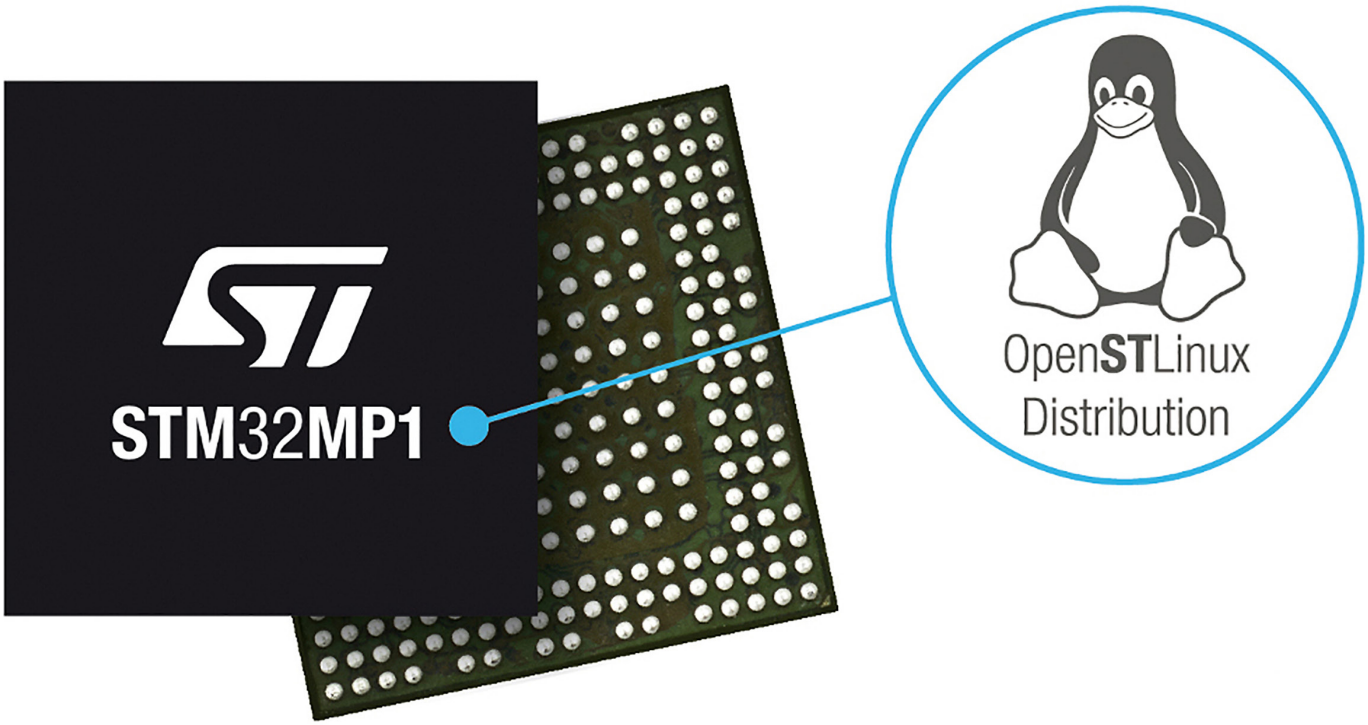


OpenSTLinux

dla procesorów z rodziny STM32MP1 (1)

Artykuł otwiera serię dotyczącą pracy z procesorami STM32MP1 i przeznaczonym dla nich systemem operacyjnym OpenSTLinux. Opiszę między innymi przygotowanie oraz modyfikację systemu, konfigurację sprzętu oraz tworzenie aplikacji z graficznym interfejsem użytkownika. Wszystkie przykłady będą uruchamiane na modułach VisionSOM-STM32MP1 z płytką bazową VisionCB-STM32MP1-STD i dołączonym wyświetlaczem SL-TFT-7-TP-800-480-P. Tematem pierwszej części cyklu będzie przygotowanie i uruchomienie systemu OpenSTLinux oraz SDK do kompilacji przykładowego programu w C.

Więcej informacji:
Opisane w artykule procedury budowania obrazu systemu i SDK są czasochłonne. Mogą wymagać nawet kilkudziesięciu godzin – zależnie od dostępnych zasobów sprzętowych. Z tego względu, na stronie ftp.somlabs.com dostępne są skompilowane wersje, gotowe do instalacji:
- OpenSTLinux: https://bit.ly/2Jl92Zd,
- SDK: https://bit.ly/3o3tj4a.

Informacje dotyczące instalacji znajdują się w artykule.

Budowanie systemu OpenSTLinux

Dystrybucja systemu OpenSTLinux została udostępniona przez firmę STMicroelectronics dla procesorów z rodziny STM32MP1. System ten, zbudowany na bazie projektu Yocto, dostępny jest w trzech wariantach:
- Starter package – obraz systemu do uruchomienia na wybranych zestawach deweloperskich;
- Developer package – obraz systemu zawierający SDK (*Software Development Kit*) do kompilacji własnego oprogramowania;
- Distribution package – zestaw receptur umożliwiający zbudowanie własnej dystrybucji.

W naszym przykładzie zastosujemy ostatni z wariantów, czyli zestaw receptur, które można zmodyfikować na potrzeby modułu VisionSOM-STM32MP1. Wszystkie przykłady będziemy wykonywali na systemie Linux Ubuntu 18.04.4 LTS. W chwili pisania artykułu OpenSTLinux był dostępny w wersji openstlinux-20-02-19. Informacje o najnowszych zmianach można znaleźć na stronie Wiki: https://bit.ly/2JhMbxE.

Pierwszym krokiem jest utworzenie katalogu roboczego i pobranie niezbędnego oprogramowania z repozytoriów ST oraz projektu *openembedded*:

```
cd <working directory path>/Distribution-Package
mkdir openstlinux-4.19-thud-mp1-20-02-19
cd openstlinux-4.19-thud-mp1-20-02-19
repo init -u https://github.com/STMicroelectronics/oe-manifest.git -b refs/tags/openstlinux-20-02-19
repo sync
```

Powyższe instrukcje pobierają receptury potrzebne do przygotowania podstawowej wersji systemu OpenSTLinux, ze wsparciem dla wybranych zestawów ewaluacyjnych.

W celu uruchomienia systemu na urządzeniu VisionSOM-ST-M32MP1 musimy pobrać receptury z repozytorium SoMLabs, zawierające konfigurację sprzętu dla tego modułu oraz płytki bazowej VisionCB-STM32MP1-STD:

```
cd layers/meta-st
```

```
git clone https://github.com/SoMLabs/openst-meta-somlabs.git meta-somlabs -b thud
```

Po ściągnięciu repozytorium openst-meta-somlabs mamy już wszystkie komponenty niezbędne do zbudowania obrazu systemu. Przechodzimy więc z powrotem do katalogu *openstlinux-4.19-thud-mp1-20-02-19* i konfigurujemy środowisko:

```
cd ../../
DISTRO=openstlinux-weston MACHINE=stm32mp157a-visionsom-mx source layers/meta-st/scripts/envsetup.sh
```

Drugie z powyższych poleceń konfiguruje aktywny terminal. Natomiast proces budowania obrazu systemu rozpoczynamy, wywołując:

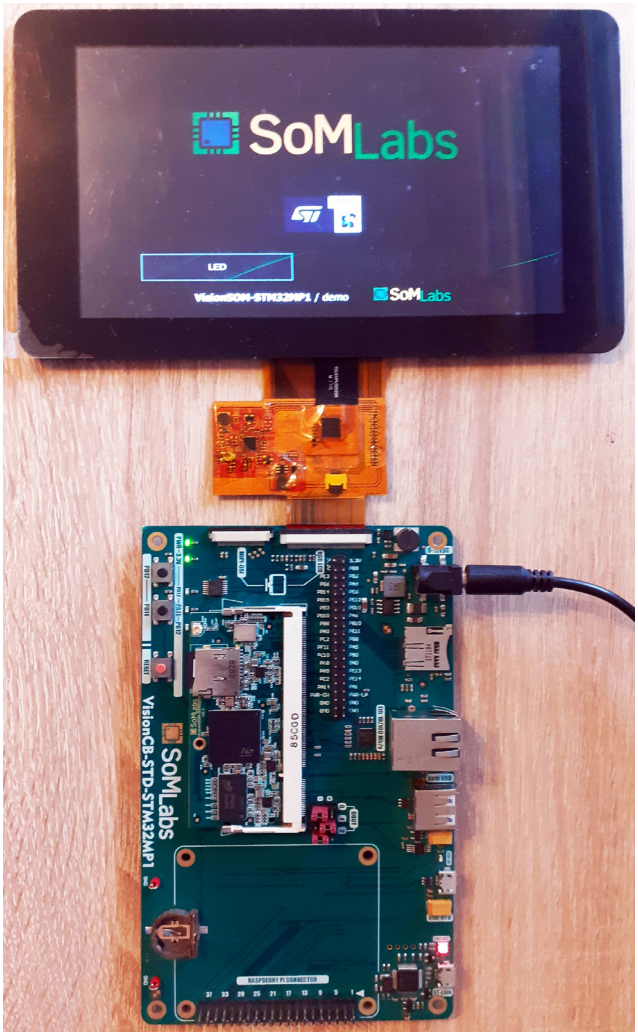
```
bitbake st-image-weston
```

Cała operacja może potrwać kilka godzin – zależnie od dostępnych zasobów sprzętowych. Po jej zakończeniu musimy jeszcze przygotować obraz karty SD, składający się z odpowiedniej konfiguracji partycji, używając przeznaczonego do tego skryptu:

```
cd tmp-glibc/deploy/images/stm32mp157a-visionsom-mx/scripts/
./create_sdcard_from_flashlayout.sh ../flashlayout_st-image-weston/FlashLayout_sdcard_stm32mp157a-visionsom-mx-basic.tsv
```

Gotowy obraz karty SD znajduje się w pliku *flashlayout_st-image-weston_FlashLayout_sdcard_stm32mp157a-visionsom-mx-basic.raw* w katalogu *tmp-glibc/deploy/images/stm32mp157a-visionsom-mx*. Możemy go skopiować na kartę poleceniem:

```
sudo dd if=flashlayout_st-image-weston_FlashLayout_sdcard_stm32mp157a-visionsom-mx-basic.raw of=/dev/sdX bs=1M
```



Fotografia 1. Uruchomiony system z aplikacją demo na VisionSOM-STM32MP1

Ścieżkę */dev/sdX* należy zmienić zgodnie z nazwą, jaką ma zamontowana w naszym systemie karta SD.

Po zainstalowaniu obrazu na karcie możemy uruchomić przygotowany system na module VisionSOM-STM32MP1, podłączonym do płytki bazowej VisionCB-STM32MP1-STD. Jeżeli podłączony jest również wyświetlacz SL-TFT7-TP-800-480-P, to po uruchomieniu systemu zobaczymy aplikację demo, odtwarzającą plik wideo oraz zawierającą przycisk do włączania diody podłączonej do pinu PA11, a także suwak umożliwiający zmianę jej jasności (**fotografia 1**).

Budowanie SDK

Pakiet SDK (*Software Development Kit*) jest niezbędny do kompilacji na systemie hosta programów, przeznaczonych dla systemu OpenSTLinux. Zawiera on kompilator wraz z niezbędnymi bibliotekami i nagłówkami systemu, na którym będziemy uruchamiali aplikacje (tzw. *sysroot*). Podobnie jak sam obraz, SDK możemy zbudować po wcześniejszym skonfigurowaniu terminalu:

```
DISTRO=openstlinux-weston MACHINE=stm32mp157a-visionsom-mx source layers/meta-st/scripts/envsetup.sh
```

i wywołaniu polecenia:

```
bitbake st-image-weston -c populate_sdk
```

Po zakończeniu procesu budowania będziemy mieli dostęp do instalatora, dzięki czemu możemy używać SDK na wielu komputerach, na których chcemy przeprowadzać kompilację naszych aplikacji. Instalator *st-image-weston-openstlinux-weston-stm32mp157a-visionsom-mx-x86_64-toolchain-2.6-snapshot.sh* znajduje się w katalogu *tmp-glibc/deploy/sdk*. Po zainstalowaniu go w swoim systemie musimy odpowiednio skonfigurować terminal, w którym będziemy przeprowadzali kompilację:

```
./opt/st/stm32mp157a-visionsom-mx/2.6-snapshot/environment-setup-cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

Konfigurację możemy zweryfikować, sprawdzając wartość zmiennej środowiskowej CC, zawierającej wywołanie kompilatora C:

```
echo $CC
```

W terminalu powinniśmy zobaczyć następujący wynik:

```
arm-ostl-linux-gnueabi-gcc -march=armv7ve -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 --sysroot=/opt/st/stm32mp157a-visionsom-mx/2.6-snapshot/sysroots/cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi
```

Mając gotowe narzędzia, możemy zbudować przykładową aplikację, która będzie zmieniała stan diody na płytce VisionCB-STD-STM32MP1.

Sterowanie diodą LED

Nasza pierwsza aplikacja będzie okresowo zmieniała stan jednej z diod, znajdujących się na płytce bazowej. W domyślnym obrazie systemu skonfigurowane są trzy diody, przypisane do następujących pinów:

- PA12 – wyjście GPIO używane przez systemową funkcję „heartbeat”;
- PA11 – wyjście licznika generującego sygnał PWM, wykorzystywane w dostępnej w systemie aplikacji demonstracyjnej;
- PG12 – wyjście GPIO;

W naszym przykładzie użyjemy ostatniego z pinów, który jest w systemie przypisany do diody „led3”. Możemy nią sterować za pomocą pliku */sys/class/leds/led3/brightness*. Dioda ta nie jest sterowana sygnałem PWM, więc wpisanie dowolnej wartości różnej od zera spowoduje ustawienie pinu PG12 w stan wysoki, natomiast wpisanie zera – w stan niski. Możemy się o tym przekonać, podłączając się do systemu za pośrednictwem portu szeregowego (port micro-USB ST-LINK) oraz np. programu *screen*: `sudo screen /dev/ttyACM0 115200` (wirtualny port szeregowy programatora ST-LINK jest w systemach Linux rozpoznawany jako urządzenie /dev/ttyACMX) i wywołując polecenia:

Listing 1. Kod przykładowego programu

```
#include <fcntl.h>
#include <unistd.h>
#define LED_PATH "/sys/class/leds/led3/brightness"
#define LED_BLINKS 2
int main(void) {
    int fd;
    fd = open(LED_PATH, O_WRONLY);
    for(int i=0; i<LED_BLINKS; i++) {
        write(fd, "1", 1);
        sleep(1);
        write(fd, "0", 1);
        sleep(1);
    }
    close(fd);
    return 0;
}
```

```
echo 1 > /sys/class/leds/led3/brightness
echo 0 > /sys/class/leds/led3/brightness
```

Spowodują one odpowiednio zapalenie i zgaszenie diody.

Przykładowa aplikacja w C

Na koniec napiszemy i skompilujemy program w C, który będzie w stanie modyfikować stan diody led3, podłączonej do pinu PG12. Uproszczony kod, niezawierający obsługi możliwych błędów, zwracanych przez funkcje *open*, *write* i *close*, znajduje się na **listingu 1**.

Program otwiera plik zdefiniowany jako *LED_PATH* (ścieżka do pliku *brightness*), wykonuje określoną liczbę zmian stanów diody, a na koniec zamyka plik. Możemy go skompilować w terminalu, w którym zostało skonfigurowane SDK:

```
$CC hello.c -o hello
```

W efekcie powinniśmy otrzymać plik wykonywalny *hello* dla architektury ARM, o czym możemy się przekonać, wywołując polecenie `file hello` i otrzymując w odpowiedzi:

```
hello: ELF 32-bit LSB executable, ARM, EABI5
version 1 (SYSV), dynamically linked, interpreter
/lib/ld-, for GNU/Linux 3.2.0, BuildID[sha1]=
7326911247aebb2361a8a2c94946453b02ac1402, with
debug_info, not stripped
```

Plik ten możemy skopiować bezpośrednio na kartę pamięci lub poleceniem `scp`, jeżeli wcześniej podłączyliśmy nasze urządzenie do sieci, za pomocą kabla Ethernet lub modemu Wi-Fi:

```
scp hello root@<Adres IP>:
```

Adres IP uzyskamy, wywołując polecenie `ip -a` w terminalu portu szeregowego. Plik zostanie skopiowany do katalogu domowego użytkownika root, czyli */home/root*. Możemy go uruchomić, przechodząc do wspomnianego katalogu i wywołując polecenie `./hello`.

Krzysztof Chojnowski