Implementacja funkcji Screen Mirroring

na platformach SoMLabs VisionSOM-6ULL oraz Raspberry Pi

W ostatnich miesiącach miałem przyjemność wziąć udział w nietechnicznym szkoleniu, gdzie prowadzący zdecydował się użyć Raspberry Pi jako niewielkiego i energooszczędnego zamiennika typowego komputera PC. Podczas prezentacji pojawiła się potrzeba przeniesienia na rzutnik informacji z telefonu prowadzącego – czy problem ten można rozwiązać w sposób czysto programowy, bez użycia dodatkowych konwerterów sprzętowych? Spróbujemy znaleźć odpowiedź na to pytanie.

Kiedy 8 lat temu zaczęły pojawiać się pierwsze informacje prasowe, dotyczące komputera jednopłytkowego Raspberry Pi, został on okrzyknięty pogromcą typowych komputerów PC w zastosowaniach biurowych i edukacyjnych. Pomimo, że szumnie ogłaszana rewolucja nie nastąpiła, to komputer Raspberry Pi mocno wpisał się w najnowszą historię informatyki i jest obecnie coraz częściej spotykany w standardowym użyciu, nawet przez osoby niezwiązane zawodowo z systemami wbudowanymi.

W artykule omówię dwa proste sposoby realizacji funkcjonalności *Screen Mirroring*, czyli przesłania i wyświetlenia zawartości ekranu urządzenia mobilnego na zewnętrzny ekran stacjonarny. Do realizacji zadania użyję telefonu z systemem Android oraz dwóch komputerów jednopłytkowych – wydajny pod kątem aplikacji multimedialnych *Raspberry Pi 4*, oraz przystosowany głównie do zadań kontrolnopomiarowych, zestaw *SoMLabs VisionSOM6ULL* [1], z energooszczędnym procesorem *NXP i.MX6ULL*. Wybór dwóch różnych platform sprzętowych pozwoli na dobór różnych rozwiązań programowych, dopasowanych pod kątem możliwości oferowanych przez sprzęt

– od prostych wywołań z pakietu funkcji dostarczanych przez projekt *GStreamer* [2], aż do kompilacji otwartoźródłowego i funkcjonalnego projektu *scrcpy* [3].

VisionSOM6ULL i GStreamer

Proces implementacji funkcji realizującej *Screen Mirroring* rozpoczniemy od platformy *VisionSOM6ULL*, z przeznaczonym dla niej wyświetlaczem LCD oraz modułem wyposażonym w gniazdo karty pamięci SD. Pomimo, że procesory i.MX6ULL nie mają wyspecjalizowanych układów sprzętowych do dekodowania strumieni wideo i wsparcia dla grafiki 2D/3D (co przekłada się na znacznie niższy koszt procesora w porównania do lepiej wyposażonych układów z rodziny i.MX), nie uniemożliwia to jednocześnie implementacji wydajnej obsługi przesyłania i dekodowania obrazu – operacja ta wymaga jednak od użytkownika systemu nieco więcej zabiegów programowych w procesie przygotowania systemu.

Prace nad projektem rozpoczynamy od przygotowania karty z systemem operacyjnym Linux. Producent komputera – firma SoMLabs – na stronach Wiki swojego produktu dostarcza użytkownikowi wsparcia w postaci gotowych obrazów z dystrybucją *Debian* oraz opisem budowy obrazów, z użyciem takich narzędzi jak *Buildroot* oraz *Yocto*. Zastosuję gotowe obrazy systemu, z dystrybucją *Debian* w wersji *Stretch* (na stronie producenta udostępniono również nowszą wersję dystrybucji – *Debian Buster* – z wbudowaną obsługą przeznaczonego wyświetlacza LCD).

Przygotowanie karty SD rozpoczynamy od pobrania obrazu systemu. W środowisku Linux operację tę można zrealizować poleceniem:

```
wget http://ftp.somlabs.com/debian-stretch-visionsom-6ull.img.xz
```

Po pobraniu pliku rozpakujmy jego zawartość za pomocą narzędzia *unxz*:

```
unxz debian-stretch-visionsom-6ull.img.xz
```

Po rozpakowaniu pliku możemy przystąpić do wgrania obrazu systemu na kartę microSD. Jedną najprostszych metod zapisania obrazu z pliku jest wykorzystanie linuksowego narzędzia *dd*.

```
sudo dd if=./debian-stretch-visionsom-6ull.img of=/dev/sdX bs=4M oflag=dsync
```

Po wgraniu obrazu na kartę do konsoli systemu możemy się zalogować, wykorzystując wbudowany w płytę bazową *VisionCBSTD* konwerter UARTUSB oraz dowolny program emulatora terminalu:

```
picocom -b 115200 /dev/ttyUSBX
```

Po otwarciu połączenia należy zalogować się na konto użytkownika root (bez hasła):

```
Debian GNU/Linux 9 localhost.localdomain ttymxc0
localhost login: root
root@localhost:~#
```

Wraz z obrazem systemu Debian Stretch firma SoMLabs dostarcza użytkownikowi wygodne środowisko konfiguracji i kompilacji jądra systemu oraz plików *Device Tree*. Środowisko to zostało zbudowane w oparciu na narzędziach *Qemu* oraz *chroot*, co pozwala na uruchomienie na komputerze PC (np. x86) nienatywnych plików wykonywalnych (np. dla architektury ARM) w standardowy dla plików natywnych sposób, tj. `./program`. Do poprawnego działania środowiska w dystrybucji Ubuntu niezbędna jest uprzednia instalacja pakietów *qemu*, *binfmt-support* oraz *qemu-user-static*:

apt-get install qemu binfmt-support qemu-user-static

Pobranie, rozpakowanie i uruchomienie (w tym wykonanie operacji *chroot*, zmieniającej katalog główny) kompletnego środowiska:

```
wget http://ftp.somlabs.com/somlabs-visionsom-6ull-debian-rootfs-qemu.tar.xz
sudo tar xf somlabs-visionsom-6ull-debian-rootfs-qemu.tar.xz
sudo ./somlabs-debian/chtoolchain
```

Po uruchomieniu skryptu *chtoolchain* źródła systemu Linux są dostępne w katalogu */home/developer/source/kernel/linux-rel_imx_4.1.15_2.1.0_ga*. Dla uproszczenia dalszego opisu przejdźmy do katalogu z kodem źródłowym:

```
cd /home/developer/source/kernel/linux-rel_imx_4.1.15_2.1.0_ga
```

W kolejnym kroku, do katalogu *arch/arm/configs* należy skopiować domyślną konfigurację jądra (plik *visionsom-6ull-linux_defconfig*), udostępnioną przez producenta płytki:

```
cp ../../somlabs-dts-1.0/visionsom-6ull-linux_defconfig arch/arm/configs/
```

W odróżnieniu od najnowszej wersji obrazu systemu Debian (*Buster*) dla płytki VisionSOM, opis Device Tree (w postaci pliku *arch/arm/boot/dts/somlabs-visionsom-6ull.dts*) w obrazie Debian Stretch nie zawierał pełnego wsparcia dla przeznaczonego dla niego ekranu LCD. Aby poprawnie uruchomić wyświetlacz, niezbędne jest zatem pobranie i zaaplikowanie dodatkowej łatki:

```
wget http://wiki.somlabs.com/images/c/cd/Enable-tft-lcd.zip
unzip Enable-tft-lcd.zip
patch arch/arm/boot/dts/somlabs-visionsom-6ull.dts ./enable-tft-lcd.patch
```

W tak przygotowanym środowisku konfiguracja i kompilacja jądra oraz opisu Device Tree może zostać zrealizowana za pomocą następujących poleceń:

```
make ARCH=arm visionsom-6ull-linux_defconfig
make ARCH=arm menuconfig
make -j4 ARCH=arm zImage
make ARCH=arm somlabs-visionsom-6ull.dtb
```

Pliki wynikowe powyższych kompilacji:

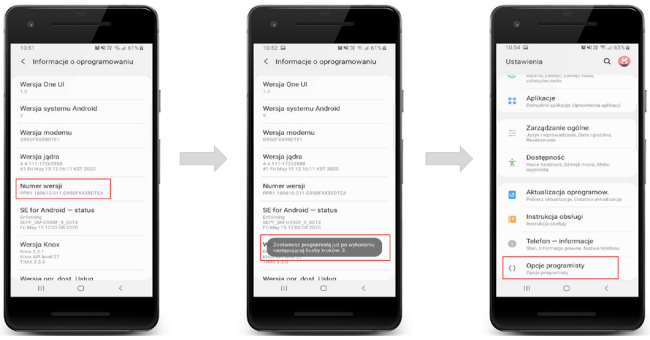
```
arch/arm/boot/zImage
arch/arm/boot/dts/somlabs-visionsom-6ull.dtb
```

powinny zostać skopiowane do katalogu */boot* na karcie SD z obrazem systemu. Po podłączeniu wyświetlacza oraz kabla Ethernet (połączenie z siecią Internet będzie niezbędne do instalacji i konfiguracji kolejnych pakietów oprogramowania – operacje te zostaną przeprowadzone bezpośrednio na systemie docelowym) płytka jest gotowa do dalszych działań.

Zanim przejdziemy do działań związanych bezpośrednio z dekodowaniem i wyświetlaniem obrazu z urządzenia mobilnego, należy rozwiązać problem „przechwycenia" takiego obrazu. Pierwszą z opcji jest wykorzystanie gotowych rozwiązań, np. w postaci standardu *Miracast,* wspieranego przez największych producentów urządzeń mobilnych. Niestety implementacja takich rozwiązań może być niezwykle czasochłonna i/lub wymagać odpowiednich certyfikatów. Alternatywnym rozwiązaniem jest przygotowanie własnej aplikacji dla systemu Android, która wykorzystując udostępnione dla programistów *MediaProjection* API [4] – umożliwiające przechwytywanie zawartości ekranu – odpowiednio skompresuje i wyśle dane do komputera jednopłytkowego. Takie rozwiązanie, choć zapewniające pełną swobodę w wyborze kompresji obrazu czy protokołu komunikacji, wymaga od programisty pewnego doświadczenia z systemem Android oraz językami Java lub Kotlin. Czy można ten problem rozwiązać znacznie prościej? Można, wykorzystując do tego celu narzędzie *ADB* (*Android Debug Bridge*) [5], przeznaczone do komunikacji i zarządzania urządzeniami Android z poziomu komputera PC. Program ADB został udostępniony dla wszystkich najpopularniejszych systemów operacyjnych (Windows, Linux, macOS), jednak ze względu na swój konsolowy interfejs nie znajduje on zastosowania w powszechnym użytku. Oprócz szeregu funkcji związanych z kopiowaniem plików, instalacją pakietów *APK* czy odczytem logów systemowych, program ADB udostępnia również możliwość wykonania zrzutów oraz przechwycenia zawartości ekranu urządzenia mobilnego – i właśnie ta opcja zostanie wykorzystana do realizacji postawionego zadania.

Instalacja pakietu ADB na module VisionSOM6ULL z systemem Debian sprowadza się wyłącznie do jednego wywołania narzędzia *apt*:

```
sudo apt install adb
```



Rysunek 1. Odblokowanie „Opcji programisty” na urządzeniu mobilnym

Aby nawiązać komunikację z telefonem Android, niezbędne jest uprzednie odblokowanie na urządzeniu „Opcji programisty” oraz włączenie „Debugowania USB”. Włączenie opcji programisty realizowane jest poprzez siedmiokrotne kliknięcie opcji „Numer wersji” (lub „Numer kompilacji”) w ustawieniach telefonu – **rysunek 1.**

Po odblokowaniu opcji programisty, włączenia trybu debugowania oraz połączeniu telefonu z modułem VisionSOM6ULL za pomocą kabla USB, możemy podjąć próbę sprawdzenia statusu połączenia pomiędzy urządzeniami. W tym celu, w konsoli użytkownika wpisujemy polecenie `adb devices`, które wyświetli identyfikatory wszystkich urządzeń z systemem Android podłączonych do komputera.

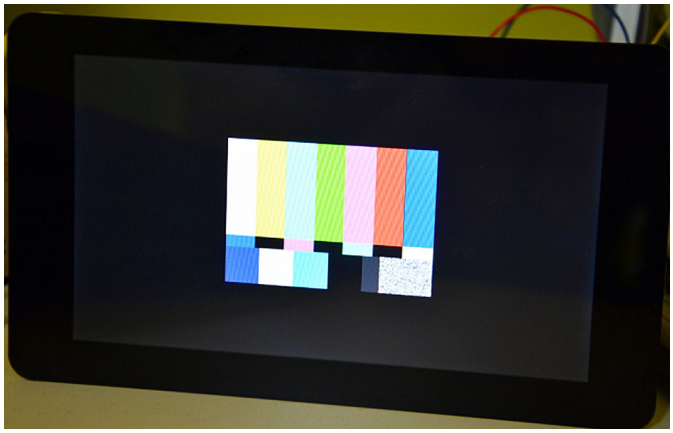
```
root@somlabs:~# adb devices
List of devices attached
ce091719631fe12f01      device
```

W przypadku wystąpienia problemów z nawiązaniem połączenia warto sprawdzić komunikaty wyświetlane na ekranie telefonu, które mogą wymagać potwierdzenia podpisu cyfrowego urządzenia lub udzielenia stosownych zezwoleń.

Do przechwycenia zawartości ekranu wykorzystamy polecenie `adb shell screenrecord` w postaci:

```
adb shell screenrecord --output-format=h264 --bit-rate 2M --size 800x480 -
```

Tym samym strumień wyjściowy zostanie poddany kompresji w formacie H264, przepływność danych jest ustawiona na wartość 2 Mbit/s (domyślna wartość 20 Mbit/s znacznie przekracza możliwości płynnej dekompresji obrazu), a rozdzielczość obrazu dopasowana



Fotografia 1. Wyświetlenie wzoru testowego z wykorzystaniem funkcji Gstreamer

do podłączonego wyświetlacza. Ostatnim z argumentów polecenia jest nazwa pliku wyjściowego, do którego na bieżąco będzie zapisywany przechwytywany obraz. Ostatni argument ma wartość „-”, co oznacza, że strumień wyjściowy będzie bezpośrednio zapisywany na standardowe wyjście. Próba uruchomienia tak zdefiniowanego polecenia zakończy się wyłącznie wyświetleniem szeregu nieczytelnych znaków – uzyskane dane muszą zostać przekazane do odpowiedniego dekodera.

Do zdekodowania strumienia wyjściowego wykorzystany zostanie pakiet *GStreamer*, którego kompleksowa instalacja w systemie może zostać zrealizowana za pomocą poleceń:

```
root@somlabs:~# apt-get install gstreamer1.0-x
gstreamer1.0-tools
root@somlabs:~# apt-get install
gstreamer1.0-plugins-good
root@somlabs:~# apt-get install
gstreamer1.0-plugins-bad
root@somlabs:~# apt-get install gstreamer1.0-libav
```

Poprawność instalacji pakietów weryfikujemy komendą:

```
root@somlabs:~# gst-launch-1.0 videotestsrc !
fbdevsink
```

Listing 1. Przebieg procesu konfiguracji i kompilacji projektu *gstreamer-imx*

```
root@somlabs:~# cd gstreamer-imx

root@somlabs:~/gstreamer-imx# ln -s /usr/lib/arm-linux-gnueabihf/gstreamer-1.0/ /usr/lib/gstreamer-1.0

root@somlabs:~/gstreamer-imx# ./waf configure --prefix=/usr --kernel-headers=/usr/src/linux-headers-4.1.15/include
Setting top to : /root/gstreamer-imx
Setting out to : /root/gstreamer-imx/build
Checking for 'gcc' (C compiler) : /usr/bin/gcc
Checking for compiler switch -O2 : yes
...
checking for linux/pxp_device.h : yes
PxP elements will be built
checking for linux/fb.h and the IPU header linux/ipu.h : yes
IPU elements will be built
Checking for 'libimxvpuapi >= 0.10.3' : not found
could not find installed imxvpuapi library - VPU elements will not be built
...
'configure' finished successfully (25.634s)

root@somlabs:~/gstreamer-imx# ./waf
Waf: Entering directory `/root/gstreamer-imx/build'
[ 1/35] Compiling src/common/canvas.c
[ 2/35] Compiling src/common/fd_object.c
[ 3/35] Compiling src/common/phys_mem_allocator.c
...
[34/35] Symlinking build/src/compositor/libgstimxcompositor.so
[35/35] Symlinking build/src/common/libgstimxcommon.so
Waf: Leaving directory `/root/gstreamer-imx/build'
'build' finished successfully (1m45.921s)

root@somlabs:~/gstreamer-imx# ./waf install
Waf: Entering directory `/root/gstreamer-imx/build'
+ install /usr/lib/libgstimxcommon.so.0.13.0
+ symlink /usr/lib/libgstimxcommon.so (to libgstimxcommon.so.0.13.0)
+ symlink /usr/lib/libgstimxcommon.so.0 (to libgstimxcommon.so.0.13.0)
+ install /usr/lib/pkgconfig/gstimxcommon.pc
...
Waf: Leaving directory `/root/gstreamer-imx/build'
'install' finished successfully (0.584s)
```

Listing 2. Sprawdzenie wtyczki *imxpxpvideosink* projektu *GStreamer*

```
root@somlabs:~/gstreamer-imx# gst-inspect-1.0 imxpxpvideosink
Factory Details:
    Rank         primary + 1 (257)
    Long-name    Freescale PxP video sink
    Klass        Sink/Video
    Description  Video output using the Freescale PxP API
    Author       Carlos Rafael Giani <dv@pseudoterminal.org>
…

Element Properties:
    name         : The name of the object
                   flags: readable, writable
                   String. Default: "imxpxpvideosink0"
    parent       : The parent of the object
                   flags: readable, writable
                   Object of type "GstObject"
```

która wyświetli na dołączonym ekranie prosty wzór testowy – **fotografia 1**.

Aby wspomóc procesor i.MX6ULL w operacjach związanych z grafiką i multimediami (wyświetlanie ekranu testowego, jak na fotografii 1, realizowane jest w pełni programowo), niezbędne będzie zainstalowanie dodatkowych wtyczek dla pakietu *GStreamer*, dostarczanych i rozwijanych przez NXP. Zestaw wtyczek rozwijanych w ramach projektu *gstreamer-imx* [6] zawiera między innymi wsparcie dla sprzętowego modułu PXP (*Pixel Processing Unit*), wspomagającego operacje związane z przetwarzaniem obrazów 2D, konwersją przestrzeni kolorów, skalowaniem, itd. Zainstalujmy zatem w systemie pakiety i zależności niezbędne do poprawnej kompilacji projektu *gstreamer-imx*:

```
root@somlabs:~# apt-get install build-essential
autoconf libtool
root@somlabs:~# apt-get install wget python
pkg-config
root@somlabs:~# apt-get install libgstreamer1.0-dev
root@somlabs:~# apt-get install
libgstreamer-plugins-base1.0-dev
```

Kolejny krok stanowi pobranie źródeł projektu:

```
root@somlabs:~# git clone git://github.com/Freescale/
gstreamer-imx.git
Cloning into 'gstreamer-imx'...
remote: Counting objects: 4349, done.
remote: Total 4349 (delta 0), reused 0 (delta 0),
pack-reused 4349
Receiving objects: 100% (4349/4349), 1.91 MiB | 1.35
MiB/s, done.
Resolving deltas: 100% (3129/3129), done.
```

Do kompilacji *gstreamer-imx* niezbędne jest zainstalowanie w systemie plików nagłówkowych jądra Linux, które dla obrazu Debian Stretch są udostępnione w postaci pakietu DEB wraz z opisanym na wstępie artykułu plikiem *somlabs-visionsom-6ull-debian-rootfs--qemu.tar.xz*:

Fotografia 2. Wyświetlenie filmu testowego z wykorzystaniem wtyczki *imxpxpvideosink*

```
root@somlabs:~# dpkg -i linux-
headers-4.1.15_4.1.15-1_armhf.deb
```

Projekt *gstreamer-imx* wykorzystuje do procesu konfiguracji i kompilacji narzędzie *Waf* – pełny przebieg tego procesu pokazano na **listingu 1**. Po zakończonym procesie instalacji lista wtyczek dla projektu *GStreamer*, powinna zostać poszerzona m.in. o wtyczkę *imxpxpvideosink* (**listing 2**).

System jest już wyposażony w komplet narzędzi programowych. Poprawność działania wtyczek pakietu *gstreamer-imx* możemy w prosty sposób przetestować, wyświetlając na ekranie fragment dowolnego filmu testowego (**fotografia 2**):

```
wget http://craftymind.com/factory/html5video/
BigBuckBunny_640x360.mp4
gst-launch-1.0 filesrc
location=BigBuckBunny_640x360.mp4 ! decodebin !
imxpxpvideosink use-vsync=true
```

Aby zdekodować i wyświetlić na ekranie obraz generowany przez polecenie `adb shell screenrecord`, połączymy polecenia `adb` oraz `gst-launch` za pomocą operacji potoku „|”, a jako źródło strumienia wejściowego dla *GStreamer* wskażemy standardowe wejście (`fdsrc fd=0`):

```
adb shell screenrecord --output-format=h264 --bit-
rate 4M --size 800x480 - | gst-launch-1.0 fdsrc fd=0
! decodebin ! imxpxpvideosink
```

Efekt działania powyższego polecenia pokazuje **fotografia 3**.

Zaprezentowana metoda nie jest pozbawiona wad, a jakość przesyłanego obrazu musiała zostać obniżona ze względu na ograniczenia, wynikające z dostępnych zasobów sprzętowych. Niemniej jednak pozwala ona na szybkie przesłanie zawartości ekranu telefonu do komputera z system Linux. Nie wymaga instalacji żadnego dodatkowego oprogramowania po stronie urządzenia mobilnego, a całość operacji jest realizowana za pomocą jednego polecenia. W przypadku większości dystrybucji systemu Linux nie jest również wymagane instalowanie żadnego dodatkowego oprogramowania po stronie komputera. Wspierany sprzętowo pakiet *GStreamer* jest instalowany domyślnie w większości dystrybucji, a zaprezentowana metoda może zostać również z powodzeniem wykorzystana na standardowych komputerach klasy PC z procesorami x86/x64.

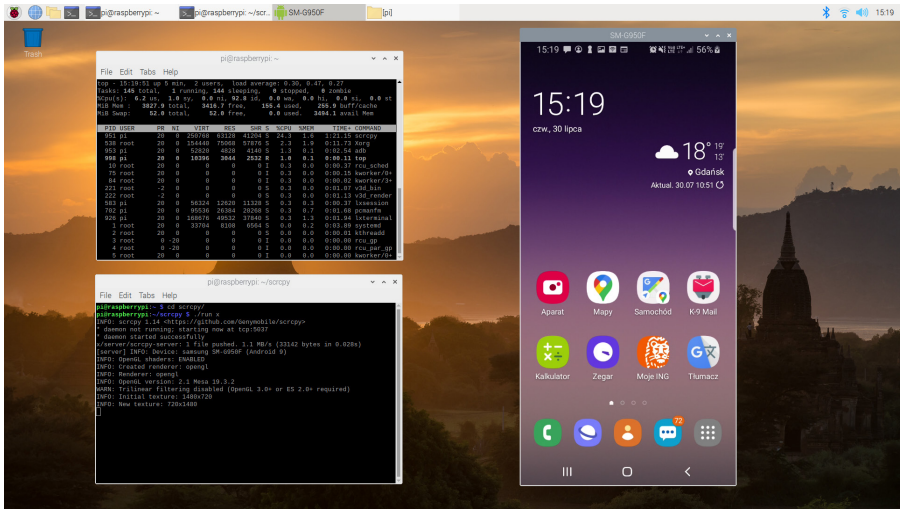
Raspberry Pi i projekt *scrcpy*

Świat miłośników oprogramowania otwartoźródłowego nie znosi pustki, więc kwestią czasu było pojawienie się alternatywnych rozwiązań dla komercyjnego projektu *Miracast* czy niskopoziomowych rozwiązań, jakim jest przedstawione wcześniej. Jednym

Fotografia 3. Funkcja *Screen Mirroring* zbudowana z wykorzystaniem pakietów *ADB* oraz *Gstreamer*

z takich otwartych projektów jest program *scrcpy*, który do realizacji zadania wykorzystuje narzędzie ADB (przesyłanie danych i zdarzeń) oraz natywną aplikację dla systemu Android do przechwytywania i kompresji obrazu. Projekt dość intensywnie wykorzystuje funkcje dostarczane poprzez interfejs ADB, oferując tym samym nie tylko proste przesyłanie obrazu, ale i obsługę wejścia (sterowanie telefonem z poziomu myszki i klawiatury na lustrzanej projekcji) czy instalację pakietów APK poprzez system *Drag&Drop*.

Aplikacja jest dostępna dla systemów Windows, Linux oraz macOS, jednak nie jest dostarczana w postaci gotowych pakietów dla platformy ARM – użytkownik musi samodzielnie wykonać kompilację kodu źródłowego. Ponieważ projekt *scrcpy* do dekodowania strumienia danych wykorzystuje biblioteki z projektu *ffmpeg* (*libav*), które na platformie i.MX6ULL nie mają sprzętowego wsparcia (całość obliczeń realizowania jest w sposób programowy i nie pozwala to uzyskać satysfakcjonujących wyników), do kompilacji i uruchomienia projektu zostanie wykorzystany komputer *Raspberry Pi 4*, którego moc obliczeniowa jest w zupełności wystarczająca do wygodnego korzystania z funkcji oferowanych przez *scrcpy*.



Rysunek 2. Funkcja *Screen Mirroring* realizowana przez projekt *scrcpy* na platformie Raspberry Pi

Po pobraniu i wgraniu na kartę SD obrazu systemu *Raspberry Pi OS* (wcześniej znanego pod nazwą *Raspbian*), proces przygotowania platformy rozpoczynamy od aktualizacji oprogramowania:

```
sudo apt-get update
sudo apt-get upgrade
```

Przypominam, że projekt *scrcpy* wykorzystuje funkcje oferowane przez narzędzie ADB, zatem kolejny krok stanowi zainstalowanie brakujących pakietów oraz zależności niezbędnych do kompilacji programu:

```
sudo apt install adb meson libavformat-dev
libsdl2-dev
```

Pobranie kodu źródłowego *scrcpy* za pomocą narzędzia *git*:

```
git clone https://github.com/Genymobile/scrcpy.git
```

Wraz z kodem źródłowym aplikacji dla systemów Windows i Linux udostępniany został kod aplikacji dla systemu Android, której zadaniem jest przechwycenie i przesłanie obrazu. Ponieważ aplikacja jest niezależna od architektury, na jaką wykonujemy kompilację programu klienta, deweloperzy umożliwili uproszczenie etapu konfiguracji i kompilacji projektu poprzez udostępnienie gotowych plików APK:

```
cd scrcpy
wget https://github.com/Genymobile/scrcpy/releases/
download/v1.14/scrcpy-server-v1.14
```

Aby wykorzystać gotowe pliki APK w procesie konfiguracji kodu, za pomocą argumentu `prebuilt_server` wskazujemy pobrany w uprzednim kroku plik dla systemu Android:

```
meson x --buildtype release --strip -Db_lto=true
-Dprebuilt_server=scrcpy-server-v1.14
```

Kompilacja źródeł umieszczonych w folderze „x”:

```
ninja -Cx
```

Zakończony proces kompilacji umożliwia nam bezpośrednie uruchomienie programu – bez potrzeby instalacji plików wynikowych:

```
./run x
```

Efekt uruchomienia aplikacji *scrcpy* pokazuje **rysunek 2**. Dodatkowo, program dostarcza również szeregu opcji konfiguracyjnych, związanych z ustawieniami rozdzielczości, przepływności, konfiguracją okna oraz umożliwia wygaszenie ekranu urządzenia mobilnego. Pełną listę opcji konfiguracyjnych uzyskamy po wydaniu polecenia:

```
./run x --help
```

Wykorzystując platformę *Raspberry Pi*, warto również przetestować alternatywne rozwiązania – będące modyfikacją metody wykorzystanej dla płytki *VisionSOM6ULL* – podmieniając pakiet *Gstreamer* na wspierany sprzętowo odtwarzacz multimedialny *omxplayer*:

```
adb shell screenrecord --output-format=h264 --bit-
rate 2M --size 800x480 - | omxplayer --no-keys -r
pipe:0 -o hdmi
```

Łukasz Skalski
contact@lukasz-skalski.com

Przypisy:
[1] https://bit.ly/3oD7OIG
[2] https://bit.ly/3jzbU0y
[3] https://bit.ly/2ToBquH
[4] https://bit.ly/3dYXCp1
[5] https://bit.ly/3ku9zFq
[6] https://bit.ly/3kyULFB