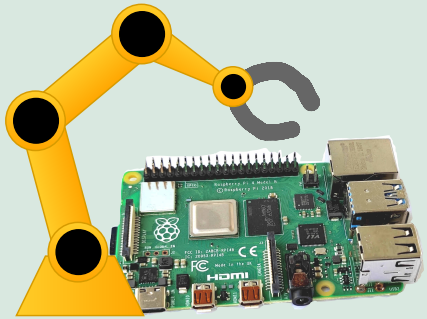
SKPS

Systemy komputerowe w sterowaniu i pomiarach

Część WZ, Wykład 1 - Wprowadzenie



dr hab. inż. Wojciech M. Zabołotny, prof. uczelni

Wydział Elektroniki i Technik Informacyjnych PW, pok. 225,
wojciech.zabolotny@pw.edu.pl (proszę dodać [SKPS] w temacie)

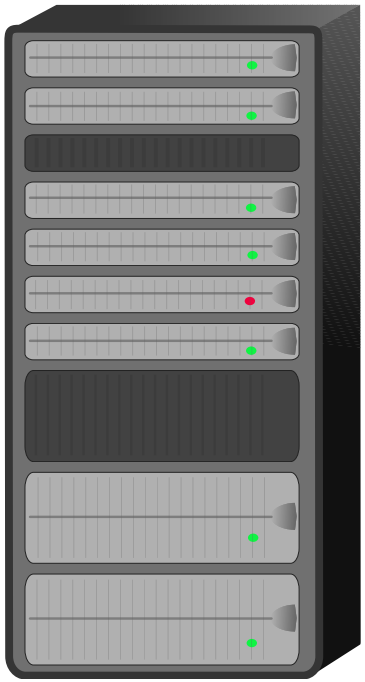
Literatura

- Łukasz Skalski, „Linux: podstawy i aplikacje dla systemów embedded”, Legionowo, Wydawnictwo BTC, 2012
- Marcin Bis, „Linux w systemach embedded”, Legionowo, Wydawnictwo BTC, 2011
- Karim Yaghmour, Building Embedded Linux Systems, Beijing, O’Reilly, 2003
- Vizuete, Daniel Manchón, „Instant Buildroot”, Packt Publishing 2013 (ISBN: 9781783289455, 9781783289462)
- Materiały internetowe – podawane na bieżąco w treści wykładu

Cele tej części

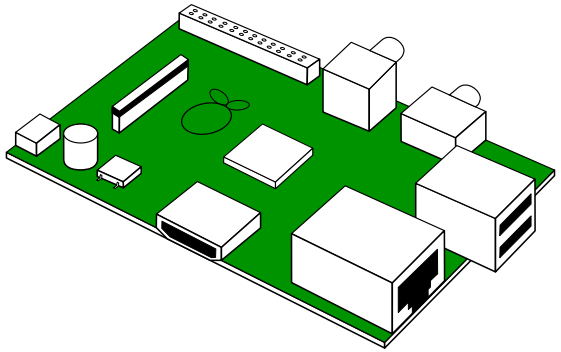
- Przekazać wiedzę i umiejętności na temat możliwości tworzenia systemów nadających się do sterowania i pomiarów, pracujących na systemach wbudowanych pracujących pod kontrolą dedykowanej wersji systemu operacyjnego GNU/Linux.
- Przygotować do części laboratoryjno-projektowej.

Co to są systemy wbudowane?



Serwery

Komputery osobiste



Systemy wbudowane

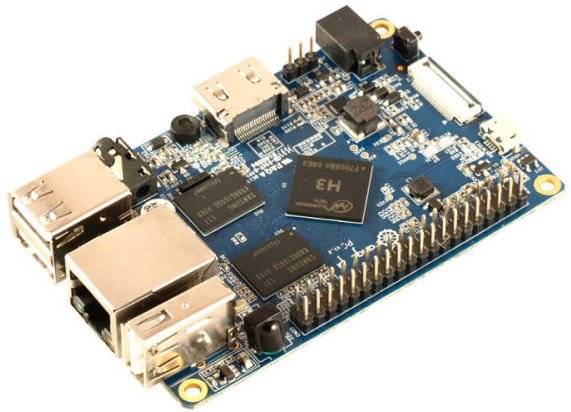
Gdzie możemy znaleźć systemy wbudowane?

- Sprzęt powszechnego użytku, np.:
  - Drukarki
  - Aparaty fotograficzne
  - Pralki
  - Punkty dostępowe WiFi
- Samochody
- Sprzęt pomiarowy
- Urządzenia medyczne
- Urządzenia przemysłowe

Możliwe realizacje systemów

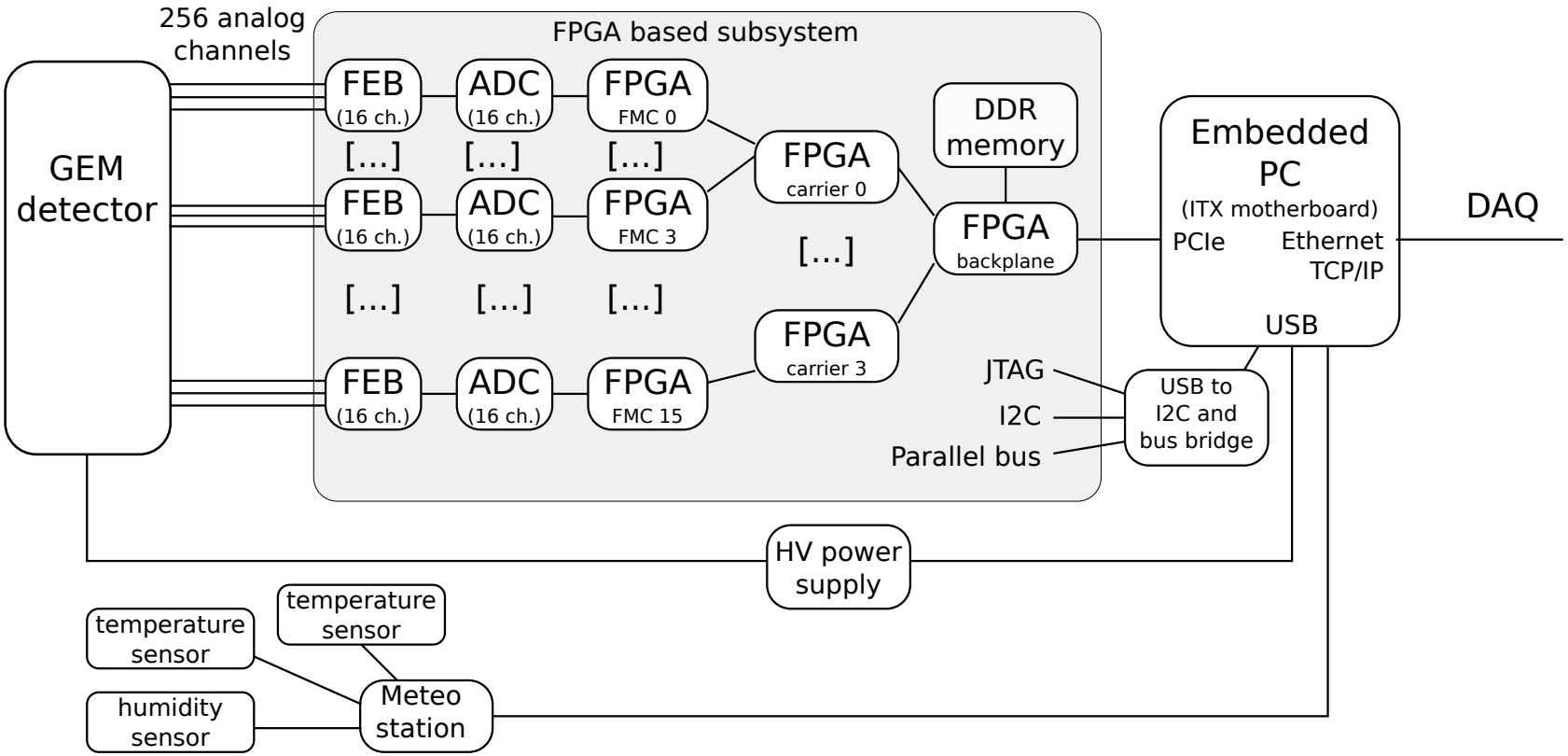
- Mikrokontrolery lub mikrokomputery
- Rozwiązanie bez systemu operacyjnego (ang. „bare metal”), z prostym systemem operacyjnym (np. FreeRTOS), lub z “normalnym” systemem operacyjnym (np. właśnie Linux).

Źródło Wikipedia



Źródło orangepi.org

Złożony system wbudowany



Systemy wbudowane bez OS (ang. „bare metal”)

- Mają swoje zalety
  - W pełni panujemy nad zasobami sprzętowymi, np. możemy ustalać priorytety obsługi przerwań.
  - Możemy stworzyć kod naprawdę zoptymalizowany pod kątem konkretnego zastosowania.
  - Możemy zminimalizować pobór mocy.
- Mają swoje wady
  - Musimy dogłębnie poznać właściwości platformy sprzętowej.
  - Łatwo o uzależnienie się od konkretnych podzespołów.
  - Często musimy “na nowo odkrywać koło”, bo nie dysponujemy standardowymi funkcjami systemu operacyjnego.

SW pracujące pod kontrolą systemu operacyjnego

- System operacyjny oferuje warstwę abstrakcji sprzętu (HAL), pozwalającą znacznie uniezależnić się od konkretnej platformy sprzętowej (choć w pewnych specjalistycznych zastosowaniach może być konieczne maksymalne wykorzystanie możliwości np. konkretnego procesora – dotyczy to jednak zazwyczaj niewielkich fragmentów kodu).
- System operacyjny oferuje zarządzanie zasobami systemu wbudowanego.
  - zarządzanie pamięcią,
  - zarządzanie czasem procesora (wielozadaniowość),
  - obsługa przerwań,
  - system plików,
  - synchronizacja dostępu do urządzeń,
  - mechanizmy komunikacji między procesami (pamięć dzielona, semafory),
  - mechanizmy ochrony (nie zawsze!),
  - standardowa obsługa złożonych mechanizmów komunikacyjnych – Internet. Bluetooth itp.

Platformy sprzętowe dostępne do eksperymentów

- Wybrane (!) platformy dobrze współpracujące z otwartym oprogramowaniem
  - Mikrokontrolery z rodziny STM32 - płytki z rodzin Nucleo lub Discovery, tanie płytki “bluepill”, “blackpill”.
  - Płytki z mikrokontrolerami ESP-32
- Możliwość tworzenia programów “Bare metal”, korzystania z systemów operacyjnych FreeRTOS lub Zephyr.
- Niektóre płytki dają możliwość szybkiego tworzenia prototypów z wykorzystaniem języka MicroPython (ograniczone możliwości pracy w czasie rzeczywistym).
- Stosując bardziej rozbudowane platformy sprzętowe, zyskujemy możliwość użycia systemów operacyjnych znanych ze “zwykłych” systemów komputerowych - na przykład Linuxa.

Dlaczego Linux?

- System używany zarówno na platformach serwerowych, jak i desktopowych, czy w systemach wbudowanych
- Otwarte źródła, możemy modyfikować źródła i dystrybuować wersję zmodyfikowaną
- Dostępne mnóstwo bibliotek i programów użytkowych, które możemy modyfikować i dystrybuować (dostępnych na licencji GPL lub LGPL).
- Możliwe jest dołączenie komponentów o zamkniętym kodzie źródłowym (przy czym należy zwracać uwagę na warunki licencji).
- Jest “w znacznej części zgodny” ze standardem POSIX.

Jakie platformy mamy do dyspozycji?

- Proste komputery jednopłytkowe – Raspberry Pi, Orange Pi, Banana Pi, Nano Pi
- Miniaturowe moduły, jak na przykład Onion Omega 2
- Komputery przemysłowe do zastosowań wbudowanych w różnych standardach, na przykład: EPIC, EBX, lub zaadaptowany z komputerów PC standard Mini-ITX.

Jakie zasoby oferują te platformy?

- Niektóre prawie dorównują starszym komputerom osobistym:
  - Odroid C4 – Amlogic ARM® Cortex®-A55 2Ghz quad core CPUs , Mali™-G31 GPU, 4Gbyte DDR4 SDRAM, Gigabit Ethernet, USB 3.0 Host x 4, USB OTG x 1 (power + data capable)
  - Raspberry Pi 4 – Broadcom 2711 1.5GHz 64-bit quad-core Cortex-A72 SoC, 802.11n WiFi, Bluetooth 5.0, Bluetooth Low Energy (BLE), up to 8GB RAM, 2x2.0+2x3.0 USB ports, Video Core VI...
- Niektóre mają znacznie skromniejsze zasoby:
  - Omega2 – 580MHz, single-core MT7688 CPU, 64MB RAM
  - Carambola 2 - 400MHz CPU AR9331, 64MB RAM

Jakiego Linuxa możemy na tych platformach uruchomić?

- Dla tych “silniejszych” istnieją dystrybucje analogiczne do dostępnych dla normalnych komputerów, np.: Armbian, Ubuntu-MATE...
- Jeśli jednak używamy płytki “mniejszej” (ze względu na cenę, wielkość, pobór mocy...), to może się to okazać niemożliwe.
- Wówczas będziemy potrzebować Linuxa specjalnie dostosowanego do platformy i do naszych potrzeb.

Ograniczenia „typowych” dystrybucji

- Duże wymagania co do pamięci, przestrzeni dyskowej,
- Wykorzystanie partycji lub pliku wymiany do wirtualnego powiększenia pamięci – zabójcze dla pamięci FLASH używanej jako dysk,
- Uzależnienie od graficznego interfejsu użytkownika (nie zawsze dostępnego w systemach wbudowanych)
- A co z możliwościami pracy w czasie rzeczywistym?

Typowe dystrybucje w systemach czasu rzeczywistego

- Aby zagwarantować szybkie rozpoczęcie realizacji zadań czasu rzeczywistego i dotrzymanie terminu ich wykonania, raczej chcemy ograniczyć do minimum liczbę zadań konkurujących o czas procesora.
- Istotne może być też ograniczenie niepotrzebnego korzystania z urządzeń generujących przerwania o długim czasie obsługi.
- Wskazane jest przygotowanie „minimalistycznej” wersji Linuxa, ściśle dostosowanej do sprzętu i funkcji, jaką ma on realizować.

Środowiska do kompilacji Linuxa dla systemów wbudowanych

- Środowiska automatycznie dbają o zapewnienie właściwego kompilatora skrośnego, bibliotek i spójny dobór opcji
- Wybrane przykłady środowisk:
  - OpenEmbedded – bazuje na tekstowych „receptach” opisujących tworzony system i jego opcje. Ostatnio wyewoluowało z niego środowisko Yocto project.
  - Buildroot – wyposażony w wygodny system konfiguracji opcji, możliwe jest względnie łatwe dodawanie nowych aplikacji, ale nie obsługuje instalowalnych (i usuwalnych) pakietów. Pakiety mogą być wybierane jedynie przed kompilacją obrazu systemu.
  - OpenWRT – podobny do Buildroot’a głównie zorientowany na tworzenie systemu dla urządzeń sieciowych. Posiada mechanizm zarządzania pakietami, co pozwala stworzyć własną dystrybucję dla urządzeń określonego typu.

Buildroot - wstęp

- Dostępny na stronie: https://buildroot.org
- Oparty na sprawdzonych i prostych technologiach – głównie kconfig i make
- Łatwy w użyciu i dający się modyfikować
- Szybki (na typowym komputerze PC z procesorem i5 lub i7, przy użyciu prekompilowanego zestawu narzędzi, kompilacja podstawowego obrazu powinna trwać kilkanaście minut, oczywiście wybór złożonych pakietów może ją znacznie wydłużyć).
- Dedykowany dla małych i średnich systemów wbudowanych (brak mechanizmu obsługi instalowalnych pakietów, często trzeba przeprowadzić pełną rekompilację obrazu, za to komponenty są zoptymalizowane do aktualnie wybranej konfiguracji)

Buildroot – początek

- Pobranie środowiska http://www.buildroot.org/download.html
- Pobieramy wersję stabilną, aktualnie: https://buildroot.org/downloads/buildroot-2021.11.1.tar.xz
- Jeśli z jakiegoś powodu potrzebujemy wersji wcześniejszej, możemy ją znaleźć tu: http://www.buildroot.org/downloads/
- Możliwy jest też dostęp do repozytorium git (187MB w dniu 27.02.2022):
  `git clone git://git.buildroot.net/buildroot`
- Lub w razie połączenia przez firewall:
  `git clone https://git.buildroot.net/git/buildroot.git`

Buildroot – źródła informacji

- The Buildroot user manual (wersja HTML)
- Prezentacje (dość stare):
  - Using Buildroot for real projects
  - Buildroot: a nice,simple and eficient embedded Linux buildsystem
- Najnowsze materiały:
  - Szkolenie z podstaw Buildroota
  - Objaśnienia jak Buildroot działa
- Książka: Vizuete, Daniel Manchón, Instant Buildroot, Packt Publishing Ltd, 2013, ISBN 9781783289455 (dostępna w ProQuest Ebook Central)
- Materiały internetowe – podawane na bieżąco w treści wykładu

BR – wstępna konfiguracja

- Po rozpakowaniu, pracujemy ze środowiskiem, używając programu make
- Programy konfigurujące z różnymi interfejsami użytkownika

  `make menuconfig`

  `make nconfig`

  `make gconfig`

  `make xconfig`
- Szokująca jest gigantyczna liczba opcji
- Dlatego bardzo istotne jest, aby skorzystać z możliwości wstępnego ustawienia domyślnej konfiguracji dla naszej platformy sprzętowej

  `make nazwa_platformy_defconfig`
- Domyślne konfiguracje dostępne są w katalogu `BRPATH/configs`
- Aktualnie (10.10.2021) są tam 261 konfiguracje. Którą wybrać?
- Na jakiej platformie będziemy uruchamiać naszego Linuxa?

Jaką platformę wybrać?

- Jeśli ktoś ma jedną z płytek obsługiwanych przez BR i chce z nią samodzielnie poeksperymentować, to może ją wybrać.
- W laboratorium będziemy używać Raspberry Pi 4B – `raspberrypi4_64_defconfig`
- Alternatywą może być użycie platformy wirtualnej, dostarczanej przez emulator QEMU, to ogranicza liczbę platform do 32.
  - Z uwagi na popularność procesorów ARM, możemy wybrać płytkę z takim procesorem.
  - Ponieważ zajmiemy się bliżej środowiskami BR i OpenWRT, to dobrym wyborem jest `qemu_aarch64_virt_defconfig`
  - Jeśli ktoś chce użyć platformy 32-bitowej, może wybrać `qemu_arm_vexpress_defconfig`

Konfiguracja środowiska

- Po ustawieniu domyślnej konfiguracji, możemy zmodyfikować aktualną konfigurację stosownie do naszych potrzeb np. `make menuconfig`
- Z uwagi na dużą liczbę opcji i rozmieszczenie ich w drzewiastej strukturze, bardzo przydatna jest funkcja wyszukiwania “/” w “menuconfig”, F8 w “nconfig” lub “Edit/Find” (CTRL+F) w “xconfig”
- Automatyczne przechodzenie do odnalezionej opcji w menuconfig – przez wciśnięcie przycisku z odpowiednią cyfrą (podaną w nawiasach w wynikach wyszukiwania), w xconfig – przez wygodne okno dialogowe.

Kompilacja pierwszego systemu

- Wstępna konfiguracja: `make qemu_aarch64_virt_defconfig`
- Żeby przyspieszyć proces kompilacji, używamy gotowego zestawu narzędzi (toolchain)
- Toolchain/Toolchain_type – wybieramy “External”
- W zasadzie system byłby gotowy do kompilacji, ale musimy zastanowić się, jak będzie uruchamiany...
- Co powstaje, gdy budowany jest system Linux?

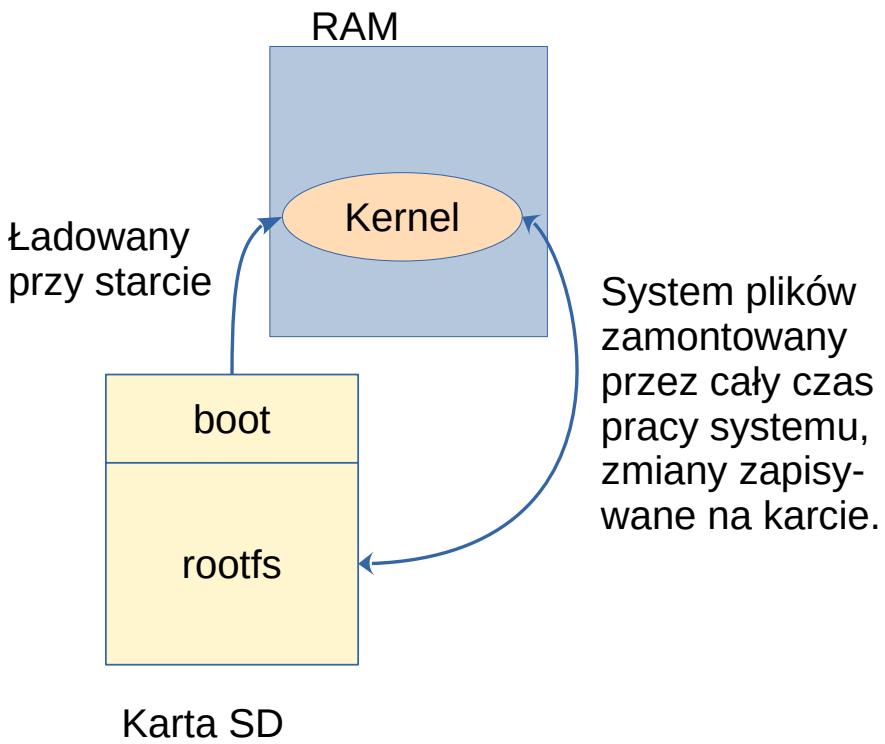
Składniki systemu Linux

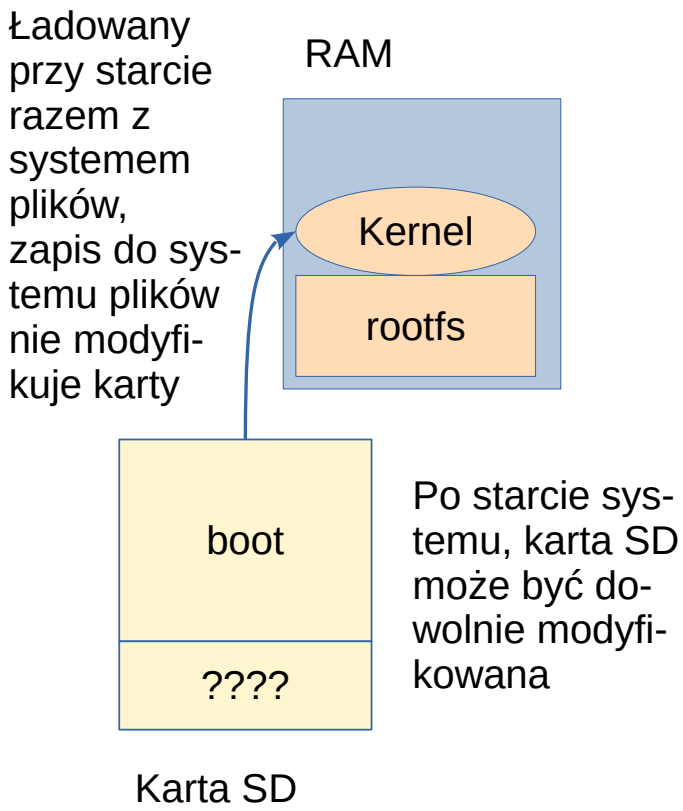
- Podstawą jest oczywiście jądro systemu Linux.
- Oprócz tego niezbędne są biblioteki systemowe, z których najważniejsza jest “libc”, lub jej odpowiednik.
- Konieczne są też aplikacje systemowe, z których najważniejszy jest “init”, uruchamiany jako pierwszy proces.
- Te dodatkowe elementy muszą być umieszczone w systemie plików – musimy wybrać właściwy.
- Często musimy też zbudować odpowiedni dla naszej platformy program ładujący (ang. bootloader).
- Wybór programu ładującego i systemu plików będzie zależał od plarformy sprzętowej.

Wybór systemu plików

- Zależy od tego, w jaką pamięć masową jest wyposażone nasze urządzenie (dysk, karta SD, pamięć FLASH) – dokładniej omówimy to później.
- Domyślna konfiguracja dla maszyny virt tworzy obraz głównego systemu plików w formacie ext4 jako plik, który może wykorzystać QEMU.
- Często zdarza się, że uruchamiamy Linuxa na nowym urządzeniu, w którym może jeszcze nie działać w pełni np. obsługa pamięci masowej. Ewentualnie chcemy móc zaktualizować obraz systemu podczas jego pracy. Wtedy rozsądnym wyborem może być dołączenie obrazu systemu plików do jądra (t.zw. Initramfs czyli ramdysk startowy).

Zwykły system plików i ramdysk startowy





Jak włączyć ramdysk startowy?

- W menu “Filesystem images”
- Uaktywniamy opcję “cpio the root filesystem” i włączamy dla niej kompresję gzip
- Uaktywniamy opcję “initial RAM filesystem...”
- Możemy wyłączyć tworzenie głównego systemu plików w formacie ext2/3/4
- Po tych zmianach wskazana jest pełna rekompilacja systemu (`make clean all`)

Kompilacja BR

- Kompilacja BR jest bardzo prosta. Wywołujemy tylko komendę “`make`”.
- Po pomyślnie zakończonej kompilacji, wyniki znajdziemy w katalogu: `output/images`.
- Możemy spróbować uruchomić nasz system, co wymaga zaznajomienia się z QEMU.

Możliwości systemu QEMU

- Możliwość emulowania systemu wbudowanego.
- Możliwość badania zachowania systemu przy różnych ograniczeniach sprzętowych (pamięć, dysk).
- Możliwość testowania z urządzeniami rzeczywistymi – np. USB.
- Możliwość testowania z urządzeniami wirtualnymi (w tym własnymi, stworzonymi przez użytkownika).
- Możliwość emulacji karty sieciowej.
- Możliwość emulacji karty dźwiękowej.

Pierwsze uruchomienie

- Informację o tym, jak uruchomić domyślną konfigurację możemy znaleźć w pliku: `board/qemu/aarch64-virt/readme.txt`
- Zasadnicza komenda uruchamiająca QEMU (zadziała z Initramfs):

```
qemu-system-aarch64 -M virt -cpu cortex-a57
 -nographic -smp 4 -kernel output/images/Image
 -append "console=ttyAMA0"
```

- Uwaga! Przy każdym uruchomieniu systemu przywracany jest początkowy stan ramdysku. Ma to swoje zalety (można eksperymentować bez obaw że uszkodzi się system plików) i wady (brak możliwości zapisania czegoś na stałe).

Pierwsze uruchomienie c.d.

- Jeśli chcemy uruchomić Linuxa korzystającego z systemu plików ext4, musimy inaczej uruchomić QEMU:

```
qemu-system-aarch64 -M virt -cpu cortex-a57 -nographic -smp 4
 -kernel output/images/Image -append "root=/dev/vda console=ttyAMA0"
 -drive file=output/images/rootfs.ext4,if=none,format=raw,id=hd0

 -device virtio-blk-device,drive=hd0
```

- Co jeśli chcemy dodatkowo umożliwić połączenie naszego emulowanego systemu z siecią (najprostszy sposób, tylko połączenia wychodzące)?

```
 -netdev user,id=eth0 -device  virtio-net-device,netdev=eth0
```

Dodatkowe sesje powłoki

- Z naszym systemem porozumiewamy się przez konsolę tekstową. Czy możemy komunikować się z kilkoma programami równocześnie?
- Jeśli chcemy uruchomić niezależnie dwa programy, może nam się przydać program tmux (Uwaga! Dla poprawnego działania musimy zapewnić obsługę UTF-8 w naszym systemie, wymaga to dodania np. “en_US.UTF-8” w opcji “System configuration/Generate locale data”).
- Innym rozwiązaniem może być użycie programu screen, jednak ma on mniejsze możliwości (Uwaga! Screen używa CTRL+A jako znaku sterującego. W przypadku pracy przez konsolę QEMU lub przez minicom, należy wysłać ten znak dwukrotnie)

Wybór komponentów systemu

- Istotnym punktem konfiguracji systemu jest wybór komponentów.
- Większość komponentów systemu możemy znaleźć w podmenu “Target_packages”.
- Niektóre z nich mogą nie być widoczne, dopóki nie zostaną spełnione ich zależności (przykład: pakiet “eudev” jest niewidoczny, dopóki nie włączymy odpowiedniej opcji w sekcji “System_configuration/dev_management”.
- Pomocne może być wówczas użycie funkcji wyszukiwania (możemy też podejrzeć listę niespełnionych zależności).

Rekompilacja BR

- Po zmianach konfiguracji BR pojawia się konieczność rekompilacji.
- Najszybszy wynik daje “`make`”, ale może on być niepoprawny.
- BR używa uproszczonego systemu badania zależności między pakietami.
- Konfiguracja pakietu “A” może wpływać na sposób kompilacji pakietu “B”. Jeśli zmienimy konfigurację “A”, to make przeprowadzi rekompilację “A”, ale nie “B”. Pakiet “B” będzie działać niewłaściwie.
- Dotyczy to zwłaszcza rekompilacji bibliotek.
- Możemy ręcznie próbować wymuszać rekompilację poszczególnych komponentów (`make <pkg1>-dirclean <pkg2>-dirclean all`).
- Jednak ostateczną “produkcyjną” wersję systemu najlepiej zbudować przez “`make clean all`” (co może zająć nawet 30 minut i więcej...).

Korzystanie z katalogu maszyny “gospodarza”

- Możemy do tego wykorzystać wirtualny system plików “Plan 9” (9P) obsługiwany przez QEMU.
- W jądrze budowanego Linuxa należy włączyć opcje:

```
CONFIG_NET_9P=y
CONFIG_NET_9P_VIRTIO=y
CONFIG_9P_FS=y
CONFIG_9P_FS_POSIX_ACL=y
```

- Kernel konfigurujemy przez “`make linux-menuconfig`”
- Jednakże zmiana konfiguracji jądra może wymagać rekompilacji całego systemu.
- Konfiguracja zapisywana w katalogu output, jest więc niszczona przez “`make clean`”. Jak ją przechować?

Przechowywanie konfiguracji jądra

- Polecenie “`make linux-update-defconfig`” wprowadza nasze zmiany do konfiguracji domyślnej dla naszej płyty.
- To gwarantuje przetrwanie rekompilacji przez “`make clean all`”, ale nie zachowa zmian w razie konieczności reinstalacji całego BR.
- Lepszym wyjściem może być skopiowanie konfiguracji jądra do pliku na zewnątrz BR i odpowiednie ustawienie ścieżki w opcjach Kernel/Kernel_configuration.
- Wtedy `make linux-update-defconfig` będzie modyfikować naszą nową konfigurację!
- Jak zrekompilować samo jądro? Można spróbować użyć `make linux-dirclean all`, ale może to wygenerować niepoprawidłowy system, jeśli konfiguracja innych komponentów zależała od zmienionych ustawień jądra.

Co, jeśli musimy dodać nasz plik do systemu tworzonego przez BR?

- Możemy chcieć dodać, lub nadpisać jakiś zbiór konfiguracyjny, albo dodać nasz własny skrypt.
- Jak możemy to zrobić?
- Pierwsze rozwiązanie – szybkie ale nie nadające się do zastosowań “produkcyjnych”: Dopisujemy dodany zbiór w katalogu output/target i ponownie wykonujemy “make”.
- Rozwiązanie poprawne – korzystamy z nakładek na system plików.
  `System_configuration/Root_filesystem_overlay_directories`
  Można wyspecyfikować kilka katalogów, oddzielając je spacją.

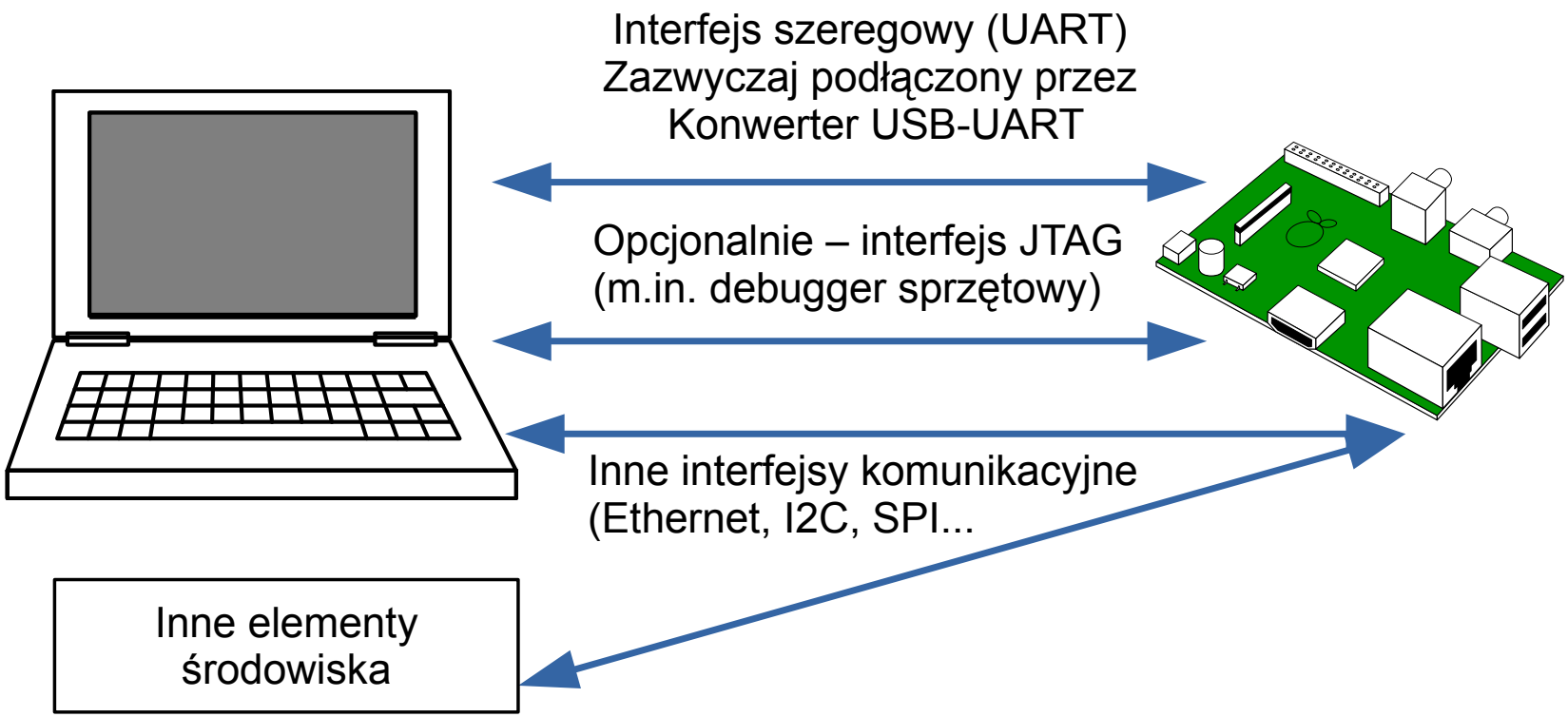
Jak przechować (zarchiwizować) stan BR

- Między sesjami laboratorium możemy musieć przechować nasz projekt, na przykład w repozytorium.
- Ważne jest żeby umieścić w repozytorium tylko wszystkie informacje niezbędne do jego odtworzenia.
- Zbiór `.config` z głównego katalogu BR.
- Zmodyfikowane zbiory konfiguracyjne komponentów (kernel, busybox itp.).
- Katalogi z nakładkami na system plików.
- Próba bezpośredniego skopiowania katalogu BR na system CIFS (SAMBA) i z powrotem skończy się uszkodzeniem tego katalogu. (Nie są zachowywane kluczowe atrybuty plików i linki symboliczne).

QEMU a praca z rzeczywistym sprzętem

- Jak praca z Linuxem na emulowanym systemie przydaje się w uruchamianiu rzeczywistego sprzętu?
- W przypadku pracy z rzeczywistym systemem wbudowanym najczęściej najpierw komunikujemy się z nim przez konsolę szeregową.
- Gdy system pracuje stabilnie, możliwa staje się komunikacja przez sieć (a w przypadku systemów z GUI, także przez GUI).
- QEMU pozwala na komunikację z niektórymi urządzeniami podłączonymi do “gospodarza”.
- QEMU możemy rozszerzać o nowe modele urządzeń (zajmiemy się tym znacznie później).
- Przydatność uruchamiania i testowania w takich warunkach została zweryfikowana w rzeczywistych projektach (np. [1]).

Środowisko do pracy z systemem wbudowanym



Sugestie do przećwiczenia

- Warto spróbować skompilować własny obraz systemu Linux w środowisku Buildroot i spróbować uruchomić go w emulatorze QEMU.
- Warto sprawdzić działanie funkcji zapewniających przechowanie zmienionej konfiguracji jądra podczas pełnej rekompilacji.
- W ramach eksperymentu można wypróbować inne emulowane platformy niż `aarch64_virt`.
- Jeśli ktoś ma dostęp do rzeczywistego komputera jednopłytkowego, obsługiwanego przez BR, można spróbować skompilować dla niego obraz systemu i go uruchomić na rzeczywistym sprzęcie.

Dziękuję za uwagę!