

Laboratorium 10

Tworzenie oprogramowania dla urządzeń wbudowanych

Plan prezentacji

Tworzenie oprogramowania dla urządzeń wbudowanych

Paweł Malczyk

Zakład Teorii Maszyn i Robotów
Instytut Techniki Lotniczej i Mechaniki Stosowanej
Wydział Mechaniczny Energetyki i Lotnictwa
Politechnika Warszawska

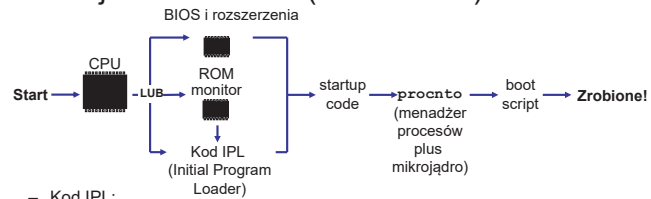
- Proces rozruchu i ładowania systemu operacyjnego QNX Neutrino
- Budowanie obrazu systemu QNX Neutrino na platformie typu BeagleBone Black
- Komputer BeagleBone Black
- Budowanie obrazu systemu z QNX Momentics

Metody programowania robotów

Metody programowania robotów

Kolejność rozruchu

- Kolejność rozruchu (bootowania):



- Kod IPL:
 - CPU wybiera kod IPL a następnie przechodzi do startup code
- startup code:
 - Ustawia urządzenia i przygotowuje środowisko dla mikrojądra `procnto`
- `procnto`:
 - Ustawia mikrojądro i ładuje skrypt boot script
- boot script zawiera:
 - Sterowniki, procesy systemowe i użytkownika

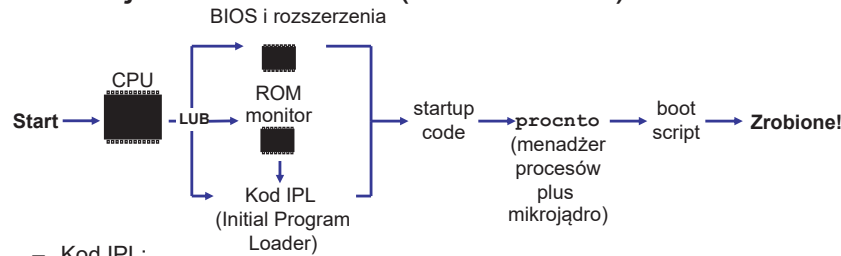
Uwaga: BBB nie posiada BIOS-u. Do ładowania systemu używane są bootloader-y.

Metody programowania robotów

Metody programowania robotów

Proces rozruchu i ładowania systemu operacyjnego QNX Neutrino

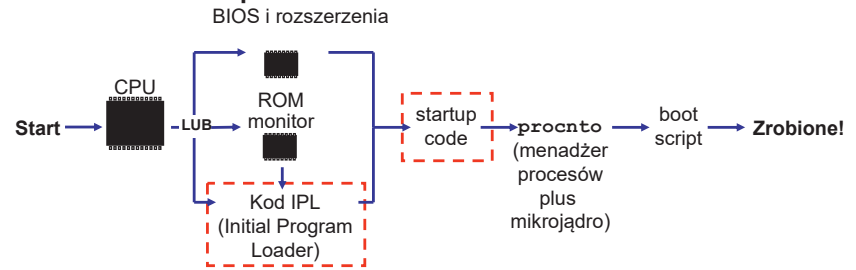
• Kolejność rozruchu (bootowania):



- Kod IPL:
 - CPU wybiera kod IPL a następnie przechodzi do startup code
- startup code:
 - Ustawia urządzenia i przygotowuje środowisko dla mikrojądra **procnto**
- **procnto**:
 - Ustawia mikrojądro i ładuje skrypt boot script
- boot script zawiera:
 - Sterowniki, procesy systemowe i użytkownika

Uwaga: BBB nie posiada BIOS-u. Do ładowania systemu używane są bootloader-y.

• IPL i startup code:



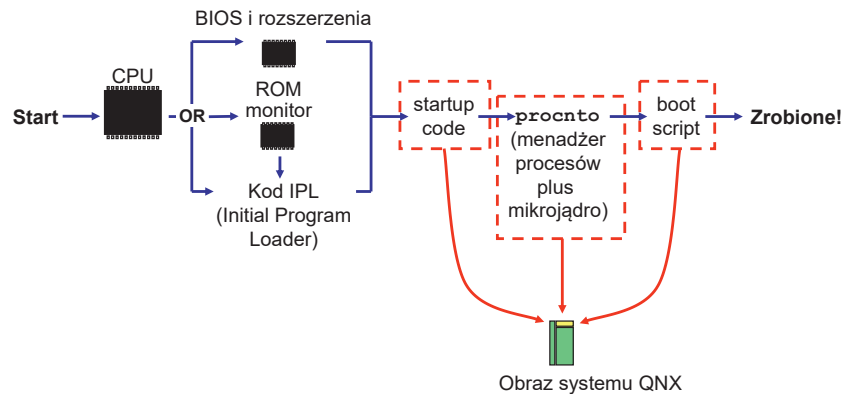
- Skrypty rozruchowe IPL i inicjujące (startup code):

- Są specyficzne dla określonych urządzeń wbudowanych
- Są częścią pakietów Board Support Package (BSP)

Co znajdują się w pliku obrazu systemu?

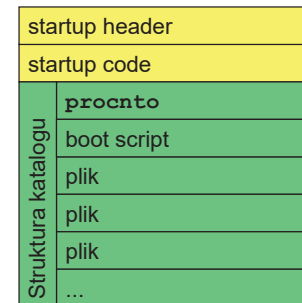
Struktura pliku obrazu systemu

• Zawartość obrazu systemu:



• Plik obrazu systemu operacyjnego (SO)

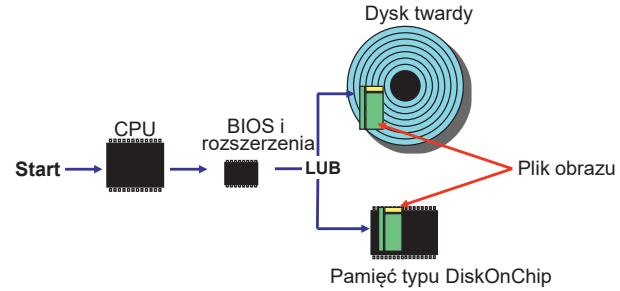
- Zawiera wiele komponentów niezbędnych do inicjalizacji
- Struktura pliku obrazu:



- Przykładowe pliki:
- Sterowniki sieci,
 - Stos TCP/IP,
 - Sterowniki dysku twardego,
 - Pliki konfiguracyjne

- Możliwość inicjalizacji SO:
 1. ROM/flash
 2. Dysk twardy
 3. Sieć/połączenie szeregowo

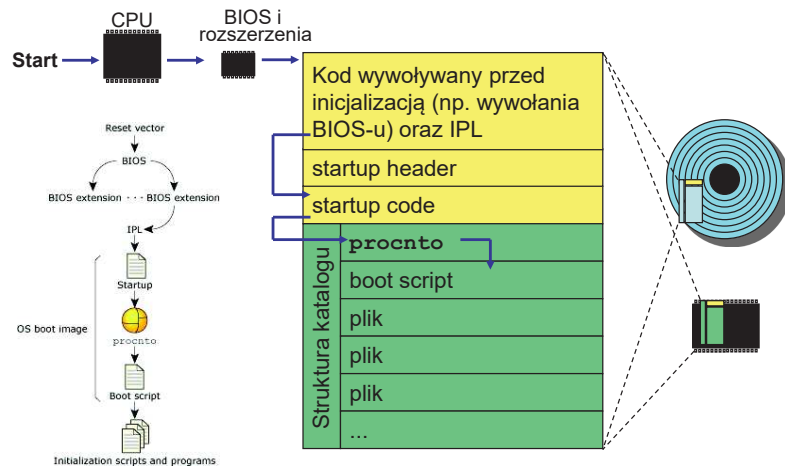
- Inicjalizacja SO z dysku twardego:



– Obraz SO jest ładowany do RAM-u za pomocą różnorodnych komponentów....

Dysk twardy

- Obraz systemu:

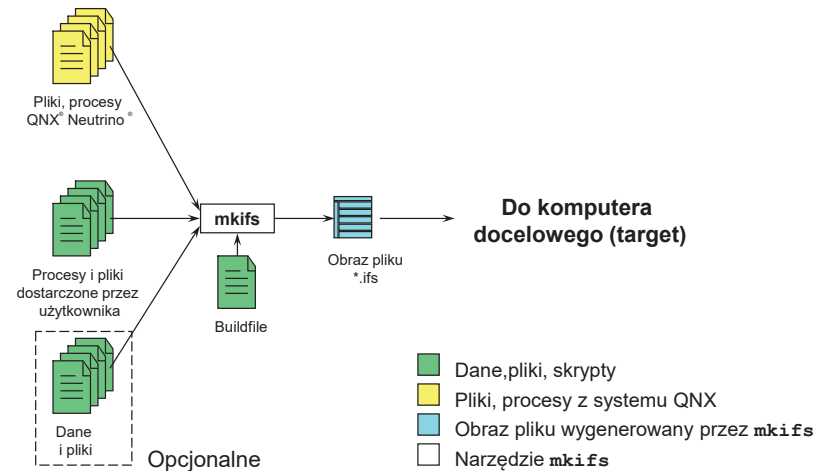


Budowanie obrazu systemu QNX Neutrino

- Plik obrazu systemu operacyjnego (SO)
 - Zawiera wiele komponentów niezbędnych do inicjalizacji
 - Struktura pliku obrazu:



Po inicjalizacji pliki znajdują się w katalogu: `/proc/boot`



- Skrypt startowy oraz mikrojądro z menadżerem procesów:
 - `startup-*`
 - `procnto`
- Pozostałe komponenty:
 - sterowniki and menadżery, e.g.: `io-net`, `devc-ser*`, `devb-eide`
 - `esh` (embedded shell), `ksh`
 - pliki wykonywalne użytkownika i dane

- Czym jest plik buildfile?
 - Określa pliki/polecenia, które będą dostępne w obrazie,
 - Porządek wywołania podczas rozruchu,
 - Procesy linii wywoływane z wiersza poleceń, zmienne środowiskowe
 - Plik zawiera opcje ładowania plików wykonywalnych

Plik buildfile dla minimalnego systemu `min_image.ifs`

Skrypt buildfile – format

```
# Plik Buildfile dla minimalnego systemu
[virtual=x86,bios +compress] boot={
  startup-bios
  PATH=:/proc/boot:/bin:/sbin:/usr/bin:/usr/sbin
  LD_LIBRARY_PATH=:/proc/boot:/lib:/usr/lib:/lib/dll
}
procnto -vv
}

[+script] startup-script={
  display_msg "Moj obraz systemu QNX..."
  devc-con &
  reopen /dev/con1
  hello
}

libc.so
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
[code=uip data=copy perms=+r,+x]
devc-con
hello
```

Bootstrap file
(skrypt inicjujący)

Startup script
(skrypt ładujący procesy, usługi, urządzenia)

Reszta pliku buildfile
(biblioteki, programy, ustawienia)

bootscript

- Format buildfile:
 - atrybut nazwa_pliku zawartość
- Skrypt może zawierać puste linie i komentarze zaznaczone za pomocą znaku „#”
- Dwa typy atrybutów:
 - - boolowski, np. [+atrybut] lub [-atrybut]
 - - wartość, np. [atrybut=wartość]
- Możliwość utworzenia pliku inline


```
readme = {
  W systemie plików zostanie utworzony plik
  dostępny w /proc/boot/readme.
}
```

Skrypt buildfile – komendy wbudowane

Plik buildfile dla minimalnego systemu `min_image.ifs`

- Komendy wbudowane:
 - `display_msg` wyświetla tekst
 - `procmgr_symlink` link symboliczny, odpowiednik wywołania `ln -s`
 - `reopen` strumienie `stdin`, `stdout`, and `stderr` są przekierowane do określonego pliku
 - `waitfor` czeka aż powstanie określony plik
- Przykłady wywołań `display_msg`, `reopen` można znaleźć w skrypcie.

```
# Plik Buildfile dla minimalnego systemu
[virtual=x86,bios +compress] boot={
  startup-bios
  PATH=:/proc/boot:/bin:/sbin:/usr/bin:/usr/sbin
  LD_LIBRARY_PATH=:/proc/boot:/lib:/usr/lib:/lib/dll
}
procnto -vv
}

[+script] startup-script={
  display_msg "Moj obraz systemu QNX..."
  devc-con &
  reopen /dev/con1
  hello
}

libc.so
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
[code=uip data=copy perms=+r,+x]
devc-con
hello
```

Bootstrap file
(skrypt inicjujący)

Startup script
(skrypt ładujący procesy, usługi, urządzenia)

Reszta pliku buildfile
(biblioteki, programy, ustawienia)

bootscript

Plik Bootstrap file (skrypt inicjujący)

```
# Plik Buildfile dla minimalnego systemu
[virtual=x86,bios +compress] boot={
  startup-bios
  PATH=/proc/boot:/bin:/sbin:/usr/bin:/usr/sbin
  LD_LIBRARY_PATH=/proc/boot:/lib:/usr/lib:/lib/dll
  procnto -vv
}
```

Bootstrap file
(skrypt
inicjujący)

Bootstrap file zawiera:

- atrybut `[virtual=x86,bios +compress]` — Adresowanie pamięci, Architektura CPU, Dla układów z BIOS-em
- nazwa pliku `boot` — Nazwa pliku, Kompresja obrazu *.ifs
- zawartość `startup-bios` — Program startowy. Inicjalizuje sprzęt i wywołuje `procnto`
- `PATH=/proc/boot LD_LIBRARY_PATH=/proc/boot procnto -vv` — Zmienne środowiskowe, Mikrojądro

Reszta pliku buildfile

```
[+script] startup-script={
  display_msg "Moj obraz systemu QNX..."
  devc-con &
  reopen /dev/con1
  hello
}
```

Startup script
(skrypt ładujący
procesy, usługi,
urządzenia)

```
libc.so
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
[code=uip data=copy perms=+r,+x]
devc-con
hello
```

Reszta pliku
buildfile
(biblioteki, programy,
ustawienia)

Narzędzie `mkifs` szuka plików w następujących katalogach:

```
${QNX_TARGET}\${PROCESSOR}/sbin oraz
.../usr/sbin, .../boot/sys, .../bin, .../usr/bin,
.../lib, .../lib/dll, .../usr/lib, .../usr/photon/bin
```

Utworzenie obrazu w IDE

- Otwieramy IDE QNX Momentics i tworzymy projekt QNX System Builder Project

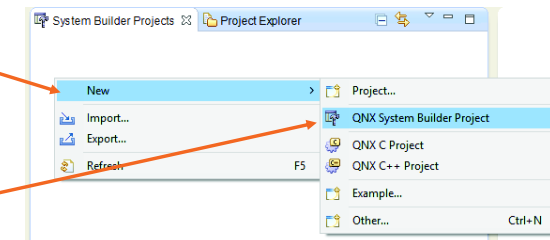
Budowanie obrazu systemu z QNX Momentics

1 Menu File – New

2 Wybrać QNX System Builder Project

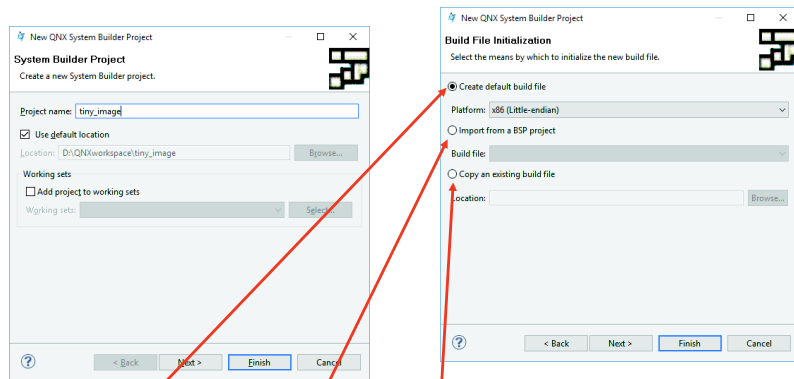
3 Wpisać nazwę projektu

4 Kliknąć Next...



Utworzenie obrazu w IDE Momentics

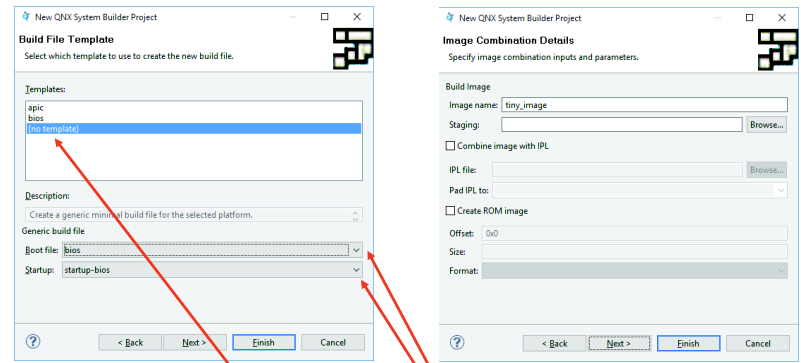
- Wybrać stosowną opcję:



Start od zera Import z BSP
Skopiuj istniejący plik

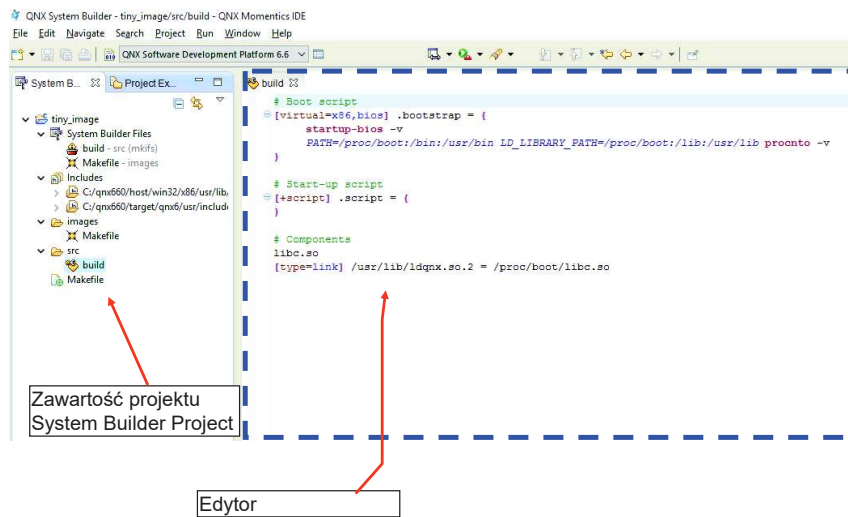
Utworzenie obrazu w IDE Momentics

- Wybrać stosowną opcję:



Minimalny plik build Maszyna z bios-em

Projekt System Builder

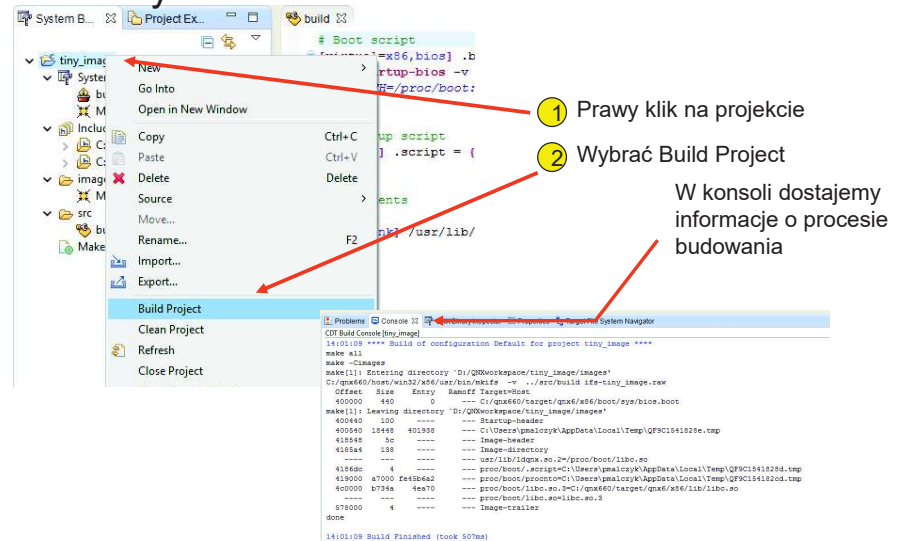


Zawartość projektu System Builder Project

Edytor

Budowanie obrazu

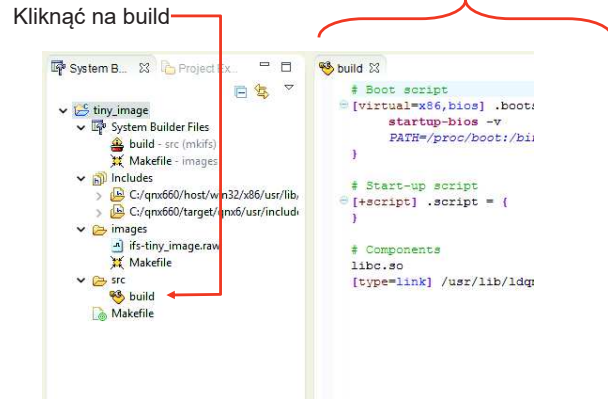
- Aby zbudować obraz



- 1 Prawy klik na projekcie
- 2 Wybrać Build Project

W konsoli dostajemy informacje o procesie budowania

- Otworzyć edytor obrazu systemu:
Edytor obrazu systemu



1. Zalogować się do systemu QNX. Wylistować zawartość katalogu `/boot` zawierającą obrazy systemu.
2. Sprawdzić zawartość załadowanego obrazu systemu znajdującego się w katalogu `/proc/boot`.
3. Napisać i zbudować w IDE program typu „Hello!”. Plik wykonywalny skopiować do katalogu `C:\qnx660\target\qnx6\x86\bin`
4. Utworzyć nowy projekt QNX Builder System Project o nazwie **min_image**; wybrać platformę x86 oraz opcje: no template, boot file bios oraz startup-bios.
5. Napisać skrypt **build** w katalogu **src** wg poniższej instrukcji.
6. Zbudować obraz systemu.
7. Przekopiować powstały obraz SO do katalogu `.boot` na maszynie wirtualnej. Obejrzeć zawartość obrazu **min_image.raw** wpisując w konsoli polecenie:


```
dumpifs min_image.raw
```
8. Zrestartować maszynę poleceniem **shutdown** i wybrać obraz systemu **min_image.raw** z menu.

```
# Plik build dla minimalnego systemu
[virtual=x86,bios +compress] boot={
    startup-bios
    PATH=:/proc/boot:/bin:/sbin:/usr/bin:/usr/sbin
    LD_LIBRARY_PATH=:/proc/boot:/lib:/usr/lib:/lib/dll
procnto -vv
}

[+script] startup-script={
    display_msg "Moj obraz systemu QNX..."
    devc-con &
    reopen /dev/con1
    hello
}

libc.so
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
[code=uip data=copy perms=+r,+x]
devc-con
hello
```

Bootstrap file (skrypt inicjujący)

Startup script (skrypt ładujący procesy, usługi, urządzenia)

Reszta pliku buildfile (biblioteki, programy, ustawienia)

bootscript

1. Należy zbudować bardziej skomplikowany obraz systemu operacyjnego QNX o nazwie **tiny_image.raw**.
2. Przetestować działanie wirtualnych konsol i dołączonych poleceń.

```
[virtual=x86,bios +compress] boot={
    startup-bios
    PATH=:/proc/boot:/bin:/sbin:/usr/bin:/usr/sbin
    LD_LIBRARY_PATH=:/proc/boot:/lib:/usr/lib:/lib/dll procnto -vv
}

[+script] startup-script={
    display_msg "Moj obraz systemu QNX..."
    hello &
    display_msg "Uruchamiam sterownik dysku..."
    devb-eide blk auto=partition cam quiet &
    display_msg "Otwieram wirtualne konsole..."
    # Ctrl+Alt+1 oraz Ctrl+alt+2
    devc-con -n2 &
    reopen /dev/con1
    [+session] HOME=/ ksh &
    reopen /dev/con2
    [+session] HOME=/ ksh &
}

# katalog /tmp w pamieci dzielonej
[type=link] /tmp=/dev/shmem
libc.so
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
# biblioteki do obsługi dysku
io-blk.so
cam-disk.so
libc.so
```

```
# Dowiazanie symboliczne
[type=link]
/usr/lib/libcam.so.2=/proc/boot/libcam.so

# system plikow QNX
fs-qnx4.so

[code=uip data=copy perms=+r,+x]
kill
cat
ls
ksh
devc-con
less
ps
pidin
hogs
uname
mkdir
devb-eide
touch
hello
mount
shutdown
```


Ćwiczenie 2 - Plik build dla systemu `tiny_image.raw`

```
[virtual=x86,bios +compress] boot={
  startup-bios
  PATH=/proc/boot:/bin:/sbin:/usr/bin:/usr/sbin

LD_LIBRARY_PATH=/proc/boot:/lib:/usr/lib:/lib/dll
procnto -vv
}
[+script] startup-script={
  display_msg "Moj obraz systemu QNX..."
  hello &
  display_msg "Uruchamiam sterownik dysku..."
  devb-eide blk auto=partition cam quiet &
  # zamontuj partycje
  waitfor /dev/hd0t79
  mount /dev/hd0t79 /pliki
  display_msg "Otwieram wirtualne konsole..."
  # Ctrl+Alt+1 oraz Ctrl+alt+2
  devc-con -n2 &
  reopen /dev/con1
  [+session] HOME=/ ksh &
  reopen /dev/con2
  [+session] HOME=/ ksh &
}
# katalog /tmp w pamieci dzielonej
[type=link] /tmp=/dev/shmem
libc.so
[type=link] /usr/lib/ldqnx.so.2=/proc/boot/libc.so
# biblioteki do obslugi dysku
io-blk.so
cam-disk.so
libcam.so

# Dowiazanie symboliczne
[type=link]
/usr/lib/libcam.so.2=/proc/boot/libcam.so

# system plikow QNX
fs-qnx4.so

[code=uiop data=copy perms=r,x]
kill
cat
ls
ksh
devc-con
less
ps
pidin
hogs
uname
mkdir
devb-eide
touch
hello
mount
shutdown
```

Komputer BeagleBone Black

Systemy wbudowane

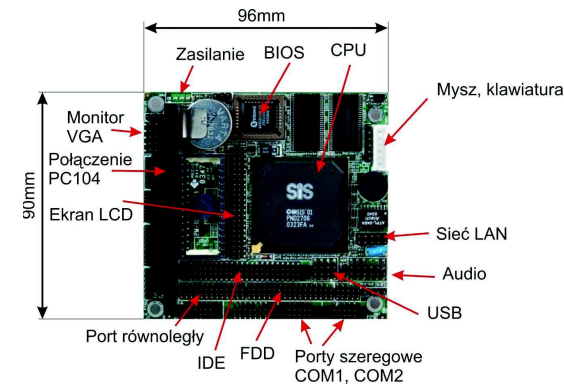
Komputer PM-1045 (SIS552) 128MB (standard PC104)

System wbudowany jest to system komputerowy będący częścią większego systemu i wykonujący istotną część jego funkcji.

Wymagania na komputery przeznaczone do sterowania i zastosowań wbudowanych.

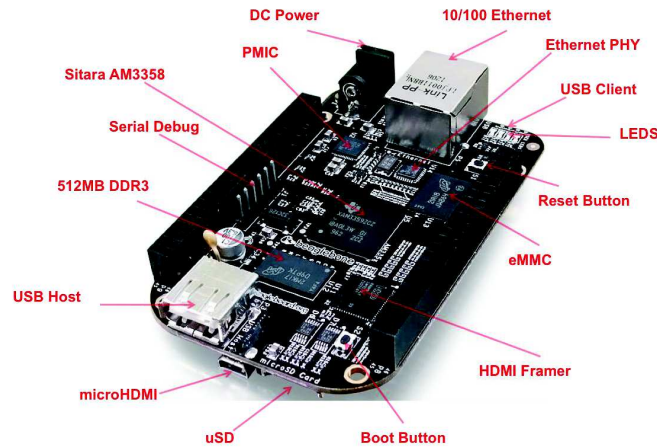
1. Wymagana jest odporność na pracę w trudnych warunkach otoczenia (wibracje, zapylenie, wilgoć), dopuszczalny jest szeroki zakres temperatur otoczenia.
2. Przeznaczone są do pracy ciągłej - brak jest elementów ruchomych (dyski obrotowe, wentylatory, napędy dyskietek), wymagana jest trwałość, łatwość serwisowania.
3. Oprogramowanie umieszczone jest w pamięci nieulotnej – ROM, flash, EPROM lub podobnej. Stosowane jest wsparcie sprzętowe dla osiągnięcia niezawodnej pracy – budzik (*ang. watchdog*), pamięci ECC, magistrala z kontrolą parzystości, poszerzona diagnostyka.

Przykład: standard PC104 ...



Standard PC104:

1. Standard mechaniczny (np. konstrukcja i wymiary, materiały)
2. Standard elektryczny (końcówki, własności elektryczne)
3. Standard logiczny (adresy, dane, magistrala, przerwania, DMA).



Komponent	Specyfikacja
Procesor	AM335x 1GHz ARM Cortex-A8 + 2x32bitPRU (Programmable Real-time Units)
Pamięć RAM	512MB DDR3 RAM
Grafika	microHDMI (do 1280x1024@60Hz) + audio
Dysk	eMMC + karta microSD
Ethernet	LAN8710A, wsparcie dla DHCP, złącze RJ45
USB	1xminiUSB2.0, 1xUSB2.0
Debug	UART, JTAG
Porty WY-WE	2x46 dostępnych pinów dla wyjść i wejść analogowych, timery, I ² C, UART, CAN, SPI, LCD,
Zasilanie	5VDC

Sprzęt ...

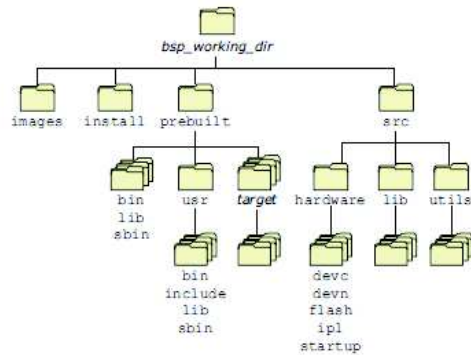
Praca z pakietami BSP



Pakiety wsparcia BSP (Board Support Packages)

- **IPL** - minimalnie konfiguruja sprzęt, aby zbudować środowisko pozwalające na uruchomienie dalszych komponentów SO.
- **startup** – kopiowanie, dekompresja obrazu SO, konfiguracja sprzętu, start mikrojądra.
- **buildfile** – określa pliki, komendy, zmienne wewnątrz obrazu SO.
- wsparcie dla urządzeń na pokładzie urządzenia wbudowanego, np. dostęp do sieci.
- sterowniki urządzeń.

Pakiety wsparcia BSP (Board Support Packages)



<http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/TiAm335Beaglebone>

1. Pobrać pakiet wsparcia BSP oraz bootloader-y (**MLO** i **u-boot.img**) dla QNX SDP 6.6.0 ze strony:
<http://community.qnx.com/sf/wiki/do/viewPage/projects.bsp/wiki/TiAm335Beaglebone>
2. W środowisku IDE zaimportować BSP do przestrzeni roboczej poprzez opcję Import oraz wybór **QNX->QNX Source Package and BSP**.
3. Obejrzeć plik **build**, który znajduje się w drzewie **System Builder Files**. Zidentyfikować skrypt inicjujący, skrypt ładujący usługi/procesy oraz listę dołączonych do obrazu plików.
4. Zbudować projekt.
5. Przygotować kartę SD poprzez utworzenie partycji FAT32 oraz ustawienie partycji jako aktywnej.
6. Skopiować na kartę pamięci MLO, u-boot.img oraz obraz systemu QNX **ifs-ti-am335x-beaglebone.bin**
7. Przełożyć kartę SD do urządzenia BBB, a następnie uruchomić BBB.
8. Za pomocą terminala szeregowego sprawdzić poprawność zainicjowania i załadowania QNX-a.

