VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Zbigniew Opioł

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV RADIOELEKTRONIKY
DEPARTMENT OF RADIO ELECTRONICS

ROZHRANÍ PRO SENZORY PHPIX
PHPIX INTERFACE BOARD

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Zbigniew Opioł

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Michal Kubíček, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Elektronika a komunikační technologie**

Ústav radioelektroniky

Student: Bc. Zbigniew Opioł ***ID:*** 154824

Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Rozhraní pro senzory PHpix

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se se senzory ionizujícího záření PHpix a navrhněte vhodný způsob implementace komunikačního rozhraní na platformě Xilinx Zynq. Komunikační rozhraní umožní zapisovat konfigurační data do čipů, číst změřená data, a zajistí správné časování signálů pro samotné měření.

Vytvořte popis jádra komunikačního rozhraní čipů PHpix, které bude prostřednictvím standardní sběrnice AXI-4 připojitelné k procesorovému systému platformy Xilinx Zynq. Důraz je kladen na minimalizaci paměťových nároků, maximální datovou propustnost a variabilitu. Navrhněte aplikaci pro PC, která umožní realizovat měření se senzory s čipy PHpix, a umožní základní vizualizaci změřených dat. Pro komunikaci PC s platformou Xilinx Zynq použijte rozhraní Ethernet. Proveďte implementaci a demonstraci funkce jádra i obslužné aplikace na PC.

DOPORUČENÁ LITERATURA:

[1] LINDH Lennart, KLEVIN Tommy. Advanced HW/SW Embedded System for Designers, 2018: FPGA - System On Chip. Independently published, 2018. ISBN: 978-1731115812

[2] CHATTOPADHYAY Santanu. Embedded System Design. PHI; Second edition, 2013. ISBN: 978-8120347304

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 20.5.2021

Vedoucí práce: Ing. Michal Kubíček, Ph.D.

prof. Ing. Tomáš Kratochvíl, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato práce se zabývá zkoumáním rozhraní a vlastností detektoru ionizačního záření s čipem PhPix, který je základem budoucího velkoplošného detektoru pro použití v nukleární medicíně. Je navrženo IP jádro pro komunikací mezi procesorovým systémem a PhPix čipy na platformě Zynq. Dále je vytvořen software pro přenos videa používající rozhraní Ethernet.

KLÍČOVÁ SLOVA

Zynq PhPix IP video stream Petalinux gui

ABSTRACT

This work investigates the interface and properties of an ionization radiation detector with the PhPix chip, which is the basis of a future large-scale detector for use in nuclear medicine. In the next stage, an IP core for communication between the processor system and the PhPix chips on the Zynq platform is designed. Furthermore, software for video transmission over Ethernet interface is developed.

KEYWORDS

Zynq PhPix IP video stream Petalinux gui

OPIOL, Zbigniew. *Rozhraní pro senzory PHpix*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2021, 38 s. Diplomová práce. Vedoucí práce: Ing. Michal Kubíček, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Zbigniew Opiol
VUT ID autora:	154824
Typ práce:	Diplomová práce
Akademický rok:	2020/21
Téma závěrečné práce:	Rozhraní pro senzory PHpix

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Michalovi Kubíčkovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Díky také patří mé rodině a partnerce, kteří mě podporovali.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

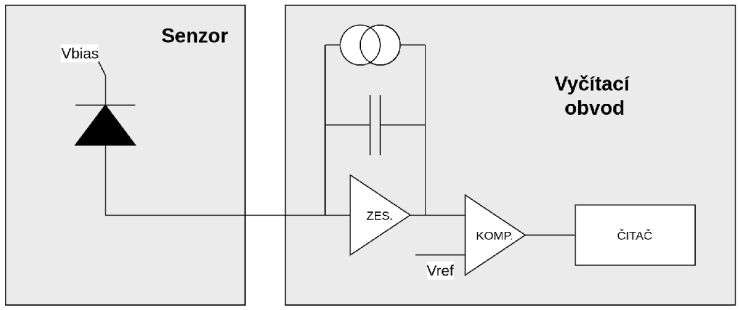
Úvod

Rentgenové záření bylo poprvé pozorováno W. C. Röntgenem v zimě roku 1895. Záření pojmenoval paprsky X. Po tomto objevu se zářením začala zabývat řada fyziků, mezi jinými také Henry Becquerel. Ten se ze začátku domníval, že se jedná o luminiscenci, brzy však došel k závěru, že uran září sám o sobě. Objevil tímto radioaktivitu. Dnešní uplatnění radioaktivity je velmi široké, například v nukleární medicíně jako kontrastní látky či během ozařování, v průmyslu atomové reaktory nebo v paleontologii radiokarbonové datování.

V této práci je zkoumáno rozhraní a vlastnosti detektoru ionizačního záření s čipem PhPix, který je základem budoucího velkoplošného detektoru pro použití v nukleární medicíně. Dále se práce zabývá vytvořením popisu jádra pro připojení čipů k platformě Red Pitaya obsahující SoC Zynq a různými způsoby přenosu obrazu přes Ethernet, včetně implementace serveru a grafického klienta.

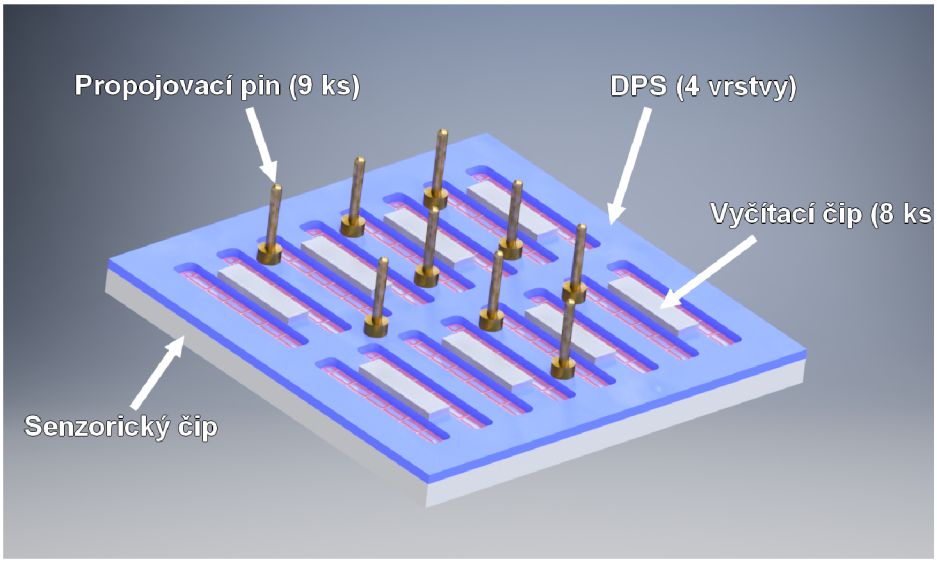
1 Detektor

Tato kapitola se věnuje popisu plošného detektoru záření. Samotný detektor se skládá ze dvou částí: čip senzoru - PN přechod na ultra čistém křemíku a čip vyčítací elektroniky. Konkrétně se jedná o derivát PH32 [1], na rozdíl od kterého čítání pulzů probíhá asynchronně.



Obr. 1.1: Ideové zapojení jednoho kanálu

Ilustrace 1.1 zobrazuje zjednodušený model detektoru. Při interakci s γ-fotonem je uvolněn náboj, který je zesilovačem převeden do vazebního kondenzátoru. Následně je vybit s pomocí konstantního zdroje proudu, což má za následek pilový signál na vstupu komparátoru, kde pokud překročí referenční úroveň, je inkrementován čítač. Kvůli zvolené technologii výroby musí být použit pseudonáhodný čítač, který je později dekódován pomocí vyhledávací tabulky.



Obr. 1.2: Prototyp šupiny - zadní strana

Phpix je určen pro použití s křemíkovými senzory, kde každý čip disponuje 32 kanály (pixely). Jednotlivé čipy jsou poskládané do ucelených šupin (angl. "scales"),

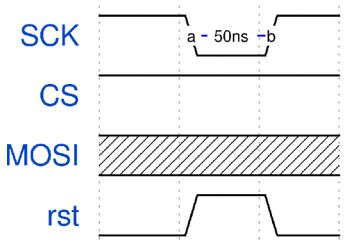
kde každou šupinu tvoří 8 PhPix čipů s výsledným rozlišením 16x16 pixelů. Velikost jednoho pixelu je $1mm^2$. Na 1.2 jsou k vidění samotné čipy. V dalších verzích jsou zality pryskyřicí. Detekční plocha se nachází na druhé straně šupiny.

1.1 Digitální rozhraní

V této práci použita šupina disponuje těmito vlastnostmi[2]:

- počet čipů: 8 (vždy 2 zapojeny sériově)
- rozhraní: 4x SPI
- maximální pracovní frekvence: 22 MHz
- standard logických úrovní: 1,8 V LVCMOS
- statická spotřeba: 43 mW

Komunikace s čipem PhPix probíhá pomocí rozhraní SPI, přestože čip používá resetovací sekvenci, která není ve standardu. Sekvence je však volitelná, a pokud je možno zahodit první snímek, není nutné ji provádět.

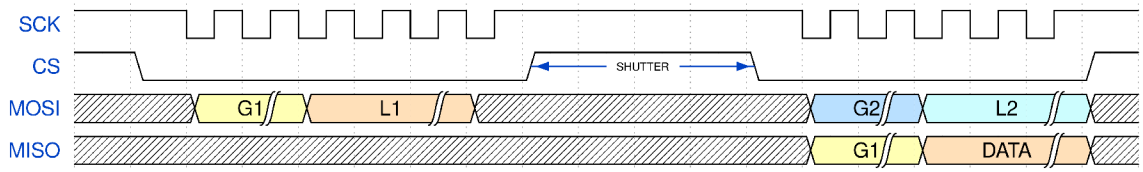


Obr. 1.3: Resetovací sekvence

Každý čip disponuje 104 bitovým globálním konfiguračním registrem a 16 bitovým registrem pro každý pixel. Celková velikost konfigurace pro šupinu (8 čipů) je:

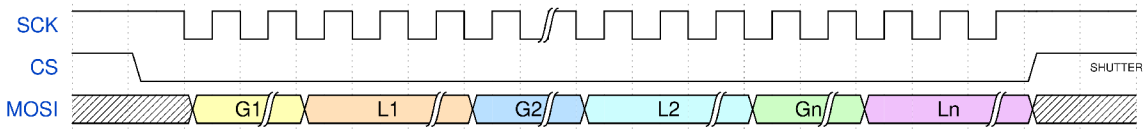
$$S = N_{chip} \times (S_{global} + S_{local} \times N_{pixel}) = 8 \times (104b + 16b \times 32) = 8 \times 616b = 4928b \quad (1.1)$$

Princip komunikace je nastíněn na obrázku 1.4. Jedná se o teoretický popis s jedním čipem, na kterém je snadnější demonstrovat funkčnost detektoru. Čip pracuje ve dvou hlavních stavech: komunikace a měření. Mezi těmito stavy je přepínáno pomocí signálu CS. Čip vzorkuje příchozí data na náběžnou hranu hodinového signálu. Jako první je vyslán nejvýznamnější bit globálního registru. Po odeslání všech (104) bitů následují konfigurace jednotlivých kanálů. Přechodem signálu CS do stavu log. '1' se spustí měření. Po vyčkání vhodné doby (v závislosti na aplikaci řádově desítky milisekund) je možno vyčíst naměřené hodnoty a zároveň nahrát novou konfiguraci.



Obr. 1.4: Konfigurace a čtení dat z jednoho čipu. G_n: Globální konfigurace (čip); L_n: Lokální konfigurace (pixel); DATA: Validní obrazová data; n: číslo snímku.

Při komunikaci s reálnou šupinou, kde jsou čipy zapojeny sériově, je vhodné konfiguraci pro všechny čipy současně, tím se předejde trhání obrazu a nekonzistentní konfiguraci. Ilustrace 1.5 ukazuje konfigurační fází čipů na šupině.



Obr. 1.5: Konfigurace čipů pro jeden snímek. G_n: Globální konfigurace (čip); L_n: Lokální konfigurace (pixel); n: číslo čipu.

1.2 Řetězení šupin

Celkovým cílem projektu je vytvoření plošného detektoru, který pro využití v praxi musí mít detekční plochu alespoň 30x30 cm a obrazový dojem musí být relativně plynulý. Pro tento detektor je uvažována matice 32x32 šupin.

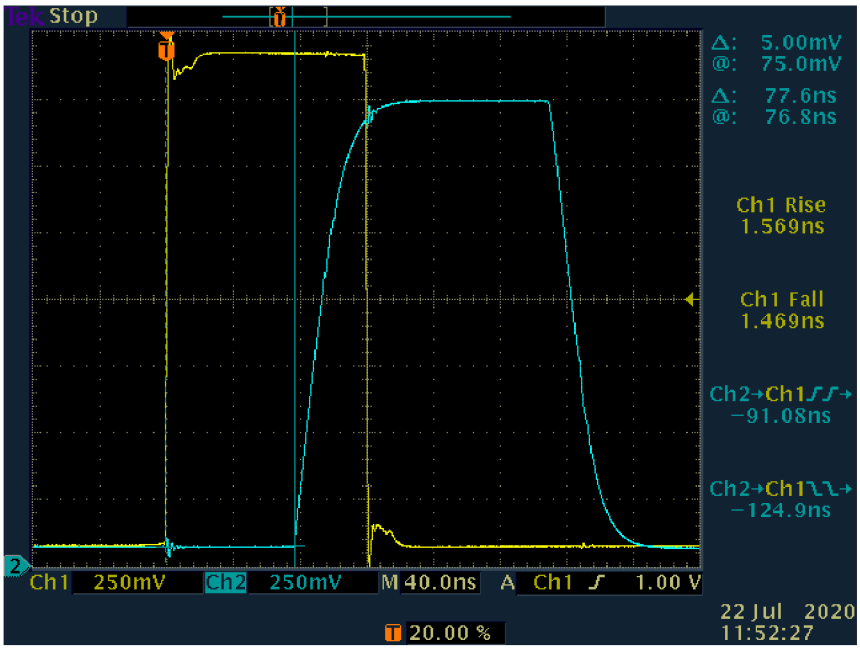
Aby bylo možné stanovit maximální počet šupin na jedné sběrnici, je zapotřebí znát časové parametry čipu. Oscilogram 1.6 demonstruje časové průběhy na vstupu a výstupu hodinového signálu. Bohužel nástupná a sestupná hrana mají rozdílné časy, což má za následek postupné zkracování stavu logické '1' až do jejího vymizení. Tento jev značně omezuje možnost řetězení šupin za sebe. Alternativou je centrální distribuce hodinového a chip select signálu nebo použití více rozhraní paralelně.

Časový rozdíl mezi nástupnou a sestupnou hranou jednoho čipu je 4,2 ns. Z tohoto zjištění je možné znázornit závislost maximální snímkové frekvence na počtu sériově zapojených šupin. Minimální doba logické '0' je 10 ns a je definována jako:

$$T_{0minIN} = T_{0minOUT} + (N \times 8 \times T_{diff}) \tag{1.2}$$

Za předpokladu $T_{0min} = T_{1min}$ lze určit maximální frekvenci jako:

$$f_{max} = 1/(T_{0minIN} + T_{1minIN}) \tag{1.3}$$



Obr. 1.6: časová asymetrie CLK signálu

N (šupiny v sérii)	**fps**	**vodiče rozhraní**
64	1,4	34
32	5,3	66
16	18,2	130
8	46,7	258

Tab. 1.1: Závislost počtu šupin v sérii na maximální snímkovací frekvenci

Přehled různého množství sériově zapojených šupin se nachází v tabulce 1.1.

2 Příprava SW

Pro vývoj byl použit vývojový kit společnosti Red Pitaya pod názvem STEMlab 125-14.Přípravek je osazen programovatelným hradlovým polem Xilinx Zynq řady 7000. Toto *field programmable gate array* (FPGA) obsahuje dvoujádrový "hard"PS architektury ARM Cortex A9, který není nutné syntetizovat. Tento PS umožňuje běh operačního systému Linux, na kterém mohou běžet služby typu HTTP serveru nebo enkodér videa.

Před začátkem sestavování je zapotřebí se rozhodnout, zda použít oficiální čisté jádro a zavaděč, či použít hotový sestavovací systém. Pro usnadnění práce embedded vývojářů vznikly iniciativy jako *buildroot* nebo *yocto project*. Tyto projekty spojují jednotlivé kroky sestavení bootovatelného obrazu do jednoho celku. V případě práce s FPGA nastává ještě jeden problém vyplývající z povahy hradlových polí - možnost implementace jakékoliv periferie. Aby kernel věděl, jaká periferie je na které adrese a mohl ji přiřadit správný ovladač, používá se mechanismus zvaný *device tree*. Společnost Xilinx nabízí nástroj Petalinux, který usnadňuje proces vývoje od implementace IP po testování v emulátoru. Jedná se o nadstavbu nad *yocto project* obsahující i upravené jádro z vlastního stromu. Pro vývojáře to nese několik implikací, jako je například nemožnost použíti nejnovějších vlastností. Na druhou stranu to nese řadu výhod, jako podpora pro IP, která ještě nejsou, nebo z principiálních důvodu nemohou být začleněna do hlavního stromu jádra. Příkladem ovladače, jenž nemůže být začleněn, může být podpora pro rekonfiguraci hradlového pole. Ve stromu Xilinxu je ovladač, který vytvoří /dev/xdevcfg, do kterého postačí přesměrovat bitstream. Toto řešení však není univerzální napříč výrobci proto nemůže být začleněno.

2.1 SDK PetaLinux

Vývojové prostředí se instaluje pomocí unifikovaného Xilinx instalátoru, kde se během dialogu volby produktu zvolí PetaLinux. Tato volba je dostupná pouze pro OS Linux. Není nutná instalace Vitis/Vivado. Po instalaci se ve zvoleném adresáři (ve výchozím stavu /tools/Xilinx) objeví binární soubor *petalinux-<verze>-final-installer.run*, který je potřeba spustit. Na rozdíl od předchozího kroku instalace pravděpodobně při prvním pokusu selže, kontroluje totiž dostupnost potřebných balíčků. V případě Ubuntu LTS 20.04 se jedná o následující: gawk, gcc, xterm, autoconf, libtool, texinfo, zlib1g-dev, gcc-multilib, build-essential, ncurses, zlig1g:i386

```
$ apt install -y gawk gcc xterm autoconf \
libtool texinfo zlib1g-dev gcc-multilib \
```

```
build-essential libncurses-dev zlib1g:i386
```

Výpis 2.1: instalace závislostí PetaLinux SDK

Po doinstalování závislosti je možno spustit samotnou instalaci. Instalace musí být spuštěna jako běžný uživatel s cílovou složkou, kde má uživatel právo zápisu, zároveň nesmí to být adresář instalátoru.

```
$ mkdir ~/petalinux
$ ./petalinux-v2020.2-final-installer.run -d ~/petalinux
```

Výpis 2.2: spuštění instalace PetaLinux SDK

Nyní je SDK nainstalováno. Posledním krokem, který je ovšem potřeba provést před každým použitím SDK (případně přidat do .bashrc nebo obdobného) je načtení proměnných prostředí do aktuálního kontextu. Konfigurační soubor je v kořenovém adresáři projektu. Pokud je využíván i program Vivado, je možné načíst konfiguraci i pro něj. Je nutno dbát na fakt, že Vivado má problém s jazykovou sadou, proto je třeba třeba upravit proměnné LANG a LC_ALL. V opačném případě dochází například k zaokrouhlování čísel, což se projeví například při nastavovaní PLL pro RAM, kde se pracuje s přesností na 6 desetinných míst.

```
source ~/petalinux/settings.sh
source /opt/Xilinx/Vivado/2020.2/settings64.sh
export LANG=en_US
export LC_ALL=en_US.UTF-8
```

Výpis 2.3: příprava proměnných prostředí

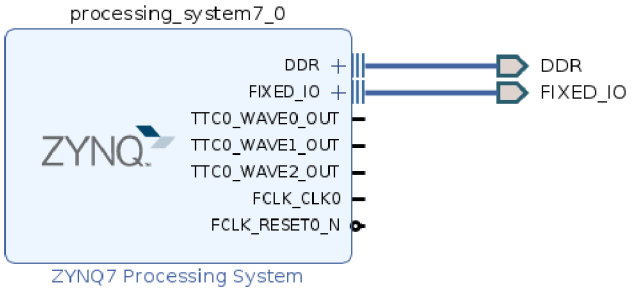
Po tomto kroku je již možno volat všechny příkazy PetaLinux SDK. Pro vytvoření projektu je však potřeba naimportovat popis hardware a bitstream.

2.2 Minimální projekt

Tato sekce popisuje kompletní proces vytvoření bootovatelného linuxového systmému, od založení projektu po ověření funkčnosti v Qemu[ref zde]. Proces je rozdělen na dvě části, kde nejdřív se v prostředí Vivado vygeneruje nastavení procesorového sobsystému a následně se zkompiluje PetaLinux.

2.2.1 Vivado projekt

Pro správnou funkci PS je zapotřebí nakonfigurovat parametry zapojení, jako je typ a množství paměti, piny, na které jsou připojené periférie, vnitřní konfigurace hodinového signálu a další. Toho je možné docílit v schématickém editoru prostředí Vivado2.1.



Obr. 2.1: Minimální zapojení PS

Konfigurace se uloží v souboru ps7_init_gpl.c. Obsahem tohoto souboru je sled maker zapisujících hodnoty z GUI do konkrétních registrů. V případě použití PetaLinuxu na tento soubor není třeba brát zřetel, používá se při použití nástrojů třetích stran, například buildroot. Ve vlastnostech PS je potřeba nastavit podle Red Pitaye RP-125 následující hodnoty:

- Peripheral I/O Pins
  - Bank 0: LVCMOS 3.3V Bank 1: LVCMOS 2.5V
  - Quad SPI Flash
    - Single SS 4bit [1:6]
  - Ethernet 0 [16:27]
    - MDIO [52:53]
  - SD 0 [40:45]
    - Card Detect [46]
    - Write Protect [47]
  - UART 0 [14:15]
  - TTC0 [EMIO]
  - GPIO MIO [0, 7:13, 28:39, 48:51]
- Clock Configuration
  - Input Frequency: 33.333333 MHz
  - CPU Clock Ratio 6:2:1
  - Processor/Memory Clocks
    - CPU: ARM PLL 666.666666 MHz
    - DDR: DDR PLL 533.333333 MHz
  - IO Peripheral Clocks
    - QSPI: IO PLL 200 MHz
    - ENET0: IO PLL 1000 Mbps
    - SDIO: IO PLL 100 MHz
- DDR Configuration

  - DDR Controller Configuration
    * DDR3 MT41J256M16 RE-125
    * Bus Width: 16 Bit
    * DDR: 533.333333

Poté je potřeba podle ilustrace 2.1 označit porty DDR a FIXED_IO jako externí (do constrainového *.xdc souboru se neuvádějí) a vygenerovat wrapper na blokový diagram. Poté je možno spustit syntézu. Posledním krokem v prostředí Vivado je vyexportování popisu hardwaru (přípona se liší dle verze, od 2020 je *.xsa) a bitstreamu. Pro zjednodušení je vhodný export do nové složky pod názvy *system.xsa* a *system.bit*.

2.2.2 Minimální PetaLinux

V tomto kroku bude vytvořen bootovatelný obraz. K tomu je zapotřebí dvou exportovaných souborů z minulé kapitoly. Nejdříve je třeba vytvořit nový projekt. Toho je docíleno příkazem:

```
$ petalinux-create --type project --template zynq \
  --name test_boot && cd test_boot
```

Výpis 2.4: vytvoření projektu

Nyní vytvořený projekt má výchozí konfiguraci pro platformu ZYNQ. V dalším kroku je zapotřebí načíst konkrétní konfiguraci z předchozí kapitoly. Toho lze docílit použitím *petalinux-config*. Tento skript načte konfiguraci a spustí textové uživatelské rozhraní pro úpravu parametrů kompilace.

```
$ petalinux-config --get-hw-description=~/exported_hw
```

Výpis 2.5: import konfigurace PS

V tomto rozhraní je zapotřebí změnit úložiště zavaděče, jádra a devicetree na SD kartu. Tato volba je v následujícím menu:

- Subsystem AUTO Hardware Settings
  - Advanced bootable images storage Settings
    * boot image settings -> image storage media: primary sd
    * kernel image settings -> image storage media: primary sd
    * dtb image settings -> image storage media: primary sd

Je vhodné poznamenat, že skript nabízí volbu *petalinux-config -c kernel*, kde lze měnit parametry jádra či přidávat ovladače pro hardware a také *petalinux-config -c rootfs*, kde lze vybírat balíčky a knihovny pro souborový systém.

V dalším kroku se stáhnou z repozitářů všechny potřebné balíčky a započne překlad. Po úspěšně provedéné kompilaci je potřeba vytvořené artefakty pčipravit pro nahrání na médium.

```
# spustit preklad
$ petalinux-build
# pripravit pro SD kartu
$ petalinux-package --boot --format BIN \
  --fsbl ./images/linux/zynq_fsbl.elf \
  --fpga ~/exported_hw/system.bit --u-boot --force
```

Výpis 2.6: kompilace a příprava pro SD kartu

Výsledek lze ověřit pomocí virtualizačního nástroje Qemu, který emuluje ARM procesor. Emulace se spustí příkazem *petalinux-boot –qemu –kernel*. Nyní je vše připraveno pro nahrání na paměťovou kartu, kterou je potřeba připravit následujícím způsobem:

- Partition 1
  - velikost: 256 MiB
  - fs: fat32
  - flag: boot
  - label: BOOT
- Partition 2
  - velikost: zbylé místo na kartě
  - fs: ext4
  - label: rootfs

Díky použití paměťové karty s dotatečnou kapacitou je možno nahradit vytvořený rootfs nějakou linuxovou distribucí, například Debianem. Tento přístup má výhodu v možnosti využití repozitářů a všech standardních aplikací, avšak na úkor velikosti.

```
# prekopirovat fsbl, uboot, kernel, bitstream
$ cp -r  images/linux/* /media/BOOT/
# extrahovat Debian na partition 2
$ sudo tar xfvp ~/Downloads/*armhf-rootfs-*.tar \
  -C /media/rootfs && sync
$ sudo chown root:root /media/rootfs/
$ sudo chmod 755 /media/rootfs/
```

Výpis 2.7: kopírováni dat na SD kartu

Nyní je systém připraven k použití. Ve výchozím stavu je zapnut SSH server a přihlašovací údaje jsou *debian:temppwd*.

3 Video stream

Důležitou součástí projektu je přenos videa, který bude probíhat přes rozhraní ethernet. Jako zdroj testovacího signálu je použit *Virtual Video Test Driver (vivid)*, který je primárně určen pro testování vstupních i výstupních video zařízení. Více informací a podrobné možností nastavení jsou součástí dokumentace kernelu, nacházejí se v *Documentation/admin-guide/media/vivid.rst* nebo online[3].

Tento ovladač není ve výchozím stavu zapnut, toto je potřeba provést před překladem jádra, jmenovitě v menu po spuštění *petalinux-config -c kernel*, kde se nachází v menu: *Device Driver->Multimedia support->Cameras/grabbers support->Media test drivers->Virtual video driver*. Po překladu je možné načíst mudul, s parametry stanovující jedno vstupní zařízení při QVGA rozlišení. Vytvořené zařízení se zobrazí jako */dev/video0* (nebo první volné v pořadí).

```
$ sudo modprobe vivid n_devs=1 node_types=0x01 \
  num_inputs=0x1 input_types=0x3 v4l2-ctl \
  --set-fmt-video=width=320,height=240
```

Výpis 3.1: spuštění ovladače virtuálního videa

3.1 HTTP Live Stream

První zkoumanou možností je přenos pomocí HLS streamu. Jedná se o streamovaci technologii založenou na HTTP s podporou adaptivního datového toku, vyvinutou v Apple Inc. a publikovanou v roce 2009. V současné době tento format nativně podporuje většina moderních webových prohlížečů (bez doinstalovavání doplňků) a přehrávačů. Další výhodou je přenos standardním HTTP protokolem, tudíž nebývá problém s konfigurací síťových prvků.

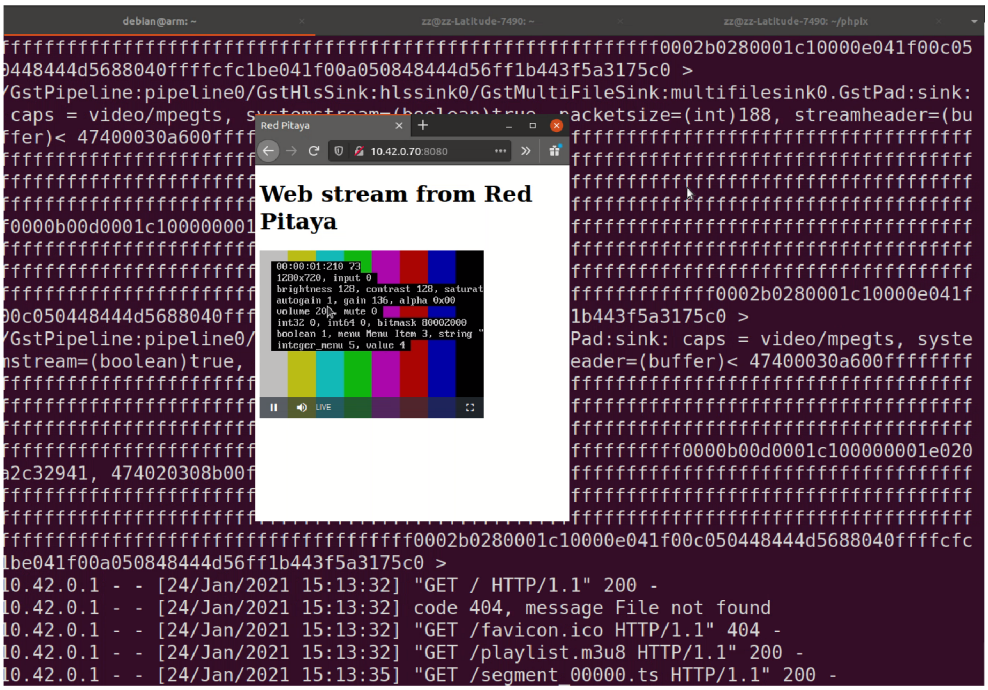
Mezi nevýhody se řadí nutnost použití h264 enkodérů. Jelikož procesorový systém nedisponuje hardwarovým enkodérem je potřeba použít softwarové řešení, které je vypočetně náročné. Další nevýhodou je velká latence (v řádu sekund) vyplývající z principu fungování. Ten spočívá v rozdělění videa na jednotlivé kousky a následné odeslání do prohlížeče, kde se opětovně poskládají do jednoho celku. V níže uvedeném příkladu se video segmentuje do pětisekundových částí.

```
$ gst-launch-1.0 -v v4l2src device="/dev/video0"          \
! videoconvert ! videoscale ! video/x-raw,width=320,      \
height=240 ! x264enc byte-stream=true                     \
speed-preset=ultrafast ! video/x-h264,profile=\"high\" ! \
mpegtsmux ! hlssink playlist-root=http://$ipaddr:8080     \
location=/home/debian/gstreamer/segment_%05d.ts           \
```

```
target-duration=5 max-files=5  &
```

Výpis 3.2: Gstreamer HLS pipeline

Výsledné řešení s použitím HLS spočívá ve vytvoření Gstreamer pipeline s výstupem (pět nejnovějších video souborů *.ts a jeden *.m3u8 playlist obsahující názvy) do adresáře, ve kterém je spuštěn webový server. Adresář taky obsahuje skriptem generovanou webovou stránku, do které je video vloženo.



Obr. 3.1: HLS stream

3.2 Real-time Transport Protocol stream

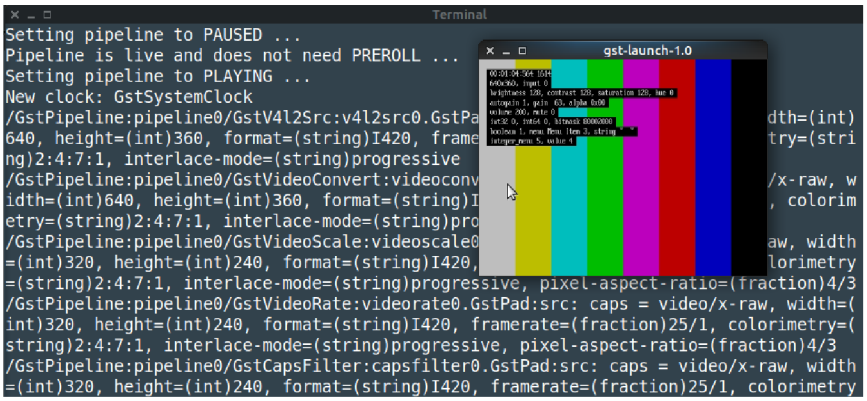
Dalším možným řešením je použití protokolu RTP, který byl navržen pro přenos audia a videa v reálném čase. Oproti HLS má výhodu ve větší variabilitě přenosových formátů a minimální latenci. Není možné však streamovat obsah do webového prohlížeče bez použití doplňků. Při poskytnutí Session Description Protocol (*.sdp) souboru, který popisuje parametry jako údaje o kolorimetrií, rozlišení, pixelclock a další, které jsou nutné ke správnému dekódování videa, je možné je přehrát ve většině moderních přehravačů (např. vlc).

```
$ gst-launch-1.0 v4l2src device=/dev/video0 ! videoconvert !\
  videoscale ! videorate ! video/x-raw,width=320,height=240,\
  format=I420,framerate=25/1 ! rtpvrawpay ! udpsink host=$ip\
```

```
port=5001 sync=false async=false -v
```

Výpis 3.3: Vysilácí Gstreamer pipeline

Výpis 3.3 popisuje vytvoření vysílací části streamu. K tomuto účelu je použit program *gst-launch*, jenž je součástí frameworku Gstreamer.



Obr. 3.2: RTP stream

Snímek 3.2 ukazuje na video přenos, kde na pozadí je spušté vzdálené vysílání a v popředí zobrazen příjmutý výstup.

```
$ gst-launch-1.0 udpsrc buffer-size=622080 port=5001\
  caps="application/x-rtp, media=(string)video,\
  clock-rate=(int)90000, encoding-name=(string)RAW,\
  sampling=YCbCr-4:2:0,depth=(string)8,width=(string)320,\
  height=(string)240,colorimetry=(string)BT601-5,\
  payload=(int)96, a-framerate=25" !\
  rtpvrawdepay !  autovideosink
```

Výpis 3.4: Příjímací RTP pipeline

Zobrazovací okno je vytvořeno pomocí *gst-launch* s pomocí pipeline popsaná v 3.4.

3.3 Vlastní protokol

Poslední zkoumanou variantou přenosu videa je implementace vlastního protokolu. Výhodou tohoto řešení je jednoduchost a přehlednost, ovšem za cenu snížené efektivity přenosu. V tomto konkrétním případě jsou data posílaná v čitelné podobě a dekadickém formátu. Jednotlivé hodnoty pixelů jsou odděleny mezerou. Snímek je ukončen symbolem nového řádku. Takovéto řešení je z hlediska kapacity přenosového kanálu vysoce neefektivní, avšak objem přenášených dat je relativně malý. Velkou

výhodou je možnost přístupu k živým datům bez dalších knihoven nebo programů. Orientační datový tok(nezahrnující paketové hlavičky) pro jednu šupinu a frekvenci 50 Hz se dá stanovit jako:

$$BW = fps((pixel_{bits} + space_{bits}) \times N_{pixels}) = 50((32 + 8) \times 256) = 64kBps \quad (3.1)$$

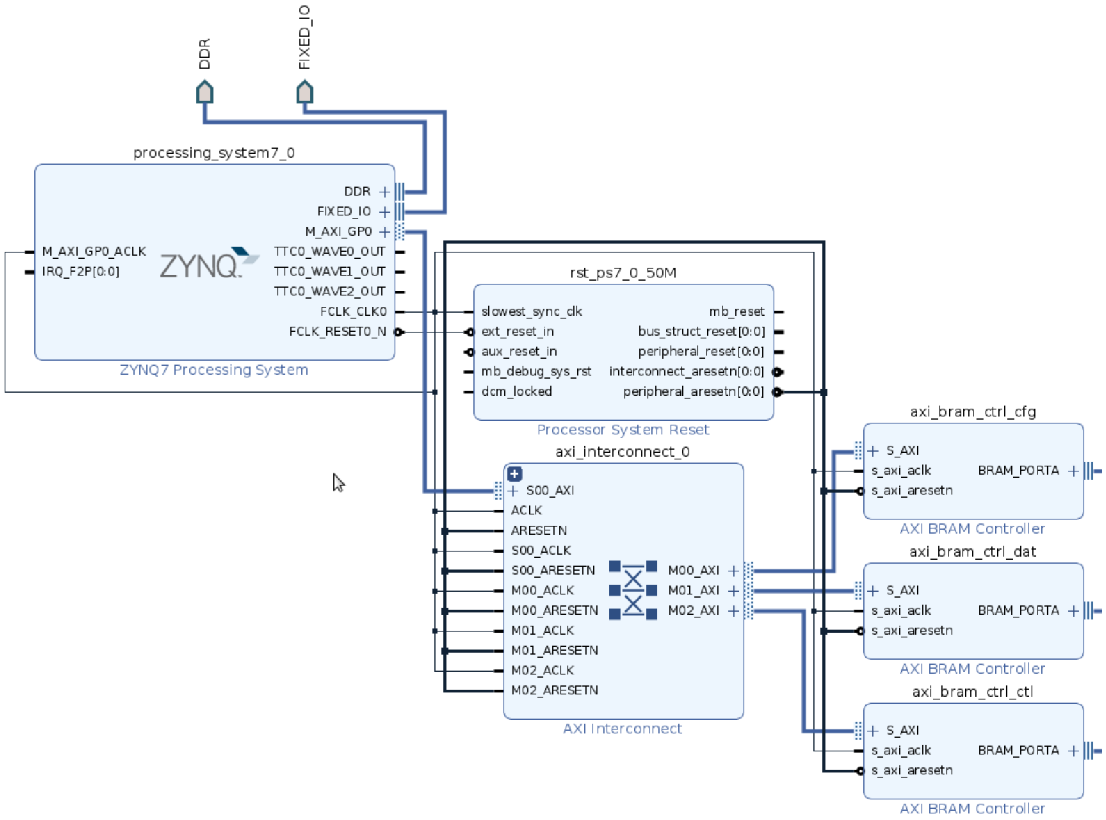
V případě použití více šupin by bylo zapotřebí snížit snímkovací periodu nebo data posílat v binární podobě. Uvážíme-li matici 32x32 šupin, datový tok dosahuje cca 500 Mbps, což představuje zvýšenou zátěž pro infrastrukturu a výpočetní prostředky.

4 Komunikace s PhPix

Tato kapitola v první části popisuje komunikaci mezi procesorovým systémem a jádrem pro komunikaci s čipem a v druhé části detailněji popisuje fungování samotného jádra.

4.1 Komunikace s programovatelným polem

Pro komunikaci mezi PS a PL je použit princip sdílené paměti. Nejedná se však o zápis přímo do DDR RAM procesoru, ale je využitá vnitřní bloková paměť hradlového pole.



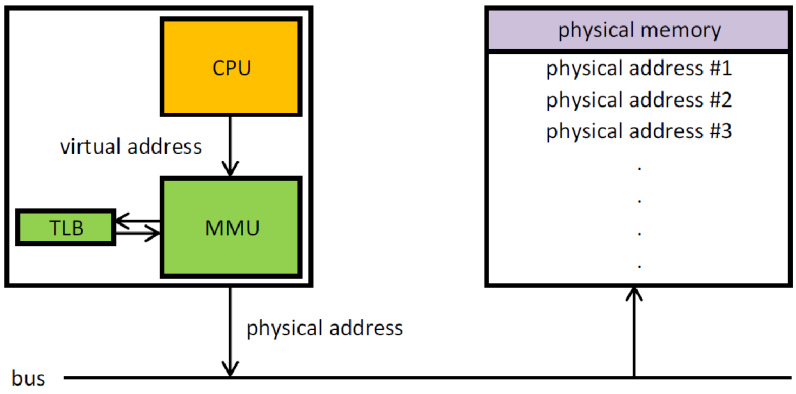
Obr. 4.1: PS-PL rozhraní

Přístup do blokové paměti je prováděn skrz IP komponentu *AXI BRAM Controller*, která překládá provoz z AXI sběrnice na méně komplexní, kompatibilní s BRAM paměti(adresa, data, enable). Tato komponenta nemusí sloužit pouze jako řadič paměti. Jelikož výstupní signály jsou přímočaré, zde blok použít jako řízení stavového automatu, kdy se na výstup místo paměti připojí uživatelská logika. Nevýhodou tohoto řešení zůstává absence notifikačního mechanismu pro změnu stavu

- přerušení. Jediný mechanismus je kontinuální čtení(pool), který ovšem není efektivní. Pokud je požadováná funkcionalita přerušení, je ji potřeba implementovat externě, v takovém případě ale uživatel musí doplnit ekvivalentní kód w softwaru. Alternativou může být například použití uartu, který disponuje ovladačem s podporou přerušení, avšak pro toto použití se příliš nehodí. Ilustrace 4.1 představuje základní zapojení pro 3 řadiče paměti napojené přes AXI směrnici na procesorový systém. Dalšími nutnými komponenty jsou resetovací blok, který zajišťuje správný reset periférií při resetu procesoru a blok pro konfiguraci AXI matice, která zprostředkovává přístup procesoru na více slave zařízení.

4.1.1 SW přístup do hradlového pole

Jelikož aplikační ARM procesor disponuje MMU(memory managment unit), není možné přímo přistupovat na fyzickou adresu, kde je připojen BRAM Memory Controller.



Obr. 4.2: Princip MMU

Memory managment unit mezi jinými zprostředkovává překlad virtuální adresy na fyzickou, ochranu paměti a arbitráž sběrnice. Jednotka rozděluje adresní prostor na jednotlivé stránky . MMU používá stránkovací tabulku, která udržuje záznam pro přiřazení virtuální adresy na fyzickou. Fyzická adresa se vytvoří pomocí kombinace adresy stránky a offsetu. Pokud požadována stránka není v TLB(Translation lookaside buffer) dojde k segmentation fault. Toto chování zaručuje ochranu systému před přepsaním uživatelským procesem. Současné ARM procesory používají, 4KB a 64KB stránky, 1MB sekce a 16 MB super-sekce. Výchozí velikost stránky v případě Red Pitayi je 4096B.

Pro přístup ke konkrétní fyzické adrese je nejdřív nutno pomocí systémového volání open() získat desktriptor k paměti. Zde je důležitý parametr O_SYNC, který zaručuje, že přístup do periférie skutečně proběhne a nepodlehne optimalizaci. Následuje skutečné mapováni do adresního prostoru procesu pomocí mmap(). Zde první

parametr určuje počáteční adresu mapování. V případě že je NULL si ji kernel určí sám, včetně vhodného zarovnání. Následuje velikost stránky, oprávnění pro přístup a nastavení vlajek(flags). Vlajka MAP_SHARED znamená, že v případě přístupu z jiného procesu se jednotlivé navzájem projeví. Poslední dva parametry jsou deskriptor otevřené paměti a offset, který musí být násobkem velikosti stránky.

```
FILE * fd = open("/dev/mem", O_RDWR | O_SYNC )
uint32_t  * ptr = mmap(NULL , 4096, PROT_READ|PROT_WRITE,
                       MAP_SHARED, fd, 0x43C00000);
```

Výpis 4.1: BRAM v paměti procesu

Jako první byl zapotřebí nástroj pro ověření funkčnosti BRAM, zda-li nedochází k přepisování adres nebo šířka slova je menší než očekávána. Program generuje náhodná čísla v rozsahu zadaným uživatelem, zapíše je do BRAM, opětovně vyčte a porovná s lokální kopií. Program "test"akceptuje následující parametry:

- - h help: vypíše nápovědu
- - a address: adresa začátku paměti v šestnáctkové soustavě
- - d depth: hloubka paměti ve slovech
- - n : počet datových bitů
- - v verbose: vypíše podrobné informace o průběhu.

Program pro otestování paměti dat pro PHpix se spustí v následující podobě:

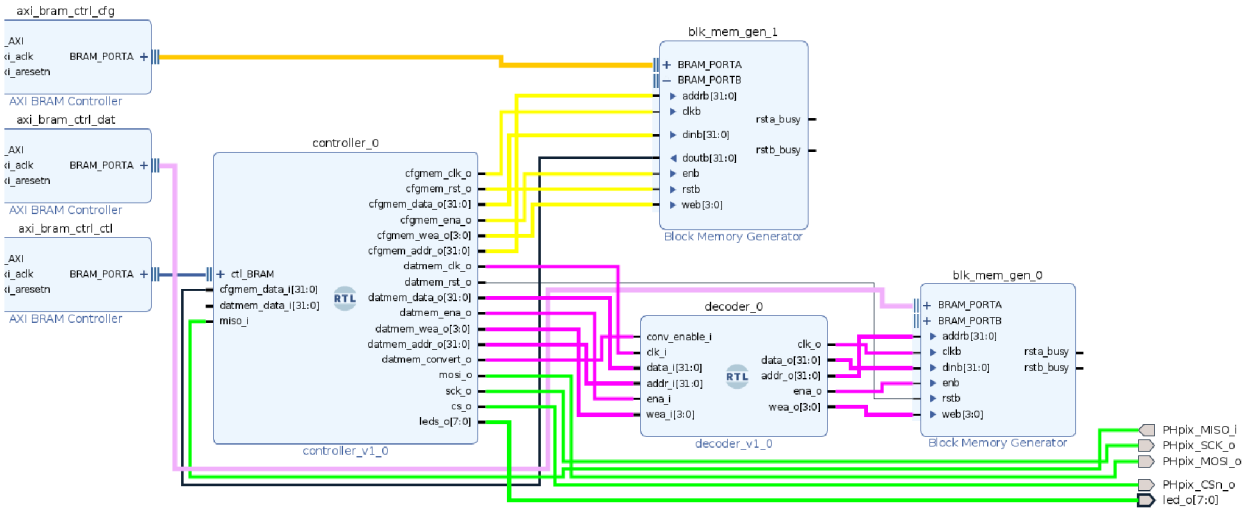
```
root@rp-f05847: ./test -a 0x43c00000 -d 2048 -n 8 -v
mask   : FF 48 0000
TEST PASSED!
0
```

Výpis 4.2: test zápisu do BRAM

4.2 Fyzická vrstva

Komunikace s jádrem probíhá výhradně přes sdílenou paměť, kde jsou použité tři řadiče paměti BRAM, z nichž jsou dva připojené ke skutečné paměti. První, konfigurační(cfgmem) slouží pro uchovávání dat která budou nahrána do Phpix. Do této paměti ARM procesor pouze zapisuje a fyzická vrstva pouze čte. Opačně funguje datová paměť(datmem), kde ARM procesor pouze čte. Mezi datovou pamětí a fyzickou vrstvou je blok dekodéru, který provádí volitelný převod mezi LFSR kódem a binárním číslem. Tato funkcionalita musí být prováděná pouze na obrazových datech - globální konfigurace každého čipu musí být přeskočena. Obě tyto funkce lze ovládat skrze fyzickou vrstvu. Poslední řadič(ctl) řídí jádro skrz zápis do registrů. Blokový diagram 4.3 zobrazuje popsané prvky, kde žlutá zvýrazňuje konfigurační a fialová

datovou paměť. Signály vyvedené ven z FPGA jsou označeny zeleně. Připojení tří řadičů k procesoru ARM je zobrazeno na ilustraci 4.1.



Obr. 4.3: Řídicí logika

Popis fyzické vrstvy je vytvořen v jazyce VHDL. Jeho funkčnost je rozdělena do pěti procesů. První proces zastává funkci generátoru clock enable signálu s variabilním kmitočtem, ovládáným registrem CLK_DIV. Díky použití clock enable je možné měnit pracovní frekvence vybraných částí bez použití fázového závěsu. Výsledná frekvence je dána vztahem 4.1. Vstupní hodinový signál je použit z řadiče BRAM, který se odvíjí od CLK signálu AXI sběrnice.

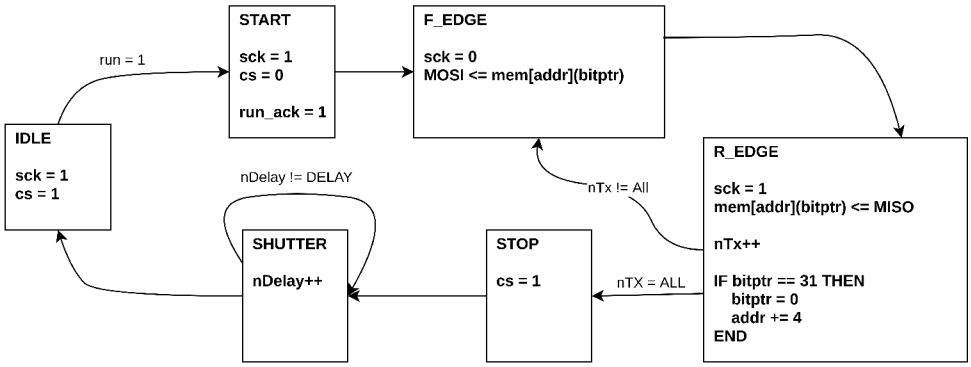
$$F_{out} = \frac{F_{IN}}{CLK\_DIV + 1} \tag{4.1}$$

ADRESA	**NÁZEV**	**BIT**			
		31 : 3	2	1	0
0x00000000	CLK_DIV	value			
0x00000004	BITS_TO_SEND	value			
0x00000008	CONTROL				RUN
0x0000000c	OPTIONS		FREERUN	CONV	RXSKIP
0x00000010	SHUTTER	value			

Tab. 4.1: Konfigurační registry

Další proces *proc_ctl* obsluhuje BRAM rozhraní. Z tohoto důvodu pracuje na plném kmitočtu. Pro přenos informací mezi tímto rychlým procesem a jinými, jenž používají clock enable, slouží synchronizační proces *proc_domain_sync*. Hlavním

procesem je *proc_fsm*, obsahující stavový automat pro komunikaci s PhPix4.4. Automat se spouští pomocí RUN bitu v CONTROL registru. Tento bit je automaticky nulován, tudíž pro kontinuální čtení je potřeba povolit bit FREERUN v OPTIONS registru. V tomto případě se ihned po návratu do stavu IDLE převod znovu spuští. Odesílaní dat je prováděno během sestupné hrany generovaného SCK signálu, poté přechází do stavu generování nastupné hrany a načtení příchozích dat. V tomto stavu také probíhá kontrola, zda-li již bylo načtené celé 32 bitové slovo a pokud se tak stane, vyšle se signál do sekundárního stavového automatu v *proc_write*, který se postará o zápis. V případě, že je povolen bit RXSKIP, je automaticky ignorováno prvních 104 bitů každého čipu, což má za následek zarovnání obrazových dat na celá 32 bitová slova. Pokud je aktivní LFSR dekódování, je nutné použít i RXSKIP, jelikož dekodér je umístěn na paměťové sběrnici a nedisponuje informací, jaká data jsou právě zapisována. Je možné použití s vypnutým LFSR dekódováním a současně zapnutým přeskočením hlavičky. Posledním stavem je prodleva pro snímaní - SHUTTER. Jeho délka je stanovena hodnotou v konfiguračním registru. Délka závěrky je udávaná v počtu clock enable pulzů. Specifikovat délku závěrky má však pouze smysl ve FREERUN módu, jelikož pro PhPix čip není rozdíl mezi IDLE a SHUTTER stavem.



Obr. 4.4: Komunikační stavový automat

5 Software

V konečné fázi je pro přenos videa použit vlastní protokol. Jelikož toto řešení nevyžaduje žádné speciální funkcionality jádra, je možné využití obrazu linuxové distribuce poskytnutém k platformě Red Pitaya STEM. I přesto bylo použito vlastní jádro kompilované projektem PetaLinux, doplněné rootfs Debianu.

5.1 Red Pitaya Server

Software běžící na Red Pitayi je napsán v jazyce C a nepotřebuje další závislosti. Primární funkcí je vyčítáání dat z datové BRAM a odesílání klientovi, zároveň však čeká na případné povely pro upravení provozních parametrů. Přestože je poskytnuta grafická aplikace, pro snadný přístup k datům je možné použít jakýkoliv program, který je schopen pracovat se sockety(nc, putty, bash). Server vyčítá a posílá data s fixní frekvencí(50 Hz), jenž nezávisí na hodnotě SHUTTER.

```
# bash style comments are enabled in this files
# on non-empty lines all characters are ignored exept '1' and '0'
# data are MSB-first: first character goes out first(SPI BUS), but are stored(sent to core) in reversed order
#
# example:
#
#   [MSB] 01101000 .......
#
#   is stored as:
#
#   0X00000004 : ........ ........ ........ ........ ........ ........ ........ ........
#   0x00000000 : ........ ........ ........ ........ ........ ........ ........ 00010110


#104b of global header
01101000 11101110 10110011 11111000 00000100 01110011 00101001 10111000 11000110 11000000 00110000 00011000 00001001

# 32x16b channel config
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
0011111001111001001111001111001
```

Obr. 5.1: Konfigurační soubor

Během spuštění server načte soubor cfginv.bb. V tomto souboru je uložená konfigurace jednoho čipu, která se následně nakopíruje do konfigurační paměti. První bit

v souboru označuje první bit, který bude odeslán. Formát taky umožňuje vkládat do souboru poznámky. Pokud soubor neobsahuje přesný počet konfiguračních bitů, je ohlášena chyba. Pokud je soubor změněn za běhu, je třeba dát pokyn pro znovu načtení konfigurace.

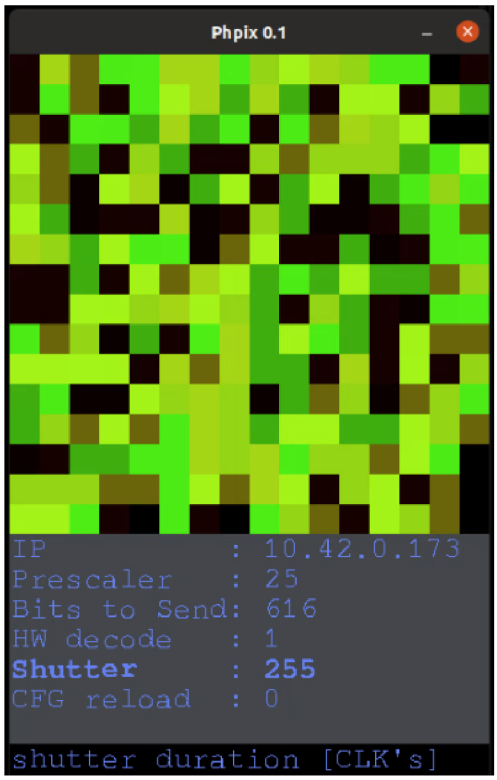
Během provozu lze serveru posílat požadavky na úpravu registrů. Tyto příkazy jsou posílané jako ASCII řetězec ve fixním formátu: *<registr>:<hodnota><CR>*, kde registr nabývá hodnot 1 až 5. Pokud je typ hodnoty "bool", je jakákoliv hodnota mino nuly brána jako "true". Nastavení počtu odesílaných bitů na 42 se tedy provede paketem "2:42".

Název	**registr**	**hodnota**
CKDIV	1	uint32
BITS TO SEND	2	uint32
DECODE	3	bool
SHUTTER	4	uint32_t
CFG RELOAD	5	bool

5.2 GUI klient

Současně s serverem byl vyvinut i grafický klient, který je napsaný v jazyce C a pro vykreslování používá knihovnu SDL2. Z tohoto důvodu je potřeba instalovat balíčky SDL2 a SDL2_ttf. Vykreslení textu se provádí pomocí fontu FreeMono a FreeMonoBold.

Po spuštění aplikace se zobrazí dynamický gradient a konfigurační menu, v němž se pohybuje pomocí kurzorových kláves nebo 'j' a 'k'. Změna hodnoty se provede klávesou Enter, zrušit zadávání lze pomocí Esc. Číselné hodnoty lze zadávat v hexadecimálním tvaru s prefixem 0x. Pro připojení k serveru je potřeba zadat IP adresu a potvrdit. Po připojení gradient je nahrazen obrazem ze senzoru. Provedené změny nastavení se ihned projeví v konfiguraci snímače. Po vypnutí dekódovaní program pracuje i nadále, avšak s barvy neodpovídají hodnotám. Změna Global nebo local registru se provádí nahráním modifikovaného cfginv.bb souboru na server a zapsaním hodnoty do CFG_RELOAD.



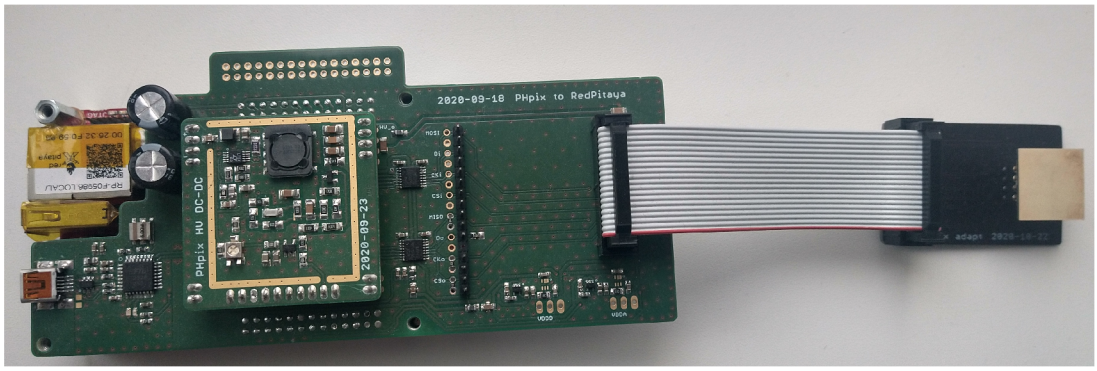
Obr. 5.2: Grafický klient

Závěr

Cílem této práce bylo se seznámit s funkcí čipu PhPix, jeho parametry a možností řetězení čipů. Hlavním výstupem je software pro platformu Red Pitaya STEM, který po rozhraní Ethernet (IEEE 802.3ab) přenáší data do vytvořené klientské aplikace.

V rámci této práce bylo navrženo komunikační jádro, které zprostředkovává komunikaci mezi samotnými čipy PhPix a systémovou sběrnicí ARM procesoru. Dále byla zkoumána možnost nasazení vlastního linuxového systému na míru vytvořeném v prostředí PetaLinux/Yocto Project, na kterým byla vyvinuta serverová aplikace. Pro přenos obrazových dat byly zkoumány varianty HLS streamu, RTP streamu a vlastního protokolu, jenž byl zvolen díky jednoduchosti a možnosti snadno pracovat s přenášenými daty. Pro zobrazovaní dat v PC a změnu parametrů čipů byl vytvořen grafický klient. Z důvodů maximalizace výkonu je napsán v jazyce C s použitím SDL2 knihovny, která umožňuje GPU akceleraci.

Při navýšení počtu šupin zapojených sériově je potřeba na straně serveru i klienta předefinovat konstanty pro množství přenášených dat. V případě zapojení více jader paralelně je také nutno přidat do serverové aplikace nové paměťové regiony. Jádro také teoreticky umožňuje použití stejné části paměti pro uložení jak konfiguračních, tak i naměřených dat. Tento přístup však vyžaduje periodické obnovování konfiguračních dat a synchronizaci mezi userspace aplikací a stavovým automatem v jádru.



Obr. 5.3: Testovací přípravek s připojenou šupinou

Literatura

[1] JANOSKA, Zdenko, Maria CARNA, Miroslav HAVRANEK, et al. Measurement of ionizing particles by the PH32 chip. *2015 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*. IEEE, 2015, 2015, , 1-5. ISBN 978-1-4673-9862-6. Dostupné z: doi:10.1109/NSSMIC.2015.7581968

[2] NEUE, G., T. BENKA, M. HAVRANEK, et al. PantherPix hybrid pixel γ-ray detector for radio-therapeutic applications. *Journal of Instrumentation*. 2018, **13**(02), C02036-C02036. ISSN 1748-0221. Dostupné z: doi:10.1088/1748-0221/13/02/C02036

[3] THE KERNEL DEVELOPMENT COMMUNITY. The Virtual Video Test Driver (vivid). *Kernel.org* [online]. [cit. 2021-5-6]. Dostupné z: https://www.kernel.org/doc/html/latest/admin-guide/media/vivid.html

[4] SEACORD, Robert C. *Effective C: an introduction to professional C programming*. San Francisco: No Starch Press, [2020]. ISBN 978-1-7185-0104-1.

[5] ASHENDEN, Peter J. *The designer's guide to VHDL*. 3rd ed. Burlington: Morgan Kaufmann, c2008. ISBN 978-0-12-088785-9.

[6] SDL2 Support. *Libsdl.org* [online]. [cit. 2021-5-20]. Dostupné z: https://wiki.libsdl.org/Support

[7] *Linux man page* [online]. 1996 [cit. 2021-5-20]. Dostupné z: https://linux.die.net/man

Seznam symbolů a zkratek

FPGA	field programmable gate array
PS	procesorový systém
f_{vz}	vzorkovací kmitočet
IP	Intelectual Propperty
SoC	System on Chip
SPI	Serial Peripheral Interface
CS	Chip Select
SW	Software
SDK	Software Development Kit
PLL	Phase Loop Lock
FSBL	First Stage Bootloader
HTTP	Hypertext Transport Protocol
HLS	HTTP Live Streaming
RTP	Real-time Transport Protocol
PL	Programmable logic
BRAM	Block Random Access Menory
GUI	Graphical User Interface

Seznam příloh

A Registry čipu PhPix

Bit number	Name	Part	Description
15	out_en	analog	Analog output enable. Can be enabled ONLY IN ONE CHANNEL!!! (default value is therefore LOW)
14	FDAC[4]	analog	FDAC
13	FDAC[3]		
12	FDAC[2]		
11	FDAC[1]		
10	FDAC[0]		
9	inject_en	analog	Enable inject
8	gain	analog	HIGH gain = 0 (default) LOW gain = 1
7	channel_en	analog	Channel enable
6	mode_1	digital	Mode select
5	mode_0		
7	TDAC[4]	analog	TDAC
6	TDAC[3]		
5	TDAC[2]		
4	TDAC[1]		
3	TDAC[0]		
2	unused		
1	unused		
0	channel_en	analog	Channel enable

Obr. A.1: Konfigurace pixelu

Bit number	functionality	Part	Default value	Bit setting / Description
103	unused			
102	INJECT_DISABLE	analog	1	turn off Injection
101:94	VINJ_LOW	analog	507 mV	1011 0111
93	VINJ_LOW_EN	analog	1	DAC enable
92:85	VINJ_HIGH	analog	507 mV	1011 0111
84	VINJ_HIGH_EN	analog	1	DAC enable
83:76	VBP_LCC	analog	507 mV	1011 0111
75	VBP_LCC_EN	analog	1	DAC enable
74:67	VDISC	analog	507 mV	1011 0111
66	VDISC_EN	analog	1	DAC enable
65:58	VHYST	analog	1.8 V	0000 0000
57	VHYST_EN	analog	1	DAC enable
56:49	VBP	analog	750 mV	1001 0101
48	VBP_EN	analog	1	DAC enable
47:40	VBN	analog	500 mV	1011 1000
39	VBN_EN	analog	1	DAC enable
38:31	VCASC	analog	800 mV	1000 1101
30	VCASC_EN	analog	1	DAC enable
29:22	VTDAC_REF	analog	900 mV	1000 0000
21	VTDAC_REF_EN	analog	1	DAC enable
20:13	VFDAC_REF	analog	900 mV	1000 0000
12	VFDAC_REF_EN	analog	1	DAC enable
11:4	VFB_BASE	analog	900mV V	1000 0000
3	VFB_BASE_EN	analog	1	DAC enable
2	unused			
1	unused			
0	unused			

Obr. A.2: Globální Konfigurace

B Konfigurační soubor PhPix

Výpis konfiguračního souboru. První bit výpisu je první bit vyslán na sběrnici. v Bram paměti uložen jako nultý bit nultého bytu.

01101000 11101110 10110011 11111000 00000100 01110011 00101001 10111000 11000110 11000000 00110000 00011000 00001001

00111110011111001001111100111100111110011111001001111100111100111110011111001001111100111100