as the application *gettemp* in */usr/bin* in our deployed image.

Copy Code

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

Save and exit.

Create Recipe

Now, we must tell bitbake where to find the source code for our new application, cross-compile it for our machine, and include it in our image.

Yocto can be picky about licenses, as it generally wants to see a licence (which can be open or closed source) for each piece of source code. We're going to include the MIT license, which is given as a text file in poky (`~/Projects/yocto/poky/meta/files/common-licenses/MIT`). We need the MD5 hash of that file to include in our recipe:

Copy Code

```
md5sum ~/Projects/yocto/poky/meta/files/common-licenses/MIT
```

Copy the output of that process (it should be a string of letters and numbers). Next, create the gettemp recipe. Paste in the MD5 hash where you see "md5=..."

Copy Code

```
vi gettemp_0.1.bb
```

Note that we are creating this file in the *recipes-apps/mytemp* directory in our custom layer. The location of files can be important in Yocto, as recipes will often search for files relative to their current location. Specifically, it will implicitly look in the *files* directory relative to the recipe's location when searching for source files. We specify "file://src" to mean `./files/src`.

Copy in the following text to create our recipe:

Copy Code

```
SUMMARY = "Get temperature recipe"
DESCRIPTION = "Custom recipe to build gettemp.c application"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

# Where to find source files (can be local, GitHub, etc.)
SRC_URI = "file://src"

# Where to keep downloaded source files (in tmp/work/...)
S = "${WORKDIR}/src"

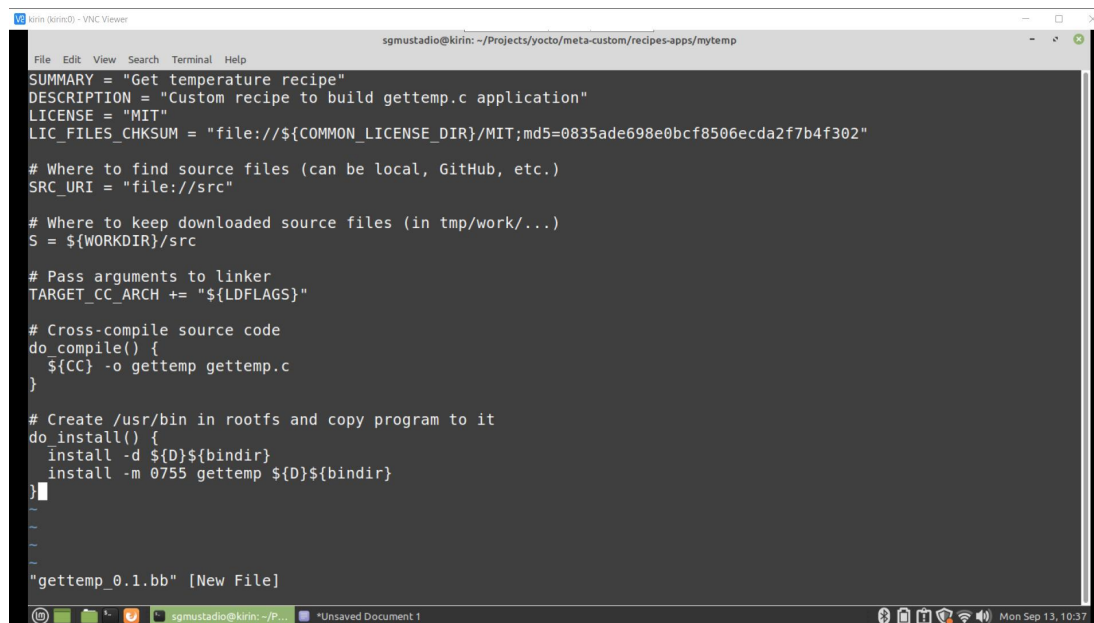
# Pass arguments to linker
TARGET_CC_ARCH += "${LDFLAGS}"

# Cross-compile source code
do_compile() {
    ${CC} -o gettemp gettemp.c
}

# Create /usr/bin in rootfs and copy program to it
do_install() {
    install -d ${D}${bindir}
```

```
}
install -m 0755 gettemp ${D}${bindir}
```

Your application recipe should look like the following:



```
File Edit View Search Terminal Help
sgmustadio@kirin: ~/Projects/yocto/meta-custom/recipes-apps/mytemp

SUMMARY = "Get temperature recipe"
DESCRIPTION = "Custom recipe to build gettemp.c application"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

# Where to find source files (can be local, GitHub, etc.)
SRC_URI = "file://src"

# Where to keep downloaded source files (in tmp/work/...)
S = ${WORKDIR}/src

# Pass arguments to linker
TARGET_CC_ARCH += "${LDFLAGS}"

# Cross-compile source code
do_compile() {
    ${CC} -o gettemp gettemp.c
}

# Create /usr/bin in rootfs and copy program to it
do_install() {
    install -d ${D}${bindir}
    install -m 0755 gettemp ${D}${bindir}
}

"gettemp_0.1.bb" [New File]
```

Save and exit.

By default, this recipe is not included in our image, so we have to tell our image recipe to call this recipe.

Copy Code

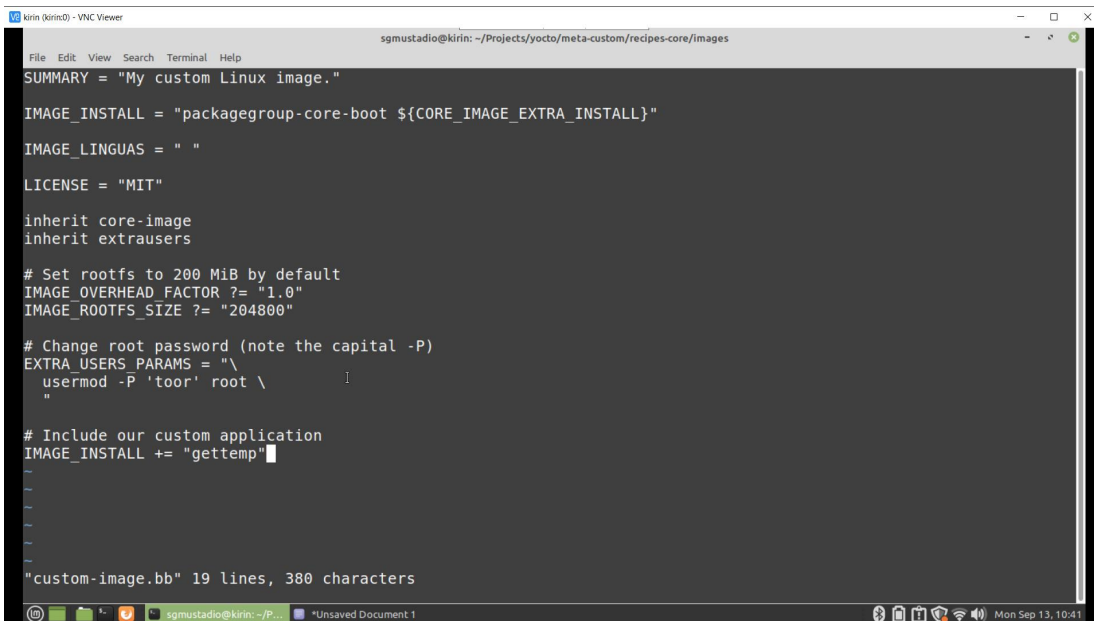
```
cd ../../recipes-core/images
vi custom-image.bb
```

Append the recipe name ("gettemp") to the IMAGE_INSTALL variable. Note that you could also put this line in build-mp1/conf/local.conf if you did not want the default image to include the application.

Copy Code

```
IMAGE_INSTALL += "gettemp"
```

Your custom-image.bb file should look like the following:



```
File Edit View Search Terminal Help
sgmustadio@kirin: ~/Projects/yocto/meta-custom/recipes-core/images

SUMMARY = "My custom Linux image."

IMAGE_INSTALL = "packagegroup-core-boot ${CORE_IMAGE_EXTRA_INSTALL}"

IMAGE_LINGUAS = " "

LICENSE = "MIT"

inherit core-image
inherit extrausers

# Set rootfs to 200 MiB by default
IMAGE_OVERHEAD_FACTOR ?= "1.0"
IMAGE_ROOTFS_SIZE ?= "204800"

# Change root password (note the capital -P)
EXTRA_USERS_PARAMS = "\
    usermod -P 'toor' root \
"

# Include our custom application
IMAGE_INSTALL += "gettemp"

"custom-image.bb" 19 lines, 380 characters
```

Save and exit.

Build and Flash Image

Enable the build environment:

Copy Code

```
cd ~/Projects/yocto
source poky/oe-init-build-env build-mp1/
```

Build the custom image with:

Copy Code

```
bitbake custom-image
```

When the build process is done, you should be able to see that our cross-compiled application is included in the deploy rootfs:

Copy Code

```
ls tmp/work/stm32mp1-poky-linux-gnueabi/custom-image/1.0-r0/rootfs/usr/bin/
```

In there, you should see the “gettemp” binary.

Flash the root filesystem (we don’t need to flash bootfs, as we did not modify the kernel):

Copy Code

```
sudo umount /media/sgmustadio/bootfs
sudo umount /media/sgmustadio/rootfs
sudo dd if=tmp/deploy/images/stm32mp1/custom-image-stm32mp1.ext4 of=/dev/mmcblk2p5 bs=1M
```

Test

Plug the SD card into the STM32MP157D-DK1 and apply power to the board. Connect to the board with a serial terminal (ttyACM0 might be different depending on your particular host computer):

Copy Code

```
picocom -b 115200 /dev/ttyACM0
```

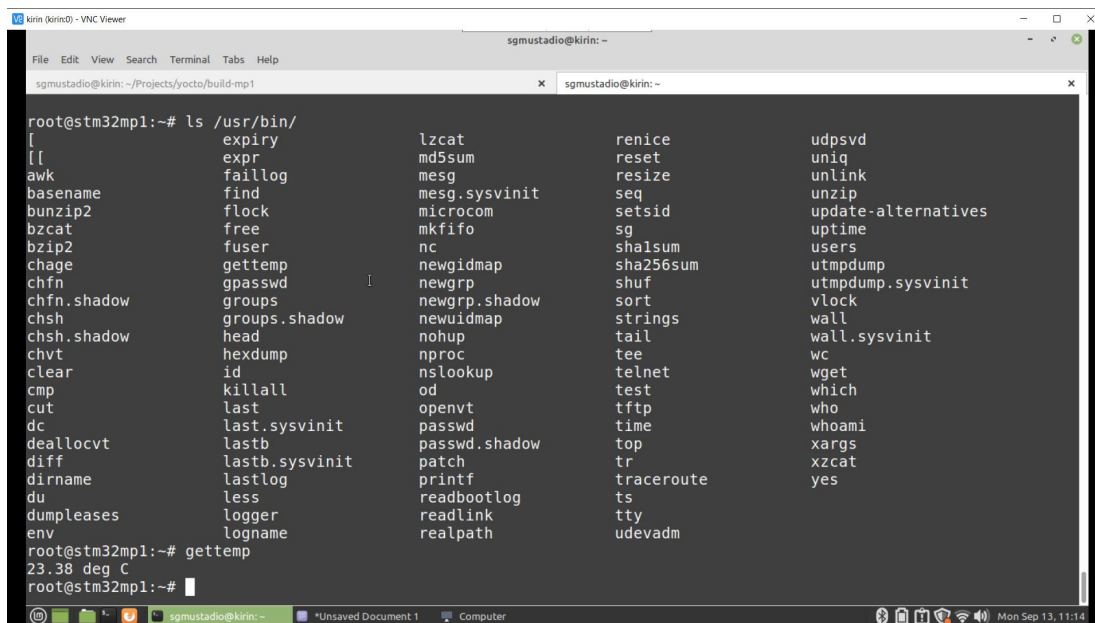
When you get the login prompt, log in with “root” as the username and “toor” as the password.

Test your custom application by entering the following into the console:

Copy Code

```
gettemp
```

With any luck, you should see the temperature printed to the console (or “Hello, world!” if you wrote the simpler program instead).



The screenshot shows a VNC viewer window titled "kirin (kirin0) - VNC Viewer". Inside the window is a terminal window titled "sgmustadio@kirin: ~". The terminal shows the following commands and output:

```
root@stm32mp1:~# ls /usr/bin/
[
[[
awk
basename
bunzip2
bzip2
bzcat
bzzip2
chage
chfn
chfn.shadow
chsh
chsh.shadow
chvt
clear
cmp
cut
dc
deallocvt
diff
dirname
du
dumpleases
env
root@stm32mp1:~# gettemp
23.38 deg C
root@stm32mp1:~#
```

The terminal also shows a list of system utilities in the background, including: lzcat, expiry, md5sum, reset, udpsvd, uniq, unlink, unzip, update-alternatives, uptime, users, utmpdump, utmpdump.sysvinit, vlock, wall, wall.sysvinit, wc, wget, which, who, whoami, xargs, xzcat, yes, renice, resize, seq, setsid, sg, sha1sum, sha256sum, shuf, sort, strings, tail, tee, telnet, test, tftp, time, top, tr, traceroute, ts, tty, udevadm, newgidmap, newgrp, newgrp.shadow, newuidmap, nohup, nproc, nslookup, od, openvt, passwd, passwd.shadow, patch, printf, readbootlog, readlink, realpath, groups, groups.shadow, head, hexdump, id, killall, last, last.sysvinit, lastb, lastb.sysvinit, lastlog, less, logger, logname.