

# Buildroot

24 maj 2023  
Wiktor Kuna



# Buildroot

- GNU GPL 2
- Potrafi zbudować:
  - Toolchain
  - Linux kernel
  - Rootfs
  - Bootloader
- Konfigurowalny:  
**{menu/n/x/g}config**
- UDOKUMENTOWANY!!
- Korzysta z **make** i **kconfig**
- Szybki (czas budowania 'na czysto' zwykle nie przekracza 15-30 min)
- **Prosty w obsłudze** (przyjazny początkującym)

×

=



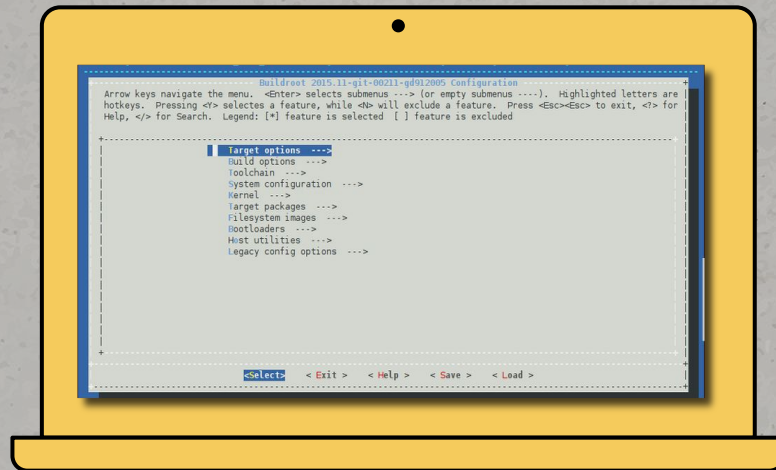


# Setup

`git clone https://github.com/buildroot/buildroot.git`

Bądź tar z ostatniego stable release'a (01.02.23):

`wget https://buildroot.org/downloads/buildroot-2023.02.1.tar.gz`  
`tar xvf buildroot-2023.02.1.tar.gz`



# Wspierane architektury

- ARC
- ARM
- AArch64
- i386
- m68k
- Microblaze AXI
- Microblaze non-AXI
- MIPS
- MIPS64
- NIOS II
- OpenRISC
- PowerPC
- PowerPC64
- RISC-V
- s390x
- SuperH
- SPARC
- SPARC64
- x86\_64
- Xtensa





# Proces budowania



**Pobieranie źródeł**

**Konfiguracja, budowa i  
instalacja paczek**

**Budowa  
bootloadera**  
(opcjonalnie)

**Konfiguracja, budowa i  
instalacja toolchaina**  
(lub import zewnętrznego)

**Budowa jądra  
Linuxa**  
(opcjonalnie)



**Utworzenie  
systemu plików**  
(opcjonalnie)



# Struktura zbudowanego projektu



<b><i>output/images/</i></b>	Wszystkie obrazy, których zbudowanie zleciliśmy podczas konfiguracji (tj. Jądro Linuxa, bootloadery, systemy plików)
<b><i>output/build/</i></b>	Wszystko co zostało zbudowane (w tym narzędzia dla buildroota, linux, pakiety)
<b><i>output/host/</i></b>	Zawiera narzędzia zbudowane dla hosta i sysroot toolchainu targetu (pliki nagłówkowe i biblioteki dla paczek w przestrzeni użytkownika)
<b><i>output/staging/</i></b>	Symlink do toolchaina targetu (potrzebny ze względów kompatybilności)
<b><i>output/target/</i></b>	<u>Prawie</u> kompletny system plików dla targetu. Brakuje w nim plików urządzeń w /dev (Buildroot nie działa z przywilejami roota) i posiada niepoprawne uprawnienia dla niektórych plików





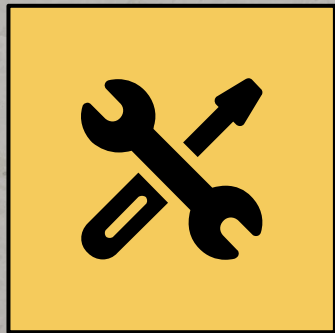


# WHOA!

---

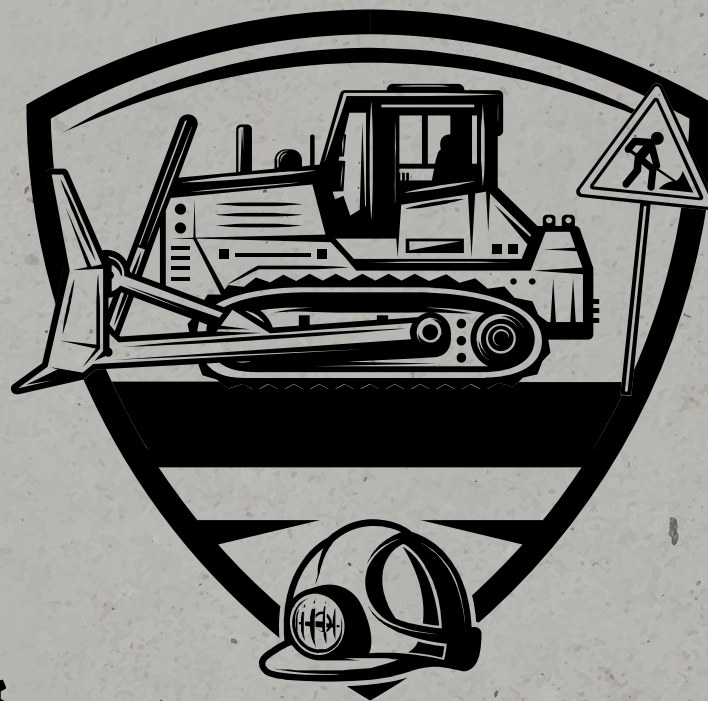
<PRZEGLĄDAMY OPCJE KONFIGURACYJNE  
BUILDROOTA>





# Toolchain

---





# Toolchain



## Buildroot

Musimy wyspecyfikować:

- Architektūrę targetu
- Bibliotekę C (domyślnie glibc)
- Pliki nagłówkowe jądra (wersję/ścieżkę nagłówków)
- Wersję binutils
- Wersję gcc

Skompilowany cross toolchain znajdziemy w **output/host/bin/**.

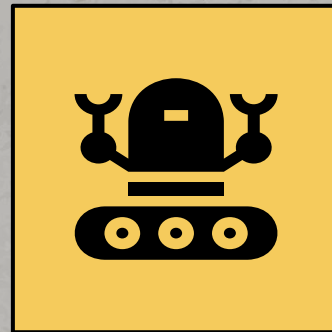


## External

Do zbudowanego wcześniej wystarczy przekazać ścieżkę.

- Toolchain od Liniario lub Bootlina
- Custom:
  - Wszystko co przy toolchainie buildrootowym
  - URL lub ścieżka do plików źródłowych
  - Dodatkowe informacje o toolchainie (wsparcie)
  - Prefix toolchainu





# Konfiguracja cd

---







# Zarządzanie /dev

Buildroot daje nam kilka opcji, jeśli chodzi o zarządzanie plikami urządzeń w /dev:

- **Statyczne, korzystające z device table** (old school, rebooty nie wpływają na pliki w /dev i nic nie dzieje się 'samo')
- **Dynamiczne, wykorzystujące wyłącznie devtmpfs** (wirtualny system plików; automatycznie zarządza plikami w /dev)
- **Dynamiczne z devtmpfs + mdev** (mdev - zamiennik udev'a w BusyBoxie - między innymi pozwala na wykonywanie akcji z przestrzeni użytkownika przy podłączeniu/odłączeniu urządzenia - na przykład - załadowaniu modułu obsługującego to urządzenie)
- **Dynamiczne z devtmpfs + eudev** (eudev - trochę cięższy i bardziej elastyczny od mdeva - daemon automatycznie obsługujący /dev - włącznie z zarządzaniem odpowiednim firmware'm)

Jeśli jako **init** wybierzemy **systemd** to obsługą /dev zajmie się **udev**.



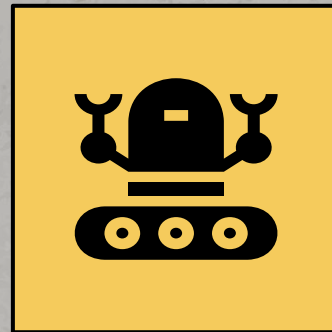


# init

- BusyBox - prosty, lekki
- systemV - systemV
- systemd - dość skomplikowany; będzie lepiej się nadawał do bardziej skomplikowanych systemów wbudowanych; dużo zależności







**Linux**  
**kernel**





# Konfiguracja Linuxa



- **defconfig** - nazwa pliku w arch/<ARCH>/configs (bez przyrostka \_defconfig)
- Domyślny dla wybranej architektury
- Inny - pozwala na podanie ścieżki do pliku







# Konfiguracja Linuxa



Konfiguracją linuxa możemy zarządzać w prosty sposób poprzez:

```
make linux-{menu/n/x/g}config
```

Zapisujemy poprzez (tylko jeśli wybraliśmy opcję 'custom configuration'):

```
make linux-update-(def)config
```

Alternatywnie, po zbudowaniu (gdy wybraliśmy inny defconfig):

```
cd output/build/linux-custom && cd menuconfig && cd ../../..  
make linux-rebuild
```





# Konfiguracja Linuxa



- Własne źródła - należy podać URL/ścieżkę do archiwum z kodem źródłowym Linuxa (należy też wtedy upewnić się, że zaznaczymy kompatybilne headery przy budowaniu toolchaina)
- Z buildroota - wystarczy wybrać wersję





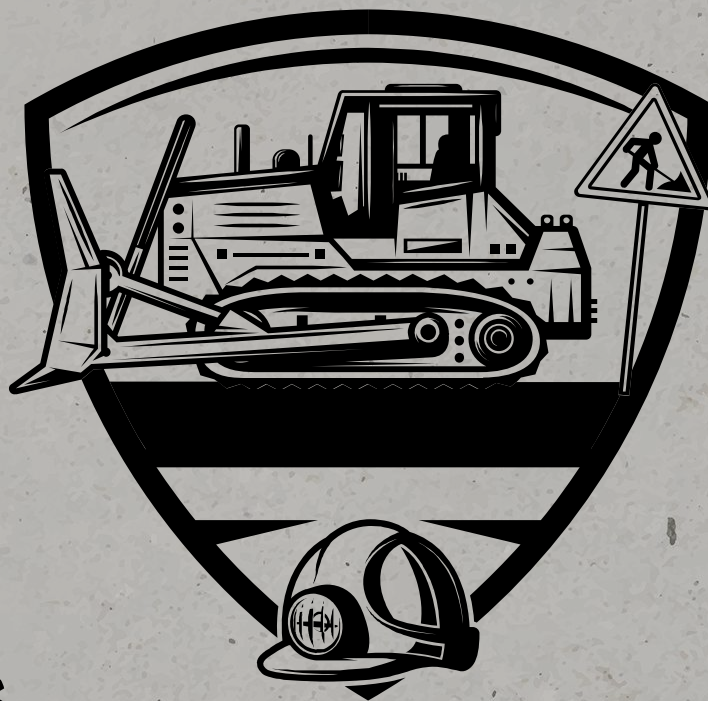
×

=



**rootfs**

---



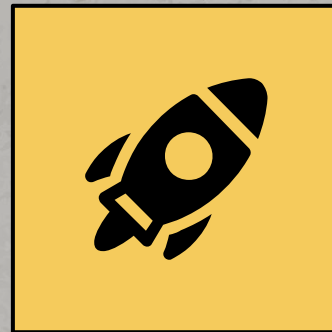
# Wspierane Systemy Plików

- axfs
- btrfs
- cloop
- cpio
- cramfs
- erofs
- ext2/3/4
- f2fs
- jffs2
- oci

- romfs
- squashfs
- tar
- ubi
- ubifs
- yaffs2
- ...







# Booloaders

---

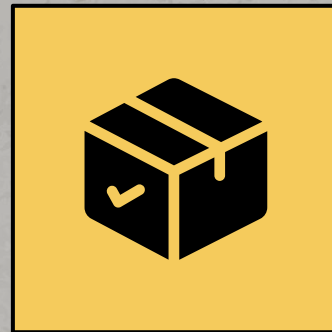


# Wspierane bootloadery

- afboot-stm32
- ARM Trusted Firmware
- Barebox
- grub2
- mxs-bootlets
- optee\_os
- s500-bootloader
- shim
- U-Boot
- ...







# Pakiety

---





# Pakiety



- Buildroot zapewnia wsparcie dla wielu systemów budowania, np. autotools, Meson, CMake; czy języków: Python, Perl, Lua, Erlang...
- Kopiowanie całego ich kodu byłoby dość... Gorszące
- Stąd, buildroot posiada infrastrukturę pakietów, w której, dla danego pakietu posiada informację:
  - Skąd pobrać pakiet
  - Jak go rozpakować
  - Jak aplikować patche
  - Jak go zbudować





# Dodawanie pakietu

## Config.in

- Dla paczek dla targetu
- Zawiera informacje, które zostaną wyświetlone w narzędziu konfiguracyjnym

```
config BR2_PACKAGE_LIBFOO
bool "libfoo"
help
```

This is a comment that explains what libfoo is. The help text should be wrapped.

<http://foosoftware.org/libfoo/>

Ponadto, może zawierać: domyślną wartość, potrzebne zależności (wobec architektury docelowej, toolchainu, innych paczek), czy opcje, które zaznacza (select).

Na koniec, należy dodać linijkę do `package/Config.in`:

```
source "package/libfoo/Config.in"
```

## Config.in.host

- Dla paczek hosta
- Analogiczna do paczek dla targetu

```
config BR2_PACKAGE_HOST_FOO
bool "host foo"
help
```

This is a comment that explains what foo for the host is.

<http://foosoftware.org/foo/>

Na koniec, należy dodać linijkę do `package/Config.in.host`:

```
source "package/libfoo/Config.in.host"
```

# Dodawanie pakietu

## .mk

- Przepis na pobieranie, konfigurację, budowanie i instalację paczki

Zawartość:

- Wersja paczki
- Źródła paczki (URL lub ścieżka do lokalnych źródeł)
- Metoda pozyskania źródeł (local, git, wget, scp...)
- Spełnienie wymagań dla którejś z dostępnych infrastruktur:
  - Generic (najbardziej pracochłonna)
  - Bazując na autotools
  - Bazując na cmake
  - Moduły Pythonowe
  - Moduły Lua

Dokładny opis (i przykłady) dla plików .mk dla różnych systemów budowania można znaleźć pod:

<https://buildroot.org/downloads/manual/manual.pdf#generic-package-tutorial>

## .hash

- Hashe dla pobieranych plików
- Pozwala sprawdzić spójność pobieranych plików (w tym też licencji)
- Wspierane hashe:
  - md5, sha1, sha224, sha256, sha384, sha512

Można pominąć - wtedy nic nie zostanie sprawdzone.



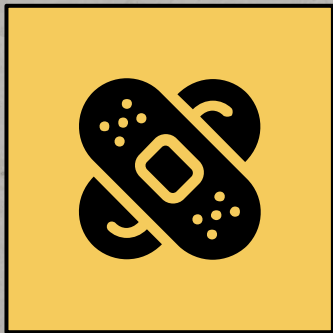


# WHOA!

---

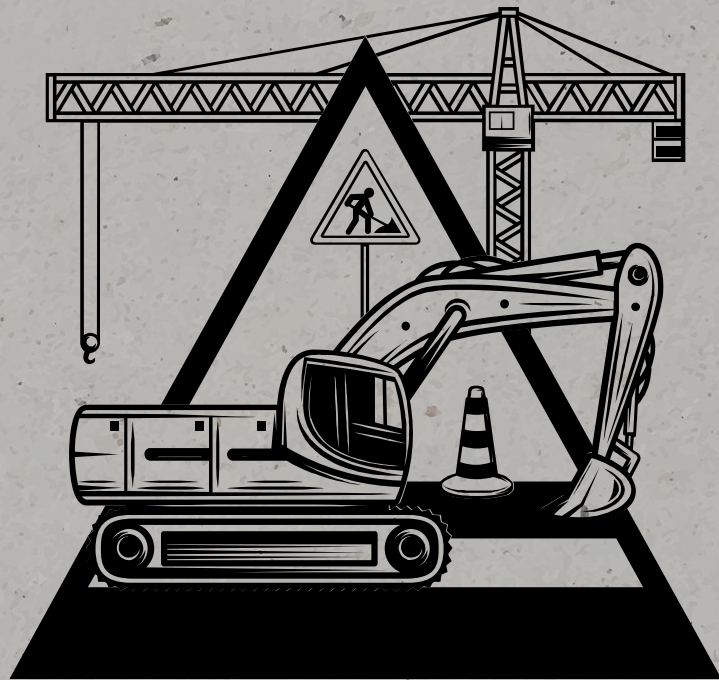
<PATRZYMYP NA STRUKTURĘ PRZYKŁADOWEGO  
PAKIETU>





**Patche**

---





# Patche dla paczek

- Czasem trzeba naprawić coś naprawić, ale zamiast zmieniać całych źródeł naszego projektu, możemy zaaplikować łatkę
- Takie łatki można przekazać Buildrootowi, by zaaplikował je podczas budowania paczki. Można zrobić to na trzy sposoby:
  - URL do zawartości łatki
  - Lokalnie, w źródłach Buildroota
  - Ścieżkę do katalogu z łatkami, rozłącznie od źródeł Buildroota





# Jak łąta Buildroot

Łatki aplikowane są w następującej kolejności:

- Pobiera i aplikuje wszystkie patche wyspecyfikowane w zmiennej `<pkg>_PATCH` (ustawianej w pliku `.mk`)
- Łatki z katalogu `package/<pkg>/*.patch`
- Łatki z zewnętrznego katalogu (ustawiane poprzez `BR2_GLOBAL_PATCH_DIR`)





# Przykład łatki

configure.ac: add C++ support test

Signed-off-by: John Doe <john.doe@noname.org>

--- configure.ac.orig

+++ configure.ac

@@ -40,2 +40,12 @@

AC\_PROG\_MAKE\_SET

+

+AC\_CACHE\_CHECK([whether the C++ compiler works],

+ [rw\_cv\_prog\_cxx\_works],

+ [AC\_LANG\_PUSH([C++])

+ AC\_LINK\_IFELSE([AC\_LANG\_PROGRAM([], [])],

+ [rw\_cv\_prog\_cxx\_works=yes],

+ [rw\_cv\_prog\_cxx\_works=no])

+ AC\_LANG\_POP([C++]))

+

+AM\_CONDITIONAL([CXX\_WORKS], [test "x\$rw\_cv\_prog\_cxx\_works" = "xyes"])





# Buildroot w CI?!





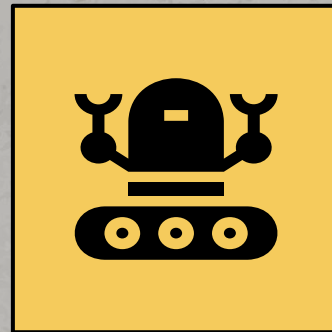


# Przykład budowania buildroota w CI



```
build-buildroot:
  stage: build
  script:
    - export ROOT=$(pwd)
    # Manifest zawierający linki do potrzebnych zależności
    - repo init -manifest-name manifest.xml && repo sync
    # Wcześniej zapisana konfiguracja buildroota, o ile
    # domyślne nie spełniają naszych potrzeb
    - cp buildroot.config buildroot/.config
    - pushd buildroot
    - make BR2_EXTERNAL="$(ROOT)/br2_ext" && popd
    # Tu (opcjonalnie) można zebrać artefakty
  artifacts:
    paths:
      - buildroot/output/images/
```





**BR2\_EXTERNAL**

---







# BR2\_EXTERNAL



- Trzymanie własnych konfiguracji w źródłach buildroota nie jest zbyt praktycznym rozwiązaniem
- Zmienna `BR2_EXTERNAL` pozwala nam przekazać ścieżk(ę/i) do zewnętrznego katalogu z własnymi konfiguracjami i/lub innymi artefaktami



# Zawartość katalogu BR2\_EXTERNAL



<b>external.desc</b>	Nazwa i opis (musi zawierać co najmniej nazwę - będzie wykorzystywana przy tworzeniu zmiennych)
<b>Config.in</b>	Opcje konfiguracyjne do wyświetlenia w menuconfig (może być pusty, bądź utworzony analogicznie do <code>packages/Config.in</code> - jako lista source)
<b>external.mk</b>	Przepis, który zostanie załączony do logiki make (może być pusty)
<b>configs/ (opcjonalnie)</b>	Katalog z domyślnymi konfiguracjami buildroota (o nazwach: <nazwa płytki rozwojowej>_defconfig)
<b>packages/ (opcjonalnie)</b>	Katalog z paczkami, strukturą analogiczny do <code>packages/</code> w drzewie Buildroota.





# Inne (opcjonalne) w BR2\_EXTERNAL



Ponadto, w takim katalogu możemy przekazać pliki specyficzne dla danej płytki rozwojowej (na której pracujemy). Zalecane jest umieścić te pliki w ścieżce `<nazwa płytki rozwojowej>/<firma>/<nazwa płytki>/` (można też po prostu w `<nazwa płytki rozwojowej>/`).

<b><i>linux.config</i></b>	Konfiguracja jądra Linuxa
<b><i>busybox.config</i></b>	Konfiguracja dla BusyBoxa
<b><i>post_build.sh</i></b>	Skrypt do wykonania PO zbudowaniu całego wybranego oprogramowania ale PRZED zbudowaniem systemu plików
<b><i>post_image.sh</i></b>	Skrypt do wykonania po zbudowaniu systemu plików
<b><i>rootfs_overlay/</i></b>	Drzewo plików, które zostanie przekopiowane do zbudowanego systemu plików
<b><i>patches/</i></b>	Katalog z paczkami





# Perks & Quirks

Przegląd opinii





A solid yellow rectangular banner is centered horizontally across the middle of the image. It contains the text "Niski Próg Wejścia" in a bold, black, sans-serif font. The banner is set against a light gray, textured background that resembles paper or concrete. At the top and bottom of the image, there are horizontal bands of black and white diagonal stripes, resembling caution tape.

**Niski Próg Wejścia**



**Brak Kompilacji Przyrostowej  
(przy każdej zmianie trzeba  
przebudować cały obraz)**

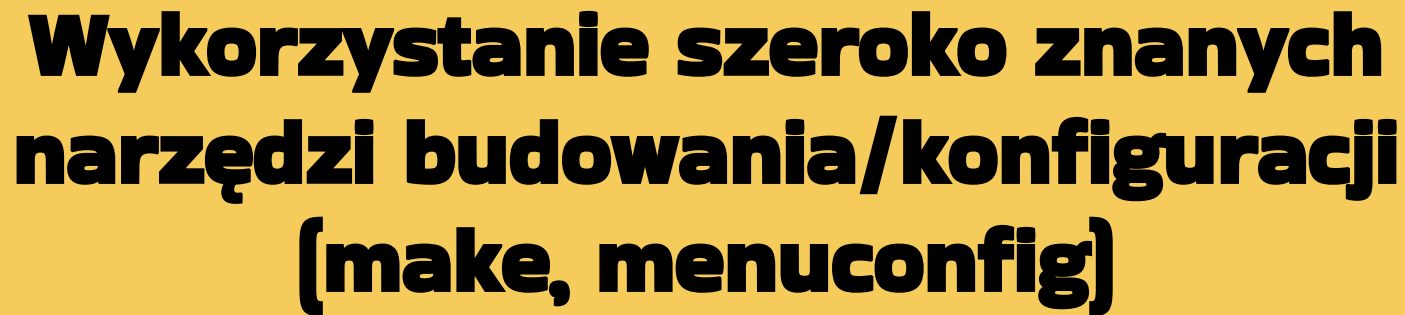






**Wsparcie dla wielu popularnych  
narzędzi – audio/video, networking,  
edytory tekstu, narzędzia systemów  
plików...**





**Wykorzystanie szeroko znanych  
narzędzi budowania/konfiguracji  
(make, menuconfig)**





# Wykorzystane przy prezentacji



## ✘ Dokumentacja Buildroota

<https://buildroot.org/downloads/manual/manual.pdf>

## ✘ Linux v5.15

<https://github.com/antmicro/linux-xlnx>

## ✘ Bootlin Buildroot training

<https://bootlin.com/doc/training/buildroot/buildroot-sli-des.pdf>

## ✘ LWM: Buildroot & Yocto

<https://lwn.net/Articles/682540/>

## ✘ Tutorial: Dodawanie paczek

<https://embeddedinn.xyz/articles/tutorial/Adding-Custom-Packages-to-Buildroot/>

## ✘ Renode

<https://github.com/renode/renode>

# Dziękuję

**CREDITS:** This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**

