Budowanie dystrybucji GNU/Linux dla Raspberry Pi z pomocą Yocto Project

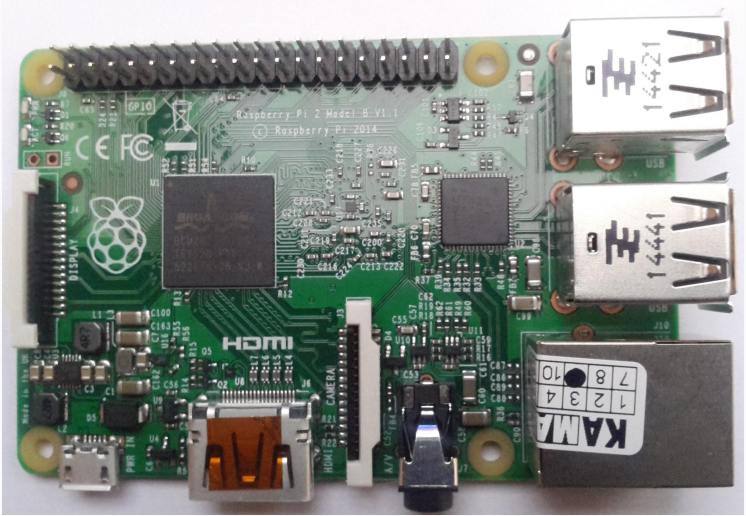
Adam Stolcenburg

15 stycznia 2019

Akademia ADB

Raspberry Pi

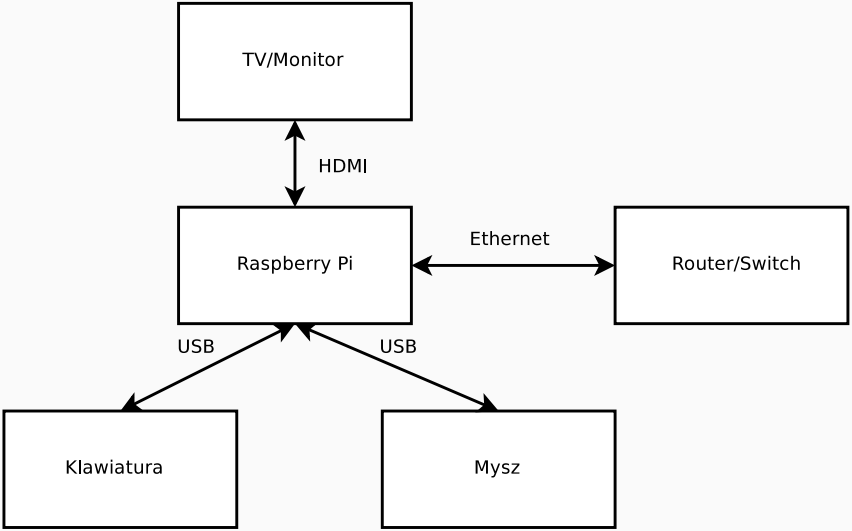
Raspberry Pi 2 Model B V1.1 i



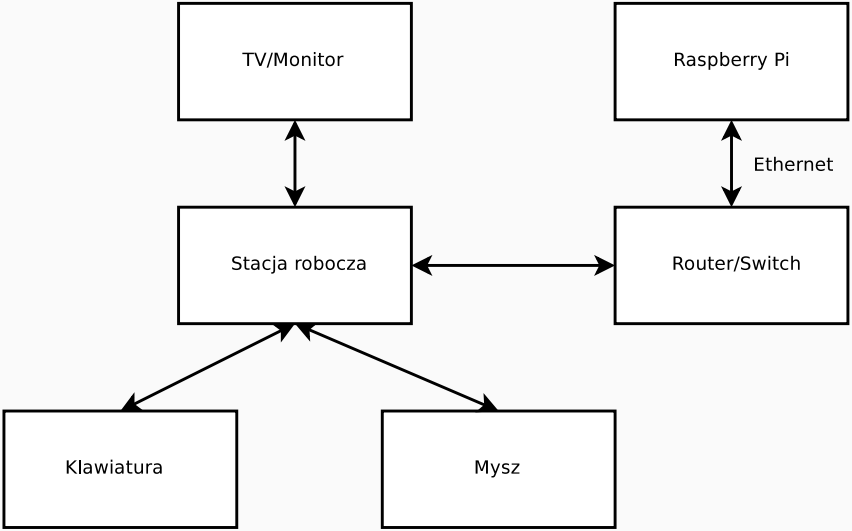
Raspberry Pi 2 Model B V1.1 ii

- BCM2836 (rodzina BCM2709)
  - czterordzeniowy procesor ARM Cortex-A7 900 MHz (ARMv7-A)
  - rdzeń graficzny 3D VideoCore IV
  - slot na kartę micro SD
  - 40 pinów ogólnego przeznaczenia
  - port HDMI
  - 3.5mm port audio i analogowego video
  - interfejs do podłączenia kamery
  - interfejs do podłączenia wyświetlacza
- SMSC LAN9514
  - port Ethernet
  - 4 porty USB
- Elpida EDB8132B4PB-8D-F
  - 1 GB DDR2 SDRAM

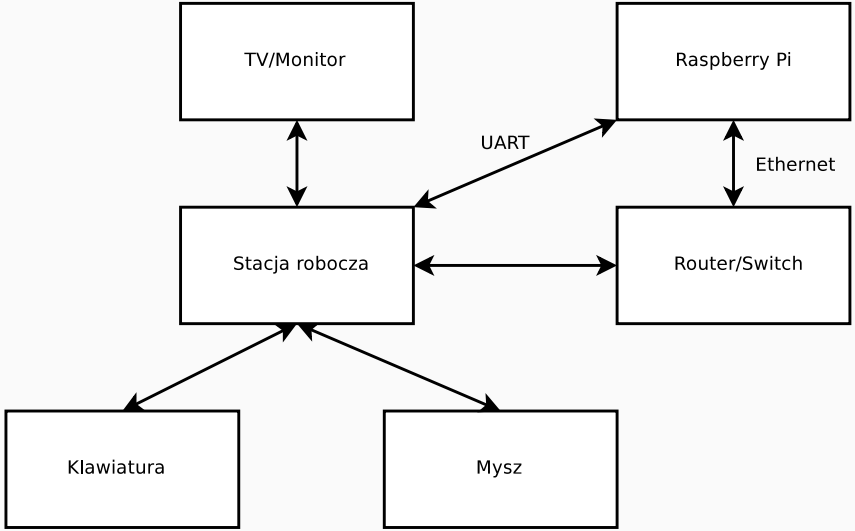
Model pracy - stacja robocza



Model pracy - dostęp zdalny



Model pracy - system wbudowany



Oprogramowanie stacji roboczej

- zazwyczaj Linux
- oprogramowanie umożliwiające komunikacje przez port szeregowy
- oprogramowanie sieciowe
  - Telnet, SSH (Secure Shell) – zdalny terminal
  - NFS (Network File System) – dzielenie plików
  - TFTP (Trivial File Transfer Protocol), HTTP (Hypertext Transfer Protocol) – przesyłanie obrazu systemu
- zestaw narzędzi do kompilacji skrośnej (ang. cross compiling toolchain)

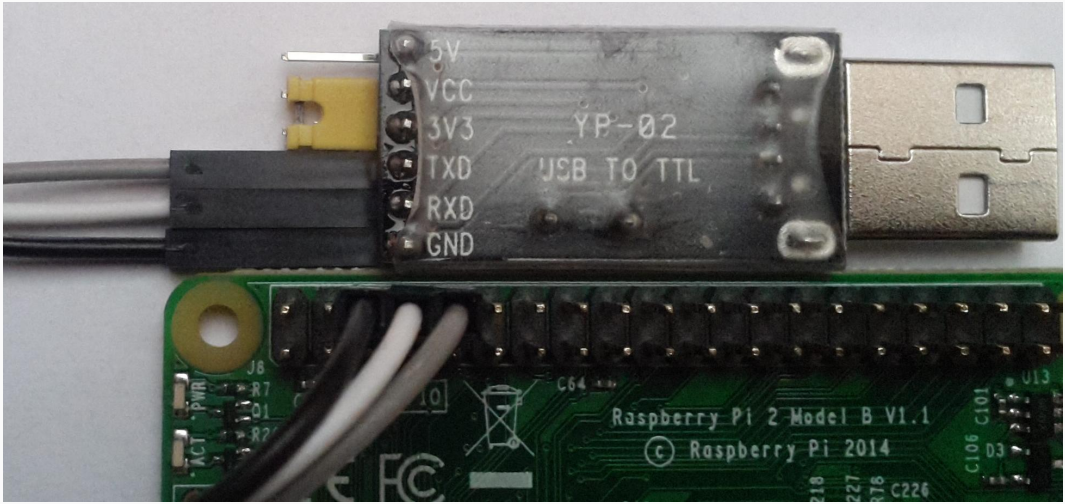
Kompilacja skrośna

- proces kompilacji wykonywany na innej architekturze procesora niż ta, dla której kod jest generowany
- wyróżnia się następujące platformy
  - platforma na której budowany jest kompilator (ang. build platform)
  - platforma na której kompilator będzie uruchamiany (ang. host platform)
  - platforma na której będzie wykonywany kod wygenerowany przez budowany kompilator (ang. target platform)
- zalety
  - brak konieczności posiadania działającego systemu na urządzeniu docelowym
  - brak konieczności posiadania kompilatorów na systemie docelowym
  - znacznie krótszy czas budowania

Komunikacja przez UART i

- UART używa dwóch sygnałów TxD (transmisja danych) – pin 8, RxD (odbiór danych) – pin 10
- aby skomunikować urządzenia należy połączyć skrośnie ich sygnały RxD z sygnałami TxD oraz ich masę – pin 6 (na przykład)
- komunikacja odbywa się na poziomie napięć TTL 3,3V
- w przypadku komunikacji ze stacją roboczą wymagany jest konwerter
  - poziomów napięć – MAX3232
  - USB-UART – FT232RL, CH340G
- domyślne parametry transmisji danych: 115200 8N1 (115200 bps, 8 bitów, bez kontroli parzystości, 1 bit stopu) bez kontroli przepływu danych

Komunikacja przez UART ii



Systemy operacyjne dla Raspberry Pi

- bazujące na Linuxie
  - Raspbian
  - Android Things
  - Gentoo Linux
  - własne dystrybucje
  - zbudowane z wykorzystaniem systemów wpierających tworzenie dystrybucji wbudowanych (ang. embedded), np. Yocto Project
  - (...)
- nie bazujące na Linuxie
  - Windows 10 IoT Core
  - FreeBSD
  - NetBSD
  - Haiku
  - (...)

https://en.wikipedia.org/wiki/Raspberry_Pi#Operating_systems

Własny system bazujący na Linux

+ idealnie skrojony do potrzeb
+ doskonały by zdobyć dogłębną wiedzę na temat Linuxa
− wymaga własnego systemu budowania
− integracja dodatkowych bibliotek wymaga ręcznego dbania o zależności
− wymaga dużej wiedzy i nakładu pracy

Dystrybucja Linux przygotowana za pomocą Yocto Project

+ w razie potrzeby umożliwia idealne dopasowanie do potrzeb
+ umożliwia łatwą integracje tysięcy pakietów wraz z ich zależnościami
+ wspierany przez społeczność
– stosunkowo wysoki próg wejścia
– długi czas budowania
– wymaga ogromnych ilości miejsca na stacji roboczej

Dystrybucja Raspbian

+ gotowa do natychmiastowego użycia
+ umożliwia łatwą instalację tysięcy pakietów wraz z ich zależnościami
+ proces instalacji pakietów taki sam jak dla dystrybucji na PC
+ wspierany przez społeczność
– duży rozmiar minimalnej instalacji (2GB dla wersji Stretch Lite)
– dostępne pakiety zazwyczaj nie są najświeższe

Yocto Project

Yocto Project

- https://www.yoctoproject.org/
- ogłoszony przez Linux Foundation w 2010, a wystartowany w marcu 2011
- 22 aktywnych członków, platynowi członkowie to Intel, Texas Instruments, Facebook, ARM
- celem jest usprawnienie procesu tworzenia wbudowanych dystrybucji Linuxa
- korzysta z
  - OpenEmbedded - framework automatyzujący budowanie dystrybucji Linuxa dla systemów wbudowanych
  - BitBake - narzędzie budujące
  - Poky - referencyjna dystrybucja Yocto Project

Wersje Yocto Project

Codename	Yocto Project Version	Release Date	Poky Version	BitBake branch
Thud	2.6	Nov 2018	20.0	1.40
Sumo	2.5	Apr 2018	19.0	1.38
Rocko	2.4	Oct 2017	18.0	1.36
Pyro	2.3	May 2017	17.0	1.34
Morty	2.2	Nov 2016	16.0	1.32
Krogoth	2.1	Apr 2016	15.0	1.30
Jethro	2.0	Nov 2015	14.0	1.28
Fido	1.8	Apr 2015	13.0	1.26
(...)	(...)	(...)	(...)	(...)
Bernard	1.0	??? 2011	5.0	1.11
(...)	(...)	(...)	(...)	(...)

https://wiki.yoctoproject.org/wiki/Releases

Warstwy Yocto Project

- repozytoria zawierające metadane mówiące systemowi budowania co robić
- zawierają recepty, klasy, pliki konfiguracyjne oraz pliki modyfikujące metadane dostarczane przez inne warstwy
- indeks oficjalnych warstw kompatybilnych z Yocto Project: https://www.yoctoproject.org/software-overview/layers/
- warstwy kompatybilne z OpenEmbedded: http://layers.openembedded.org/layerindex/branch/master/layers/
- podstawowe warstwy dostarczane w ramach dystrybucji Poky:
  - `meta` - rdzeń OpenEmbedded z metadanymi wspierającymi architektury ARM, ARM64, x86, x86-64, PowerPC, MIPS, MIPS64 oraz QEMU
  - `meta-poky` - metadane specyficzne dla dystrybucji Poky
  - `meta-yocto-bsp` - metadane dotyczące sprzętu referencyjnego

Warstwa BSP (board support package) dla Raspberry Pi

- http://layers.openembedded.org/layerindex/branch/master/layer/meta-raspberrypi/
- wymaga warstw
  - `openembedded-core` - meta z dystrybucji Poky - git://git.yoctoproject.org/poky.git
  - `meta-oe` - git://git.openembedded.org/meta-openembedded
- dokumentacja opisująca możliwości oraz zmienne konfiguracyjne dostępna w katalogu `docs`, a w postaci już wygenerowanej pod adresem https://media.readthedocs.org/pdf/meta-raspberrypi/latest/meta-raspberrypi.pdf
- wspierane maszyny: raspberrypi, raspberrypi0, raspberrypi0-wifi, raspberrypi2, raspberrypi3, raspberrypi3-64, raspberrypi-cm, raspberrypi-cm3

Konfiguracja stacji roboczej na potrzeby Yocto Project

- zalecana jedna z dystrybucji Linuxa: Fedora, openSUSE, Debian, Ubuntu albo CentOS
- co najmniej 50GB wolnej przestrzeni na dysku
- przygotowanie dystrybucji Ubuntu 16.04

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib
    build-essential chrpath socat libsdl1.2-dev xterm bmap-tools make xsltproc
    docbook-utils fop dblatex xmlto cpio python python3 python3-pip python3-
    pexpect xz-utils debianutils iputils-ping python-git bmap-tools python3-git
     curl parted dosfstools mtools gnupg autoconf automake libtool libglib2.0-
    dev python-gtk2 bsdmainutils screen libstdc++-5-dev libx11-dev
$ git config --global user.email "adres@email"
$ git config --global user.name "Imie Nazwisko"
```

Budowanie minimalnego obrazu dla Raspberry Pi i

- pobranie dystrybucji Poky w wersji sumo

```
$ git clone --single-branch -b sumo git://git.yoctoproject.org/poky.git
```

- pobranie kompatybilnej warstwy BSP dla Raspberry Pi

```
$ git clone --single-branch -b sumo git://git.yoctoproject.org/meta-raspberrypi
```

- pobranie zależności warstwy BSP dla Raspberry Pi

```
$ git clone --single-branch -b sumo git://git.openembedded.org/meta-
    openembedded
```

- ładowanie skryptu przygotowującego środowisko budujące, jeśli podany katalog nie istnieje utworzy domyślną konfigurację w podkatalogu `conf`

```
$ source ./poky/oe-init-build-env <opcjonalna nazwa katalogu>
```

Budowanie minimalnego obrazu dla Raspberry Pi ii

- modyfikacja `conf/local.conf` na potrzeby Raspberry Pi
  - podstawowa configuracja

```
MACHINE="raspberrypi2"
DISTRO = "poky"
```

  - aktywacja konsoli na porcie szeregowym i ustawienie hasła dla root

```
ENABLE_UART = "1"
INHERIT += "extrausers"
EXTRA_USERS_PARAMS = "usermod -P haslo root; "
```

  - zalecana opcjonalna konfiguracja

```
INHERIT += "rm_work"
DL_DIR = "/mnt/yocto/sumo/downloads"
SSTATE_DIR = "/mnt/yocto/sumo/sstate-cache"
```

Budowanie minimalnego obrazu dla Raspberry Pi iii

- modyfikacja `conf/bblayers.conf` na potrzeby Raspberry Pi

```
$ bitbake-layers remove-layer meta-yocto-bsp
$ bitbake-layers add-layer ../meta-raspberrypi
$ bitbake-layers add-layer ../meta-openembedded/meta-oe
```

- rozpoczęcie procesu budowania

```
$ bitbake core-image-minimal
```

- obraz wynikowy pojawi się w katalogu TMPDIR, domyślnie w podkatalogu o nazwie tmp, na przykład `./tmp/deploy/images/raspberrypi2/core-image-minimal-raspberrypi2-20190113222415.rootfs.rpi-sdimg`

Programowanie obrazu na karcie micro SD

- wykrycie nazwy urządzenia przypisanego do czytnika kart SD za pomocą dmesg

```
sd 6:0:0:1: [sdc] 30318592 512-byte logical blocks: (15.5 GB/14.5 GiB)
sd 6:0:0:1: [sdc] Write Protect is off
 sdc: sdc1 sdc2
sd 6:0:0:1: [sdc] Attached SCSI removable disk
```

- zapisanie obrazu na karcie SD – wartość <sdx> należy zastąpić odpowiednią nazwą urządzenia

```
$ cd ./tmp/deploy/images/raspberrypi2
$ sudo dd bs=4M if=core-image-minimal-raspberrypi2-20190113222415.rootfs.rpi-
    sdimg of=/dev/<sdx> status=progress conv=fsync
```

Komunikacja za pomocą portu szeregowego

- instalacja Minicom

```
$ sudo apt-get install minicom
```

- dodanie uprawnień do terminala (po tej zmianie należy się ponownie zalogować)

```
$ sudo adduser $USER dialout
```

- uruchomienie (nazwę `ttyUSB0` należy zastąpić odpowiednią nazwą terminala do którego został przypisany konwerter USB-UART informacja dostępna przez `dmesg`)

```
$ minicom -con -D /dev/ttyUSB0 ttyUSB0
```

- wyłączenie sprzętowej kontroli transmisji danych (ang. Hardware Flow Control) – należy wcisnąć `CTRL + a`, następnie `o`, wybrać `Serial port setup` i wcisnąć `f` aby przełączyć ustawienie `Hardware Flow Control` na `No`
- zapisanie ustawień – należy wyjść z poprzedniego menu za pomocą `ESC` i wybrać `Save setup as ttyUSB0`

Włączenie wsparcia dla kamery

- modyfikacja `conf/local.conf`

```
GPU_MEM = "128"
VIDEO_CAMERA = "1"
CORE_IMAGE_EXTRA_INSTALL += "userland"
```

- ponowne budowanie

```
$ bitbake core-image-minimal
```

- test kamery z poziomu Raspberry Pi

```
root@raspberrypi2:~# raspistill -o image.jpg
```

Dodanie biblioteki OpenALPR

- ściągnięcie dodatkowej warstwy

```
$ git clone https://github.com/maxinbjohn/meta-homeassistant-backup.git ../meta
    -homeassistant-backup
```

- modyfikacja `conf/bblayers.conf`

```
$ bitbake-layers add-layer ../meta-homeassistant-backup
```

- modyfikacja `conf/local.conf`

```
CORE_IMAGE_EXTRA_INSTALL += "openalpr"
```

Dodanie nowego pakietu

- utworzenie nowej warstwy

```
$ bitbake-layers create-layer ../meta-main
```

- dodanie nowej warstwy do `conf/bblayers.conf`

```
$ bitbake-layers add-layer ../meta-main
```

- utworzenie recepty main

```
$ rm -rf ../meta-main/recipes-example/example
$ mkdir -p ../meta-main/recipes-example/main
$ touch ../meta-main/recipes-example/main/main_1.0.bb
```

Przykładowa zawartość main_1.0.bb i

```
DESCRIPTION = "Parking"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835
    ade698e0bcf8506ecda2f7b4f302"

SRCREV = "7a010cbf8fca587128c25ae95458952a57cbdfb4"
SRC_URI = "git://adbacademy.imei.uz.zgora.pl/bitbucket/scm/par/raspberry.git;
    protocol=https \
    file://Makefile.patch \
    file://license_plates_region_fix.patch \
    file://header_include_fix.patch"

DEPENDS = "openalpr userland"
```

Przykładowa zawartość main_1.0.bb ii

```
TARGET_CC_ARCH += "${LDFLAGS}"

S = "${WORKDIR}/git"

do_compile_prepend () {
    ${MAKE} ${EXTRA_OEMAKE} clean
}

do_install() {
    install -d ${D}/opt/main/
    install ${S}/main ${D}/opt/main
}

FILES_${PN} = "/opt/main/main"
```

Wybrane zadania

- `do_build` - domyślne zadanie, wywołuje wszystkie normalne zadania
- `do_fetch` - pobiera źródła z `SRC_URI` do katalogu wskazywanego przez `DL_DIR`
- `do_unpack` - rozpakowuje źródła do katalogu wskazywanego przez `WORKDIR`
- `do_patch` - nakłada patche
- `do_configure` - konfiguruje źródła
- `do_compile` - kompiluje kod źródłowy, działa w katalogu roboczym `${B}`
- `do_install` - kopiuje pliki które mają trafić do pakietów do `${D}`
- `do_package` - rozdziela pliki dostarczone `${D}` na wiele pakietów

https://www.yoctoproject.org/docs/latest/mega-manual/mega-manual.html#ref-tasks-build

Konfiguracja Git dla repozytoriów wymagających autoryzacji

- plik `.netrc` umieszczony w katalogu domowym użytkownika

```
machine adbacademy.imei.uz.zgora.pl
login Twoj.Login
password Twoje.haslo
```

Rozwiązywanie problemów z budowaniem pakietów

- wyczyszczenie pakietu (`main` to nazwa czyszczonego pakietu)

```
$ bitbake -c cleanall main
```

- uruchomienie powłoki deweloperskiej

```
$ bitbake -c devshell main
```

- wyjście z powłoki

```
# exit
```

- wykonanie zadania kompilacji oraz zadań od niego zależnych

```
$ bitbake -C compile main
```

SDK

- budowanie SDK

```
$ bitbake core-image-minimal -c populate_sdk
```

- SDK pojawi się w katalogu TMPDIR, domyślnie w podkatalogu o nazwie `tmp`, na przykład `./tmp/deploy/sdk/poky-glibc-x86_64-core-image-minimal-cortexa7hf-neon-vfpv4-toolchain-2.5.2.sh`
- instalacja SDK

```
$ ./poky-glibc-x86_64-core-image-minimal-cortexa7hf-neon-vfpv4-toolchain-2.5.2.
    sh
```

- ładowanie skryptu przygotowującego środowisko dla SDK (zakładając że SDK jest zainstalowane w katalogu ./sdk)

```
$ source ./sdk/environment-setup-cortexa7hf-neon-vfpv4-poky-linux-gnueabi
```

Zmienne środowiskowe ustawiane przez SDK (fragment)

```
(...)
CC="arm-poky-linux-gnueabi-gcc -march=armv7ve -marm -mfpu=neon-vfpv4 -mfloat-abi=
    hard -mcpu=cortex-a7 --sysroot=$SDKTARGETSYSROOT"
CXX="arm-poky-linux-gnueabi-g++ -march=armv7ve -marm -mfpu=neon-vfpv4 -mfloat-abi=
    hard -mcpu=cortex-a7 --sysroot=$SDKTARGETSYSROOT"
CPP="arm-poky-linux-gnueabi-gcc -E -march=armv7ve -marm -mfpu=neon-vfpv4 -mfloat-abi
    =hard -mcpu=cortex-a7 --sysroot=$SDKTARGETSYSROOT"
AS="arm-poky-linux-gnueabi-as "
LD="arm-poky-linux-gnueabi-ld --sysroot=$SDKTARGETSYSROOT"
GDB=arm-poky-linux-gnueabi-gdb
STRIP=arm-poky-linux-gnueabi-strip
RANLIB=arm-poky-linux-gnueabi-ranlib
OBJCOPY=arm-poky-linux-gnueabi-objcopy
OBJDUMP=arm-poky-linux-gnueabi-objdump
AR=arm-poky-linux-gnueabi-ar
NM=arm-poky-linux-gnueabi-nm
(...)
```

Instalacja i użycie SSH

- modyfikacja `conf/local.conf` dodająca serwer SSH

```
CORE_IMAGE_EXTRA_INSTALL += "openssh"
EXTRA_USERS_PARAMS += "useradd -P password user; "
```

- instalacja klienta SSH na stacji roboczej

```
$ sudo apt-get install openssh-client
```

- logowanie do Raspberry Pi (hasło takie jakie podaliśmy po parametrze `-P useradd`)

```
$ ssh user@192.168.0.180
user@192.168.0.180's password:
```

Instalacja i użycie serwera NFS

- instalacja

```
$ sudo apt-get install nfs-kernel-server nfs-common
```

- konfiguracja

```
$ sudo mkdir /mnt/nfs
$ sudo chmod 777 /mnt/nfs
$ sudo sh -c 'cat << EOF >> /etc/exports
/mnt/nfs    *(rw,sync,no_subtree_check)
EOF'
```

- uruchomienie

```
$ sudo service nfs-kernel-server restart
```

- montowanie katalogu z poziomu Raspberry Pi

```
root@raspberrypi2:~# mkdir /mnt/nfs
root@raspberrypi2:~# mount -t nfs -o nolock 192.168.0.100:/mnt/nfs /mnt/nfs
```

Warto przeczytać

- Embedded Linux Development Using Yocto Project Cookbook - Second Edition, Alex González
- https://www.yoctoproject.org/docs/
- https://www.yoctoproject.org/docs/latest/bitbake-user-manual/bitbake-user-manual.html
- https://www.yoctoproject.org/docs/latest/mega-manual/mega-manual.html

Dziękuję

Makefile.patch i

```
Index: git/Makefile
===================================================================
--- git.orig/Makefile
+++ git/Makefile
@@ -1,21 +1,14 @@
-
-GCC = g++
-
-
-
-
-
 OUTPUT_FILE = main
```

Makefile.patch ii

```
 all: CarPlateRecognizer.o CarPlateDataPC.o CarPlateDataRaspberry.o CarPlateSender.o
-       $(GCC) main.cpp CarPlateRecognizer.o CarPlateDataPC.o CarPlateDataRaspberry.o
     CarPlateSender.o -L/home/others/projects/ADBProject/libs -lopenalprarm -o $(
    OUTPUT_FILE)
+       $(CXX) main.cpp CarPlateRecognizer.o CarPlateDataPC.o CarPlateDataRaspberry.o
     CarPlateSender.o -L/home/others/projects/ADBProject/libs -lopenalpr -o $(
    OUTPUT_FILE)
 CarPlateRecognizer.o: CarPlateRecognizer.h CarPlateRecognizer.cpp data/
     CarPlateDataPC.h data/CarPlateDataRaspberry.h data/CarPlateSender.h
-       $(GCC) CarPlateRecognizer.cpp -c -o CarPlateRecognizer.o
+       $(CXX) CarPlateRecognizer.cpp -c -o CarPlateRecognizer.o
 CarPlateDataPC.o: data/CarPlateDataPC.cpp data/CarPlateDataPC.h
-       $(GCC) data/CarPlateDataPC.cpp -c -o CarPlateDataPC.o
+       $(CXX) data/CarPlateDataPC.cpp -c -o CarPlateDataPC.o
```

Makefile.patch iii

```
 CarPlateDataRaspberry.o: data/CarPlateDataRaspberry.cpp data/CarPlateDataRaspberry.
     h
-       $(GCC) data/CarPlateDataRaspberry.cpp -c -o CarPlateDataRaspberry.o
+       $(CXX) data/CarPlateDataRaspberry.cpp -c -o CarPlateDataRaspberry.o
 CarPlateSender.o: data/CarPlateSender.cpp data/CarPlateSender.h
-       $(GCC) data/CarPlateSender.cpp -c -o CarPlateSender.o
+       $(CXX) data/CarPlateSender.cpp -c -o CarPlateSender.o
 clean:
        rm -rf *.o $(OUTPUT_FILE)
```

license_plates_region_fix.patch

```
Index: raspberry/CarPlateRecognizer.cpp
===================================================================
--- raspberry.orig/CarPlateRecognizer.cpp
+++ raspberry/CarPlateRecognizer.cpp
@@ -19,7 +19,7 @@ CarPlateRecognizer::CarPlateRecognizer()

 void CarPlateRecognizer::init()
 {
-    alpr::Alpr openalpr("au");
+    alpr::Alpr openalpr("eu");

     if(openalpr.isLoaded() == false) {
         std::cerr << "OpenALPR not loaded" << std::endl;
```

header_include_fix.patch i

```
Index: git/CarPlateRecognizer.h
===================================================================
--- git.orig/CarPlateRecognizer.h
+++ git/CarPlateRecognizer.h
@@ -1,7 +1,7 @@
 #ifndef CAR_PLATE_RECOGNIZER_H
 #define CAR_PLATE_RECOGNIZER_H

-#include "libs/alpr.h"
+#include <alpr.h>
 #include "data/CarPlateData.h"
 #include <vector>

Index: git/data/CarPlateSender.h
```

header_include_fix.patch ii

```
===================================================================
--- git.orig/data/CarPlateSender.h
+++ git/data/CarPlateSender.h
@@ -1,7 +1,7 @@
 #ifndef CAR_PLATE_SENDER_H
 #define CAR_PLATE_SENDER_H

-#include "../libs/alpr.h"
+#include <alpr.h>

 class CarPlateSender {
 public:
```