VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

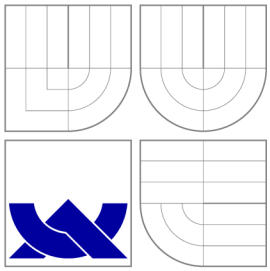
# VGA GRABBER PRO FITKIT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

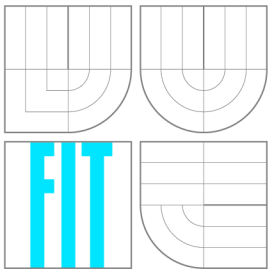
AUTOR PRÁCE
AUTHOR

Bc. JAKUB LOJDA

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# VGA GRABBER PRO FITKIT

FITKIT VGA GRABBER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAKUB LOJDA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK VAŠÍČEK, Ph.D.

BRNO 2015

# Abstrakt

Práce pojednává o možnostech realizace VGA grabberu pro přípravek FITkit. Text je zaměřen na softwarové i hardwarové možnosti realizace. Úvod zavádí čtenáře do teorie dané problematiky. Následně práce uvádí několik možností realizace VGA grabberu a lehké zhodnocení variant. Druhá polovina práce je zaměřena na implementaci nejvýhodnější architektury VGA grabberu z uváděných variant a obsahuje stručné shrnutí poznatků o procesoru *LPC4370* od firmy *NXP* a USB třídě UVC, na které je výsledná architektura založena. Závěr práce obsahuje stručné zhodnocení.



# Abstract

This paper discusses the possibilities of realization of VGA grabber for FITkit. Text is focused on software and hardware implementation possibilities. The first part introduces the reader to the theory of the issue. Next, the paper proposes several options of VGA grabber implementation and brief evaluation of alternatives. The second part describes a chosen architecture of VGA grabber of the featured options and includes a brief summary of the findings of the processor *LPC4370* from *NXP* and USB Video Class UVC, on which the resulting architecture is based. The conclusion includes a brief summary.



# Klíčová slova

VGA, grabber, grabbing, FITkit, USB, DMA, LPC43xx, LPC4370, NXP, video, UVC, USB Video Class, vzorkovani

# Keywords

VGA, grabber, grabbing, FITkit, USB, DMA, LPC43xx, LPC4370, NXP, video, UVC, USB Video Class, sampling

# Citace

Jakub Lojda: VGA grabber pro FITkit, diplomová práce, Brno, FIT VUT v Brně, 2015

# VGA grabber pro FITkit

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Vašíčka, Ph.D.

.....
Jakub Lojda
3. června 2015

## Poděkování

Tímto děkuji vedoucímu práce, Ing. Zdeňku Vašíčkovi Ph.D., za odbornou pomoc při řešení diplomové práce.

# Obsah


**1 Úvod** **4**

**2 Rozhraní VGA (Video Graphics Array)** **5**
2.1 Rozhraní VGA . . . 5
2.1.1 Synchronizační signály . . . 5
2.1.2 Signály reprezentující intenzity barev . . . 6
2.2 Protokol rozhraní VGA . . . 6
2.2.1 Princip vykreslování bodů . . . 6
2.2.2 Oblasti zatemnění a aktivity . . . 7
2.2.3 Synchronizace v rozhraní VGA . . . 7
2.2.4 Režimy VGA . . . 8

**3 Přípravek FITkit** **10**
3.1 Řadič VGA . . . 10
3.2 Vlastnosti VGA na přípravku FITkit . . . 11

**4 Rozhraní USB (Universal Serial Bus)** **12**
4.1 Verze sběrnice USB . . . 12
4.2 Enumerace nově připojených zařízení . . . 12
4.3 Typy přenosů na sběrnici USB . . . 13
4.3.1 Řídicí přenosy . . . 13
4.3.2 Nárazové přenosy . . . 14
4.3.3 Přerušované přenosy . . . 14
4.3.4 Izochronní přenosy . . . 14
4.4 Deskriptory USB zařízení . . . 15
4.5 Třída zařízení UVC . . . 16
4.5.1 Předávání obrazových dat . . . 16
4.5.2 Třídně specifické příkazy . . . 16
4.5.3 Barevné modely . . . 17
4.6 Ovladače pro platformu Windows . . . 17
4.7 Závěr o USB . . . 18

**5 Možnosti realizace VGA grabberu** **19**
5.1 Požadovaná propustnost pro přenos videa . . . 19
5.2 Požadovaná frekvence vzorkování . . . 19
5.3 Volba rozhraní pro připojení k PC . . . 20
5.3.1 Rozhraní Fast Ethernet . . . 20
5.3.2 Rozhraní USB . . . 20

5.4 Možnosti realizace VGA grabberu . . . 20
5.4.1 FPGA s externím ADC převodníkem . . . 20
5.4.2 Architektura s kompresí . . . 21
5.4.3 MCU s vestavěným ADC převodníkem . . . 22
5.5 Volba software na straně PC . . . 22
5.5.1 Transparentní ovladač video zařízení . . . 22
5.5.2 Klientská aplikace . . . 23
5.5.3 Univerzální ovladač video zařízení UVC . . . 23

**6 Mikrokontrolér NXP LPC4370** **24**
6.1 Architektura procesoru . . . 24
6.1.1 Procesor ARM Cortex-M4 . . . 25
6.1.2 Procesor ARM Cortex-M0 . . . 25
6.1.3 Koprocesor ARM Cortex-M0 . . . 25
6.1.4 Subsystém ARM Cortex-M0 . . . 25
6.1.5 Meziprocesorová komunikace . . . 25
6.2 Periferie a jednotky obvodu LPC4370 . . . 26
6.2.1 NVIC (Nested Vectored Interrupt Controller) . . . 26
6.2.2 Vstupně/výstupní piny pro všeobecné využití GPIO . . . 26
6.2.3 SCT (State Configurable Timer) . . . 26
6.2.4 Řadiče GPDMA . . . 27
6.2.5 USB rozhraní . . . 27
6.2.6 Rozhraní Ethernet . . . 28
6.2.7 ADC převodníky . . . 28
6.2.8 Sériové rozhraní pro debugging a JTAG . . . 28
6.2.9 Univerzální deska LPC-Link2 . . . 28
6.3 Odhad realizovatelnosti pomocí LPC4370 . . . 29

**7 Návrh a realizace firmware** **30**
7.1 Koncept firmware . . . 30
7.1.1 Komunikace s počítačem . . . 30
7.1.2 Získávání obrazových dat z rozhraní . . . 30
7.1.3 Synchronizace mezi USB a VGA . . . 30
7.1.4 Mapování úkolů na dostupná jádra . . . 31
7.1.5 Získávání obrazových dat . . . 31
7.1.6 Odesílání dat po rozhraní . . . 31
7.2 Realizace firmware . . . 31
7.2.1 Implementace třídy UVC na sběrnici USB . . . 31
7.2.2 Inicializace . . . 32
7.2.3 Deskriptory USB . . . 32
7.2.4 Odchycení chybových stavů . . . 35
7.2.5 Příkazy specifické pro třídu UVC . . . 35
7.2.6 Přenos obrazových dat . . . 36
7.2.7 Chyby ovladače USBD ROM Stack . . . 39
7.3 Příjem dat z rozhraní VGA . . . 40
7.3.1 Inicializace . . . 40
7.3.2 Vzorkování řádků . . . 40
7.3.3 Konec snímku . . . 40

7.3.4 Podvzorkování . . . . . 41
7.3.5 Latence přístupu do paměti . . . . . 41
7.3.6 Propojení LPC-Link2 a FITkitu . . . . . 41
7.4 Sloučení obou projektů . . . . . 42

**8 Výsledné řešení** **43**
8.1 Ověření činnosti . . . . . 43
8.2 Parametry výsledného řešení . . . . . 44

**9 Závěr** **45**

**A Seznam použitých zkratek** **48**

**B Obsah CD** **49**

# Kapitola 1

# Úvod

VGA grabber je hardwarový prostředek, který umožňuje zpřístupnit obrazový signál přenášený po rozhraní VGA. Často se chová jako virtuální VGA monitor, který je schopen posílat obrazový výstup do příslušné PC aplikace. Aplikace následně umožňuje výstup nejen prohlížet, ale také nahrávat, popř. jinak nakládat se získanými obrazovými daty.

Cílem práce je navrhnout a později také prakticky realizovat VGA grabber, který by bylo možné využít ve spolupráci s VGA výstupem přípravku FITkit. Primárním cílem je vytvořit zařízení, které by bylo vhodné pro výukové účely při práci s VGA rozhraním na přípravku FITkit. Zařízení se připojí k osobnímu počítači, na jehož zobrazovacím zařízení zpřístupní obraz, který přijalo z rozhraní VGA přípravku FITkit. Zařízení by mělo být kompatibilní s operačním systémem *Windows*.

Kapitola 2 této práce se zabývá popisem rozhraní VGA. Následuje krátký popis přípravku FITkit, pro který má být VGA grabber určen. Popis je uveden v kapitole 3. Další kapitola 4 popisuje rozhraní USB, které je určeno pro propojení VGA grabberu s klientským PC. V kapitole 5 je uvedeno několik realizací. Kapitola rozvádí popis zvolené architektury. Předposlední část textu 6 shrnuje základní poznatky o vybraném mikrokontroléru *LPC4370* od firmy *NXP*. Poslední kapitola 9 uvádí závěrečné shrnutí.

# Kapitola 2

# Rozhraní VGA (Video Graphics Array)

Následující kapitola čerpá z [5], [9], [25] a [6]. Rozhraní VGA poprvé spatřilo světlo světa v roce 1987 v řadě počítačů IBM PS/2, časem se ovšem stalo standardním analogovým video rozhraním. Slouží k připojení zobrazovače ke grafickému výstupu zdroje obrazového signálu. Standardně se nachází v roli zobrazovače VGA monitor a v roli zobrazujícího zařízení grafická karta.

V souvislosti s rozhraním VGA je často zmiňován pojem *Array*. Je tomu tak především z historických důvodů, kdy bylo VGA rozhraní implementováno jako jediný čip. Tento čip přitom nevyžadoval příliš mnoho dalších součástí, to mj. umožnilo jeho jednodušší nasazení přímo na základovou desku PC.

## 2.1 Rozhraní VGA

Tato část textu čerpá ze zdroje [6]. Rozhraní VGA sestává z celkem pěti signálů:

- synchronizační: Hsync, Vsync
- intenzity barevných složek: R, G, B

Lze rozlišit dva druhy signálů. Signály *Hsync* a *Vsync* jsou synchronizační, zobrazující zařízení dle těchto signálů odvozuje rozlišení obrazu. Signály $R$, $G$ a $B$ reprezentují intenzity jednotlivých barevných složek určitých obrazových bodů a tedy výslednou barvu pixelu.

### 2.1.1 Synchronizační signály

Synchronizační signály nabývají logických hodnot *0* a *1*. Logická hodnota *0* je reprezentována napětím $0\ V$, logická *1* je reprezentována napětím $5\ V$ [1]. Signály jsou aktivní v logické nule.

V rámci jednoho řádku pixelů není určena synchronizace, tedy není přímo jasné, kdy vzorkovat signály $R$, $G$, $B$. Toto je často řešeno přidáním dalšího vodiče, který určuje tzv. *pixel clock*. Vzorkování hodnot barevných intenzit je pak provedeno na každou náběžnou hranu tohoto signálu.

[1] Rozhraní rovněž akceptuje napětí $3,3\ V$ jako logickou *1*.

**Hsync** Pulzy na vodiči Hsync napomáhají správnému umístění řady obrazových bodů v rámci šířky zobrazující plochy. Zdroj signálu zasílá zobrazujícímu zařízení po rozhraní vždy jeden pulz na začátku a jeden pulz po konci přenosu obrazových dat po signálech $R$, $G$, $B$.

**Vsync** Pulzy signálu Vsync značí začátek a konec přenášeného obrazového snímku. Stejně, jako v případě signálu Hsync, zdroj signálu zasílá vždy jeden pulz na začátku snímku[2] a jeden pulz po konci.

### 2.1.2 Signály reprezentující intenzity barev

Signály reprezentující intenzity barevných složek mohou nabývat hodnot $0\ V$ pro temnou až $0,7\ V$ pro maximální intenzitu jasu dané barevné složky. Jedná se tedy o signály analogové.

Tyto signály lze chápat rovněž tak, že u CRT zobrazovačů řídí každý z těchto vodičů dané barvě příslušející elektronové dělo.

## 2.2 Protokol rozhraní VGA

Rozhraní VGA definuje jednoduchý protokol předávání obrazových informací analogovou cestou. Kompletní specifikace protokolu není dostupná zdarma, je tedy nutné vycházet z informací dostupných na webu, které byly zjištěny jejich autory převážně experimentálně. Přenos obrazových dat se odehrává po jednotlivých obrazových bodech. Směr přenosu obrazových bodů je zleva doprava po jednotlivých bodech a shora dolů po horizontálních řadách. Za standardní režim VGA je považován ten s rozlišením 640 x 480 obrazových bodů a snímkovou frekvencí 60 $Hz$. Tomuto režimu odpovídá frekvence vykreslování bodů 25 $MHz$ a řádkový kmitočet $31,25\ kHz$.

### 2.2.1 Princip vykreslování bodů

Vykreslování vychází z principu funkce zobrazovačů CRT. Tyto zobrazovače byly založeny na proudu elektronů, který byl postupně vychylován pomocí tzv. vychylovacích obvodů. Pro správnou funkčnost potřebují tyto obvody informace o časování, aby byly schopny patřičně rozprostřít přijímaný obraz po obrazovce. Obvody postupně vychylují vysílané elektrony směrem zleva doprava a shora dolů.

Okamžik, kdy je paprsek vychylován směrem zleva doprava a zanechává tak stopu na stínítku CRT zobrazovače, je nazýván *řádkový aktivní běh*. Po dosažení pravého okraje stínítka se paprsek navrací zpět do nejlevější pozice. Okamžik od zahájení návratu paprsku po dosažení nejlevější pozice je nazýván *řádkový zpětný běh*. Při *řádkovém zpětném běhu* se paprsek zároveň pohybuje svislým směrem. Paprsek se tedy přímočaře navrací pod mírným sklonem do místa, odkud opět započne nový *řádkový aktivní běh*.

Svislý pohyb paprsku shora dolů se nazývá *snímkový aktivní běh*. Svislý pohyb paprsku směrem nahoru se nazývá *snímkový zpětný běh*. *Snímkový zpětný běh* neprobíhá přímočaře do nové výchozí pozice, protože během návratu stále dochází k průběžnému vychylování *řádkovými běhy*. V případě zpětných běhů nedochází na stínítku zobrazovacího zařízení k promítání stopy, barva je v těchto případech nastavena na zcela temnou.

[2] Jeden snímek se skládá z několika řádků, řádek se skládá z několika obrazových bodů.

### 2.2.2 Oblasti zatemnění a aktivity

Během vykreslování probíhají obrazové signály *R*, *G* a *B* tzv. oblastmi zatemnění. Jde právě o tyto oblasti, které nespadají do času, kdy probíhají *aktivní běhy*, jinými slovy o časové mezery, které zobrazovací zařízení využije k návratu vychýlení elektronového děla do následující pozice. Z pohledu času můžeme rozlišit tzv. intervaly *Front Porch* a *Back Porch*. Interval *Front Porch* označuje čas od dokončení kreslení obrazu na stínítko obrazovky do přijetí začátku synchronizačního pulzu. Dále můžeme rozlišit tzv. *Back Porch* interval, který časově spadá mezi dobu skončení synchronizačního pulzu a začátek vykreslování. Právě intervaly *Front Porch*, *Back Porch* a délka synchronizačního pulzu poté tvoří oblast zatemnění.

Aktivní oblast odpovídá oblasti dopadající na stínítko zobrazovače. Během průchodu elektronového paprsku touto oblastí dochází k vykreslování obrazu na stínítku, odtud označení *aktivní*.

### 2.2.3 Synchronizace v rozhraní VGA


Obrázek 2.1: Ukázka průběhu signálů na rozhraní VGA

#### Vertikální synchronizace

Vertikální synchronizace určuje dobu, za kterou na analogových vodičích $R$, $G$ a $B$ proběhne jeden obrazový snímek. Vertikální synchronizace je určena vodičem *Vsync*. Tento vodič je dle [25] aktivní v logické nule. Počátek snímku je tedy signalizován nastavením tohoto vodiče do logické *0*. Délky intervalu mezi jednotlivými pulzy na vodiči *Vsync* jsou obvykle počítány v jednotkách period horizontálního signálu. Signál vertikální synchronizace je možno pozorovat v levé části obrázku 2.1, kde je znázorněn jeho průběh v souvislosti se signálem horizontální synchronizace.

#### Horizontální synchronizace

Tyto signály určují frekvenci vykreslování jednotlivých řádků na stínítko obrazovky. Synchronizace je založena na obdobném principu, jako výše uvedená synchronizace vertikální. Doba periody tohoto signálu určuje dobu, za kterou má být vykreslen jeden řádek obrazu na stínítko obrazovky. Zároveň udává začátek každého nového řádku obrazu. Signál je rovněž aktivní v logické nule, signalizace začátku nového řádku tudíž probíhá na základě sestupné hrany logické úrovně signálu. Délku periody bývá zvykem u tohoto signálu uvádět v počtu obrazových bodů, tedy šířce obrazu v bodech. Průběh signálu je možno vidět na obrázku 2.1 v horní části. Zde je zobrazena rovněž návaznost na signály složek barevných intenzit $R$, $G$ a $B$. Na obrázku jsou intenzity barev znázorněny barevnými obdélníky, které představují výslednou barvu viditelnou lidským okem.

### 2.2.4 Režimy VGA

#### Grafické režimy

Přenos po rozhraní VGA může probíhat v několika grafických režimech. Tabulka 2.1 shrnuje nejčastější režimy VGA.

| Rozlišení (pixelů) | Vert. synch. | Horiz. synch. | Frekvence pixelů | Poznámka |
|---|---|---|---|---|
| 320 x 200 | 70 $Hz$ | 31,778 $kHz$ | 25,175 $MHz$ | VGA |
| 320 x 200 | 70 $Hz$ | 31,778 $kHz$ | 25,175 $MHz$ | VGA |
| 640 x 200 | 70 $Hz$ | 31,778 $kHz$ | 25,175 $MHz$ | VGA |
| 640 x 350 | 70 $Hz$ | 31,778 $kHz$ | 25,175 $MHz$ nebo 28,322 $MHz$ | Režim VGA kompat. s předchůdcem EGA |
| 640 x 480 | 60 $Hz$ | 31,778 $kHz$ | 25,175 $MHz$ | VGA; monochr. varianta nazývána též MCGA |
| 640 x 480 | 72 $Hz$ | 37,7 $kHz$ | 31,5 $MHz$ | VGA |
| 640 x 480 | 75 $Hz$ | 37,5 $kHz$ | 31,5 $MHz$ | VGA |
| 640 x 480 | 85 $Hz$ | 43,3 $kHz$ | 36 $MHz$ | VGA |
| 800 x 600 | 56 $Hz$ | 35,1 $kHz$ | 36 $MHz$ | SVGA |
| 800 x 600 | 60 $Hz$ | 37,9 $kHz$ | 40 $MHz$ | SVGA |
| 800 x 600 | 72 $Hz$ | 48,1 $kHz$ | 50 $MHz$ | SVGA |
| 800 x 600 | 75 $Hz$ | 46,9 $kHz$ | 49,5 $MHz$ | SVGA |
| 800 x 600 | 85 $Hz$ | 53,7 $kHz$ | 56,25 $MHz$ | SVGA |

Tabulka 2.1: Nejčastější grafické režimy VGA s informacemi o časování, čerpá z [5], [4] a [9]

Kromě tzv. standardních grafických režimů, které musí podporovat každý adaptér, je možné dosáhnout téměř libovolné kombinace rozlišení:

- 512 až 800 pixelů šířky - při 16 barvách
- 256 až 400 pixelů šířky - při 256 barvách

se

- 200 nebo 350 až 410 řádků - při obnovovací frekvenci 70 $Hz$
- 224 až 256, nebo 448 až 512 - při obnovovací frekvenci 60 $Hz$
- 512 až 600 - při snížené obnovovací frekvenci alespoň na 50 $Hz$

### Textové režimy

Kromě grafických režimů existují také režimy textové, které umožňují vykreslovat pouze textové znaky. O vykreslování znaků se následně stará grafický hardware. Příklady nejčastějších textových režimů shrnuje tabulka 2.2.

| Rozlišení (znaků) | Rozlišení znaku (pixelů) | Výsledné efektivní rozlišení (pixelů) |
|---|---|---|
| 80 x 25 | 9 x 16 | 720 x 400, 16 barev, nebo monochr. |
| 80 x 30 | 8 x 16 | 640 x 480, 16 barev |
| 80 x 43 | 8 x 8 | 640 x 344, 16 barev |
| 80 x 50 | 9 x 8 | 720 x 400, 16 barev |
| 80 x 50 | 8 x 8 | 640 x 400, 16 barev |
| 40 x 25 | 9 x 16 | 360 x 400, 16 barev |

Tabulka 2.2: Nejčastější standardní textové režimy VGA, čerpáno z [18] a [9]

# Kapitola 3

# Přípravek FITkit

FITkit je hardwarový přípravek dostupný v rámci Fakulty informačních technologií (FIT) Vysokého učení technického v Brně. FITkit obsahuje mikrokontrolér, hradlové pole FPGA a řadu dalších periferií. Přípravek využívá pokročilého reprogramovatelného hardware na bázi hradlových polí, který je možné neomezeně modifikovat pro potřebné účely a tím se vyhnout potřebě vytvářet nový hardware pro každou aplikaci znovu. Cílem přípravku je seznámit studenty s problematikou HW/SW codesign a umožnit snadný přístup k realizaci nejen softwarových, ale také hardwarových projektů.

Přípravek FITkit obsahuje jako jednu z mnoha dostupných možností také schopnost generovat výstup v podobě video signálu pomocí řadiče implementujícího rozhraní VGA. Tato kapitola čerpá z informací uvedených v [25].

## 3.1 Řadič VGA

Pro správnou funkčnost řadiče na přípravku je nutné nejprve nastavit patřičnou frekvenci hodinového signálu *CLK* dle zvoleného grafického režimu. Perioda hodinového signálu závisí na horizontálním počtu obrazových bodů.

Řadič na přípravku je schopen generovat signály pro VGA rozhraní v téměř libovolném grafickém režimu. Grafický režim je možno specifikovat pomocí 61 bitového signálu *MODE*.

Po inicializaci parametrů a nastavení signálu *RST* do logické *0* a signálu *ENABLE* do logické *1* začne řadič pracovat. Pomocí signálů *ADDR_ROW* a *ADDR_COLUMN* řadič VGA určuje pozici obrazového bodu, jehož hodnoty mají být předány v následujících hodinových taktech[1] na barvonosné datové signály *DATA_RED*, *DATA_GREEN* a *DATA_BLUE*. Každý z těchto datových signálů nese tříbitovou informaci o barvě. Jakých hodnot mohou signály *ADDR_ROW* a *ADDR_COLUMN* nabývat je určeno zvoleným grafickým režimem[2].

Pro správnou funkčnost je nutné zajistit, aby výstupní digitální signály *VGA_RED*, *VGA_GREEN* a *VGA_BLUE* řadiče VGA byly pomocí D/A převodníku převedeny na analogové signály před zapojením do VGA rozhraní. Výstupní digitální signály *VGA_HSYNC* a *VGA_VSYNC* mohou být připojeny přímo do výstupního VGA rozhraní.

Signály *VGA_RED*, *VGA_GREEN* a *VGA_BLUE* jsou na přípravku FITkit přístupné rovněž v digitální formě, tedy není nutné k nim přistupovat pouze přes zmíněný D/A převodník. Na přípravku FITkit 2.0 jsou signály v digitální formě označeny *X27* až *X35*

[1] Maximální zpoždění, se kterým spolupracující systém předá data VGA řadiči, je možno nastavit pomocí generického parametru REQ_DELAY, implicitně 1 takt.

[2] Pro zvolený režim např. 640 x 480 obrazových bodů nabývá signál *ADDR_COLUMN* postupně hodnot 0 až 639 a signál *ADDR_ROW* hodnot 0 až 479.

a jsou vyvedeny na piny *34* až *42* konektoru *JP10*. Schéma přípravku FITkit verze 2.0 je dostupné z [3].

## 3.2 Vlastnosti VGA na přípravku FITkit

Z dříve uvedených údajů je nutné vyvodit parametry, které umožní, případně usnadní tvorbu VGA grabberu určeného pro přípravek FITkit.

První vlastností VGA na přípravku je počet barev na pixel. Ten je dán tříbitovým rozlišením barvonosných signálů *VGA_RED*, *VGA_GREEN* a *VGA_BLUE*. Odtud plyne maximální počet zobrazitelných barev $2^9 = 512$.

Další parametr se netýká přímo VGA rozhraní na přípravku, ale spíše účelu využití. Jelikož má jít o pomůcku při výuce, je možné počítat s nižší obnovovací frekvencí jednotlivých snímků.

Poslední limitaci, kterou budeme uvažovat, je velikost rozlišení. Vzhledem k účelu je možné snížit nároky na podporu rozlišení video signálu přenášeného po VGA na maximálně 640 x 480 obrazových bodů.

# Kapitola 4

# Rozhraní USB (Universal Serial Bus)

Následující kapitola je zaměřena na popis rozhraní USB. V kontextu práce je sběrnice USB zvolena jako komunikační prostředek k propojení klientského PC s VGA grabberem.

Vývoj USB byl zahájen v roce 1994. Na vývoji se podílelo celkem sedm významných společností. Jejich cílem bylo vytvořit jednotné rozhraní, které by bylo snadno použitelné pro připojení řady periferií. Důvodem byl stále narůstající počet periferních zařízení a s tím spojeny rostoucí požadavky na počet konektorů v systémové desce. Při vývoji byl kladen důraz na jednoduchost použití rozhraní a rychlost přenosu dat. Obvody podporující prvotní verze USB byly produkovány již v roce 1995. Používáním sběrnice USB odpadají dosavadní problémy spojené s instalací nového periferního zařízení do systému, zejména odpadá nutnost manuální konfigurace a vypnutí systému při instalaci nového periferního zařízení.

Sběrnice USB je tzv. řízenou sběrnicí, každou komunikaci zde zahajuje hostitelský řadič sběrnice. Řadič sběrnice zároveň slouží jako brána operačního systému pro komunikaci se zařízeními připojenými k dané sběrnici. Hostitelský řadič postupně oslovuje zařízení připojená ke sběrnici a dotazuje se na přítomnost dat. Tímto principem je zároveň zajištěn výlučný přístup ke sběrnici[1]. Sběrnice je rovněž schopna do jisté míry poskytovat napájení připojeným zařízením. Tato kapitola čerpá z [11], [10], [7] a [2].

## 4.1 Verze sběrnice USB

Existuje několik verzí sběrnice USB, které se liší svými parametry. Tabulka 4.1 zobrazuje základní přehled jednotlivých verzí.

## 4.2 Enumerace nově připojených zařízení

Po připojení nového zařízení ke sběrnici je započat proces jeho tzv. enumerace. Ten spočívá v zaslání signálu *reset* připojenému zařízení. Během resetu zařízení probíhá rovněž proces zjištění podporované rychlosti přenosu daného zařízení. Nově připojenému zařízení je přidělena unikátní 7 bitová adresa. Poté jsou po USB sběrnici vyčteny informace o typu zařízení.

[1] Výlučný přístup ke sběrnici je zajištěn tím, že hostitelský řadič nikdy najednou neosloví více, než jedno zařízení

| Verze | Rok vydání | Teoretická propustnost |
|---|---|---|
| USB 1.1 | 1998 | $1,5\ Mbit/s$ (režim *Low Speed*)<br>$12\ Mbit/s$ (režim *Full Speed*) |
| USB 2.0 | 2000 | $480\ Mbit/s$ (režim *High Speed*) |
| USB 3.0 | 2008 | $5\ Gbit/s$ (režim *Super Speed*) |
| USB 3.1 | 2013 | $10\ Gbit/s$ (režim *Super Speed+*) |

Tabulka 4.1: Přehled verzí sběrnice USB

Pokud jsou na hostitelském systému dostupné ovladače daného zařízení, je tímto okamžikem nově připojené zařízení označeno stavem *nakonfigurováno* a připraveno k použití pro ovladač zařízení.

## 4.3 Typy přenosů na sběrnici USB

Komunikace na sběrnici USB je zahajována hostitelským řadičem. Řadič pravidelně v daných intervalech postupně vyzývá připojená zařízení a dotazuje se, zda zařízení nemají nová data k odeslání. Tento princip je nazýván *polling*, neboli *vyzývání*.

Veškerá komunikace na sběrnici se odehrává pomocí rour. Rouru lze chápat jako logický kanál. Každá roura je svázána s koncovým bodem. Koncové body již přímo náleží připojeným zařízením. Zařízení může mít maximálně 32 koncových bodů, z nichž polovina je vstupních a další polovina je výstupních. Směr komunikace je vždy označován z pohledu hostitelského řadiče. Koncové body jsou označovány čísly. Příslušející roura může být použita pro jednosměrnou nebo obousměrnou komunikaci. Obecně lze rozlišit čtyři typy přenosů na sběrnici USB:

- řídicí přenosy (Control Transfers)
- datové přenosy
  - nárazové přenosy (Bulk Transfers)
  - přerušované přenosy (Interrupt Transfers)
  - izochronní přenosy (Isochronnous Transfers)

### 4.3.1 Řídicí přenosy

Řídící přenosy jsou důležité především v souvislosti s enumerací zařízení. Zároveň je lze použít pro konfiguraci a dotazování se stavu připojených zařízení. Jde o shluky dat, přenášených v náhodných intervalech. Přenosy jsou iniciovány ze strany hostitelského řadiče. Pro *Low Speed* zařízení je velikost paketu v rámci řídícího přenosu omezena na $8\ B$. Pro *Full Speed* je to až $64\ B$. Pro zařízení podporující režim *High Speed* je velikost paketu pevně stanovena na velikost $64\ B$. Tyto přenosy probíhají ve třech fázích, během nichž se přenáší standardem udávané pakety. V první fázi, zvané *Setup Stage*, je zahájen řídicí přenos. Při zahájení je stanovena adresa zařízení a číslo koncového bodu příslušného zařízení. Během následující fáze *Data Stage* probíhá přenos vyžádaných dat. Při překročení délky jednoho paketu jsou data rozdělena do více paketů. Přenosy dat mohou probíhat směrem k zařízení případně i směrem k hostitelskému řadiči. V poslední fázi zvané *Status Stage* probíhá nahlášení výsledného stavu celého řídicího přenosu. Směr hlášení stavu je závislý

na směru přenosu v předchozí fázi. Pokud proběhl přenos dat směrem k zařízení, je nutno, aby zařízení uvědomilo hostitelský řadič o úspěšném přijetí dat. V případě přenosů směrem k hostitelskému řadiči je proces obdobný.

Řídící přenosy jsou vždy svázány s koncovým bodem *0*. Jde o tzv. *control endpoint*, který je ustanoven již před samotným začátkem enumerace zařízení.

### 4.3.2 Nárazové přenosy

Nárazové přenosy využívají především zařízení, která potřebují doručit velký objem dat v předem nespecifikovaných intervalech. Příkladem zařízení využívající tohoto typu přenosů může být např. flash disk. Data doručována pomocí tohoto typu přenosu jsou zabezpečena kontrolním součtem CRC. Zároveň je zajištěno doručení, tedy že nedojde ke ztrátě paketu, pomocí opakovaného doručení. Nárazové přenosy naopak nemohou zaručit určitou šířku pásma ani minimální latenci. Tento druh přenosů je možné uskutečnit pouze v režimech *Full Speed* a rychlejších. Velikosti paketů jsou omezeny. V režimu *Full Speed* je velikost paketu limitována na maximálně 64 $B$, režim *High Speed* umožňuje velikosti až 512 $B$.

Při požadavku na přenos dat, která délkou neodpovídají násobku velikosti paketu nemusí u posledního paketu docházet k zarovnání nulami. Nárazový přenos končí ve chvíli, kdy je:

(a) přenesena požadovaná délka dat

(b) přenesen paket délky menší, než je délka koncového bodu

(c) přenesen paket nulové délky

### 4.3.3 Přerušované přenosy

Přerušované přenosy jsou vhodné v případě, kdy je nutno zajistit doručení dat do určité doby od vzniku požadavku na přenos, tedy garantovat maximální latenci. Zároveň však není garantována minimální přenosová rychlost. Tento druh přenosu je vhodný pro použití s polohovacími zařízeními, jako jsou např. myši či klávesnice. Typicky se jedná o malé objemy dat, které nevznikají v pravidelných intervalech. Zároveň je však nutné tato data doručit s co nejmenším zpožděním.

I v těchto typech přenosů, zařízení, pokud má data k odeslání, musí čekat na výzvu od kořenového řadiče. Teprve pak může informovat hostitelský řadič o nových datech, která čekají na přenos. Maximální velikost efektivních dat paketů spjatých s tímto druhem přenosu je omezena na 8 $B$ pro *Low Speed* zařízení, 64 $B$ pro *Full Speed* a 1024 $B$ pro *High Speed* zařízení.

### 4.3.4 Izochronní přenosy

Izochronní přenosy jsou určeny pro přenos souvislých dat, jejichž datový tok se výrazně nemění. Důraz je kladen na doručení dat s co nejmenším zpožděním. Tento druh přenosů je vhodný pro periferie generující data v reálném čase, která je nutno zpracovat ihned. Tento typ přenosů garantuje požadovanou propustnost sběrnice. Detekce chyb pomocí kontrolního součtu CRC jde zde dostupná, ale k opravě chyb opětovným zasláním nedochází. Vychází se z účelu těchto přenosů, kdy chyba přenosu není kritická[2].

[2]Např. neopravená chyba přenášení při přenosu dat z USB webové kamery se projeví pouze nepostřehnutelným pozastavením obrazu, popř. u USB mikrofonu krátkým výpadkem zvuku.

Maximální délka efektivních dat paketu u přenosů tohoto typu je omezena do 1023 *B* pro *Full Speed* zařízení a 1024 *B* pro *High Speed* zařízení. V průběhu enumerace dochází ze strany hostitelského řadiče k rezervování potřebné kapacity pro možnost garantování propustnosti těchto přenosů. Kapacita se odvíjí od velikosti paketu, jež je specifikována v příslušném deskriptoru koncového bodu. Pro zajištění alespoň částečné funkčnosti v případě odepření požadované šířky páska je vhodné specifikovat alternativní rozhraní s nižšími požadavky na přenosové pásmo. Data odesílaná pomocí izochronních přenosů mohou být kratší, než vyjednaná velikost koncového bodu a délka dat se může lišit mezi jednotlivými transakcemi.

## 4.4 Deskriptory USB zařízení

Každé USB zařízení obsahuje tzv. deskriptory. Ty společně tvoří hierarchickou strukturu. Deskriptory umožňují hostitelskému řadiči snadné rozpoznání periferního zařízení. Jsou zde uchovány informace o zařízení samotném, podporované verzi sběrnice USB, identifikaci výrobce a způsobu možných nastavení zařízení vč. čísel koncových bodů a jejich typů. Existuje celkem pět běžných druhů deskriptorů:

- deskriptor zařízení (Device Descriptor)
- konfigurační deskriptor (Configuration Descriptor)
- deskriptor rozhraní (Interface Descriptor)
- deskriptor koncových bodů (Endpoint Descriptor)
- řetězcový deskriptor (String Descriptor)

USB zařízení může mít pouze jeden deskriptor zařízení. Ten obsahuje informace o kompatibilní revizi USB, *Product ID* a *Vendor ID*[3], které jsou využity pro identifikaci příslušejícího ovladače zařízení na straně operačního systému. Dále deskriptor zařízení obsahuje informaci o počtu konfiguračních deskriptorů. Ty představují počet možných konfigurací, do kterých je možno zařízení uvést.

Konfigurační deskriptory uvádějí informace jako zdroj napájení nebo potřebný proud pro napájení po sběrnici. Je možné zde vyčíst rovněž informaci o počtu rozhraní. Hostitelský řadič po zjištění potřebných informací z deskriptorů zasílá příkaz *SetConfiguration*, který vybírá jednu z možných konfigurací[4].

Na deskriptory rozhraní lze nahlížet jako na logickou skupinu sdružující koncové body. Sdružení je nejčastěji provedeno na základě příslušnosti jednotlivých koncových bodů k určitým funkcím či vlastnostem zařízení.

Dalším druhem jsou deskriptory koncových bodů. Tyto deskriptory popisují koncové body, především jejich typ a velikost. Z těchto informací následně hostitelský řadič odvozuje požadavky na šířku pásma na sběrnici USB. Deskriptory koncových bodů neobsahují popis tzv. *control endpoint* s číslem *0*, který je využíván pro řídicí přenosy a enumeraci. Jeho přítomnost se předpokládá implicitně.

[3] Jedná se o identifikaci zařízení (*Product ID*) a výrobce (v případě *Vendor ID*) pomocí dvoubytového identifikačního čísla, často zapisovaného v hexadecimální formě.

[4] Většina zařízení podporuje pouze jedinou konfiguraci

Posledním typem jsou řetězcové deskriptory. Jejich obsahem jsou člověkem čitelná data. Tento typ deskriptorů je volitelný, je ovšem vhodné jej uvádět. Obsah slouží např. pro snadnou identifikaci periferního zařízení člověkem, který pracuje v operačním systému. Textová data jsou zde uložena pomocí kódování Unicode. Je možné uvést více variant řetězců v různých jazycích.

## 4.5 Třída zařízení UVC

Třída UVC je třídou, která definuje zařízení pro zpracování případně poskytnutí video obsahu pomocí sběrnice USB. Jedná se o preferovaný způsob při tvorbě nových video zařízení. Operační systém Windows disponuje podporou těchto zařízení od verze XP. Zařízení podléhající standardům UVC tedy nepotřebuje vlastní ovladač, stačí zařízení navrhnout tak, aby splňovalo specifikaci této třídy. Následující sekce čerpá z [21].

### 4.5.1 Předávání obrazových dat

Standard UVC definuje dva způsoby předávání obrazových dat. Prvním jsou přenosy typu *bulk*. Tento způsob je jednodušší z pohledu implementovaného zařízení, protože nevyžaduje řešit synchronizaci snímků. Oproti tomu u druhého způsobu implementujícího přenosy pomocí izochronních rour je potřeba vystavit snímek přibližně v době, která odpovídá počtu snímků za sekundu definovaném v příslušejících deskriptorech.

Třída definuje několik formátů, vč. možnosti přenášet *streamovaná* data komprimovaná pomocí některého ze známých kodeků, jako je např. MJPEG, příp. MPEG-4. Oproti tomu zařízení s nižším výkonem, která nedisponují HW podporou komprese mohou využít nekomprimovaného kodeku *Uncompressed*.

### 4.5.2 Třídně specifické příkazy

Třída UVC definuje několik požadavků. Přehled je uveden v následujícím výčtu.

- GET_CUR
- GET_MIN
- GET_MAX
- GET_RES
- GET_DEF
- GET_LEN
- GET_INFO
- SET_CUR

Požadavky jsou spjaty s rozhraním, kterého se týkají. Ne všechny požadavky implementují všechna rozhraní. Jednodušší zařízení mohou implementovat pouze podmnožinu těchto příkazů.

#### GET_CUR

Příkaz *GET_CUR* vydává ovladač ze strany počítače v případě, že chce zjistit momentální nastavení zařízení. Zařízení na tento dotaz odpovídá sadou bytů definovaného formátu s názvem *Video Probe and Commit Controls*. Před prvním nastavením hodnot pomocí příkazu *SET_CUR* jsou tyto hodnoty nedefinované. V takovém případě by zařízení mělo odpovídat *stall*.

#### GET_MIN, GET_MAX, GET_DEF

Tyto příkazy navrací minimální, maximální, resp. výchozí možné hodnoty. Příkazy jsou využívány v procesu negociace parametrů přenosu mezi počítačem a zařízením.

#### GET_RES

Příkaz *GET_RES* navrací rozlišení jednotlivých polí v procesu negociace. Tímto příkazem ovladač na straně počítače zjistí granularitu jednotlivých nastavení.

#### GET_LEN

Příkaz *GET_LEN* umožňuje ovladači na straně operačního systému podat dotaz na velikost struktury *Video Probe and Commit Controls*.

#### GET_INFO

Slouží pro zjištění podporovaných možností a stavu zařízení.

#### SET_CUR

Slouží k nastavení parametrů přenosu. Jde o příkaz používaný k vyjednávání parametrů přenosu.

### 4.5.3 Barevné modely

V rámci nekomprimovaného, tzv. *Uncompressed*, video kodeku je možné specifikovat několik barevných modelů, které jsou ovladačem třídy UVC podporovány. Mezi podporované formáty patří *YUY2* a *NV12*. Jde v podstatě o kompresní formáty, které podvzorkují barevnou složku, zatímco světelná zůstává zachována. Operační systém podporuje navíc barevné modely *RGB16* a *RGB24*. Formát je v specifikován v rámci konfiguračního deskriptoru identifikátorem *GUID*. Příslušející identifikátory *GUID* je možné dohledat na [19].

## 4.6 Ovladače pro platformu Windows

Vzhledem k primární orientaci realizace pro použití se systémem Windows nastává drobná komplikace. Od verze Windows 8 je totiž vyžadována instalace pouze digitálně podepsaných ovladačů[5], které prošly testy firmy *Microsoft*. Na webové stránce [8] je uvedeno více podrobností o této problematice, včetně způsobu vypnutí kontroly digitálních podpisů ovladačů. Jedná se však o poměrně drastický zásah do konfigurace klientské stanice, který je založen

[5]Verze Windows 7 a nižší v souvislosti s instalací ovladačů, které nejsou digitálně podepsané, zobrazily pouze varovný dialog. V tomto dialogu bylo nutno tento fakt odsouhlasit.

na uvedení operačního systému do testovacího režimu. Přitom samotný podpis ovladačů není jednoduché získat.

Tento nedostatek je odstranitelný použitím univerzálního ovladače zařízení, který je již součástí operačního systému a jehož instalace proběhne automaticky po detekci zařízení příslušejícího do patřičné třídy a to bez zásahu uživatele.

## 4.7 Závěr o USB

Závěrem lze konstatovat, že sběrnice USB je vhodným kandidátem pro realizaci propoje mezi VGA grabberem a klientským PC. Sběrnice USB od verze 2.0 disponuje dostatečnou kapacitou přenosu. Zároveň je z hlediska konfigurace uživatelsky přívětivá. Drobným nedostatkem může být náročnost implementace a dodržení požadovaných standardů.

# Kapitola 5

# Možnosti realizace VGA grabberu

Z pohledu praktické realizace VGA grabberu je nutno přihlédnout k výše uvedeným faktům a také účelu použití tohoto zařízení. V neposlední řadě je potřeba respektovat časovou náročnost realizace, včetně tvorby prototypu a firmware pro takové zařízení. Následující kapitola proto pojednává o možných realizacích VGA grabberu.

## 5.1 Požadovaná propustnost pro přenos videa

Pro správnou volbu rozhraní je potřeba znát jejich požadovanou propustnost. Přípravek FITkit využívá ke generování VGA výstupu komponentu VGA. Proto budeme vycházet z vlastností komponenty VGA, jež jsou podrobně shrnuty v sekci 3.2. Budeme tedy uvažovat maximální rozlišení 800 x 600 obrazových bodů, snímkovací frekvenci 10 Hz a rozlišení maximálně 512 barev. Tyto parametry by měly pro výukové účely plně dostačovat.

Datový tok videa je poté možno určit vzorcem $B = F \cdot R$, kde $F$ představuje velikost jednoho snímku videa a $R$ značí snímkovou frekvenci videa v počtech snímků za sekundu. Velikost jednoho snímku $F$ je určena vertikálním rozlišením $H$, horizontálním počtem bodů $W$ a počtem bitů pro specifikaci barvy jednoho pixelu $D$. Tedy $F = W \cdot H \cdot D$.

Výsledný vzorec tedy zní $B = W \cdot H \cdot D \cdot R$. Po dosazení maximálního rozlišení 800 x 600 obrazových bodů, maximální snímkové frekvence 10 Hz a 9 bitů pro specifikaci barvy jednoho obrazového bodu dostáváme $B = 800 \cdot 600 \cdot 9 \cdot 10$, tj. $B = 43.2\ Mbit/s$ datového toku nekomprimovaného videa.

## 5.2 Požadovaná frekvence vzorkování

Dále je nutno zvážit počet analogových vzorků za sekundu. Tento parametr ovlivňuje volbu ADC převodníku, případně mikrokontroléru, jehož součástí by patřičný ADC převodník byl. Frekvence vzorkování lze odvodit z frekvence snímku a rozlišení podle vzorce $f_{RGB} = W \cdot H \cdot R \cdot C$, kde parametry $W$ a $H$ představují šířku a výšku přenášeného videa v obrazových bodech, parametr $R$ značí snímkovou frekvenci a proměnná $C$ určuje počet barevných kanálů[1]. Po dosazení je tedy možno určit $f_{RGB} = 800 \cdot 600 \cdot 10 \cdot 3$, tj. frekvence vzorků pro všechny barevné kanály je $f_{RGB} = 14.4\ Ms/s$. Pro jeden barevný kanál činí frekvence vzorkování $f_1 = 4.8\ Ms/s$.

[1] Rozhraní VGA přenáší obraz za pomoci třech kanálů barevných intenzit - R, G a B.

## 5.3 Volba rozhraní pro připojení k PC

Za vhodné kandidáty pro realizaci propojení mezi PC a VGA grabberem lze považovat rozhraní *Fast Ethernet* a *USB*.

### 5.3.1 Rozhraní Fast Ethernet

Rozhraní *Fast Ethernet* se vyznačuje teoretickou propustností 100 $Mbit/s$. Pro přenos by bylo vhodné využít protokolu IP s pevně danou IP adresou cíle přijímajícího proud videa[2]. Na vyšší vrstvě by bylo použito protokolu UDP s pevně danou velikostí efektivních dat.

Výhodou je nezávislost na dostupnosti ovladačů zařízení pro operační systém klientského počítače. Navíc aplikaci pro příjem datového video proudu lze snáze portovat pro různé operační systémy.

Možným problémem při použití rozhraní *Fast Ethernet* je přívětivost konfigurace zařízení. Jednou z možných variant se jeví malý webserver s jednoduchou webovou stránkou uvnitř zařízení, který by po zadání přihlašovacích údajů zpřístupnil nastavení. Jednalo by se především o konfiguraci cílové IP adresy pro zasílání video streamu.

Možnou překážkou použití tohoto rozhraní, pokud by nebylo využito nějaké již existující vývojové platformy založené např. na operačním systému Linux, je implementace DHCP klienta pro případ připojení do sítě, která není navržena pro zařízení s pevně danou, uživatelem zvolenou, IP adresou. Východiskem by mohlo být využití externího adaptéru USB, za pomocí kterého by byla mezi klientským počítačem a VGA grabberem vytvořena oddělená síť, ve které by byla umožněna statická adresace, kde by tento protokol nebylo potřeba implementovat.

### 5.3.2 Rozhraní USB

Další variantou je využít sběrnici USB. Sběrnice USB verze 2.0 disponuje dostatečnou teoretickou propustností 480 $Mbit/s$, jak již bylo popsáno v 4.1. Nevýhodou této volby je nutnost implementace ovladačů pro cílovou platformu. Dále je nutno pamatovat, že zařízení implementující daný standard USB stále nemusí podporovat přenosy v maximálních propustnostech, které tento standard definuje.

Výhodou sběrnice USB je, že nově připojená zařízení jsou konfigurována automaticky za pomocí enumerace a ovladačů instalovaných do operačního systému. Není tedy nutno zasahovat do konfigurace počítačové sítě, či jinak řešit adresaci zařízení. Z pohledu uživatelské přívětivosti se tedy jeví rozhraní USB jako lepší volba.

Pro následující varianty se tedy omezíme na propojení VGA grabberu s PC pomocí sběrnice USB.

## 5.4 Možnosti realizace VGA grabberu

### 5.4.1 FPGA s externím ADC převodníkem

Jako první varianta se nabízí využití hradlového pole FPGA ve spojení s rychlým externím ADC převodníkem. Vzorkování by probíhalo probíhalo v daných intervalech pomocí externích ADC převodníků. Ty by pomocí rozhraní SPI posílaly do FPGA navzorkované hodnoty jednotlivých barevných kanálů. Zde by mohlo probíhat případné podvzorkování.

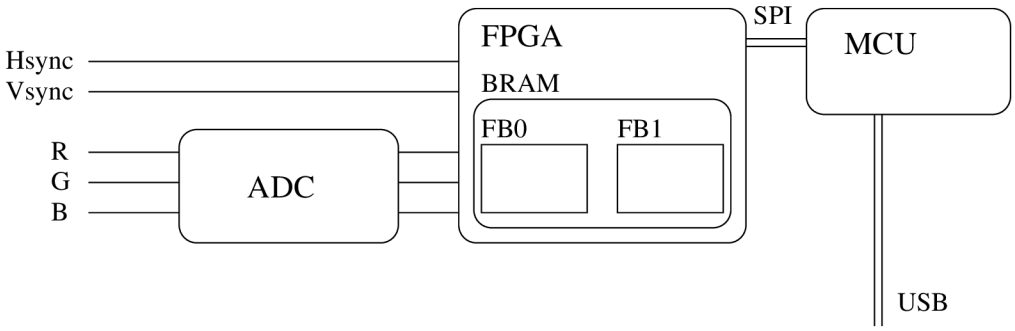
[2]Označováno též jako unicast streaming.

Následně je nutné dopravit navzorkované hodnoty do PC. Jelikož implementace USB v FPGA by byla příliš nákladná, bylo by následně nutné vzorky dopravit do mikrokontroléru, který by zajišťoval implementaci komunikace po USB sběrnici.

Jde o naivní implementaci, postavenou na záměru vytvořit framebuffer, ve kterém by byl obraz přenášený pomocí VGA rozhraní průběžně rekonstruován. Obraz by byl vyčítán pomocí mikrokontroléru jeho vlastní rychlostí, kterou případně dovoluje sběrnice USB. Blokové schéma architektury je možné vidět na obrázku 5.1. Tento způsob implementace je však nerealizovatelný. Hlavním problémem je vyžadovaná velikost paměti RAM pro tento účel. Velikost potřebné paměti by byla rovna $F \cdot 2$, kde $F$ představuje velikost jednoho snímku videa, dvakrát pro realizaci eliminující trhání obrazu, tzv. *screen tearing*, kvůli zápisu vždy do uzamčeného a čtení vždy z volného zásobníku. Tedy $F = 8,64\ Mbit = 1,08\ MB$.

Při volbě varianty, kdy by nedocházelo k sestavování obrazu ve framebufferu a data by byla odesílána přímo pomocí mikrokontroléru ztrácí význam použití paralelního zpracování pomocí hradlového pole FPGA.

Nemalým problémem je rovněž časová náročnost návrhu a výroby několikavrstvé desky plošných spojů pro takový systém. Tato architektura byla proto zavržena.




Obrázek 5.1: Blokové schéma naivní implementace VGA grabberu pomocí framebufferu v FPGA a MCU

### 5.4.2 Architektura s kompresí

Jako další varianta se nabízí možnost využít architektury podobné té předchozí. Tedy hradlové pole FPGA, ke kterému by byly připojeny rychlé externí ADC převodníky. Hradlové pole FPGA by provádělo vzorkování napěťových úrovní barevných kanálů VGA rozhraní. Hradlové pole by zároveň implementovalo kompresní metodu.

Z výše uvedených hodnot vyplývá, že hlavním problémem není přenosová kapacita mezi VGA grabberem a PC. Zavedení vhodné komprese ovšem může odlehčit vytížení klientského PC, na kterém je VGA grabber provozován. Jde především o situaci, kdy je záměrem video obsah přeposílat dále po počítačové síti. Jedná se o případ tzv. video streamingu. V takovém případě je vhodné stream videa komprimovat vhodnou metodou. Mezi běžné kompresní kodéry video obsahu patří např. MJPEG, MPEG-1, MPEG-2, MPEG-4 apod. Z hlediska jednoduchosti se zdá být nejvhodnějším kodérem MJPEG[3], podrobnosti jsou uvedeny v [12].

[3]Tento kompresní formát je často využíván webovými kamerami, které podporují hardwarovou kompresi přímo uvnitř kamery. Do PC poté proudí již komprimovaný tok videa.

Kompresní metoda MJPEG, označovaná také M-JPEG, případně Motion JPEG, je tzv. vnitrosnímkovou kompresní metodou. Jde o metodu s obecně nižší výpočetní náročností. Nižší náročnost metody spočívá v nezávislém komprimování každého obrazového snímku, ovšem za cenu nižší účinnosti komprese. Komprese jednotlivých snímků využívá standardu JPEG.

Závěrem byla tato architektura zavržena z důvodu náročnosti implementace byť té nejjednodušší komprese uvnitř hradlového pole FPGA.

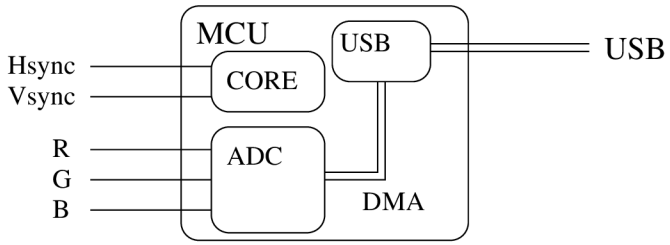
### 5.4.3 MCU s vestavěným ADC převodníkem

Následující variantou je použití mikrokontroléru, který by měl ADC převodník patřičné rychlosti přímo vestavěn na čipu. Takový mikrokontrolér existuje pod označením LPC4370 od firmy *NXP Semiconductors*. Tento mikrokontrolér disponuje mj. 12 bitovým šestikanálovým ADC převodníkem, který zvládá vzorkování frekvencí až 80 $Ms/s$.

Tato realizace spočívá v mikrokontroléru LPC4370, který by byl připojen přímo na rozhraní VGA. Interní šestikanálový ADC převodník by sloužil pro vzorkování barevných intenzit přicházejících po rozhraní. Zároveň by mikrokontrolér zpracovával řídicí vodiče *Hsync* a *Vsync* rozhraní VGA a podle toho odvozoval rozlišení a tedy i časování a potřebnou frekvenci vzorkování. Blokové schéma je znázorněno v obrázku 5.2.

Odesílání obrazových dat do PC by probíhalo pomocí jednoho ze dvou vestavěných USB rozhraní, které pracují v režimu *High Speed* a kterým procesor LPC4370 disponuje.

Uvedený mikrokontrolér disponuje dostatečnými parametry pro zvolený úkol. Tato architektura se zdá být vhodnou cestou pro tvorbu VGA grabberu.




Obrázek 5.2: Blokové schéma architektury VGA grabberu využívající MCU s vestavěním ADC převodníkem

## 5.5 Volba software na straně PC

### 5.5.1 Transparentní ovladač video zařízení

Na straně klientského systému, který realizuje zobrazení dat z VGA grabberu je nutné zajistit vhodný prostředek pro zobrazování dat zpracovaných periferním zařízením. Možným způsobem by zřejmě bylo vytvořit ovladače USB zařízení, které by zpřístupnily VGA grabber jako klasické zařízení pro snímání videa[4]. S takovým zařízením je následně možné pracovat pomocí dostupných aplikací, které umožňují přehrávání proudu videa z periferií, jako je např. video přehrávač *VLC*. Tímto lze video následně přenášet i pomocí počítačové

[4]Zařízení by následně bylo v operačním systému dostupné obdobným způsobem jako např. televizní tuner, či webová kamera.

sítě, takže není nutné obrazový přenos sledovat pomocí vzdáleného sezení či jiného systému pro vzdálené ovládání PC, které na takový způsob využití často nebývají navrženy. Vzhledem k problémům popsaným v 4.6 bylo ovšem od této varianty upuštěno.

### 5.5.2 Klientská aplikace

Další variantou realizace klientského software zůstává možnost realizovat klientskou aplikaci za pomocí knihovny *libusb*. Tato knihovna umožní přistupovat ke sběrnici USB na základě *generických ovladačů* sběrnice. Hardware je poté svázán s klientskou aplikací, tím je následně omezen funkcemi, které tato aplikace poskytuje. V souvislosti s primárním zaměřením na platformu Windows se toto řešení jeví jako lépe realizovatelné.

### 5.5.3 Univerzální ovladač video zařízení UVC

Pravděpodobně nejvýhodnější cestou je navrhnout HW zařízení tak, aby bylo možné jej použít ve spojení s univerzálním ovladačem video zařízení *UVC*. Třída *UVC* definuje jak výzvy a reakce na přenosy dat po sběrnici USB, tak i formát obrazových dat, přenášených rovněž po sběrnici USB. Třída zařízení *UVC* je navržena pro vstupní i výtupní video zařízení. VGA grabber podporující tuto třídu USB zařízení by tedy byl použitelný ve spojení se standardně dodávaným ovladačem, který je součástí operačního systému.

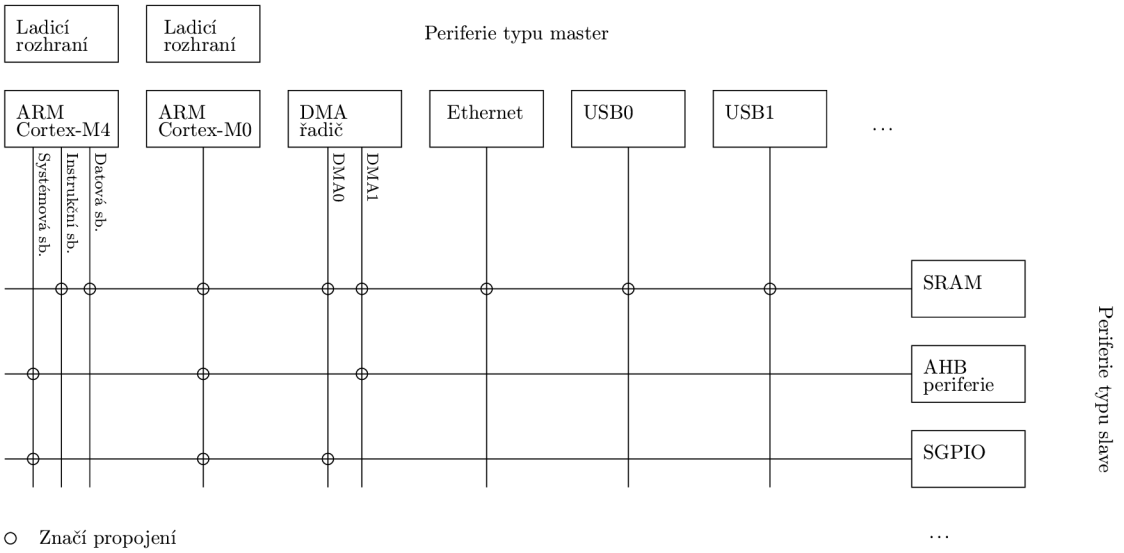
# Kapitola 6

# Mikrokontrolér NXP LPC4370

Tato kapitola čerpá z dokumentů [13] a [15]. Mikrokontrolér *LPC4370* je založen na 32 bitovém jádře *ARM Cortex-M4*. Navíc disponuje dvěma dalšími jádry, *ARM Cortex-M0*, *ARM Cortex-M0* koprocesorem, a systémem řízení periferií.

## 6.1 Architektura procesoru

Jádro *ARM Cortex-M4* zahrnuje tři sběrnice *AHB-Lite*. Jedná se o systémovou, instrukční a datovou. Poslední dvě jmenované umožňují souběžný přístup k instrukcím a zároveň datům z různých portů. *LPC4370* využívá víceúrovňovou matici, pro spojení zmíněných sběrnic a jiných *master* zařízení s periferiemi. Propojení je prováděno efektivním způsobem. Propojení umožňuje přístup k periferiím umístěným na různých *slave* portech matice zároveň z různých zařízení typu *master*. Způsob zapojení je znázorněn na obrázku 6.1.




Obrázek 6.1: Zjednodušená ukázka propojení v matici *AHB*, originál je dostupný z [13]

### 6.1.1 Procesor ARM Cortex-M4

Procesor *ARM Cortex-M4* využívá třístupňové zřetězené linky. Jedná se o procesor s Harwardskou architekturou[1], sběrnice pro data a instrukce jsou odděleny. Procesor disponuje rovněž třetí sběrnicí pro připojení periferií. Procesor podporuje instrukce typu SIMD[2].

Procesor obsahuje jednotku pro přednačítání dat z paměti, která podporuje i tzv. spekulativní větvení. Jádro obsahuje jednotku pro hardwarovou podporu čísel s plovoucí řádovou čárkou.

### 6.1.2 Procesor ARM Cortex-M0

*ARM Cortex-M0* jsou 32 bitové procesory bez zaměření na určitou funkčnost. Tyto procesory využívají rovněž třístupňovou zřetězenou linku ve spojení s redukovanou instrukční sadou. Procesory nabízí vysoký výkon při nízké spotřebě energie.

### 6.1.3 Koprocesor ARM Cortex-M0

Koprocesor *ARM Cortex-M0* umožňuje odlehčit zátěž hlavního jádra *ARM Cortex-M0*. Propojení je zde realizováno pomocí *interprocessor communication protocol*. Koprocesor je situován na stejné vrstvě propojovací matice, jako zmiňovaný *ARM Cortex-M4*.

### 6.1.4 Subsystém ARM Cortex-M0

Subsystém *ARM Cortex-M0* je možné využít k řízení SPI a SGPIO periferií nacházejících se ve vícevrstvé propojovací matici *AHB*, nejen v subsystému *M0*. Subsystém *M0* je od hlavní části propojovací matice *AHB* oddělen přemostěním. V subsystému *M0* se nachází rovněž dva bloky paměti SRAM, které umožňují provozovat *ARM Cortex-M0* plnou rychlostí, nezávisle na hlavní části vícevrstvé propojovací matice *AHB*. Jeden z bloků je typicky využíván pro uložení kódu programu, který je provozován v subsystému *M0*. Druhý blok je využíván pro uložení dat. Tento způsob umožňuje přístup k datům i z jiných částí obvodu, aniž by docházelo k nucenému pozastavování při vyčítání instrukcí z první paměti SRAM.

Komunikační matice subsystému *M0* je provozována na frekvencích nezávislých na kmitočtech hlavní matice *AHB*. Ze subsystému je tudíž možné ovládat příslušející *SGPIO* bez zbytečné latence, která naopak vzniká, pokud dochází k řízení ze systému *M4* pomocí zmíněného přemostění.

Příkladem aplikace subsystému *M0* je např. využití pro snížení příkonu obvodu *LPC4370*. Hlavní propojovací matice *AHB* je provozována v režimu sníženého kmitočtu, zatímco subsystém *M0* může nezávisle hlídat případnou potřebu zvýšení kmitočtu hlavní matice a také zvýšení kmitočtu provést.

### 6.1.5 Meziprocesorová komunikace

Meziprocesorová komunikace mezi systémy *ARM Cortex-M0* a *ARM Cortex-M4* probíhá na základě principu sdílené paměti. Jedná se o sdílenou paměť typu SRAM. Komunikace

[1] Jedná se o architekturu s oddělenými paměťovými prostory pro data a instrukce programu.

[2] Instrukce SIMD (Single Instruction Multiple Data) umožňují využití datového paralelismu a zpracovávat tak jedinou instrukcí více nezávislých dat za pomoci nezávislých výpočetních elementů.

využívá tuto paměť jako poštovní schránku, do které jsou uložena data určená pro příjemce. Odesílatel následně může informovat příjemce vyvoláním přerušení. Přerušení probíhá za asistence řadiče NVIC[3]. Příjemce zprávy může odesílateli oznámit úspěšné vyzvednutí zprávy ze sdílené paměti rovněž za pomoci vyvolání přerušení na straně odesílatele.

## 6.2 Periferie a jednotky obvodu LPC4370

### 6.2.1 NVIC (Nested Vectored Interrupt Controller)

Jednotka NVIC je nedílnou součástí mikrokontroléru *ARM Cortex-M4*. Každý *ARM Cortex-M0* koprocesor má přidělenu svou vlastní jednotku NVIC. Každá z jednotek má 32 vektorů přerušení. Jednotky řídí přerušení vzniklá periferními zařízeními a výjimkami. Jednotky navíc disponují čtyřmi programovatelnými úrovněmi priority. Většina přerušení od periferií je sdílených mezi dvěma jádry *ARM Cortex-M0* a *ARM Cortex-M4*.

Jednotky NVIC příslušející subsystému *ARM Cortex-M4* mají až 53 vektorů přerušení, osm programovatelných úrovní priority, podporují nemaskovatelná přerušení NMI a generování softwarových přerušení.

Jednotka NVIC je spojena s každou periferií pomocí jedné linky přerušení. Periferie mohou specifikovat různé příznaky přerušení.

### 6.2.2 Vstupně/výstupní piny pro všeobecné využití GPIO

Obvod nabízí celkem osm GPIO portů, z nichž každý může obsahovat až 31 pinů. GPIO piny, které nejsou připojeny k řídicím vstupům určité periferie, mohou být řízeny pomocí speciálních registrů. Registry mohou dynamicky měnit jejich využití jako vstupní či výstupní piny. Další registry ovládají úroveň napětí dostupnou na jednotlivých pinech. Piny jsou ve výchozím stavu nastaveny jako vstupní s *pull-up* rezistory.

Registry pro řízení portů GPIO jsou připojeny na *AHB* sběrnici, čímž je dosaženo nízké latence při změnách parametrů portů. Registry jsou adresovatelné po bytech. Celá konfigurace hodnoty portu[4] je zapisovatelná pomocí jediné instrukce. Jedinou instrukcí je možno nastavit libovolný počet hodnot pinů v rámci portu na logickou úroveň *1*, popř. *0*. Až osm pinů lze zvolit pro generování přerušení na základě hrany případně logické úrovně napětí na pinu. Rovněž lze specifikovat až dvě skupiny pinů daného portu, které mohou vyvolávat přerušení při detekci hrany případně dané logické úrovně napětí.

### 6.2.3 SCT (State Configurable Timer)

Subsystém SCT je standardní digitální periferií obvodu *LPC4370*. Umožňuje nejrůznější způsoby časování, modulace, zachytávání vstupu a porovnávání vstupů a výstupu 32-bitových čítačů. Lze jej nakonfigurovat jako jeden 32 bitový čítač, případně jako dva nezávislé 16 bitové čítače. V případě dvou 16 bitových čítačů jsou navíc nezávislé také stavové proměnné, podmínky spuštění, zastavení a hodnoty porovnávacích registrů pro každý 16 bitový čítač. Čítače je možno nakonfigurovat pro čítání nahoru, popř. střídavě nahoru a dolů.

[3] Jedná se o řadič Nested Vectored Interrupt Controller, umožňuje nízkou latenci a efektivní zpracování přerušení.

[4] Nastavení napěťových úrovní v rámci portu na jednotlivých pinech.

### 6.2.4 Řadiče GPDMA

Řadič GPDMA je připojen na propojovací sběrnici *AHB*. Jeho náplní práce je provádět přenos dat autonomně bez asistence dalších řídicích jednotek. Po dokončení přenosu, případně pokud nastane chyba při přenášení dat je možno informovat procesor vyvoláním přerušení. Řadič umožňuje datové transakce:

- z periferie do paměti
- z paměti do periferie
- z periferie do periferie
- z paměti do paměti

Každý DMA tok představuje jednosměrný sériový přenos dat ze zdroje do cíle. Při potřebě obousměrného přenosu je toto nutno řešit ustanovením dvou DMA kanálů. Na obvodu jsou dostupné dva *master* řadiče DMA. Řadič *master1* umožňuje přenosy mezi paměťovými místy a periferiemi. Řadič *master0* umožňuje přenosy pouze v rámci paměti. Celkem je k dispozici osm jednosměrných DMA kanálů.

Řadič umožňuje pracovat v nárazových režimech, kdy je nutno konfigurací řadiče určit velikost nárazových dat, případně jednoduchých *single* režimech. Režim přenosu dat může volit přímo periferie pomocí tzv. *request lines*, kterými řadič disponuje celkem šestnácti. Při přenosech je možno nakonfigurovat inkrementaci zdrojové nebo cílové adresy. Řadič lze nakonfigurovat pro podporu konvencí *little-endian*, případně *big-endian*. Výchozím stavem je nastavena konvence *little-endian*.

Jednotka DMA podporuje rovněž operace *scatter* a *gather*, pomocí vázaných seznamů. Tím je možno přenášet také data, která nezabírají souvislý blok paměti.

### 6.2.5 USB rozhraní

Obvod *LPC4370* disponuje celkem dvěma USB rozhraními. První rozhraní *USB0* je dostupné přímo na vývodech čipu. Další řadič *USB1* je dostupný na vývodech čipu pouze v USB režimu *Full Speed*. Režim *High Speed* je u tohoto rozhraní dostupný pouze pomocí rozhraní ULPI. Rozhraní ULPI definuje standardní rozhraní pro připojování USB čipů, které implementují rozhraní na fyzické úrovni modelu. Tyto čipy jsou obecně označovány jako *PHY*. Zkratka ULPI, UTMI+ Low Pin Interface, vychází z primárního účelu tohoto rozhraní, což je snížení počtu vývodů na pouzdře čipu a tím i snížení výsledné ceny při zachování požadované funkčnosti. Obecné informace o UPLI a důvodu jeho vzniku byly přejaty z [16].

Rozhraní *USB0* umožňuje připojení mikrokontroléru k hostitelskému řadiči[5], popřípadě v hostitelském režimu umožňuje realizovat hostitelský řadič na čipu a tím i připojit jiné USB periferie k čipu. Rovněž je podporován protokol HNP a SRP[6] pro zajištění kompatibility se zařízeními USB OTG[7]. Samotné rozhraní sběrnice USB je kompatibilní se specifikací verze USB 2.0. Rozhraní podporuje automatickou detekci verze sběrnice USB a tím i režimy zpětně kompatibilní s verzí sběrnice USB 2.0. Modul *USB0* obsahuje svůj vlastní DMA řadič, který pracuje nezávisle na výše zmíněných řadičích GPDMA.

[5] Nejčastěji v podobně PC.

[6] Jedná se o Host Negotiation Protocol a Session Request Protocol, které jsou využívány v zařízeních USB OTG při ustanovení sezení.

[7] Sběrnice USB OTG (On The Go) je často využívána pro propojení přenosných zařízení.

Druhým rozhraním sběrnice USB je *USB1*, které umí propojit obvod s hostitelským řadičem a tvářit se tak jako periferní zařízení, případně realizovat hostitelský řadič na čipu a vystupovat tak v roli řadiče sběrnice USB. Při vynechání čipu *PHY* je možné toto rozhraní provozovat maximálně v režimu *Full Speed* a tím degradovat jeho rychlost na 12 $Mbit/s$. Rozhraní obsahuje, stejně jako předchozí *USB0*, vlastní DMA řadič, automatickou detekci verze a zpětnou kompatibilitu.

### 6.2.6 Rozhraní Ethernet

Součástí čipu *LPC4370* je také řadič rozhraní Fast Ethernet. Řadič implementuje podporu DMA, podporu vzdáleného probuzení zařízení z režimu spánku. Rozhraní je schopno pracovat v režimech *Full Duplex*, případně *Half Duplex*, rozhraní implementuje protokol CSMA/CA.

### 6.2.7 ADC převodníky

Převodník *ADCHS* je analogovou periferií, dostupnou na čipu *LPC4370*. Umožňuje snímat úroveň napětí na až šesti kanálech. Převodník umožňuje vzorkovat napětí v rozmezí 0 $V$ až $1,2$ $V$. Rozlišení převodníků je 12 bitů. Převodník je označen jako vysokorychlostní, umožňuje snímkování frekvencí až 80 $Ms/s$ při nejvyšší přesnosti 12 bitů. Součástí převodníku je interní 14 bitový čítač. Konverzi[8] je možno iniciovat na základě změny napětí na vstupním pinu obvodu, případně na základně některého z vnitřních signálů obvodu. Převodník ADC disponuje schopností využití řadiče DMA a vyrovnávací pamětí *FIFO*, která slouží pro tyto případy.

Kromě převodníku *ADCHS* disponuje obvod *LPC4370* rovněž dvěma pomalejšími převodníky označovanými *ADC0*, případně *ADC1*. Tyto převodníky umožňují frekvenci konverze až 400 $ks/s$ při maximálním rozlišení 10 bitů.

### 6.2.8 Sériové rozhraní pro debugging a JTAG

Kromě standardního *JTAG* rozhraní jsou podporovány rovněž funkce pro přímé trasování a ladění kódu programu. *ARM Cortex-M4* podporuje až osm breakpointů a čtyři body hlídání. Koprocesory *ARM Cortex-M0* podporují pouze tzv. *Boundary Scan*[9]. Základní popis této metody je dostupný v [14].

### 6.2.9 Univerzální deska LPC-Link2

S obvodem *LPC4370* souvisí rovněž přípravek *LPC-Link2*, který je navržen přímo výrobcem *NXP*. Přípravek je možné používat jako pomocnou sondu pro ladění vestavěných systémů, ale rovněž je využitelná jako univerzální vývojová deska. Piny obvodu jsou dostupné na celkem šesti konektorech po obvodu desky. USB rozhraní je vyvedeno do konektoru na hraně desky plošných spojů.

#### LPC-Link2 jako pomocná sonda

*LPC-Link2* je samostatnou sondou pro účely ladění firmware a hledání chyb. Sonda umožňuje spolupracovat s vývojovým softwarem různých firem. Podpora je zajištěna ve formě dostup-

[8] Převod analogové hodnoty úrovně napětí na digitální reprezentaci je označován jako konverze.

[9] Jedná se o metodu navrženou především pro testování správného usazení čipů do patic, popřípadě desek tištěných spojů.

ných firmware pro tato zařízení. Firmware se do zařízení nahrává pomocí portu USB, který je vyvedený v podobě konektoru na desce. Následně je možné zařízení propojit s vývojovou deskou, případně systémem obsahující podporovaný mikrokontrolér. Deska umožní přístup k ladicím informacím ve zvoleném vývojovém prostředí na uživatelově PC. Výchozím podporovaným softwarem je *LPCXpresso IDE* od firmy *NXP*, který je dostupný ke stažení pro operační systémy *Linux*, *Windows* a *Mac OS X*.

Výhodou sondy je mj. to, že je modulárně rozšiřitelná. Pomocí modulu od *Embedded Artists* je možné desku využít jako dvoukanálový osciloskop se vzorkovací frekvencí 80 $MHz$, případně jedenáctikanálový logický analyzátor pracující až do frekvence 100 $MHz$. Modul dále umožňuje generování dvou analogových signálů frekvencí 40 $kHz$ a jedenácti digitálních signálů frekvencí 80 $MHz$.

Zajímavým faktem je, že sonda *LPC-Link2* je založena na výše popisovaném mikrokontroléru *LPC4370*.

#### LPC-Link2 jako vývojová deska

Desku *LPC-Link2* je možné rovněž využít jako vývojový kit pro procesor, který sama obsahuje. Tímto procesorem je výše zmiňovaný *LPC4370*. Pro vývojové účely je snadné desku využít jako vývojový kit právě z důvodu snadné změny firmware mikrokontroléru. Deska je ve výchozím stavu nastavena právě pro zavádění programu z USB rozhraní. Pro případ, kdy je nutno zařízení provozovat samostatně, je na desce dostupná 1 $MB$ flash paměť. V této paměti je možné firmware uchovat.

Pro plné využití vč. možnosti ladit provozovaný firmware přímo na této desce je zapotřebí mít tedy desky *LPC-Link2* dvě. Jedna deska poté slouží v režimu ladicí sondy a druhá je provozována v režimu vývojové desky s procesorem *LPC4370*.

## 6.3 Odhad realizovatelnosti pomocí LPC4370

Pro zajištění správných parametrů řešení je nutné zajistit dostatečné kapacity na straně mikrokontroléru. Jde především o to, zda bude možné zařízení sestrojit i za daných podmínek. Především se jedná o detekování času, kdy přichází obrazová data na rozhraní pomocí vodičů nesoucích barevné intenzity. Tuto funkčnost by měla zajišťovat komponenta SCT. Dle aplikační poznámky [17], která realizuje čtení dat z digitální kamery, lze tuto komponentu konfigurovat podobně jako konečný automat, který následně přepíná mezi stavy podle vyvolaných událostí. Vyvolanou událostí může být i např. změna logické hodnoty úrovně napětí na některém ze vstupních pinů obvodu. Dle strany *950* v dokumentu [15], je rovněž možné generovat události za pomoci časovače a tím nastavit požadovanou frekvenci generování událostí. Šestikanálový převodník ADCHS plně dostačuje svou frekvencí vzorkování a rozlišením.

Kanály DMA by měly zajistit požadovanou propustnost a realizaci přenosu dat mezi převodníkem ADCHS a rozhraním USB, které bude využito pro zasílání navzorkovaných dat do PC. Na straně uživatelského PC bude spuštěna aplikace realizující zobrazování, případně síťový streaming dat obdržených od zařízení.

Procesorová část čipu bude zajišťovat pouze výpočet potřebné frekvence pro vzorkování, konfiguraci SCT, konfiguraci DMA transakcí, reakce na jisté typy požadavků na sběrnici USB a ošetření případných výjimek.

# Kapitola 7

# Návrh a realizace firmware

## 7.1 Koncept firmware

### 7.1.1 Komunikace s počítačem

Pro komunikaci s klientským počítačem je zvolena sběrnice USB. Tato bude sloužit zároveň jako zdroj napájecího napětí. Na mikroprocesoru LPC4370 je k dispozici *USBD ROM Stack*, ovladač sběrnice USB, který odstiňuje nezbytné náležitosti komunikace po rozhraní USB pro snazší realizaci. *USBD ROM Stack* tak implementuje rozhraní[1] pro přístup k registrům ovládajícím subsystém USB na čipu LPC4370. Ze strany hostitelského řadiče zajišťuje bezproblémovou enumeraci zařízení dle dodaných deskriptorů USB. Takto je možné v nově tvořeném firmware explicitně uvádět pouze ty části, které je nutné doplnit oproti základním funkcím, které musí podporovat každé USB zařízení.

Ovladač *USBD ROM Stack* podporuje několik základních tříd zařízení USB, jako jsou HID, MSC a další. Zvolená třída zařízení *UVC* zde implementována není. Proto je nutné ji implementovat vlastní programovou vrstvou.

Samotná obrazová data je vhodné přenášet pomocí mechanismu DMA.

### 7.1.2 Získávání obrazových dat z rozhraní

Na přípravku *FITkit* jsou signály rozhraní VGA dostupné rovněž v digitální podobě, jak uvádí sekce 3.1. Tímto je možné dosáhnout jistého zjednodušení a zároveň snížení nároků na rychlost mikrokontroléru. Digitální signály je možné přivést na GPIO porty čipu LPC4370, které budou nastaveny jako vstupní, a průběžným načítáním hodnot z registrů příslušejících těmto portům lze získávat obrazová data. Data budou ukládána do paměti, odkud je převezme část implementující komunikaci s hostitelským řadičem USB.

Synchronizační signály *Hsync* a *Vsync* je možné obsloužit za pomoci přerušení citlivých na náběžnou hranu u příslušejícího pinu mikrokontroléru.

### 7.1.3 Synchronizace mezi USB a VGA

Pro správnou funkčnost je nutné zajistit synchronizaci mezi částí, která implementuje komunikaci s hostitelským řadičem a částí, která získává obrazová data z rozhraní VGA. Problém je zejména v odlišných časových intervalech, které se vyskytují na těchto stranách.

[1] Kromě využití vrstvy *USBD ROM Stack* je možné k registrům USB subsystému přistupovat přímo, pomocí instrukcí zápisu do paměti a adres registrů definovaných v [15].

Nejjednodušší a zároveň realizovatelnou možností je rekonstruovat celý obraz rozhraní VGA v tzv. *framebufferu* v paměti SRAM. Obraz bude nutné udržovat v paměti v podvzorkované formě. Komunikační část následně může obraz číst z vyhrazené paměti v libovolném okamžiku, bez nutnosti vzájemné synchronizace obou částí.

### 7.1.4 Mapování úkolů na dostupná jádra

Jak uvádí kapitola 6, mikrokontrolér NXP LPC4370 disponuje kromě hlavního jádra Cortex-M4, také dvěma dalšími jádry Cortex-M0. Celý systém VGA grabberu můžeme obecně rozlišit na dvě základní úlohy. První je získávání obrazových dat a druhou je propagace těchto dat po zvoleném rozhraní.

### 7.1.5 Získávání obrazových dat

Část zabývající se získáváním obrazových dat a rekonstrukcí obrazu v paměti by zřejmě bylo vhodné přiřadit jádru s vyšším výkonem. Proto zde volíme Cortex-M4, který, byť provozován na stejné taktovací frekvenci, obsahuje některé DSP instrukce. To jej dělá vhodnějším kandidátem na tuto pozici.

### 7.1.6 Odesílání dat po rozhraní

Oproti tomu na část zabývající se propagací nasbíraných dat směrem k hostitelskému řadiči USB není díky využití DMA přenosů kladen tak velký důraz na instrukční sadu. Toto jádro se bude převážně zabývat nastavováním přenosových deskriptorů pro řadič DMA, což představuje zápisy do paměti na předem vyhrazená místa.

## 7.2 Realizace firmware

Implementace tvoří celkem dva projekty v prostředí LPCXpresso. Oba projekty jsou navzájem svázány, což je dáno jejich nastavením, tvoří tedy dohromady jediný firmware pro VGA grabber. První projekt se nazývá `vga_grabber_m4_core` a jde o tzv. *master* projekt, tedy ten, který je spouštěn na jádře Cortex-M4 a je zodpovědný za inicializaci a spuštění dalšího, tzv. *slave* projektu na druhém procesorovém jádře. Druhý projekt je v prostředí označen názvem `vga_grabber_m0_core`. Následující sekce a samotná implementace čerpá informace ze zdrojů [24] a [23].

Tímto jsou tvořeny celkem dva paralelní programy, z nichž každý běží na vyhrazeném procesorovém jádře.

### 7.2.1 Implementace třídy UVC na sběrnici USB

Prvním krokem je vytvořit pomocí zvolené vývojové desky LPC-Link2 zařízení, které po zasunutí do USB konektoru počítače bude v operačním systému enumerováno jako webová kamera, případně jako zařízení pro digitalizaci videa. Takové zařízení by následně mělo být viditelné v podporovaných video přehrávačích[2].

Dalším krokem je vyhradit na takovém zařízení část paměti, která by představovala *framebuffer* a implementovat reakce na požadavky o zahájení a ukončení přenosu videa

[2] Mezi podporované přehrávače patří např. VLC

směrem k hostitelskému řadiči do počítače. Tak by se po výběru našeho zařízení v podporovaném video přehrávači a spuštění tlačítkem *play* měl objevit statický obraz, který by byl odrazem framebufferu z paměti mikrokontroléru.

### 7.2.2 Inicializace

Vstupním místem projektu `vga_grabber_m0_core` je standardně funkce `main()`. Tato funkce volá `setup_m0_core()`, zajišťující inicializaci příslušejícího jádra Cortex-M0, na kterém je spouštěna, tj. obsahuje volání pro nastavení specifických pro použitou vývojovou desku LPC-Link2, inicializaci kontextu aplikace a nastavení související se sběrnicí USB. Tyto funkce sloužící pro nastavení jsou definovány v souboru `init.c`.

#### Kontext aplikace

Kontext aplikace označuje datovou strukturu, která je předávána většině funkcí v aplikaci. Struktura obsahuje důležité ukazatele a stavové proměnné. Tímto způsobem se lze vyhnout velkému množství globálních proměnných a problémům s tím spojených. Kvůli možnosti předávat ukazatel na strukturu s kontextem aplikace také do obsluh přerušení je definován jako globální proměnná `app_context_t g_ctx` s tím, že zdrojový kód aplikace, kromě obsluh přerušení, důsledně dodržuje zásadu nepřistupovat ke kontextu aplikace pomocí globální proměnné, ale pouze pomoci předávaného ukazatele na tuto strukturu. To umožňuje lepší přenositelnost implementovaných funkcí. Inicializace kontextu probíhá ve funkci `init_context()`, mj. zde probíhá také počáteční vynulování framebufferu na konstantní černou barvu.

#### Inicializace USB rozhraní

Nastavení související s rozhraním USB zajišťuje funkce `setup_usb()`. Jedná se především o inicializaci ovladače *USBD ROM Stack*, což představuje nastavení ukazatelů na datová pole obsahující USB deskriptory. Tyto deskriptory jsou uloženy v souboru `usb_uvc_desc.c`. Zde lze pouze dodat, že pro správnou funkčnost je nutné konfigurační deskriptor zařízení definovat bez kvalifikátoru `const`, na rozdíl od ostatních deskriptorů zde ovladač *USBD ROM Stack* potřebuje mít možnost zápisu do jednobytového pole *bDescriptorType*. Deskriptory jsou definovány s použitím makra `ALIGN(4)`, což způsobí zarovnání v paměti na 32-bitovou hranici.

Dále se zde nachází vyhrazení paměti pro uložení celkem osmi DMA deskriptorů, registrace obslužné funkce `usb_ep0_handler()` pro konfigurační koncový bod 0 a registrace obslužné funkce `usb_ep1_handler()`, která zpracovává události spjaté s koncovým bodem 1.

### 7.2.3 Deskriptory USB

Důležitou součástí jsou deskriptory, které popisují vlastnosti a možnosti jednotlivých USB zařízení. Tyto deskriptory jsou uloženy v paměti zařízení, které je musí na popud hostitelského řadiče vydat pomocí řídicích přenosů, které se odehrávají přes konfigurační koncový bod. Odesílání je v režii ovladače *USBD ROM Stack*, na návrháři ovšem zůstává úkol tvorby těchto deskriptorů. Deskriptory jsou definovány v souboru `usb_uvc_desc.c`.

### Deskriptor zařízení

Z pohledu USB třídy UVC je v datových polích tohoto deskriptoru důležitá identifikace třídy `bDeviceClass`, tj. `0xEF`, identifikace podtřídy `bDeviceSubClass`, tj. `0x02` a protokolu `b DeviceProtocol`, tj. `0x01`. Tato nastavení zajistí, že ovladač na straně počítače bude respektovat nastavení v části konfiguračního deskriptoru, která se nazývá *Interface Association Descriptor*[3].

### Konfigurační deskriptor

V konfiguračním deskriptoru je soustředěna většina informací o zařízení, které má splňovat specifikaci UVC. Konfigurační deskriptor je dělen do kratších částí. První z nich je klasický začátek konfiguračního deskriptoru, který musí být přítomen na všech zařízeních splňujících standard USB. Další deskriptory, které byly zvoleny pro popis tohoto zařízení jsou popsány v následujících odstavcích v pořadí, v jakém jsou uspořádány.

**Interface Association Descriptor** Tímto deskriptorem identifikujeme především počet rozhraní a index prvního rozhraní. Počet rozhraní je zde stanoven na celkem 2. První rozhraní představuje způsob ovládání zařízení, jako např. pro kamery možnosti volby jasu, kontrastu apod. Tato podčást zároveň obsahuje informace o formátu obrazových dat. Další rozhraní sdružuje prostředky spojené se samotným přenášením obrazových dat.

**VideoControl Interface Descriptor** VideoControl Interface Descriptor je označení pro deskriptor, který popisuje rozhraní umožňující nastavení video zařízení a také způsob získávání stavových informací zpět ze zařízení. Navrhované řešení nevyužívá ani jednu z těchto možností, proto zde uvádíme nulový počet koncových bodů, které k tomuto rozhraní přísluší.

**Input Terminal Descriptor** Uvedený deskriptor popisuje část zařízení, které je schopno získávat obrazová data. Obecně existuje varianta *kompozitního* vstupu a kamery. Vzhledem k tomu, že s enumerací zařízení, které mělo na tomto místě uveden jako zdroj videa kompozitní vstup, zde používáme deskriptor popisující jako zdroj videa kameru. Důležité je především index tohoto terminálu, který je označen číslem 1.

**Processing Unit Descriptor** Tento deskriptor popisuje část zařízení, které se stará o úpravu vstupního obrazu. V našem případě není tato jednotka využívána, protože zdrojem obrazových dat je ve skutečnosti vstup VGA, tudíž není nutné implementovat prostředky pro vyvážení bílé barvy, případně úpravu jasu apod. Všechny bity představující možnosti této jednotky jsou zde nulovány.

Vstupem této jednotky je další jednotka s indexem 1, což v našem případě představuje kameru, kterou popisuje výše uvedený deskriptor. Tato jednotka je označena indexem 2.

**Extension Unit Descriptor** Extension Unit Descriptor popisuje jednotku, která může implementovat reakce na požadavky specifické pro daný výrobek. Počet vstupů je volitelný. Zde volíme jediný vstup. Index této jednotky je 3. Vstupem jednotky je jednotka *Processing Unit*, tedy jednotka s indexem 2.

[3] *Interface Association Descriptor* je dle specifikace UVC povinný pro zařízení s jedním a více streamy videa.

**Output Terminal Descriptor** Poslední jednotkou v celém řetězci je v našem případě jednotka *Output Terminal*. Tato jednotka představuje místo výstupu video signálu ze zařízení. V našem případě je vstupem této jednotky *Extension Unit*, tj. jako index vstupu je zde uvedeno číslo 3.

**VideoStreaming Interface Descriptor - Zero Bandwidth** Tento deskriptor uvádí druhou část konfiguračního deskriptoru. Popisuje část, která se v našem případě zabývá přenosem obrazových dat směrem k hostitelskému řadiči. Video rozhraní, které je zvoleno implicitně po enumeraci, je stanoveno na tzv. *zero bandwidth*, které nedisponuje koncovými body a tudíž negeneruje přenos obrazových dat. Tímto je zajištěn způsob, jak počítač sděluje zařízení, že nemá zájem o obrazová data a tudíž není nutné těmito daty vytěžovat sběrnici. Proto je v tomto deskriptoru uveden nulový počet koncových bodů příslušejících k tomuto rozhraní.

**VideoStreaming Format Descriptor** Tento deskriptor určuje formát obrazových dat. Pro naši potřebu je zde nastaven nekomprimovaný formát, který představuje přenos pixelových dat v podobě obrazových snímků. Některá zařízení zde mohou uvádět různé typy kompresních formátů, jako je např. formát MJPEG. V případě nekomprimovaného formátu dat je zde dále uváděno tzv. *GUID*, tedy identifikátor upřesňující formát pixelových dat v rámci jednoho obrazového snímku. Zde specifikujeme formát *RGB16*, tedy formát, který představuje celkem dva byty dat pro jeden pixel. Formát dat je v pořadí R, G, B ukládán postupně po pěti, šesti a pěti bitech. Tento formát představuje rozšíření operačního systému Windows, tedy zařízení využívající tento formát není implicitně kompatibilní se standardem UVC. Pro naši potřebu je ovšem nezbytné zvolit formát, který nevyžaduje dodatečnou konverzi barevných složek do odlišného barevného modelu, jako je tomu u implicitně podporovaných formátů typu *YUYV*.

Tento deskriptor zároveň nese informace o poměru stran obrazového snímku videa, zde specifikujeme klasický poměr stran 4:3.

**Uncompressed VideoStreaming Frame Descriptor** Následující deskriptor popisuje velikost jednoho snímku obrazových dat v pixelech. Zde s ohledem na paměťové nároky volíme velikost 256 x 192 pixelů. Další pole tohoto deskriptoru popisují především snímkovou frekvenci, která je zde uváděna v délce trvání jednoho snímku v jednotkách 100 ns. Např. pro snímkovou frekvenci 30 fps se zde uvádí hodnota 333 333. Pro naše potřeby postačí frekvence snímků 15 fps, což představuje hodnotu dvakrát větší.

**VideoStreaming Interface Descriptor - Alternate Setting 1** Dále následuje deskriptor popisující rozhraní pro přenos obrazových dat, které je zvoleno požadavkem ovladačem v případě, že si uživatel vyžádá přenos obrazu z našeho zařízení. Důležitou změnou oproti předchozímu deskriptoru pro *zero bandwidth* je počet koncových bodů. Uvádíme zde jeden koncový bod. Následuje deskriptor samotného koncového bodu pro *izochronní* přenosy a ukončení konfiguračního deskriptoru nulovým bytem.

### Řetězcový deskriptor

Posledním uváděným deskriptorem je řetězcový deskriptor, který obsahuje řetězce, na které se lze odkazovat pomocí čísel indexů z různých míst předešlých deskriptorů. Jako minimální konfiguraci zde uvádíme popis výrobce a jméno zařízení.

### 7.2.4 Odchycení chybových stavů

Knihovny ovladačů, případně další volané funkce nabízí informaci o korektním dokončení pomocí návratových hodnot. U vestavěného systému však není jednoduché signalizovat chybová hlášení v případě, že k systému není připojena sonda debuggeru. Proto se v souborech `error.c` a `error.h` nachází jednoduchá funkce `error_exit()`, která je svým použitím podobná příkazu `return` např. v jazyce C. Po předání chybového kódu zmíněná funkce cyklí v nekonečné smyčce, zároveň však signalizuje číslo chybového kódu počtem krátkých záblesků LED diody, která je dostupná na vývojové desce LPC-Link2. Tak je možné identifikovat přibližné místo vzniku chyby v programu i při běžném provozu, kdy k vývojové desce není připojena programovací sonda.

### 7.2.5 Příkazy specifické pro třídu UVC

Jak již bylo uvedeno v odstavci 7.2.2, události vzniklé na koncových bodech jsou obsluhovány funkcemi vyhrazenými pro tento účel. Pro obsluhu událostí, které se týkají koncového bodu je zvolena funkce `usb_ep0_handler()`. Tato funkce na základě testování polí obdrženého *setup* paketu určuje, zda se jedná o požadavek třídy USB. Pokud ano, jsou obsluhovány výhradně události `USB_EVT_SETUP` a `USB_EVT_OUT`.

První ze dvou zmíněných představuje přijetí *setup* paketu. V takovém případě je volána funkce `setup_event()`, která zpracuje přijatý paket a připraví na něj odpověď. V tomto případě je funkce bezprostředně následována funkcí `reply_request()`, která se postará o odeslání vyžádané odpovědi.

V případě události `USB_EVT_OUT` je volána funkce `out_event()` pro zpracování přijatých dat. Následuje rovněž volání funkce `reply_request()` pro odeslání odpovědi.

Je důležité věnovat pozornost směru prováděných přenosů. V USB je směr přenášení dat pojmenován výhradně podle směru přenosu z pohledu hostitelského řadiče. Proto událost `USB_EVT_OUT` představuje vlastně přijetí dat. Zároveň je potřeba respektovat fakt, že sběrnice USB 2.0 funguje na bázi vyzývání, tedy data, která jsou vyžádána ve fázi *setup* je nutné doručit až v následující fázi *in*. Tuto funkcionalitu nabízí již ovladač *USBD ROM Stack*. U transakce opačného směru je tomu ale naopak. Data oznamovaná ve fázi *setup* je nutno přečíst až ve fázi *out*, proto je zpracování požadavků rozděleno na tyto dvě události.

#### Minimální požadavky

Pro bezproblémové zprovoznění postačuje implementovat odezvu na požadavky uvedené v následujícím výčtu:

- GET_DEF
- GET_MIN
- GET_CUR
- SET_CUR

Tato minimální množina požadavků byla odpozorována za pomoci ladicích výpisů z programu v zařízení. První tři výzvy v podstatě implementují stejnou činnost. Poslední požadavek lze snadno vyřešit odpovědí typu *acknowledge*.

#### Zpracování událostí SETUP

Ve funkci `setup_event()`, která se nachází v souboru `usb_uvc.c`, jsou implementovány reakce ve fázi *setup*, které reagují na výzvy třídy UVC. Dle typu požadavku, který je identifikován na základě pole `bRequest` ve přijatém *setup* paketu je spuštěna příslušející obslužná funkce.

Požadavky typu *GET* jsou vyřizovány funkcemi `uvc_get_def_se()` a `uvc_get_cur_se()`. Tyto obslužné funkce způsobí odeslání datové struktury *Video Probe and Commit Control*, která obsahuje specifické informace o nově spouštěném, případně probíhajícím video přenosu.

Požadavek typu *SET* je, co se týče *setup* fáze přenosu, ošetřen funkcí `uvc_set_cur_se()`. Příkaz spočívá v naslouchání na příchozí data směrem od klientského počítače, tj. zde nastavením volného datového bufferu pro zápis přijatých dat na konfiguračním koncovém bodě.

#### Zpracování událostí OUT

Funkce `out_event()`, nacházející se v `usb_uvc.c`, implementuje reakce v *datové* fázi transakcí na sběrnici USB. Typ požadavku je možno identifikovat opět na základě pole `bRequest` ve dříve obdrženém *setup* paketu, jelikož tento zůstává uchován po celou dobu transakce za asistence ovladače *USBD ROM Stack*.

V této fázi ošetřujeme pouze požadavek typu *SET*, protože se jedná o fázi příjmu dat směrem od hostitelského řadiče, čili požadavky typu *GET* touto fází neprostupují. Požadavek `SET_CUR` obsluhuje funkce `uvc_set_cur_ds()`. Protože naše zařízení implementuje pouze základní podmnožinu operací, která je nutná pro bezproblémovou enumeraci zařízení a pouhé přeposílání video obsahu bez možnosti dalších úprav a bez možnosti výběru několik kvalit přenášeného videa, je postačující pouhé potvrzení přijatých dat pomocí paketu *acknowledge*. Není nutné data parsovat, případně ukládat. Ovladač v režii operačního systému na straně počítače je schopen zjistit, jakými funkcemi zařízení disponuje pomocí dvojic příkazů *SET* a *GET*. Po příkazu *SET_CUR* následuje obvykle *GET_CUR*, kterým ovladač UVC ověří, zda parametry, které si během první transakce vyžádal byly pro zařízení splnitelné.

### 7.2.6 Přenos obrazových dat

#### Zahájení a ukončení přenosu

Zařízení standardu UVC umožňuje přenos obrazových dat pomocí přenosů typu *bulk* a *izochronních* přenosů. V případě, že na straně operačního systému nedochází ke zobrazení, příp. jinému zpracování dat, je zbytečné je přenášet po sběrnici do počítače a zaplňovat tak její pásmo pro jiná zařízení, která jej mohou potřebovat. Z tohoto důvodu disponuje naše zařízení dvěma alternativními konfiguracemi rozhraní pro přenos videa. První konfigurace rozhraní je označována *zero bandwidth* a je volena implicitně po zasunutí zařízení do USB konektoru a úspěšné enumeraci. Tato konfigurace nezahrnuje koncové body, proto ani nemůže generovat datový tok po sběrnici. V případě, že chce na straně operačního systému některá aplikace přistupovat k UVC zařízení, ovladač UVC zajistí přepnutí aktuálního rozhraní na takovou konfiguraci, která již koncový bod obsahuje. Tím dojde k navázání *roury* a přenos obrazových dat může začít.

Volba konfigurace rozhraní se odehrává pomocí konfiguračních příkazů *set interface*. Pro možnost odchycení události příchodu takového příkazu nabízí zvolený ovladač *USBD ROM*

*Stack* možnost zaregistrovat si obslužnou funkci, která je spouštěna vždy po příchodu takové události. Příslušející obslužná funkce je potom v představované implementaci definována v souboru `usb_handlers.c` pod názvem `usb_interface_event()`. Ve funkci je pomocí pole `wIndex`, které je součástí *setup* paketu zjištěn index rozhraní, na které se vztahuje tento požadavek. Zpracováváme výlučně pakety týkající se rozhraní s indexem 1, což představuje rozhraní ke komunikaci obrazových dat. Pole *setup* paketu s názvem `wValue` představuje index alternativní konfigurace, která je na tomto rozhraní právě volena. Pokud je toto pole nastaveno na 1, znamená to zahájení přenosu. Zahájení přenosu odpovídá chvíli, kdy uživatel ve svém přehrávači stiskne tlačítko *play*, tedy na úrovni ovladače volání `open()`. Pokud je pole rovno hodnotě 0, znamená to návrat zpět k alternativní konfiguraci *zero bandwidth*, což odpovídá chvíli, kdy uživatel ve svém přehrávači stiskne tlačítko *stop*, příp. aplikaci ukončí. Na úrovni ovladače jde o volání typu `release()`. Na tomto místě je rovněž vhodné provést opětovnou inicializaci některých stavových proměnných tak, aby byly opět připraveny pro následující spuštění video proudu.

Samotné spuštění, příp. ukončení přenosu obrazových dat probíhá pomocí volání ovladače *USBD ROM Stack* s názvem `EnableEvent()`. Pomocí parametrů předaných tomuto volání lze povolit, příp. pozastavit spouštění obslužné funkce při události *start of frame*. Tato událost nastává na sběrnicích USB vždy po příchodu paketu *SOF*, což při USB *high-speed* představuje spuštění každých 125 $\mu s$. Pro *full-speed* zařízení chodí *SOF* paket každou 1 $ms$. Naše zařízení se omezí na podporu standardu USB *high-speed*, proto můžeme počítat s hodnotou 125 $\mu s$, což dělá z události *start of frame* vhodného kandidáta na umístění kódu, který bude plánovat přenosy DMA.

### Plánování DMA přenosů

Pro naši aplikaci je vhodné využít řadiče USB DMA přenosů, který je ve zvoleném mikrokontroléru LPC4370 standardně k dispozici. Tím celkově snížíme náročnost komunikační části aplikace.

Přenos na sběrnici USB pomocí mechanizmu DMA jsou plánovány na základě tzv. *DMA deskriptorů*. Tyto deskriptory popisují délku dat a adresu do paměti obsahující data. Obecně DMA přenos na USB můžeme rozdělit do dvou fází. První fáze zahrnuje zařazení DMA deskriptoru do fronty reprezentující data, která čekají na odeslání. Další fáze nastává v okamžiku, kdy si hostitelský řadič vyžádá data pro daný koncový bod, na který jsou DMA přenosy vázány. Teprve v okamžiku vyžádání pomocí paketu *IN* dojde v datové fázi k odeslání. Pokud je fronta okamžikem přijetí vyzývacího paketu prázdná, je přenesena nulová délka dat. Tuto funkcionalitu je DMA řadič schopen vykonávat autonomně bez zásahu firmware. Jediným úkolem, který tak zbývá na firmware je doplňování fronty čekajících deskriptorů ve vhodný okamžik.

Navržené zařízení používá pro přenos obrazových dat *izochronních* přenosů. Počet přenosů je udáván v deskriptorech zařízení a je pevně spjat s událostí *start of frame*. V našem případě půjde o jednu izochronní transakci délky 512 $B$ na jednu událost *start of frame*.

Standardně lze frontu DMA deskriptorů doplňovat v libovolný okamžik, ovšem pouze v případě, že nedochází ke změnám těch datových struktur, které jsou právě řadičem používány. To vyžaduje poměrně komplikovanou synchronizaci. Proto se omezíme na jednodušší případ, kdy pomocí události *start of frame* budeme frontu plnit osmi deskriptory vždy každých devět událostí *start of frame*. Takto máme jistotu, že nedojde k přerušení právě probíhajících transakcí kvůli tomu, že by se nestihly dokončit včas[4].

[4]Izochronní přenosy na sběrnici USB garantují propustnost právě svou návazností na událost *SOF*.

Zajímavým zjednodušením by bylo naplnit frontu DMA deskriptorů vždy pro celý jeden obrazový snímek. Toto bohužel není realizovatelné z důvodu omezené velikosti rychlé USB paměti, ve které se fronta deskriptorů nachází, a tím i omezeného počtu deskriptorů, které se v daný okamžik mohou nacházet ve frontě. Řadič DMA pro přenosy na USB na čipu LPC4370 sice dovoluje jedním deskriptorem popsat až 20 $kB$ dat, ovšem pouze pro přenosy typu *bulk*. Pro *izochronní* přenosy je podmínkou dokončení celého jednoho DMA deskriptoru v rámci jedné USB transakce. Tím jsme nuceni DMA deskriptory omezit tak, že maximální délka dat, kterou popisují je 512 $B$.

Samotné plánování DMA přenosů provádí funkce `uvc_queue_dma()`, kde je implementováno doplňování fronty DMA deskriptorů. Počet odeslaných bytů je udržován v kontextu aplikace. Funkce `uvc_queue_dma()` navrací chybové kódy definované v souboru `errors.h`. Pokud funkce vrátí chybový kód `E_LAST_TRANSACTION`, potom byla právě zařazena do fronty poslední transakce v rámci jednoho obrazového snímku. Pro volajícího je toto znamením, že má uvést stavové proměnné do výchozího stavu a zároveň začít čekat na vhodný okamžik k zařazení dalšího obrazového snímku tak, aby byla dodržena hodnota *fps* uvedená v konfiguračním deskriptoru.

### Paměť obrazu

Z důvodů popsaných v podsekci 7.1.3 je nezbytné udržovat celý obraz v paměti v tzv. *framebufferu*. Velikost *framebufferu* se odvádí od velikosti jednoho obrazového snímku. Pokud zvážíme, že na realizaci takové paměti použijeme největší souvislý blok paměti, který je na mikrokontroléru LPC4370 k dispozici, dostáváme maximální velikost obrazové paměti přibližně 100 $kB$. Jednoduchým výpočtem lze odvodit přibližnou maximální velikost obrazu v pixelech.

Pokud máme k dispozici 100 $kB$ paměti RAM, budeme předpokládat barevný prostor RGB16, tj 2 $B$ na jeden obrazový bod, tím dostáváme celkem 50 000 pixelů. Pokud uvažujeme obraz formátu 4:3, lze obraz chápat jako plochu celkem dvanácti stejně velikých čtverců. Určíme počet pixelů na jeden takový čtverec, tj. 50000 / 12 $\doteq$ 4167. Nyní zjistíme počet pixelů příslušejících na jednu stranu čtverce, tj. $\sqrt{4167} \doteq 64$ pixelů. Nyní vynásobením konstantou 4, případně 3 zjistíme velikost jednotlivých hran obrazové paměti, tj. $64 \cdot 4 = 256$ pixelů a $64 \cdot 3 = 192$ pixelů. Tak dostáváme maximální velikost snímkového bufferu, který se vejde do paměti, tj. 256 x 192 pixelů[5].

**Vyhrazení paměti** Protože k implementaci využíváme vícejaderný mikroprocesor a protože budeme chtít více jader využít, umístění obrazového bufferu na fixní adresu v paměti částečně usnadní práci. Kompilátor, který je součástí prostředí LPCXpresso, bohužel toto nepodporuje. Proto musíme jít na úroveň *linkovacích* skriptů.

Prvním krokem je vytvořit samostatnou oblast v paměti `RAMLoc128` tím, že odebereme přibližně 100 $kB$ část paměti a pojmenujeme ji např. `FB_RAM`. Následně modifikujeme *linkovací* skript tak, že vytvoříme novou sekci `.FRAMEBUFFFER` a přiřadíme ji pomocí symbolu `> FB_RAM` výše vyhrazené oblasti v paměti RAM.

Samotný framebuffer je následně definován a deklarován ve funkci `init_context()` jako statická proměnná pole patřičné velikosti a za pomoci atributu `__attribute__((section(".FRAMEBUFFER")))` se odkazujeme do vyhrazené části paměti. V našem případě je *framebuffer* umístěn na fixní adresy začínající hodnotou `0x10007000`.

[5]Při výpočtu jsme zanedbali velikost tzv. *Payload Headers*, což je 2 $B$ na každých 510 $B$.

Adresa je definována makrem `FB_MEM_ADDR`. V dalším jádře mikrokontroléru je pak možné pomocí této fixní adresy přistupovat k obrazové paměti.

**UVC Payload Header** Součástí obrazových dat přenášených do počítače jsou také tzv. *UVC Payload Headers*. Jde o hlavičky uvozující každý obrazový paket. V našem případě USB transakcí, z nichž každá zahrnuje datový paket dlouhý 512 $B$, tvoří z přenášených dat 2 $B$. Jednou USB *izochronní* transakcí se tedy přenese 510 $B$ užitečných dat.

V rámci minimalizace rozhraní je hlavička zvolena ve své minimální variantě. První byte hlavičky určuje její délku včetně tohoto bytu. Další byte je významný z pohledu synchronizace obrazových snímků.

Pro každý snímek je potřeba invertovat bit `FID`, který označuje pořadové číslo snímku. Tento bit slouží také k synchronizaci v případě ztráty posledního datového paketu. Poslední paket obsahuje nastavený bit `EOF`, který značí konec daného snímku. Implementace spočívá v jednoduché úpravě patřičných bytů před každým zařazením příslušejícího deskriptoru DMA přenosu, což je součástí funkce `uvc_queue_dma()`.

### 7.2.7 Chyby ovladače USBD ROM Stack

Ovladač *USBD ROM Stack* je z důvodu šetření paměti implementován přímo na čipu LPC4370. Jak již název ovladače napovídá, jedná se o knihovnu, která je uložena v nezměnitelné paměti ROM. Vývojové prostředí následně počítá s tímto faktem a všechna volání na funkce tohoto ovladače přesměruje na patřičnou adresu do implementující paměti ROM. Takto není nutné knihovnu přibalovat k výslednému firmware. Drobným problémem ovšem může být snížená schopnost oprav chyb v této knihovně.

Během implementace popisovaného zařízení se ve verzi *USBD ROM Stack*, která je dostupná na čipu LPC4370 nacházely dvě chyby, které měly potenciál ovlivnit správnou funkčnost zařízení.

#### Race condition u koncového bodu 0

Dle [22] je v kódu ovladače *USBD ROM Stack* riziko vzniku tzv. *race condition*. Jedná se o problém, kdy výchozí obslužná funkce pro konfigurační koncový bod připravuje DMA řadič pro přenos poté, co hostitelský řadič odeslal po USB paket *NAK*.

Kvůli zmíněné chybě existuje riziko, že další paket *NAK* dorazí dříve, než byl připraven přenos v důsledku prvního *NAK*.

Řešení spočívá ve zjištění adresy původní obslužné funkce, změně obslužné funkce pro konfigurační koncový bod na funkci definovanou v rámci uživatelského firmware. Tato funkce využije původní funkci pro obsluhu událostí, ale zároveň nedovolí vznik *race condition*. Implementace opravy je součástí souborů `usb_ep0_patch.c` a `usb_ep0_patch.h`. Zdrojový program implementace byl přejat z [22], kde je možné dohledat více podrobností o tomto problému.

#### Nulování polí DMA deskriptoru

DMA deskriptory jsou uspořádány ve frontě. Fronta sestává z tzv. *hlavy* fronty a zbývajících deskriptorů, které čekají na vyřízení. *Hlava* fronty je dostupná vždy a je součástí pole, které obsahuje právě jednu *hlavu* pro každý koncový bod.

Tato chyba se projevuje pouze u izochronních přenosů při použití DMA. Chyba spočívá v nesprávném přenastavení některých polí této *hlavy*, které se děje po každém přijetí konfiguračního paketu *set interface*. Proto je nutné tuto konfigurace vždy po změně rozhraní, příp. zde i po změně alternativní konfigurace, opět vrátit do korektního stavu. Konkrétně pole `MULT`, které určuje počet paketů v rámci jednoho rámce na sběrnici USB je chybně nastaveno na 0 a pole `ZLT` je nastaveno na 1. Pokud je délka paketu v rámci jedné transakce nastavena na hodnotu 1024 $B$, potom je navíc nutné i tuto hodnotu vrátit zpět na původní požadovanou. V případě jiných délek se chyba u posledního pole neprojeví. Kód realizující opravu tohoto problému a další podrobnosti je možné dohledat rovněž v [22].

## 7.3 Příjem dat z rozhraní VGA

Druhým projektem v rámci prostředí LPCXpresso, který je součástí firmware popisovaného zařízení je uložen pod názvem `vga_grabber_m4_core`. Projekt je spouštěn na *master* jádře Cortex-M4. Tento projekt realizuje vzorkování dat z rozhraní VGA a následný zápis do obrazové paměti, popsané v podsekci 7.2.6. Pro tento úkol bylo vybráno obecně výkonnější z jader, které jsou k dispozici na zvoleném čipu, jelikož se jedná o časově kritický úkol.

Vstupní funkcí projektu je klasicky hlavní funkce `main()`, která je navíc i vstupní funkcí celého firmware po finálním slinkování. Zde dochází k inicializaci a startu *slave* jádra Cortex-M0, označovaného jménem *M0APP*, jehož úkolem je implementace třídy UVC.

### 7.3.1 Inicializace

Inicializační funkce `setup_m4_core()` je volána z funkce `main()`. Provádí základní konfiguraci využitých GPIO portů pro vstup. Dále na dvou zvolených pinech nastavuje přerušení citlivé na úroveň, které odpovídá vstupním signálům *Hsync* a *Vsync*.

### 7.3.2 Vzorkování řádků

Signál *Hsync* je obsluhován rutinou přerušení pod názvem `HSYNC_PININT_IRQ_HANDLER`. Ve skutečnosti se jedná o makro, které je definuje název rutiny `GPIO0_IRQHandler()`.

Obslužná rutina samotná, která provádí vzorkování signálu na rozhraní VGA je časově kritickou částí programu. Nesmí být přerušována a při každém průchodu musí trvat stejný počet taktů. Pokud by rutina obsahovala např. nevyvážené konstrukce `if`, docházelo by při každém průchodu k jinak dlouhému zdržení, což by se neblaze promítlo do výsledného obrazu, jinými slovy, každý řádek obrazu by se začal vzorkovat v jiný okamžik od synchronizačního intervalu, který v tomto případě tvoří signál *Hsync*.

Vzorkování barvonosných signálů se odehrává ve smyčce `for`, která pravidelně čte hodnoty barev z paměťových adres odpovídajících adresám GPIO registrů.

### 7.3.3 Konec snímku

Konec obrazového snímku na rozhraní VGA je signalizován logickou 1 na synchronizačním vodiči *Vsync*. Toho opět využijeme a pin, na který je tento vodič připojen, nastavíme tak, aby umožňoval vyvolat přerušení citlivé na náběžnou hranu hranu.

Obslužná rutina má název `VSYNC_PININT_IRQ_HANDLER`. Jedná se rovněž o makro, které se rozgeneruje na původní název obslužné rutiny `GPIO4_IRQHandler`. Úkolem této obslužné rutiny je především nastavení hodnot ukazatelů do *framebufferu* na výchozí hodnoty.

### 7.3.4 Podvzorkování

Protože vzorkujeme obraz o rozlišení 640 x 480 obrazových bodů, náš *framebuffer* ale obsahuje obraz o rozlišení 256 x 192 obrazových bodů, musíme provádět podvzorkování přijímaného signálu v obou prostorových rozměrech.

Vertikální podvzorkování lze řešit jednoduchým přeskakováním řádků. Tato funkcionalita je implementována v obslužné rutině `HSYNC_PININT_IRQ_HANDLER` pomocí čítače řádků. Míru podvzorkování řádků definuje makro `HSYNC_DOWNSAMPLING_CYCLES`.

Podvzorkování na horizontální úrovni je implicitní. Procesor jednoduše vzorkuje data z portů GPIO v takové rychlosti, aby přečetl právě 256 obrazových bodů, což odpovídá jednomu řádku *framebufferu*.

### 7.3.5 Latence přístupu do paměti

Po prvotních testech VGA grabberu bylo zřejmé, že něco způsobuje rozdílné latence pro smyčku vzorkující data na rozhraní VGA. To způsobovalo jev vlnění obrazu rekonstruovaného ve *framebufferu*. Po bližším ohledání se ukázalo, že příčinou je rozdílná latence přístupu do paměti vyhrazené pro *framebuffer*, která se projevovala rozdílnou dobou vzorkování různých horizontálních řádků. Problémem byly časté přístupy do paměti ve spojení s probíhajícími přenosy DMA. Dle [15] jsou přístupy do stejné paměti z více různých bloků čipu vyřizovány systémem *round robin*.

Řešení spočívalo ve vypnutí vzorkování ve chvíli, kdy probíhá *izochronní* DMA přenos po sběrnici USB a tím i časté vyčítání dat ze sdílené paměti. Toto přerušování vzorkování má pouze zanedbatelný vliv na kvalitu obrazu, jelikož frekvence snímků na rozhraní VGA činí 60 $fps$, zatímco přenos do počítače pomocí rozhraní UVC dosahuje snímkové frekvence 15 $fps$.

### 7.3.6 Propojení LPC-Link2 a FITkitu

Tabulka 7.1 uvádí propojení vývodů na $JP10$ přípravku FITkit s konektorem $J9$ na desce LPC-Link2. Propojení bylo zvoleno s ohledem na méně náročnou implementaci bitových posuvů jednotlivých bitů na správné pozice v rámci barevného modelu RGB16, tj. RGB565, na procesoru LPC4370.

| | Barvonosné signály | | | Synchronizační signály | |
|---|---|---|---|---|---|
| Označení signálu | RED0<br>RED1<br>RED2 | GREEN0<br>GREEN1<br>GREEN2 | BLUE0<br>BLUE1<br>BLUE2 | HSYNC | VSYNC |
| Vývod na $JP10$ (FITkit) | 40<br>41<br>42 | 39<br>38<br>37 | 36<br>35<br>34 | 44 | 43 |
| Vývod na $J9$ (LPC-Link2) | 10<br>12<br>16 | 18<br>20<br>22 | 4<br>6<br>8 | 14 | 24 |
| GPIO procesoru LPC4370 | P1_3<br>P1_4<br>P1_13 | P3_0<br>P3_1<br>P3_3 | P1_0<br>P1_1<br>P1_2 | P1_12 | P3_4 |

Tabulka 7.1: Propojení přípravku FITkit s vývojovou deskou LPC-Link2.

Schéma přípravku FITkit je dostupné z [3] a schéma desky LPC-Link2 je dostupné z [1].

## 7.4 Sloučení obou projektů

Pro správné slinkování obou projektů do jednoho výsledného firmware je potřeba splnit následující body. Seznam je stručným výtahem ze zdroje [20].

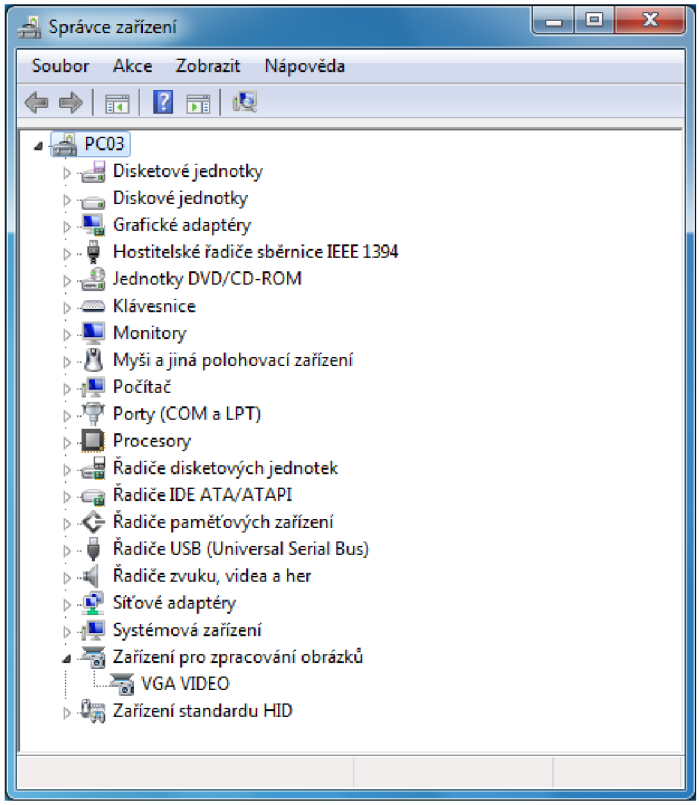
- Zajistit, aby oba projekty měly stejnou mapu paměti
- U *slave* projektů odstranit z mapy paměti paměť typu *flash*, tím se zajistí běh programů z paměti
- První paměť v paměťové mapě u *slave* projektů určuje paměť, ze které bude program spouštěn. Doporučuje se takto nastavit na každém z jader jiný blok paměti z důvodu rychlosti.
- Ve vlastnostech *master* projektu v části *Linker*, sekci *Multicore* je třeba vybrat zvolený *slave* projekt.
- Na stránce z předchozího bodu musí paměť, ze které je spouštěn *slave* kód, odpovídat nastavenému prvnímu bloku z příslušejícího *slave* projektu.

# Kapitola 8

# Výsledné řešení

## 8.1 Ověření činnosti

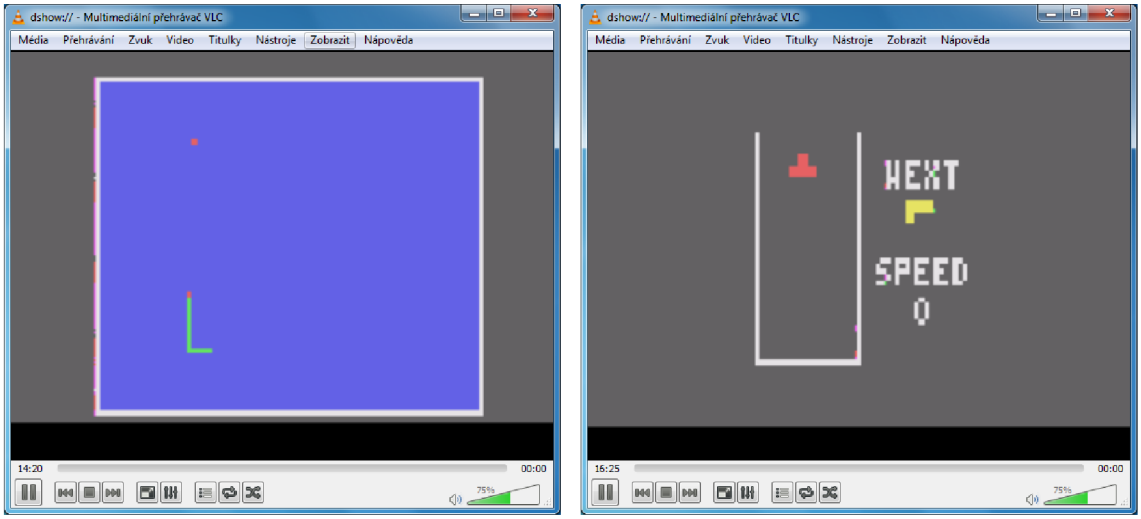
K ověření je nutné využít počítač s operačním systémem Windows. Ověření činnosti VGA grabberu je poměrně jednoduché. Protože nevyžaduje instalaci speciálních ovladačů, je možné grabber pouze zasunout do volného konektoru USB sběrnice a vyčkat chvíli na instalaci univerzálního ovladače. Po automatické instalaci se zařízení objeví ve *Správci zařízení* systému Windows jako *Zařízení pro zpracování obrázků*, jak ilustruje snímek 8.1.




Obrázek 8.1: Ukázka enumerace našeho zařízení *VGA VIDEO* v systému Windows.

Pro samotné prohlížení videa je vhodné využít některý z video přehrávačů, který podpo-

ruje režim snímání *DirectShow*. Touto funkcí disponuje např. přehrávač VLC. Pro spuštění v přehrávači VLC zvolíme menu *Média*, následně vybereme *Otevřít zachytávací zařízení*. V okně, které se zobrazí vybereme u parametru *Název video zařízení* zdroj signálu *VGA VIDEO*, což je název, pod kterým se VGA grabber zobrazuje v operačním systému. Po stisku tlačítka *Přehrát* by mělo dojít k zobrazení video streamu z VGA grabberu. Obrázek 8.2 ilustruje funkčnost na hrách *Had* a *Tetris* využívajících VGA rozhraní, dostupných z repozitářů pro přípravek FITkit.




Obrázek 8.2: Ukázka funkčnosti VGA grabberu na hrách *Had* (vlevo) a *Tetris* (vpravo).

Data z VGA grabberu je možné dále *streamovat* po síti, příp. nahrávat na pevný disk. Záleží na funkcích zvoleného přehrávače.

## 8.2 Parametry výsledného řešení

Poměrně nízkému rozlišení 256 x 192 obrazových bodů odpovídá také *bitrate* obrazových dat. Teoretická šířka pásma je dána rozlišením, počtem bitů na pixel a snímkovou frekvencí, tedy $256 \cdot 192 \cdot 16 \cdot 15 \doteq 11520\ kb/s$. Hodnota naměřená prostřednictvím statistik přehrávače VLC činí přibližně 11720 $kb/s$, což odpovídá vypočítané hodnotě.

Maximální rychlost snímání obrazových dat z rozhraní VGA odpovídá frekvenci 25 $MHz$, což představuje výchozí nastavení řadiče VGA na přípravku FITkit.

Odezva na změnu obrazu je téměř okamžitá, pouhým okem nepostradatelná. Je nutné ovšem vzít v potaz, že některé konfigurace operačních systémů mohou do cesty vkládat buffery, jejichž délku nelze ze zařízení ovlivnit. Pro dosažení minimální latence v přehrávači VLC je možné snížit velikost vyrovnávacího bufferu na straně přehrávače.

# Kapitola 9

# Závěr

V úvodních kapitolách 2 a 4 práce byly shrnuty teoretické informace nezbytné pro výrobu VGA grabberu. V kapitole 5 byly nastíněny možnosti realizace, z nichž se nejlépe jevila architektura s mikrokontrolérem využívajícím digitální výstup rozhraní VGA na přípravku FITkit a další možností případného rozšíření pomocí ADC převodníku, jež byla popsána v 5.4.3.

Zvoleným mikrokontrolérem pro praktickou realizaci je *LPC4370* od firmy *NXP*, který je popsán v kapitole 6. Vývoj byl proveden na vývojových deskách *LPC-Link2*, popsaných v 6.2.9, za pomoci softwarových prostředků, které jsou na Internetu dostupné zdarma.

Kapitola 7 se zabývá návrhem a popisem firmware, který je rozdělen na dvě části, z nichž každá je provozována na vyhrazeném procesorovém jádře. Kanály DMA zajišťují požadovanou propustnost a realizaci přenosu dat mezi pamětí a rozhraním USB, které bude využito pro zasílání navzorkovaných dat do PC. Na straně uživatelského PC není zapotřebí programovat specializovanou klientskou aplikaci, jelikož je zařízení implementováno dle standardu UVC. Firmware zařízení využívá vícejaderné architektury zvoleného mikroprocesoru, kdy jedno jádro obstarává komunikaci s počítačem a druhé jádro vzorkuje data z rozhraní VGA.

Kapitola 8 hodnotí konečný výsledek z pohledu koncového uživatele.

Závěrem lze dodat, že i přes relativně velký výkon zvoleného mikrokontroléru je stále poměrně obtížné úlohu realizovat optimálně.

Následujícím rozšířením by mohla být realizace vlastní desky plošných spojů, případně rozšíření RAM pomocí externí paměti, což by mohlo umožnit vyšší rozlišení zprostředkovaného obrazu.

# Literatura


[1] LPC-Link2 Schematics [online]. http://www.embeddedartists.com/sites/default/files/support/xpr/link2/LPC-Link-II_Rev_C.pdf, [cit. 2015-04-14].

[2] Axelson, J.: *USB Complete, Third Edition, Everything You Need to Develop Custom USB Peripherals*. Lakeview Research LLC, 2005, iSBN 1931448027.

[3] Fučík, O.; Bardas, R.: Dokumentace FITkit - Schéma přípravku FITkit [online]. http://merlin.fit.vutbr.cz/FITkit/download/schematic_v20.pdf, [cit. 2015-01-14].

[4] Hinner, M.: VGA modes and timing overview [online]. http://martin.hinner.info/vga/timing.html, [cit. 2015-01-14].

[5] Hradecký, D.: VGA řadič na FPGA [online]. http://amber.feld.cvut.cz/fpga/studenti/david_hradecky/vga_fpga.pdf, [cit. 2015-01-14].

[6] JavierValcarce.eu: VGA Video Signal Format and Timing Specifications [online]. http://javiervalcarce.eu/wiki/VGA_Video_Signal_Format_and_Timing_Specifications, [cit. 2014-12-20].

[7] Kotásek, Z.; ostatní vyučující předmětu IPZ: Mareriály k předmětu IPZ - Charakteristika rozhraní USB [online]. https://www.fit.vutbr.cz/study/courses/IPZ/public/texty/usb/usb2014.pdf, [cit. 2014-12-22].

[8] RevRYL.com: Force installation of unsigned drivers in Windows 8... [online]. http://revryl.com/2013/02/19/force-driver-windows-8/, [cit. 2015-01-07].

[9] WWW stránky: Video Graphics Array [online]. http://en.wikipedia.org/wiki/Video_Graphics_Array, [cit. 2014-12-20].

[10] WWW stránky: Universal Serial Bus [online]. http://en.wikipedia.org/wiki/USB, [cit. 2014-12-22].

[11] WWW stránky: USB in a NutShell - Making sense of the USB standard [online]. http://www.beyondlogic.org/usbnutshell/usb1.shtml, [cit. 2014-12-23].

[12] WWW stránky: Motion JPEG [online]. http://en.wikipedia.org/wiki/Motion_JPEG, [cit. 2015-01-02].

[13] WWW stránky: The NXP LPC4370 Datasheet [online].
http://www.nxp.com/documents/data_sheet/LPC4370.pdf, [cit. 2015-01-02].

[14] WWW stránky: JTAG - About boundary-scan [online].
http://www.jtag.com/en/content/about-boundary-scan, [cit. 2015-01-04].

[15] WWW stránky: The NXP LPC43xx User Manual [online].
http://www.nxp.com/documents/user_manual/UM10503.pdf, [cit. 2015-01-04].

[16] WWW stránky: ULPI - The Standard for High-Speed USB PHYs [online].
http://www.mentor.com/products/ip/usb/usb20otg/phy_interfaces, [cit. 2015-01-04].

[17] WWW stránky: AN11365: SCT camera interface design with LPC1800 and LPC4300 [online]. http://www.lpcware.com/content/nxpfile/an11365-sct-camera-interface-design-lpc1800-and-lpc4300, [cit. 2015-01-07].

[18] WWW stránky: Text Modes [online]. http://en.wikipedia.org/wiki/Text_mode, [cit. 2015-01-14].

[19] WWW stránky: Media Type Identifiers [online].
https://msdn.microsoft.com/en-us/library/windows/desktop/dd757532%28v=vs.85%29.aspx, [cit. 2015-02-23].

[20] WWW stránky: Multicore M0 app and memory selection [online].
http://www.lpcware.com/content/forum/multicore-m0-app-and-memory-selection#comment-1137715, [cit. 2015-02-23].

[21] WWW stránky: USB Device Class Definition for Video Devices, rev. 1.1 [online].
http://www.usb.org/developers/docs/devclass_docs/USB_Video_Class_1_1_090711.zip, [cit. 2015-02-23].

[22] WWW stránky: Errata sheet LPC4370 [online].
http://www.nxp.com/documents/errata_sheet/ES_LPC43X0.pdf, [cit. 2015-05-03].

[23] WWW stránky: GoogleCode Cortex-M3 UVC Video Device Class Example on LPC23xx [online]. http://cortexm3.googlecode.com/svn-history/r87/trunk/ST/Project/USB_UVC_LPC/, [cit. 2015-05-03].

[24] WWW stránky: How to Implement an Image Sensor Interface with EZ-USB®FX3$^{TM}$in a USB Video Class (UVC) Framework [online].
http://www.cypress.com/?rID=62824, [cit. 2015-05-03].

[25] Čapka, L.: Dokumentace FITkit - VGA rozhraní [online].
http://merlin.fit.vutbr.cz/FITkit/docs/firmware/fpga_vga.html, [cit. 2014-12-20].

# Příloha A

# Seznam použitých zkratek

VGA Video Graphics Array

HW Hardware

CRT Cathode Ray Tube

USB Universal Serial Bus

FPGA Field Programmable Gate Array

MCU Microcontroller Unit

SPI Serial Peripheral Interface

GPIO General Purpose I/O

GUID Globally Unique Identifier

DMA Direct Memory Access

NVIC Nested Vectored Interrupt Controller

SCT State Configurable Timer

ULPI UTMI+ Low Pin Interface

NMI Non-Maskable Interrupt

HNP Host Negotiation Protocol

SRP Session Request Protocol

USB OTG USB On The Go

# Příloha B

# Obsah CD

```
CD ROOT
|
|- text
| |- src
| |- dp.pdf
|
|- firmware
```