Embedded Linux remote debugging mit RPI zero, Buildroot und QT

Stefan Scholz - 2022

Inhaltsverzeichnis

Embedded Linux remote debugging mit RPI zero, Buildroot und QT 1.1
Erstmal Ausprobieren 1.2
QtCreator konfigurieren 1.3
Debuggen mit dem QtCreator 1.4
Ans Eingemachte 1.5

Einleitung

Dieser Artikel richtet sich an Entwickler, die Embedded-Linux Anwendungen in QT erstellen wollen und eine komfortable Arbeitsumgebung dafür suchen.

Warum dieser Artikel

Ich habe bereits an mehreren größeren Embedded-Linux Projekten mitgearbeitet bei denen es keine Möglichkeit zum Debuggen der Applikation auf dem Gerät gab. Die Vorgehensweise war dort (und ist immer noch), die Plattformunabhängigkeit von QT zu nutzen um Features auf dem PC auszuprobieren. Dann warten, dass ein neues Image gebaut hat - Komplettes Gerät flashen - und hoffen, dass es dort auch funktioniert.

Aber es geht auch anders - dass der QtCreator Applikationen über SSH installieren und remote Debuggen kann hab ich vor einiger Zeit bei boot2qt bewundern können. Leider ist boot2qt etwas lizenzkritisch, die Compile- und Bootzeiten extrem langsam und das Buildsystem echt kompliziert

Ich hab mir angeschaut, wie boot2qt das macht und festgestellt, dass für das remote-Debugging lediglich standard Linux Komponenten (SSH-server und gdbserver) benötigt werden

Warum QT ?

Na das versteht sich von selbst :-)

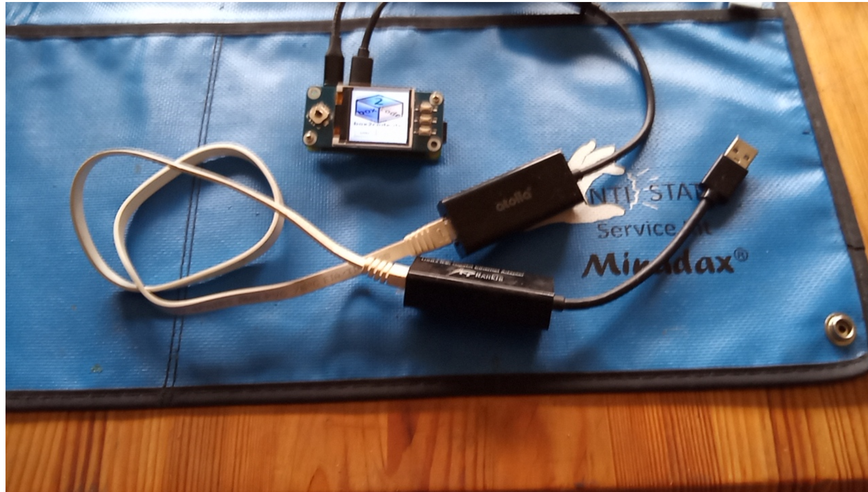
Warum Buildroot ?

- ist einfach - eigentlich kann man nicht viel falsch machen
- kann, ohne viel Aufwand, sehr kleine und flinke images erstellen
- Baut relativ schnell und zuverlässig.

Warum der RPI zero ?

Weil der günstig ist (v1.3 für ca. 6€ bei BerryBase), wenig Strom braucht (ca. 65 mA mit Display, ohne sogar nur 45 mA - Rekordverdächtig) und sehr klein ist. Hoffentlich sind die Raspberrys bald wieder lieferbar und behalten dabei ihren alten Preis. Was den zero auch interessant macht ist, dass er keine Ethernet-Schnittstelle besitzt (v1.3 auch kein WLAN) und eine Lösung gebraucht wird um Software auf das Board zu bringen ohne dauernd die SD-Karte neu zu bespielen.

Ich hab etwas gebraucht bis ich drauf gekommen bin - hier die simple Lösung:



Einfach zwei USB-Ethernet-Adapter zusammenschließen, dann hast du ein kleines Netz mit statischen IPs. Für den PI hab ich die Adresse `192.168.42.1/24` vergeben.

Vorsicht, der Adapter muss bei dem erstellten Image vor dem Einschalten des PI gesteckt werden.

Das root-Passwort ist übrigens `box2code` aber später mehr.

Erstmal Ausprobieren

Windows-Benutzer

Du brauchst natürlich ein Linux-System. Anders geht das nicht. Windows-Benutzern sei https://ubuntu.com/wsl empfohlen.

Pakete installieren

Welche Pakete Buildroot benötigt findest du unter https://buildroot.org/downloads/manual/manual.html#requirement-mandatory Und natürlich eine aktuelle Version des fabelhaften QtCreators

Hardware

Du kannst einen RPI zero v1.3 oder W verwenden - es werden beide unterstützt. Für Display-Spaß brauchst du noch den LCD-HAT hier https://www.waveshare.com/wiki/1.44inch_LCD_HAT. Und das oben beschriebene Adapterkabel aus zwei USB-ETH Adaptern.

Erst mal bauen

Das geht sehr einfach, mit ein paar Kommandos:

```
cd yourWorkFolder
git clone git://git.buildroot.net/buildroot
git clone git@bitbucket.org:bobbery/box2code_br_ext.git
cd buildroot
make BR2_EXTERNAL=../box2code_br_ext/
make pi0MitDisplayHat_defconfig
make
```

Der RPI zero bootet mit dem erstellten Image innerhalb von etwa 10 Sekunden in die QT-Applikation. Natürlich müssen wir erst eine Applikation drauf tun - machen wir später mit dem QtCreator. Der erste Start dauert etwas länger - danach gehts dann flott.

Das generierte image ist 300 MB groß (davon ca. 150 MB frei für Applikationen).

Wie flashen ?

Dafür verwendest du am Besten https://www.balena.io/etcher/ (Die Zeiten für `dd` sind, glaub ich, vorbei :-)) Das Programm ist so einfach, dass ich garnix erklären muss. Der Pfad für das image ist

`yourWorkFolder/buildroot/output/images/sdcard.img`

Erster Boot

Nichts Besonderes. Karte in den PI, Strom an und schaun was passiert. Mit etwas Glück kannst du schon den Bootprozess auf dem kleinen Display beobachten

Das Image bootet in etwa 10 Sekunden. Davon fallen circa 3s auf den Bootloader des Raspi. Der eigentliche Kernel benötigt nur etwa 7,5 Sekunden bis die Applikation gestartet wird.

QtCreator konfigurieren

SSH-Verbindung testen

Verbinde das Adapterkabel von oben mit dem RPI und deinem PC und starte den PI neu. Richte den USB-ETH-Adapter auf deinem PC so ein, dass er die statische Adresse `192.168.42.2/24` besitzt.

Nun führe im Terminal `ssh root@192.168.42.1` aus.

Es kommt eine Sicherheitswarnung mit einer genauen Beschreibung, was du tun musst: `ssh-keygen -f "~/.ssh/known_hosts" -R "192.168.42.1"`

Danach nochmal `ssh root@192.168.42.1` - es kommt die Passwortabfrage. Gib `box2code` ein - Du bist verbunden.

Gerät anlegen

Öffne jetzt den QtCreator und gehe auf `Extras->Einstellungen->Kits->Geräte->Hinzufügen->Generisches Linux Gerät`

Danach 2x auf `Weiter` und abschließen. QtCreator testet jetzt die Verbindung zum PI - nach unserer Vorbereitung sollte das klappen. Das Passwort kennst du ja schon.

Compiler hinzufügen

Öffne jetzt den QtCreator und gehe auf `Extras->Einstellungen->Kits->Compiler->Hinzufügen->GCC->C++`

Dann `Anwenden`

Das selbe mit `Extras->Einstellungen->Kits->Compiler->Hinzufügen->GCC->C` nur dass der Pfad jetzt `~/yourWorkDir/buildroot/output/host/usr/bin/arm-linux-gcc` ist.

Debugger hinzufügen

Öffne jetzt den QtCreator und gehe auf `Extras->Einstellungen->Kits->Debugger->Hinzufügen`

Der Pfad ist diesmal fest. Findest du dort keinen `gdb-multiarch` , dann musst du diesen noch installieren.

QT-Version hinzufügen

Öffne jetzt den QtCreator und gehe auf `Extras->Einstellungen->Kits->Qt-Versionen->Hinzufügen`

Vergib noch einen schönen Namen - ich hab `buildroot-QT` genommen.

Kit hinzufügen

Fast geschafft :-)

Öffne jetzt den QtCreator und gehe auf `Extras->Einstellungen->Kits->Kits->Hinzufügen`

Projekt öffnen und konfigurieren

Mit `Datei->Datei oder Projekt öffnen` öffnest du jetzt das Demo-Projekt unter `yourWorkDir/box2code_br_ext/demo/demo.pro`.

Jetzt gehst du auf den Schraubenschlüssel um das Projekt zu konfigurieren und wählst unseren neu angelegten Kit aus. Weiter gehts mit `Configure Project`

Eigentlich passt jetzt schon Alles - nur noch eine Kleinigkeit.

Damit die Applikation später beim Neustarten des PIs auch gestartet wird, legen wir noch einen Link auf unser Programm an:

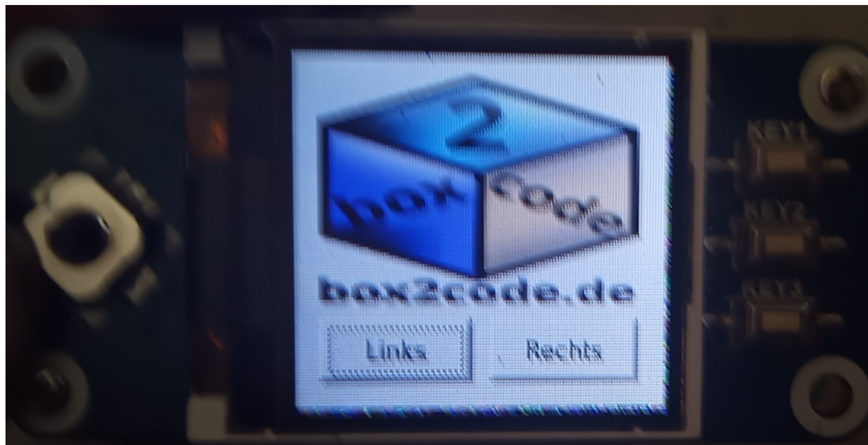
Nochmal auf den Schraubenschlüssel und unter unserem Kit `Ausführen` wählen. Dann auf `Schritt für Deployment hinzufügen` und `Benuzerdefiniertes entferntes Kommando ausführen`. In die Kommandozeile kommt: `rm /opt/qtApp; ln -s /opt/demo/bin/demo /opt/qtApp`

Jetzt aber - FERTIG :-)

Debuggen mit dem QtCreator

Setze einen Breakpoint irgendwo in der `rotate` - Methode in `mainwindow.cpp` und drücke den grünen Playknopf mit Käferchen.

Auf dem Display des PI solltest du jetzt das box2code-Logo und zwei Knöpfe sehen.



Drücke auf den Joystick

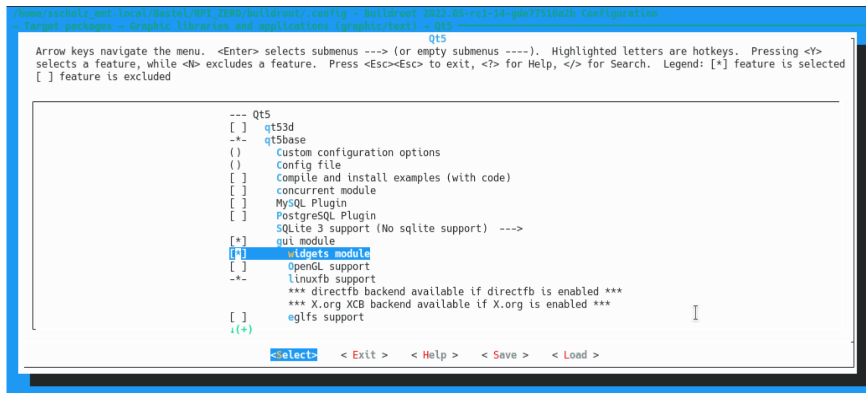
TADAA:

```
 5  MainWindow::MainWindow(QWidget *parent)
 6      : QMainWindow(parent), ui(new Ui::MainWindow) {
 7    ui->setupUi(this);
 8  }
 9
10  MainWindow::~MainWindow() { delete ui; }
11
12  void MainWindow::rotate(qreal value) {
13    QPixmap pixmap = ui->lblBox2Code->pixmap(Qt::ReturnByValue);
14    QTransform tr;
15    tr.rotate(value);
16    pixmap = pixmap.transformed(tr);
17    ui->lblBox2Code->setPixmap(pixmap);
18  }
19
20  void MainWindow::on_btnLinks_clicked() { rotate(-90); }
21
22  void MainWindow::on_btnRechts_clicked() { rotate(90); }
23
```

Ans Eingemachte

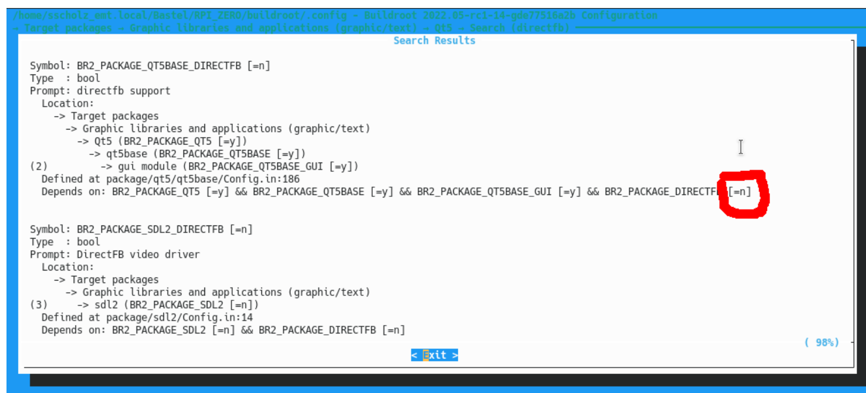
Wie funktioniert Buildroot eigentlich ?

Buildroot benutzt `KConfig` , das schon seit Ewigkeiten für die Konfiguration des Linux-Kernels eingesetzt wird. Der Trick ist, dass man die Konfiguration nicht von Hand scripten, sondern quasi grafisch zusammenklicken kann. `make menuconfig` liefert folgendes Interface:



Das Megafeature von Buildroot ist, dass es Abhängigkeiten kennt. Im Bild sieht man, dass das directfb-plugin nicht ausgewählt werden kann, weil die Abhängigkeit zu directfb fehlt.

Drückt man `/` , so kann man nach dem Paket suchen und sieht, warum es nicht auswählbar ist:



Wir sehen also, dass wir die QT-Unterstützung für directfb erst aktivieren können, wenn wir auch das Paket directfb aktiviert haben. Simpel, oder ?

Etwas komplizierter sind die Einstellungen für Platform und Toolchain. Dort kann man sich aber an den vielen vorgefertigten Standardkonfigurationen von im Moment fast 300 Boards bedienen. Natürlich hab ich das auch gemacht und hab mir die defconfig für den RPI zero erstmal kopiert.

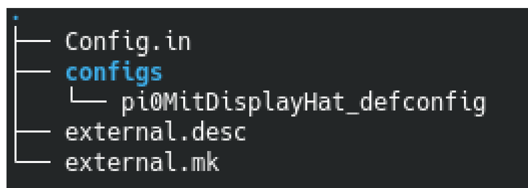
Warum ein Buildroot External ?

Natürlich geht das Ganze auch ohne ein External anzulegen. Wenn man aber Buildroot forked und direkt darin Dinge ändert hat man folgende Probleme:

- Wie hält man Buildroot auf einem aktuellen Stand ? - einmal geforked und man ist auf eine Version festgelegt. Mit External reicht ein simples `pull`.
- Wie gibt man sein Ergebnis an andere weiter ? - Externals sind klein und übersichtlich und einfach zu benutzen. Das Ergebnis ist einfach reproduzierbar. Mehrere Boarddefinitionen in einem External sind kein Problem.
- Im Endeffekt ist die Erstellung eines External sogar einfacher als an vielen Stellen im Buildroot-Baum Dinge zu ändern.

Wie baut man ein External ?

Wie oben beschrieben konfiguriert man sich sein System erstmal mit `make menuconfig`. Wenn man zufrieden damit ist kann man schon mal die wichtigsten Teile des Externals füllen:



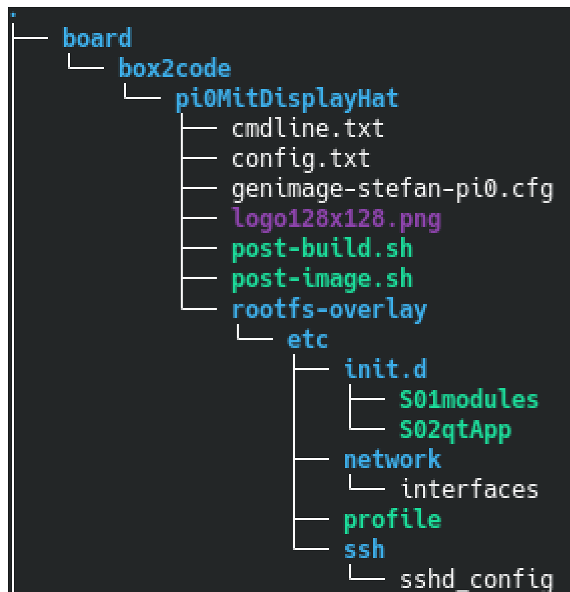
Das Bild zeigt dir das kleinstmöglich External, dass man überhaupt bauen kann. `pi0MitDisplayHat_defconfig` enthält die von `make menuconfig` erstellte Konfigurationsdatei `buildroot/.config`. `external.desc` hat den selbsterklärenden Inhalt:

```
name: BOX2CODE
desc: External for Projects on box2code.de
```

`Config.in` und `external.mk` sind bei mir leer - müssen aber da sein. Diese können verwendet werden, wenn du neue Pakete zu buildroot hinzufügen willst. Dafür gibt es dann noch den Ordener `packages` den ich nicht verwendet hab.

Das wars im Grunde schon - und ja, das ist wirklich so einfach :-)

Natürlich wollen wir unabhängig von den binaries im System noch ein paar Sachen für die Laufzeit konfigurieren, daher das `board`-Verzeichnis:



Was wir hier finden ist:

- die Konfiguration der Kernelparameter mit `cmdline.txt` und `config.txt`.
- Eine Konfiguration des Dateisystems mit `genimage-stefan-pi0.cfg` - hier steht drin welche Partitionen es gibt und was initial dort hin kopiert werden muss
- Mit `post-image.sh` Der Aufruf für das Programm zum generieren des Images (verwendet die .cfg-Datei) - hab ich quasi direkt von der Boarddefinition aus dem `buildroot/configs`-Ordner übernommen
- `post-build.sh` ist bei mir leer - aber gut sowas zu haben :-)
- Im `rootfs-overlay` kannst du gemäß der Linux Verzeichnisstruktur ablegen, was du gerne anders hättest (wird überschrieben) oder ergänzen möchtest. Ich gehe jetzt nicht im Detail darauf ein, da das sehr individuell sein kann. Für ein Verständnis der meisten Dateien gibt es gute manuals an anderer Stelle. Ich werde nur noch die `init.d`-Skripte kurz beschreiben:

In `S01modules` lade ich einfach mit `modprobe` Alles was das System so an Treibern benötigt wird. Dabei ist z.B. das Display und die Buttons des HATs. In `S02qtApp` führe ich deine QT-Applikation, für den du den link `/opt/qtApp` durch den QtCreator gesetzt hast aus.

Falls du `init.d` nicht kennst, `S01xxxx` bedeutet "starte als Erstes" `S02xxxx` "starte als Zweites" und so weiter, einfach oder ? (Die Verwendung von `systemd` ist natürlich auch möglich)

Noch ein letztes Wort zu der Datei `config.txt`, dann reichts aber :-)

```
...
dtoverlay=fbtft,spi0-0,st7735r,reset_pin=27,dc_pin=25,led_pin=24,width=128,hei

dtoverlay=gpio-key,gpio=6,keycode=103,label="KEY_UP",gpio_pull=up
...
```

mit der direktive `dtoverlay` kannst du Treiber des Linux-Systems im Vorfeld, schon beim Starten des Kernels konfigurieren. Wie du das machst ist sehr sehr gut in https://github.com/raspberrypi/firmware/blob/master/boot/overlays/README beschrieben. Bei unserem System musst du die Treiber dann in `S01modules` nur noch laden - hier z.B. `fbtft, fb_st7735r und gpio-key`. Du kannst aber auch `udev` aktivieren, dann passiert das automatisch. Der Bootvorgang dauert aber dadurch etwas länger.

So, das wars - Zeit für ein Bier :-)