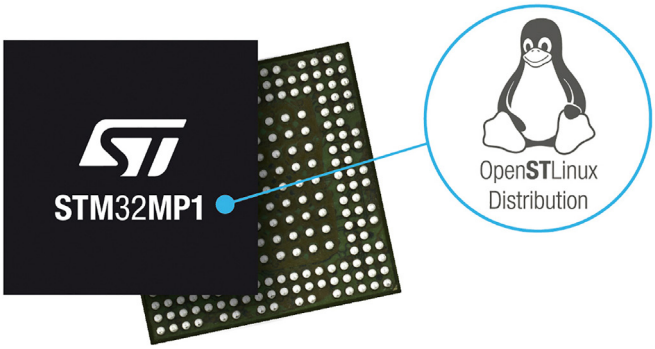
OpenSTLinux

dla procesorów z rodziny STM32MP1 (2)

W drugiej części serii przyjrzymy się bliżej sposobowi modyfikacji domyślnego obrazu systemu OpenSTLinux. System ten bazuje na projekcie Yocto, dlatego obowiązuje ten sam schemat receptur. W ramach przykładu, wprowadzimy modyfikacje do receptury jądra, włączając do kompilacji sterowniki następujących modułów: odbiornik GPS U-Blox, sterownik LED PCA9532 i akcelerometr z żyroskopem LSM6DS3. Do ich poprawnego działania wymagane są także zmiany w plikach Device Tree, jednak tym zajmiemy się w kolejnej części.



Warstwy i receptury

Wszystkie komponenty systemu są zorganizowane w receptury, opisujące sposób ich budowania oraz wzajemne zależności. Pliki receptur mają rozszerzenie *bb* i znajdują się w katalogach o nawie *recipes-**, które z kolei umieszczone są w odpowiednich warstwach budowania systemu. Receptury w *Yocto* mogą być modyfikowane poprzez pliki *bbappend*, które wprowadzają zmiany do oryginalnych plików *bb* zgodnie z kolejnością wyznaczoną przez priorytet warstwy, w której się znajdują. W ten sposób została także przygotowana warstwa *meta-somlabs* [1], którą mieliśmy okazję pobrać przy okazji budowania obrazu systemu w poprzedniej części artykułu. Warstwa ta zawiera szereg katalogów *recipes-**, wprowadzających zmiany do receptur, pochodzących z innych warstw lub zawierających nowe receptury, dodawane do budowanego systemu. Jedną z tych receptur jest *linux-stm32mp*, opisująca sposób budowania jądra systemu *Linux*. Poniższy diagram przedstawia strukturę receptur na przykładzie receptur jądra (*recipes-kernel*) i aplikacji demonstracyjnej (*recipes-somlabs*):

```
├── recipes-kernel – katalog główny warstwy dostarczanej przez SoMLabs.
│   └── linux – receptura budowania jądra systemu.
│       ├── linux-stm32mp – pliki używane przez recepturę w tej warstwie.
│       │   ├── 0001-Add-support-for-VisionSOM-STM32MP1-board.patch
│       │   ├── 0001-Fixed-display-transition-between-u-boot-and-kernel.patch
│       │   ├── 0001-Uptated-ILI9881C-driver-for-Powertip-PH720128T003.patch
│       │   └── stm32mp157a_visionsom_mx_defconfig
│       └── linux-stm32mp_4.19.bbappend – plik z modyfikacją oryginalnej receptury.
├── recipes-somlabs – receptury utworzone przez SoMLabs.
│   └── somlabs-demo – receptura aplikacji demonstracyjnej.
│       ├── somlabs-demo – pliki używane przez recepturę.
│       │   ├── SomlabsDemo
│       │   └── somlabs_demo_run.sh
│       └── somlabs-demo.bb – plik z implementacją receptury.
```

Dodatkowo na diagramie widać, że każda z receptur (*bb*) lub jej rozszerzenie (*bbappend*) może używać dodatkowych plików, zlokalizowanych w katalogu o nazwie odpowiadającej recepturze. Mogą to być łaty do kodu, pliki graficzne, skrypty itp.

Receptura linux-stm32mp

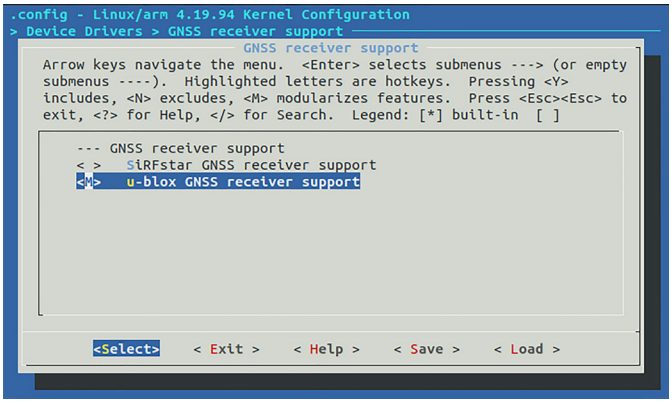
W dystrybucji OpenSTLinux używana jest wersja jądra, przygotowana przez firmę ST, dla procesorów z rodziny *STM32MP1*. W chwili pisania artykułu obowiązującą wersją jądra była 4.19, budowana przez recepturę *linux-stm32mp_4.19*. Bazowy plik receptury (*linux-stm32mp_4.19.bb*) znajduje się w warstwie *meta-st-stm32mp*. Jest w nim informacja na temat źródeł kodu, konfiguracji, budowania oraz instalacji jądra systemu. Warstwa *meta-somlabs* wprowadza do oryginalnej receptury modyfikacje w postaci łat, nakładanych na kod źródłowy jądra, przed uruchomieniem procesu kompilacji. Łaty te dostarczają wsparcia dla modułu *VisionSOM-STM32MP1*, płytki bazowej *VisionCB-STM32MP-1-STD* oraz wyświetlacza *Powertip* z interfejsem RGB lub DSI. Są one zlokalizowane w katalogu *meta-somlabs/recipes-kernel/linux/linux-st-m32mp* jako pliki z rozszerzeniem *patch*. Plik *linux-stm32mp_4.19.bbappend* w katalogu *meta-somlabs/recipes-kernel/linux/*, definiuje kolejność nakładania wspomnianych łat, katalog, w którym będą one poszukiwane oraz wskazuje na plik z konfiguracją jądra, użytą podczas budowania.

Modyfikacja receptur

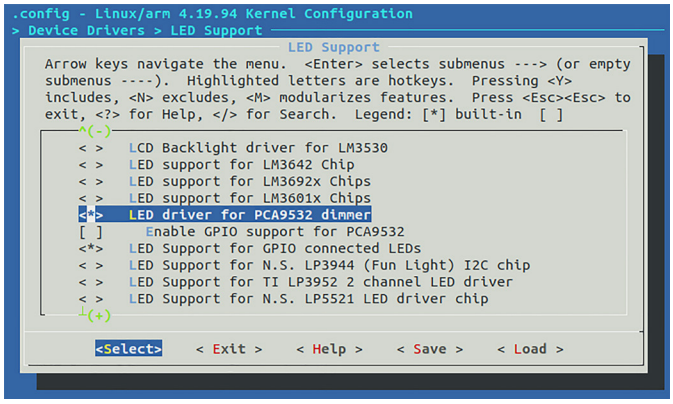
Modyfikacje receptur można wprowadzać ręcznie, zmieniając pliki *bb* i *bbappend* oraz dodając własne łaty. Jednak w naszym przykładzie skorzystamy z narzędzia *devtool*, które znacznie ułatwi pracę. Zacznijmy od konfiguracji terminalu, tak jak w poprzedniej części, przed rozpoczęciem budowania systemu:

```
cd <working directory path>/Distribution-Package/
openstlinux-4.19-thud-mp1-20-02-19
DISTRO=openstlinux-weston MACHINE=stm32mp157a-
visionsom-mx source layers/meta-st/scripts/envsetup.sh
```

Mamy teraz dostępne wszystkie narzędzia, w tym potrzebny *devtool*. W pierwszej kolejności trzeba pobrać kod źródłowy jądra, do którego będziemy wprowadzać modyfikacje:



Rysunek 1. Włączenie sterownika u-blox jako modułu jądra



Rysunek 3. Włączenie sterownika PCA9532 do kompilacji jądra

```
devtool modify linux-stm32mp
```

Polecenie to pobiera kod z repozytorium oraz rozpakowuje do katalogu *workspace/sources/linux-stm32mp*. Możemy teraz przejść do źródeł i sprawdzić, jakie łaty zostały na nie nałożone:

```
cd workspace/sources/linux-stm32mp
git log
```

Zobaczymy, że w logu na samej górze znajdują się łaty, pochodzące z warstwy *meta-somlabs*. Oznacza to, że polecenie `devtool modify` nie tylko pobiera kod ze zdefiniowanych źródeł, ale nanosi także wszelkie zmiany, które zdefiniowaliśmy w kolejnych warstwach – w tym wypadku *meta-somlabs*. Możemy więc przystąpić do wprowadzania własnych zmian. Naszym zadaniem jest włączenie wspomnianych na wstępie sterowników, więc zaczynamy od skopiowania z receptury i utworzenia domyślnej konfiguracji jądra:

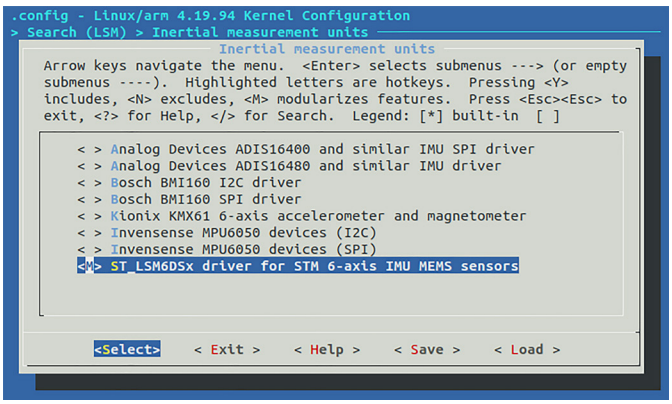
```
cp  ../../../../layers/meta-st/meta-somlabs/recipes-kernel/linux/linux-stm32mp/stm32mp157a_visionsom_mx_defconfig arch/arm/configs/
ARCH=arm make stm32mp157a_visionsom_mx_defconfig
```

W wyniku wywołania tego polecenia zostanie utworzony plik *.config*, na bazie pliku konfiguracyjnego *stm32mp157a_visionsom_mx_defconfig*. Do jego edycji możemy użyć klasycznego narzędzia *menuconfig*:

```
ARCH=arm make menuconfig
```

Jest to graficzne menu umożliwiające włączanie lub wyłączanie poszczególnych sterowników i opcji jądra. Włączmy więc sterowniki *U-blox GNSS* (Device Drivers > GNSS receiver support > u-blox GNSS receiver support), *LSM6DS3* (Device Drivers > Industrial I/O support > Inertial measurement units > ST_LSM6DSx driver for STM 6-axis IMU MEMS sensors), *PCA9532* (Device Drivers > LED Support > LED driver for PCA9532 dimmer), tak jak zostało to pokazane na **rysunkach 1**, **2**, **3**. W przykładzie pierwsze dwa sterowniki zostały włączone jako moduły (klawiszem M), natomiast ostatni został skompilowany z jądrem (klawisz Y).

Plik *.config* jest tworzony dynamicznie przy każdej konfiguracji, dlatego nie możemy go zapisać bezpośrednio w recepturze. Zamiast



Rysunek 2. Włączenie sterownika LSM6DSx jako modułu jądra

tego użyjemy go do dodania nowego pliku konfiguracyjnego w źródłach jądra:

```
cp .config arch/arm/configs/stm32mp157a_visionsom_mx_defconfig
```

Możemy teraz dodać plik do kontroli wersji i sprawdzić, jakie zmiany zostały wprowadzone w kodzie za pomocą poleceń:

```
git add arch/arm/configs/stm32mp157a_visionsom_mx_defconfig
git status
git diff
```

Będziemy wiedzieć, które pliki zostały zmodyfikowane oraz jakie zmiany w nich zaszły. Na ich podstawie możemy utworzyć nowy *commit*, z którego narzędzie *devtool* wygeneruje odpowiednią łatę w recepturze:

```
git commit
devtool update-recipe -a ../../../../layers/meta-st/meta-somlabs/ linux-stm32mp
```

Ostatnie z poleceń wymaga podania ścieżki do modyfikowanej warstwy oraz receptury.

Po wykonaniu wszystkich powyższych poleceń, w katalogu *meta-somlabs/recipes-kernel/linux/linux-stm32mp/* zobaczymy nową łatę o nazwie zgodnej z nazwą *commitu*, którą nadaliśmy przy okazji wywołania `git commit`. Ponadto została ona dodana na końcu listy, znajdującej się w pliku *meta-somlabs/recipes-kernel/linux/linux-stm32mp_4.19.bbappend*:

```
SRC_URI += "file://stm32mp157a_visionsom_mx_defconfig \
            file://0001-Add-support-for-VisionSOM-STM32MP1-board.patch \
            file://0001-Uptated-ILI9881C-driver-for-Powertip-PH720128T003.patch \
            file://0001-Fixed-display-transition-between-u-boot-and-kernel.patch \
            file://0001-Added-custom-defconfig.patch \
            "
```

Po zakończeniu wprowadzania zmian powinniśmy wywołać polecenie usuwające kopię roboczą źródeł z procesu budowania systemu:

```
devtool reset linux-stm32mp
```

Zgodnie z informacją, która pojawi się w terminalu, możemy ten katalog usunąć ręcznie.

W tym momencie plik konfiguracji jądra znajdzie się w źródłach, po nałożeniu patcha. Możemy zatem usunąć dotychczasowy plik *defconfig*:

```
rm ../layers/meta-st/meta-somlabs/recipes-kernel/linux/linux-stm32mp/stm32mp157a_visionsom_mx_defconfig
```

Musimy również zmodyfikować recepturę linux-stm32mp_4.19.bbappend tak, aby plik konfiguracyjny nie był poszukiwany w plikach receptury, ale w źródłach jądra:

```
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"

SRC_URI += "file://0001-Add-support-for-VisionSOM-
STM32MP1-board.patch \
            file://0001-Uptated-ILI9881C-driver-for-
            Powertip-PH720128T003.patch \
            file://0001-Fixed-display-transition-
            between-u-boot-and-kernel.patch \
            file://0001-Added-custom-defconfig.patch
            \
            "

KERNEL_DEFCONFIG =
"stm32mp157a_visionsom_mx_defconfig"

KERNEL_EXTERNAL_DEFCONFIG = ""
```

Pozostało już tylko przebudowanie systemu i wygenerowanie nowego obrazu poleceniem:

```
bitbake st-image-weston
```

W przykładzie zajęliśmy się modyfikacją konfiguracji jądra poprzez dodanie do kompilacji dodatkowych sterowników. Opisana metoda może być użyta do nanoszenia zmian, w dowolnym miejscu kodu źródłowego jądra oraz innych komponentów systemu, budowanych podczas przygotowywania systemu *OpenSTLinux* i innych dystrybucji projektu *Yocto*.

Krzysztof Chojnowski

Przypisy:
[1] http://bit.ly/2KrGBsQ