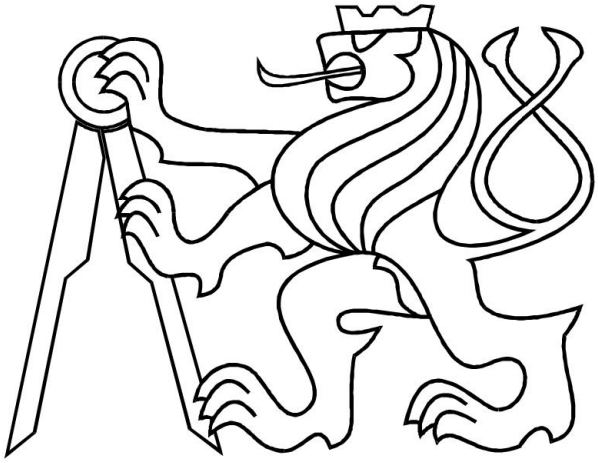
České vysoké učení technické v Praze
Fakulta elektrotechnická



DIPLOMOVÁ PRÁCE

Síť měřících obrazových snímačů

Praha, 2011 **Autor práce: Bc. Vojtěch Šváb**

Vedoucí práce: Doc.Ing.Jan Fischer, Csc.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra měření

Akademický rok **2010-2011**

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student:	**Bc. Vojtěch Šváb**
Program:	**Kybernetika a robotika**
Obor:	**Senzory a přístrojová technika**
Název tématu česky:	**Síť měřicích obrazových snímačů**
Název tématu anglicky:	**Network of Measuring Image Sensors**

Pokyny pro vypracování:

Analyzujte možnosti využití snímačů s vnitřním zpracováním redukované obrazové informace pro vícekanálové vyhodnocení polohy označených objektů v prostoru. Navrhněte metody rychlého průběžného zpracování obrazu pro nalezení polohy optických značek na objektu a implementujte je s využitím signálových procesorů Analog Devices ADSP BF533 a BF504. Navrhněte a realizujte modul obrazového snímače se senzorem CMOS a vnitřním zpracováním signálu pomocí DSP, který bude podporovat práci v síti. Posuďte možnost funkce snímačů v síti s využitím rozhraní RS485 a rozhraní Ethernet. Vytvořte potřebné programové vybavení pro signálový procesor a programové vybavení pro nadřazené PC včetně ovladačů pro spolupráci s platformou na bázi LabView. Sestavte síť s realizovanými měřicími snímači a prověřte její funkčnost při sledování polohy označeného objektu pohybujícího se v prostoru.

Seznam odborné literatury:

[1] ADSP-BF50x Blackfin® Processor Hardware Reference, Analog Devices 2010
[2] Heijden, F.: Image Based Measurement Systems. Hoboken, NJ, Wiley, 1994
[3] Fischer, J.: Optoelektronické senzory a videometrie. Skripta ČVUT, Praha 2002

Vedoucí diplomové práce:	doc. Ing. Jan Fischer, CSc.
Datum zadání diplomové práce:	5. ledna 2011
Platnost zadání do[1]:	29. června 2012

Prof. Ing. Pavel Ripka, CSc.
vedoucí katedry

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická
L.S.

Prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 5. 1. 2011

[1] Platnost zadání je omezena na dobu tří následujících semestrů.

Anotace

Tato práce se zabývá konstrukcí SMART kamery se sensorem CMOS, založené na průběžném zpracováním obrazu a dále konstrukcí dvou vývojových kitů. Navržená SMART kamera podporuje práci v síti a to pomocí rozhraní USB 2.0 a RS-485. Do sítě lze připojit až 32 takto navržených kamer. V SMART kameře jsou implementovány 3 algoritmy zpracování obrazové informace a to určení pozice jednoho markeru, určení pozice několika markerů a určení úhlu náklonu od horizontu. Dále je SMART kamera schopna vytvářet snímky, jež se na straně PC ukládají v BMP formátu. Pro možnost portace SMART kamer k nadřazenému systému byla vytvořena DLL knihovna, sloužící jako driver. Driver je navržen tak, aby mohly být SMART kamery začleněny do vývojového prostředí LABVIEW.

Annotation

This diploma thesis describes a construction of a SMART camera with a CMOS sensor, based on the continuous image processing and construction of two development kits. Proposed SMART camera supports networking through USB 2.0 and RS-485. Up to 32 SMART cameras can be connected in the network. Three image processing algorithms were implemented, namely determination of the position of one marker, determination of the positions of several markers and determination of the roll angle from the horizon. In addition, SMART camera is capable of creating images that can be stored on the PC side in BMP format. There were created DLL libraries, serving as drivers, for the possibility of porting the SMART cameras into the system. SMART cameras can be integrated in development environment LabVIEW.

Čestné prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Tato práce vznikla v laboratoři videometrie, katedry měření ČVUT - FEL v Praze pod vedením doc. Ing. Jana Fischera, CSc. Navazuje též na výzkum v rámci MSM6840770015 - "Výzkum metod a systémů pro měření fyzikálních veličin a zpracování naměřených dat", jehož některé poznatky a výstupy v oblasti optoelektronických senzorů také využívá.

V Praze dne _____ _____

Podpis

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu své diplomové práce doc.Ing.Janu Fischerovi, CSc za veškeré cenné rady v průběhu její realizace.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Kapitola 1

Úvod

V mnoha technických aplikacích vzniká potřeba měření polohy rozličných předmětů. Pokud je požadováno měření polohy jakéhokoliv předmětu v prostoru, naskýtá se možnost využití kamery, jako měřícího prvku. Pokud je tedy pro měření nasazena kamera, lze již pouhou změnou programu zcela změnit druh měření. Mimo určování polohy předmětu, lze také předměty počítat, měřit úhel náklonu, určovat barvu, či mnohé jiné aplikace.

Samotné kamery lze rozdělit do dvou skupin. Do první skupiny patří kamery, které snímají obraz, jež je následně zaslán do nadřazeného systému, bez jakéhokoliv vyhodnocení. Vyhodnocení obrazu probíha až na straně nadřazeného systému. Do druhé skupiny patří kamery, které sejmutý obraz požadovaným způsobem vyhodnotí a do nadřazeného systému zašlou jen výsledky měření, tyto kamery se nazývají inteligentní kamery tzv. SMART kamery.

SMART kamery lze dále rozdělit na SMART kamery s velkou vnitřní pamětí, schopné uložení celého snímku do paměti, umožňující veškeré úlohy strojového vnímání. Nevýhodou těchto kamer je jejich nedostatečná rychlost vyhodnocení scény. Další podskupinu SMART kamer, tvoří SMART kamery s průběžným zpracováním obrazu, jež se vyznačují tím, že požadované informace o scéně jsou k dispozici ihned po obdržení posledního řádku z CMOS sensoru, popřípadě s minimálním zpožděním.

V této práci bude navrhována SMART kamera, s průběžným zpracováním obrazu. V praxi může být požadováno rozmístění jednotlivých SMART kamer na mnoha místech, někdy i stovky metrů vzdálených. Z tohoto důvodu je třeba implementovat takové rozhraní, jež by takovouto komunikaci umožňovalo.

Kapitola 2

Analýza snímačů s průběžným zpracováním redukované informace pro vícekanálové vyhodnocení polohy

Nejprve je vhodné zdůraznit, že z vnějšího pohledu se průběžné zpracování obrazu příliš neliší od klasického přístupu, kde se snímek nejdříve přenese z CMOS snímače do RAM paměti a až následně je vyhodnocen. Vždy je výsledkem shodná informace o měřeném objektu.

Dříve, než-li bude přistoupeno k tématu samotných SMART kamer, je vhodné se seznámit s pojmem "Marker", který bude v textu hojně používán. Marker si lze představit jako "značku" umístěnou na měřeném objektu, jež se svojí barevností odlišuje od daného objektu a pozadí za objektem. Většinou se marker odlišuje tím, že má v obraze nejvyšší jas. Markery lze rozdělit do dvou druhů:

1. Aktivní markery
   Aktivní markery se vyznačují tím, že jsou samy o sobě zdroji světla. Nejčastěji se v roli aktivních markerů osvědčily LED diody.

2. Pasivní markery
   Mezi pasivní markery lze zařadit například odrazky, či jakékoliv bílé předměty umístěné na měřeném objektu. Pasivním markerem může být i bílý papír.

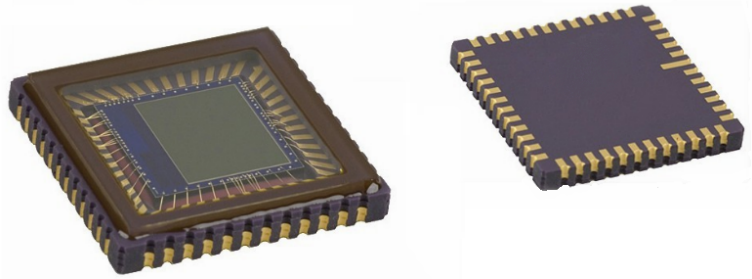
SMART kamery jsou v dnešní době často nasazovány k měření různých parametrů, mohou sloužit jen ke kontrole provozu, nicméně v nemálo případech jsou použity i k

řízení samotného procesu. V průmyslu se obvykle využívají kamery renomovaných značek, jejichž cena se včetně obslužného softwaru pohybuje řádově od desítek po stovky tisíc korun.

V mnoha případech je požadováno sledování polohy nějakého předmětu či osoby v prostoru. Například k identifikaci člověka a jeho sledování lze použít náročné algoritmy strojového vnímání, které jsou následně velmi náročné na výpočetní výkon procesorové jednotky, což v konečném důsledku prodražuje výslednou SMART kameru. Jistým východiskem z této situace je umístění na pozorovanou osobu marker, na který bude zaměřena pozornost. SMART kamera se tedy nemusí zaobírat náročným rozpoznáváním samotného člověka, ale jen sleduje daný marker v prostoru. Tímto zjednušením měření, zároveň dochází ke zrychlení určení pozice.

2.1 Přepočet získaných dat ze SMART kamer na jednotky délky

CMOS snímač viz. obr. (2.1) SMART kamery na který se promítá obraz scény, poskytuje veškerá data v jednotkách pixelů. Aby bylo možné měřit například pozici markeru v prostoru, je třeba její určení v jednotkách délky. Na obr.(2.2) je vyobrazena zjednodušená zobrazovací soustava, kde je objektiv nahrazen pro jednoduchost tenkou čočkou. Z obrázku je zřejmé, jakým způsobem se promítá měřený objekt na CMOS snímač. Z obrázku je také zřejmé, že se zvyšující se vzdáleností "a" měřeného objektu od objektivu, dochází ke zmenšení jeho promítnutého obrazu na CMOS sensoru.



Obrázek 2.1: CMOS snímač MT9M001C12STM

Každý CMOS sensor se skládá z několika stovek tisíc, či miliónů jednotlivých pixelů. Každý pixel na CMOS snímači má velikost přibližně 5 μm (5,2 μm pro sensor

MT9M001C12STM). Pokud je tedy znám rozměr každého pixelu, lze určit měřené rozměry v jednotkách délky. Pro přepočet jednotek pixelů na jednotky délky, lze vycházet z Gaussovy zobrazovací rovnice (2.1). Dále využitím rovnice pro zvětšení(2.2), kde se za velikost obrazu X' dosadí rovnice (2.3) sloužící k vlastnímu přepočtu jednotek pixelů na CMOS snímači na délkové jednotky, lze úpravou nalézt vzorec popisující již přímo daný přepočet (2.4). Nahrazením proměnné "X" za "Y" ve vzorci (2.4), lze získat přepočet souřadnic v ose y. Ze vzorce je zřejmé, že pří oddalování měřeného předmětu od objektivu se předmět jeví zmenšeně. Pro správný odečet délkových souřadnic SMART kamerou, musí být tedy nejdříve změřena vzdálenost "a".

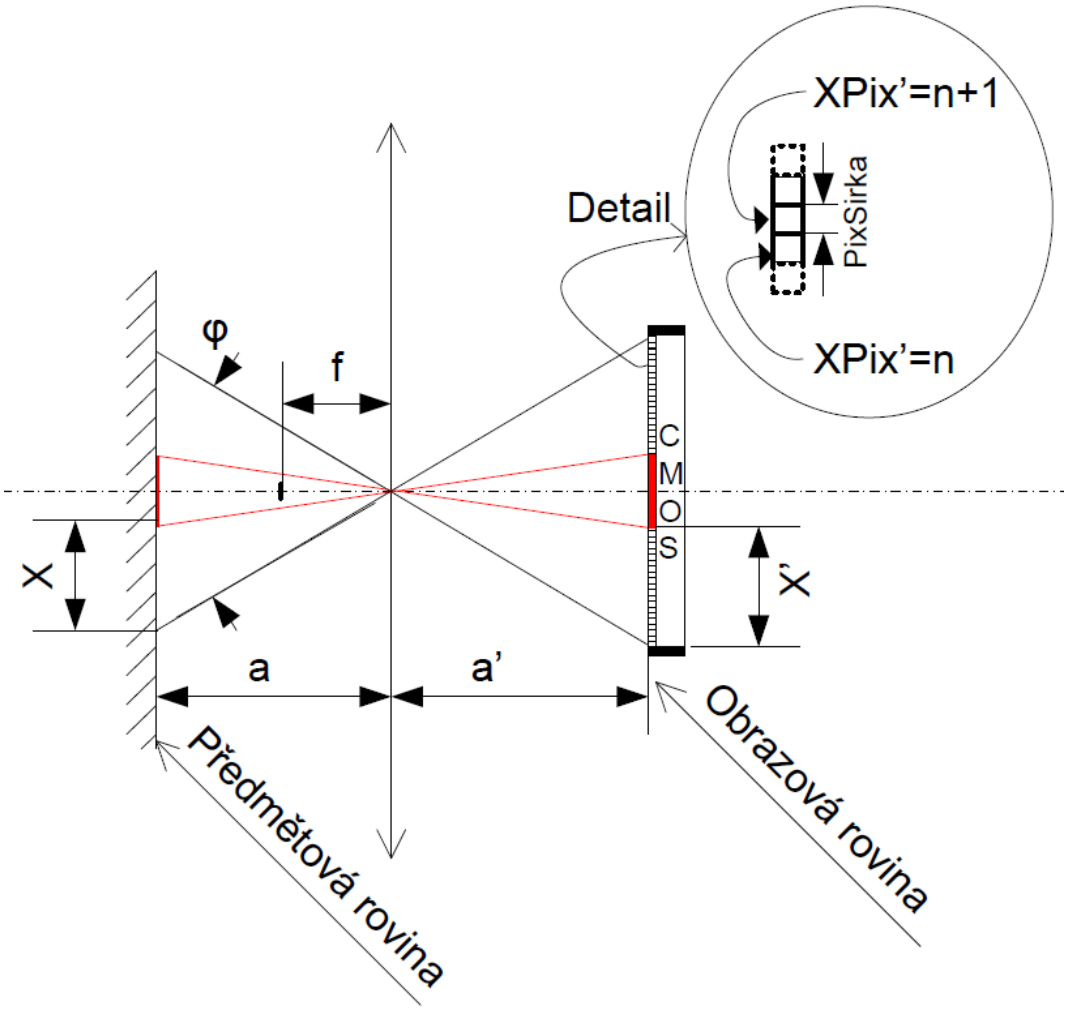
$$\frac{1}{a} + \frac{1}{a'} = \frac{1}{f} \tag{2.1}$$

$$-\beta = \frac{X'}{X} = \frac{a'}{a} \tag{2.2}$$

$$X' = XPix'.PixSirka \tag{2.3}$$

$$X = \frac{a.XPix'.PixSirka}{a'} = \frac{XPix'.PixSirka\,(a-f)}{f} \tag{2.4}$$

- f = ohnisková vzdálenost [mm]
- a = vzdálenost předmětové roviny od hlavního bodu [mm]
- a' = vzdálenost obrazové roviny od hlavního bodu [mm]
- φ = zorný úhel objektivu [°]
- X = x-ová souřadnice v předmětové rovině [mm]
- X' = x-ová souřadnice v obrazové rovině(na CMOS sensoru) [mm]
- XPix' = X souřadnice na CMOS snímači v jednotkách pixelů [pix]
- PixSirka = délka strany jednoho pixelu [mm]
- n = pozice pixelu na řádku CMOS snímače [-]



Obrázek 2.2: Zobrazovací soustava

2.2 Výpočet pozice markeru v 3D užitím více SMART kamer

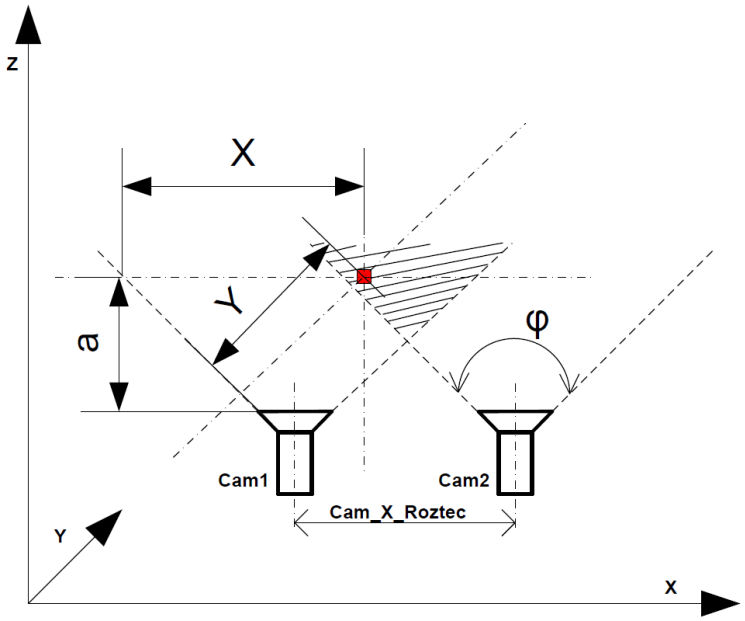
Vztahy uvedené v kapitole(2.1) umožňují přepočítat velikost měřeného objektu v pixelech na délkové jednotky v mm. Pokud by nicméně došlo ke změně vzdálenosti měřeného objektu od objetivu SMART kamery, musela by být daná vzdálenost ručně změřena a následně do vztahů opět zadána. Zde se nicméně nabízí možnost užitím dvou SMART kamer vzdálenost markeru od objektivu taktéž měřit. Pokud je tedy možné automaticky změřit vzdálenost předmětu od SMART kamery, lze již tedy kompletně určovat pozici markeru v prostoru. Geometrické uspořádání SMART kamer vychází z obr.(2.3). Z obrázku je zřejmé, že SMART kamery musejí být od sebe vzdáleny o určitou, předem

známou vzdálenost "CamXRoztec". Určení x-ové souřadnice na obou SMART kamerách by mělo být shodné dle rov. (2.4), jen se lišit právě vzdáleností "CamXRoztec" obou SMART kamer dle rov. (2.5).

$$X_1 = X_2 - CamXRoztec \tag{2.5}$$

Dosazením tedy rovnice (2.4) do (2.5) a následnou úpravou vznikne vztah (2.6)

$$a = f\left(1 - \frac{CamXRoztec}{PixSirka\,(XPix'_1 - XPix'_2)}\right) \tag{2.6}$$



Obrázek 2.3: Uspořádání SMART kamer pro stereo vidění

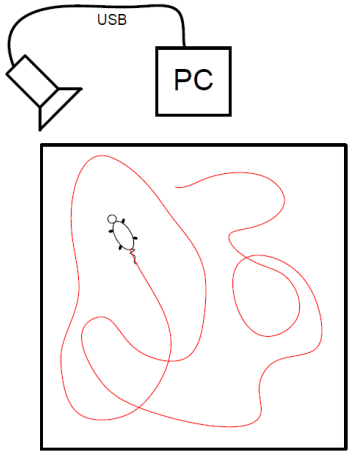
Aby byla vzdálenost "a" správně určena, musí se měřený marker nalézat ve vyšrafované oblasti dle obr. (2.3). Pokud by se marker ve vyšrafované oblasti nevyskytoval, nebyl by tedy lokalizován oběma SMART kamerama najednou, načež by tedy vzdálenost markeru od SMART kamery "a" nemohla být určena.

Z výše uvedeného textu je zřejmé, že použitím dvou SMART kamer je již možné určovat pozici markeru v 3D prostoru. Pokud by byla požadována vyšší přesnost systému, což souvisí i s vyšší robustností systému, je možné geometricky uspořádat více jak 2 SMART kamery. Výsledné souřadnice by se určily jen úpravou rovnice (2.6).

2.3 Uplatnění sítě SMART kamer v praxi

Nasazení SMART kamer se v posledních deseti letech velmi rozšířilo. SMART kamery svojí pestrou konfigurovatelností v mnoha případech vytlačují klasické senzory založené na jiných principech. Jedna SMART kamera může určit během jednoho měření barvu měřeného objektu, velikost, úhel natočení, pozici atd. Pokud by k danému účelu byly použity jednoúčelové sensory, jejich cena by pravděpodobně převyšovala jednu SMART kameru, nehledě na složitější geometrické uspořádání. Využití SMART kamer, případně jejich síťové spolupráce je názorné z několika příkladů níže:

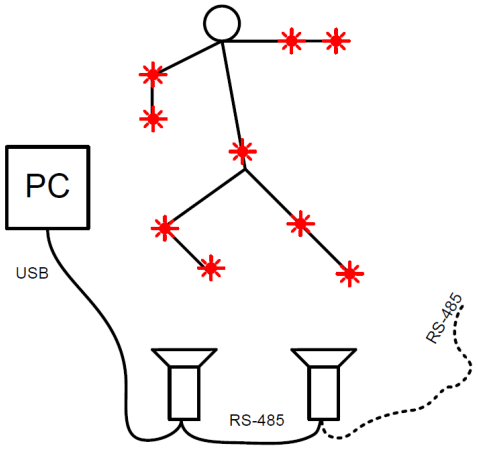
1. **Snímání trajektorie myši**
   Laboratorní myši se často využívají např. ve farmacii k pokusným účelům. Po aplikaci zkoušených látek dané myši, může být požadováno sledování pohybu myši v dlouhodobém časovém horizontu. Tohoto lze docílit využitím jedné SMART kamery a algoritmu vyhledání jednoho markeru dle(2.4). Z PC je jen v časových intervalech volána funkce určení pozice jednoho markeru, jež je následně zaznamenávána. Vzhledem k tomu, že myš je sama o sobě bílá, tvoří rovnou pasivní marker.



Obrázek 2.4: Lokalizace trajektorie myši

2. **Snímání polohy člověka**
   V mnoha oborech (Textilní průmysl, Zdravotnictví atd.) vzniká potřeba lokalizace člověka v prostoru a jeho momentálního pohybového rozpoložení. Z obr (2.5) je zřejmá postava člověka, na které je připevněno 9 aktivních LED markerů. Člověk je snímán v pravidelných intervalech dvěma kamerami pomocí algoritmu rozpoznání více markerů. Kamery jsou vhodně geometricky uspořádány tak, aby bylo možné

zpracováním dat z obou kamer rekonstruovat 3-D pohyb člověka v prostoru. Přídáním více kamer, lze dosahovat lepší přesnosti určení pohybu.



Obrázek 2.5: Rozpoznání pohybu člověka

3. **Řízení bezpilotního letadla**
   Pří řízení bezpilotních letadel je v mnoha případech využíváno autonomního módu, tzv. autopilota. Aby mohl autopilot bezpilotní letadlo bezpečně řídit, musí znát mnoho dat o aktuální pozici letadla v prostoru. K určení úhlu natočení (dopředného i bočního), lze využít např. 4 SMART kamery umístěné na přídi, zádi a na křídlech v režimu určování úhlu natočení od horizontu(resp. by stačily dvě, nicméně z důvodu bezpečnosti je vhodné využití více SMART kamer). SMART kamery mohou být umístěny na servomechanismech, umožňujícím správný odečet úhlu i při velkém natočení bezpilotního letadla. Takto navržený měřící systém by mohl v praxi sloužit jako záložní měřící systém pro určení náklonu a zároveň by dané SMART kamery mohly poskytovat pohled operátorovi při ručním řízení.

Z výše uvedených příkladů je zřejmé, že nasazení SMART kamer může být velmi různorodé. Tím že SMART kamery disponují prací v síti, lze měřící systém nasadit pro mnohé operace.

Kapitola 3

Stanovení cíle práce a použitých prostředků

Cílem této práce je návrh SMART kamery s průběžným zpracováním obrazu, jak po hardwarové, tak po softwarové stránce. Před vlastním návrhem SMART kamery bude kladen důraz na výběr vhodného procesoru. Vzhledem k tomu, že objem přenášených dat je poměrně velký a některé algoritmy budou výpočtově náročné, bude vybíráno z výkonných signálových procesorů řady BLACKFIN. Aby bylo možné CMOS snímač k procesoru připojit, musí daný procesor disponovat PPI rozhraním.

Před návrhem vlastní SMART kamery budou zkonstruovány a vyrobeny dva vývojové kity s danými procesory. Jeden kit bude navržen s procesorem BF53x a druhý s procesorem BF504F, jež je novinkou na trhu. Na těchto kitech bude možné otestovat vhodné obvodové zapojení výsledné SMART kamery.

Pro komunikaci SMART kamery s PC, by měl být využit obvod FT2232HL, jež by se pro maximalizaci přenosové rychlosti s procesorem propojil pomocí asynchronního FIFO módu přes EBIU rozhraní. Vzhledem k tomu, že procesor BF504F nedisponuje externě vyvedeným EBIU rozhraním, musel by se vytvořit komunikační kanál softwarově. Dále bude třeba vyzkoušet, jaké maximální přenosové rychlosti lze s obvodem FT2232HL dosáhnout. Cílem bude dosažení přenosové rychlosti alespoň 8 MB/s, která by již umožňovala "real-time" přenos kvalitních snímků z CMOS sensoru v plném rozlišení. Dále budou vyzkoušeny možné bootovací režimy procesoru a bude stanoveno, zda-li je třeba na plošný spoj SMART kamery umisťovat externí bootovací paměť. Pokud bude procesor BF504F splňovat veškeré požadavky, bude použit při konstrukci samotné SMART kamery. Navržené kity budou posléze využívány studenty katedry měření k výuce mikroprocesorové techniky na téma DSP.

V rámci návrhu softwaru SMART kamery, budou navrženy celkem tři algoritmy průběžného zpracování obrazu. Vzhledem k tomu, že algoritmy budou obraz zpracovávat průběžně, budou kladeny vysoké nároky na efektivitu kódu. První implementovaný algoritmus, bude algoritmus určení středu jednoho markeru. Tento algoritmus nebude výkonově náročný, tudíž bude celý napsán v jazyce C. Další algoritmy, které budou navrženy jsou tyto. Algoritmus vyhledání několika markerů najednou a algoritmus určení úhlu náklonu od horizontu. U těchto algoritmů lze očekávat vysoké nároky na výkon procesoru, proto bude pravděpodobně nutné jejich implementace v assembleru.

Při návrhu softwarového vybavení bude brán zřetel na možnost zapojení SMART kamer do sítě. Síťová komunikace bude podporována rozhraním USB a dodatečně ještě ETHERNETEM, či RS-485. V případě poruchy jedné SMART kamery v síti, bude kladen důraz nato, aby nedošlo ke kolapsu celé síťové komunikace.

Aby bylo možné SMART kamery připojit k nadřazenému PC, bude navržen příslušný driver. Driver bude navržen ve formě dynamické knihovny (DLL), z které bude možné jednotlivé funkce nadřazenou aplikací volat. Vzhledem k tomu, že je třeba přenášet obraz v real-time režimu, bude nutné maximalizovat rychlost driveru. Z tohoto hlediska je vhodné použít takový programovací jazyk, kde dochází ke kompilaci kódu na cílovou platformu ještě před jeho použitím. Tento požadavek splňuje jazyk C++. Takto napsaná DLL knihovna, již může být přímo importována v prostředí LABVIEW, nicméně komfort volání jednotlivých funkcí je pro programátora nadřazené aplikace velice nízký. Z tohoto důvodu bude vytvořena ještě jedna DLL knihovna v jazyce C#, která bude tvořit interface mezi DLL knihovnou napsanou v C++ a prostředím LABVIEW. Veškerý časově kritický kód programu bude tedy napsán v C++ DLL knihovně, s tím že C# DLL bude jen "zprostředkovávat"komunikaci mezi LABVIEW a C++ knihovnou.

3.0.1 Návrh vývojových kitů

Student se nejlépe seznámí s danou architekturou procesoru, pokud si může vývojový kit na dobu semestru zapůjčit na domácí experimentování. Vzhledem k tomu, že se jedná o vývojový kit, měla by být zajištěna možnost jednoduchého přístupu k jednotlivým pinům procesoru, čímž mohou být jednotlivé periferie procesoru důkladně vyzkoušeny.

V nynější době je na katedře měření k dispozici omezený počet zakoupených vývojových kitů ADSP-EZ kit BF533. Kity obsahují kromě procesoru BLACKFIN BF533, také analogový a video kodek. Vstupy a výstupy z kodeků jsou přímo přístupné na čelním panelu, nicméně jednolivé I/O piny již tímto způsobem vyvedeny nejsou. Výhodou tohoto kitu

je integrovaný debugger přímo na desce, což je pro vývojové účely nutností. Naopak nevýhodou kitu je použití mnoha integrovaných obvodů a různých propojek, což v celku na začátečníka působí příliš "složitě". Vyvedení jednotlivých pinů z procesoru také není zcela triviální. V případě připojení nevhodných log. úrovní, či zkratování vstupních obvodů, může dojít ke znehodnocení některých integrovaných obvodů. Vzhledem k použití BGA pouzder by byla případná oprava kitu složitá, ne-li nemožná. Vzhledem k pořizovací ceně kitu 450 $ by to bylo velmi nepříjemné. Z tohoto důvodu by tento kit měli používat jen tací studenti, kteří již mají nějaké zkušenosti s mikroprocesorovou technikou.

Dále byly v rámci diplomových prací [18] a [17] navrženy dva vlastní vývojové kity. Vývojový kit dle [17] je složen z procesoru BF53x a obvodů nutných pro jeho funkci. Většina pinů je přímo vyvedena na headery, což je pro seznámení s daným procesorem ideální. Nicméně kit neumožňuje přímou komunikaci s PC a k přehrání softwaru musí být k dispozici externí programátor. Vývojový kit dle [18] již obsahuje obvod FT2232HL, jež je zapojen jako konvertor USB/RS232, čímž již lze komunikovat s PC. Tento kit je taktéž ideální pro vývojové účely, nicméně k nahrání nového softwaru musí být použit externí programátor. Z vývojového hlediska jsou tedy na kity kladeny následující nároky:

1. Vyvedení většiny pinů na přístupné headery
2. Nízká pořizovací cena
3. Možnost výměny každé součástky na kitu
4. Zajištění příme komunikace kitu s PC
5. Jednoduchá změna software

Snaha o dodržení veškerých požadavků na vývojový kit, dala vstupní impuls k vytvoření vlastních vývojových kitů. Jak již bylo psáno výše, je třeba aby kit komunikoval přes určité rozhraní s počítačem. V dnešní době jsou počítače vybaveny porty USB, občas RS-232 a vyjímečně paralelním rozhraním. Komunikace pomocí USB je sice rychlá, nicméně její zprovoznění je pro začátečníka nepřiměřeně náročné. Jednoduché na implementaci a zároveň velmi vhodné z pedagogických důvodů je rozhraní RS-232. Negativem tohoto rozhraní je fakt, že na mnoha noteboocích již není k dispozici. Jistým východiskem z této situace je použití obvodu FT2232HL, jež lze nakonfigurovat jako převodník USB/RS-232. Počítač se s kitem propojí přes USB, nicméně FT2232HL s procesorem komunikuje přes RS-232. Pro uživatele se dané spojení v konečném důsledku tváří, jako by bylo použito hardwarového RS-232.

Celkem budou tedy navrženy dva vývojové kity. První bude s procesorem BF53x a druhý s procesorem BF504F.

3.0.2 Posouzení rozhraní ETHERNET a RS-485 pro podporu síťové komunikace

SMART kamera by měla dle požadavků podporovat USB rozhraní a následně dle uvážení ještě ETHERNET či RS-485.

Síťová komunikace přes ETHERNET je dnes standardem u mnoha zařízeních. Hlavní výhodou ETHERNETU je, že zařízení může být umístěno kdekoliv po celém světě. K tomuto účelu je vyráběno mnoho podpůrné spotřební elektroniky od routerů, modemů až po různé switche.

Procesor BF504F nedisponuje hardwarovou podporou pro ETHERNET, z tohoto důvodu by musel být použit externí integrovaný obvod. Ze schema SMART kamery (17.5) je zřejmé, že procesor nemá dostatek volných pinů k paralelnímu připojení externího obvodu, proto by komunikace musela být sériová v tomto případě přes SPI. K tomuto účelu by mohl být použit například integrovaný obvod ENC424J600.

RS-485 je asynchronní sériové rozhraní, které je často využíváno v průmyslovém prostředí. Na logické úrovni je podobné rozhraní RS-232, od kterého se liší především jinou definicí napěťových úrovní. RS-485 je často užíváno pro síťové účely, kde může být připojeno až 32 zařízení. Procesor BF504F hardwarově podporuje UART rozhraní. K zprovoznění RS-485 tedy stačí jen připojení externího obvodu ADM3485.

Při zvažování, které rozhraní je pro SMART kameru vhodnější bylo nakonec přistoupeno k RS-485. ETHERNET je sice výhodný v tom, že lze zařízení připojit téměř kdekoliv, nicméně toto není hlavní požadavek na SMART kameru. Rozhraní RS-485 může být provozováno až na vzdálenost 1200m, což je pro dané účely dostačující. Použitím opakovačů lze komunikační vzdálenost dále zvyšovat. Integrovaný obvod ENC424J600 je oproti ADM3485 několikanásobně větší a při návrhu PCB by mohli nastat komplikace, vzhledem k tomu že bude pro návrh použita jen 2-vrstvá PCB. V případě, že by někdy bylo třeba komunikovat se SMART kamerou přes ETHERNET, bude PCB SMART kamery navržena jako modulární. Na PCB budou navrženy upevňovací otvory a bude vyvedeno SPI rozhraní na headery, tudíž bude kdykoliv později možné ETHERNET zprovoznit připojením externího modulu.

Kapitola 4

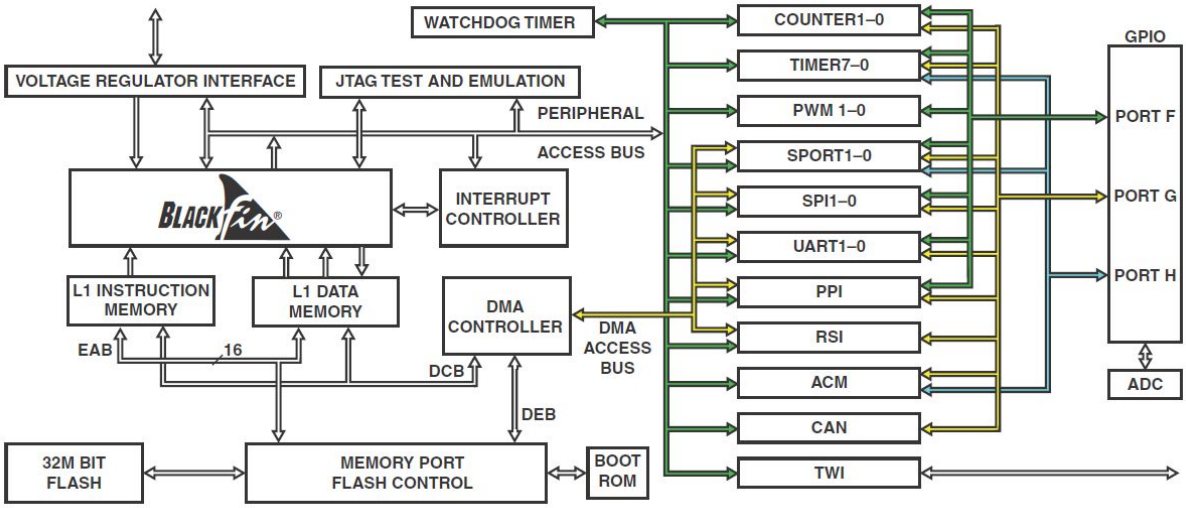
DSP Procesory BF533 a BF504F

4.1 DSP BF533

Ačkoliv byl v této práci procesor BF533 využíván, nebudou zde rozepisovány jeho specifikace. Podrobné materiálu o tomto procesoru již byly vypracovány v rámci diplomových prací Lit.[18] a Lit.[19]

4.2 DSP BF504F

Signálový procesor BF504F je novinkou od firmy Analog Devices. Obsahuje integrovanou 4MB bootovatelnou FLASH paměť, z které lze zároveň vykonávat program. Tím že není třeba využívat externí bootovací paměť, je možné výsledný produkt navrhnout s menšími rozměry a menší cenou. Samotná cena procesoru je tak nízká(cca 8$), že představuje milník v možné implementaci signálových procesorů do spotřební elektroniky. Za tuto cenu lze pořídit obdobné procesory jen s polovičním výkonem. Procesor je také vyráběn bez interní FLASH paměti pod označením BF504. Procesor lze uplatnit v řízení motorů, UPS, SMART kamerách atd., nicméně ve všech aplikacích lze těžit z výkonu kterým tento procesor disponuje. Na obr. (4.1) je blokové schéma procesoru, z kterého je zřejmá vnitřní struktura procesoru a podporované periferie. Níže budou uvedeny jen základní informace o procesoru, pro podrobnější popis lze nahlédnout do datasheetu [9]. Popis jednotlivých periferií a jejich příslušných registrů je uveden v hardwarové příručce [10].



Obrázek 4.1: Blokové schema BF504F

4.2.1 Napájecí systém

Procesor BF504F ke své funkci potřebuje celkem tři různá napětí.

- VDDINT(napětí jádra) je závislé na požadované frekvenci CCLK viz tab. (4.1)
- VDDEXT(napětí pro periferie) = 3.3V
- VDDFLASH(napětí pro paměť FLASH) = 1.8 V

Tabulka 4.1: Maximální frekvence jádra v záv. na napětí jádra pro BF504F

	VDDINT	max. CCLK
veškeré modely	1.400 V	400 MHz
průmyslové-komerční modely	1.225 V	300 MHz
průmyslové modely	1.200 V	200 MHz
komerční modely	1.150 V	200 MHz

Maximální proud, který lze odebírat z jedné skupiny pinů, nesmí přesáhnout 76mA. Jednotlivé skupiny pinů jsou uvedeny v tab. (4.2).

4.2.2 Fázový závěs

Procesor obsahuje fázový závěs, pomocí kterého je možné nezávisle na sobě volit frekvenci jádra (CCLK) a frekvenci pro periferie(SCLK). Obě frekvence je možné dynamicky

Tabulka 4.2: Maximální dovolený proud na skupinu pinů u BF504F

skupina	piny ve skupině
1	PF[10-11]
2	PF[12-15]
3	PG0
4	PG[1-4]
5	PG[5-11]
6	PG[12-15],SDA,SCL
7	ÊMU,EXT_WAKE,P̂G
8	PF[0-1],PH[0-2]
9	EXTCLK
10	PF[2-9]

měnit během chodu programu. Maximální frekvence pro periferie je F_SCLK=100 MHz. Maximální frekvence jádra je závislá na napětí VDDINT a na typu procesoru viz. tab. (4.1). Blokový diagram fázového závěsu je na obr. (4.2). Na obrázku je vyobrazen pin EXTCLK, jež v závislosti na nastavení může poskytovat frekvenci oscilátoru, či přímo SCLK.

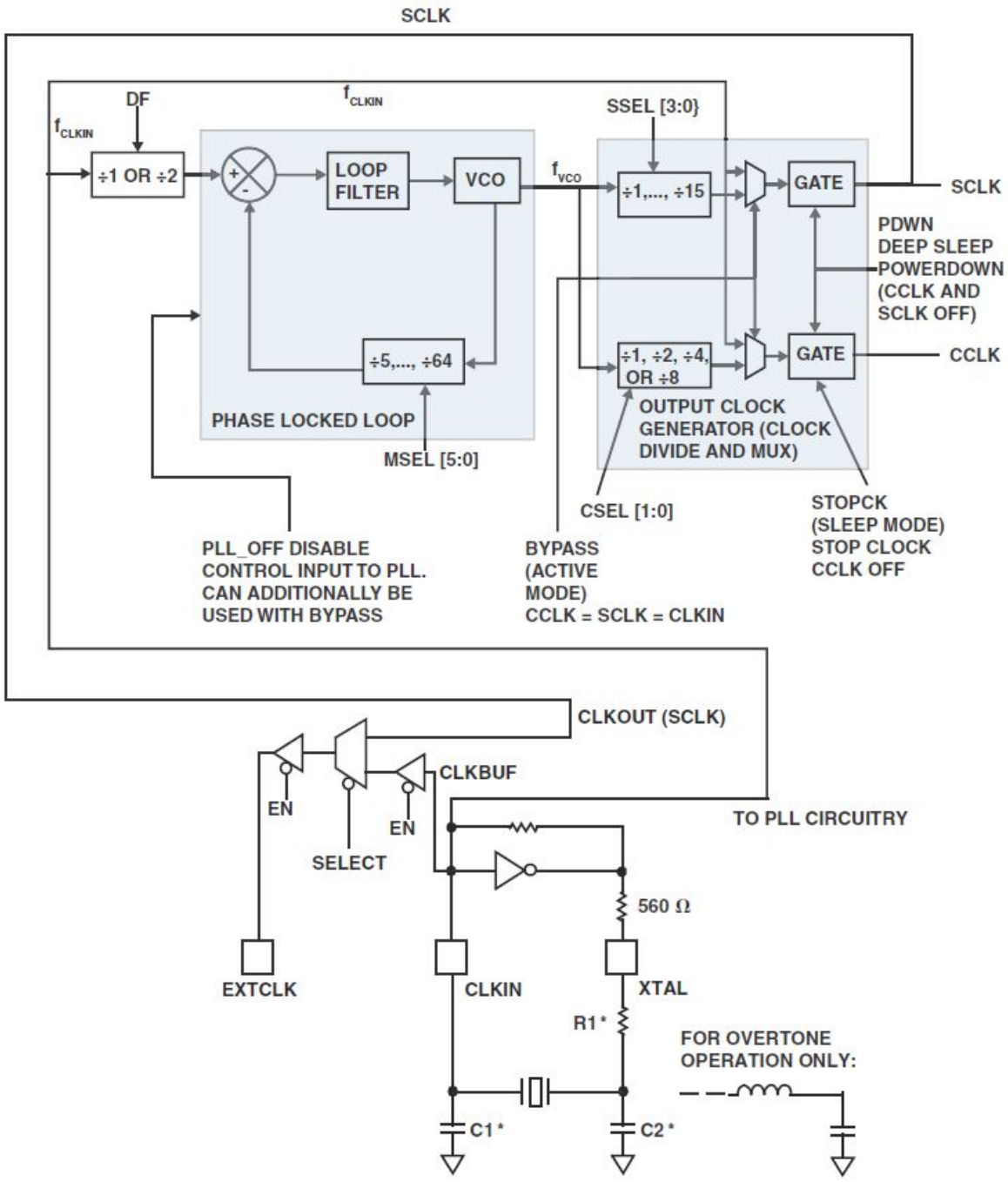
4.2.3 Architektura jádra

Jak je vyobrazeno na obr. (4.3), jádro procesoru obsahuje dvě 16-bitové násobičky, dva 40-bitové akumulátory, dvě 40-bitové ALU a jeden 40-bitový posuvný registr.

Registrový soubor je tvořen osmi 32-bitovými registry R(7-0). Při výpočetních operacích s 16-bitovými operandy registrový soubor pracuje jako 16 nezávislých 16-bitových registrů.

Každá MAC jednotka může v jednom cyklu vykonávat 16-bitové násobení, sčítání a uložení výsledku do 40-bitového akumulátoru. Podporovány jsou oba znaménkové formáty, zaokrouhlování a saturace hodnot.

ALU vykonává obvyklé aritmetické a logické operace na 16-bitových a 32-bitových operandech. Dále je přidáno mnoho speciálních instrukcí ke zrychlení různých úloh zpracování signálu. Až dvě 16-bitové ALU operace mohou být vykonány najednou v registrových párech(vyšších 16 bitů a nižších 16 bitů výpočetního registru). Pokud je využita i druhá ALU, mohou být vykonány až čtyři 16-bitové operace najednou.



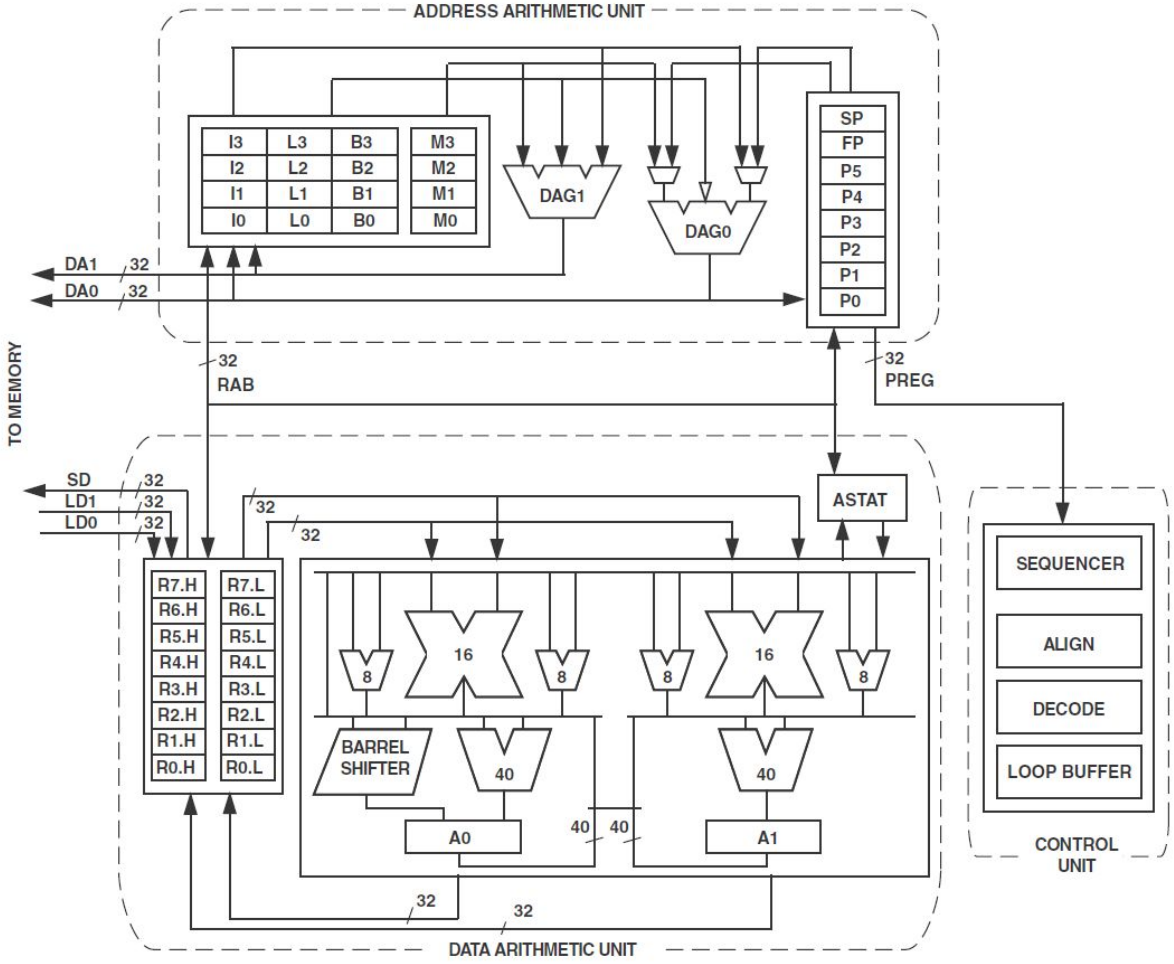
Obrázek 4.2: Blokový diagram fázového závěsu BF504F

Posuvný registr, který je 40-bitový, vykonává operace posunu a rotace. Dále je využíván k podpoře instrukcí: vyjmutí x bitů z registru, záloha x bitů do registru či k normalizaci.

Programový sequencer řídí tok vykonávání instrukcí včetně instrukčního zarovnání a dekódování. K programovému řízení instrukčního toku, sequencer disponuje PC relativními a nepřímými podmíněnými skoky a voláním subrutin. Dále jsou hardwarově

podporovány smyčky s nulovou režií.

Adresní aritmetická jednotka poskytuje dvě adresy k současnému přístupu do paměti. Obsahuje registrový soubor skládající se ze čtyř skupin registrů(index, modify,length a base registry) a z 8-mi 32-bitových pointerových registrů.

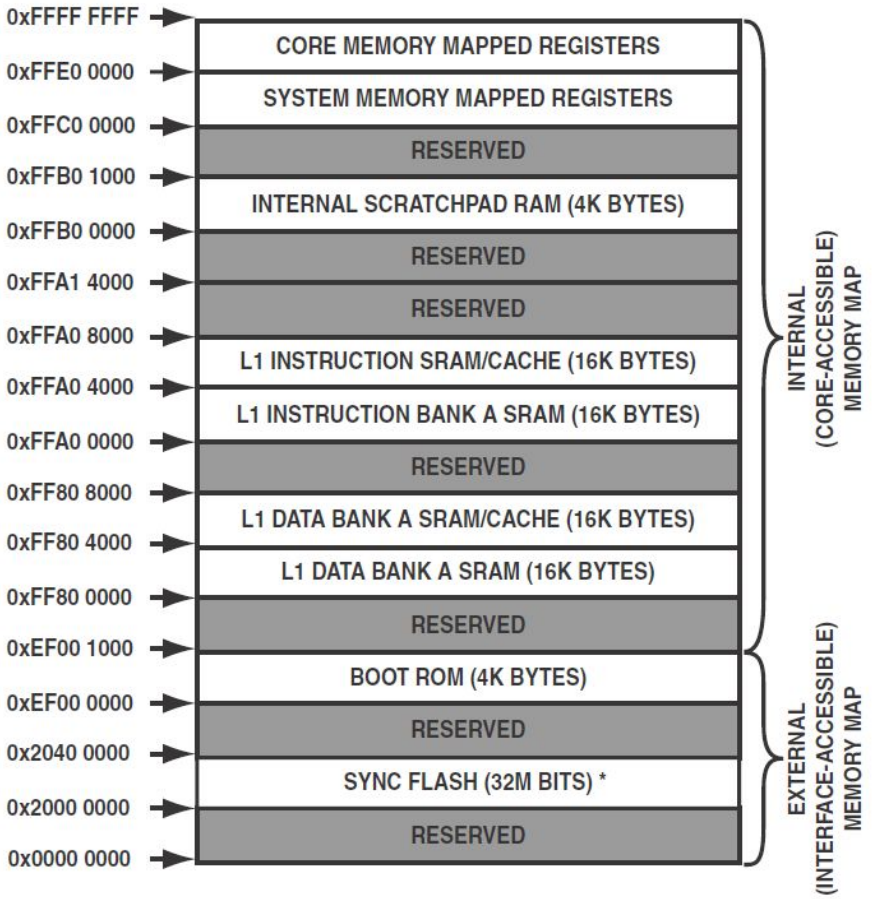


Obrázek 4.3: Architektura jádra BF504F

4.2.4 Architektura paměti

Procesor nahlíží na paměť jako na celistvý 4GB adresní prostor, užíváním 32-bitových adres. Veškeré prostředky zahrnující interní paměť, externí paměť, I/O kontrolní registry jsou umístěny v oddělených sekcích tohoto společného paměťového prostoru. Jednotlivé části paměti adresního prostoru, jsou sestaveny v hierarchické struktuře a to tak, aby poskytovaly dobrý poměr cena/výkon. Skládají se z velmi rychlých, jádrem

přístupných pamětí s nízkou latencí, jako cache či SRAM a z velkých, levných přes interface přístupných pamětí. Celá paměťová struktura je vyobrazena na obr. (4.4) Jádrem



Obrázek 4.4: Architektura paměti BF504F

přístupná L1 paměť je nejrychlejší dostupná paměť v procesoru. Paměťový systém přístupný přes EBIU, poskytuje přístup k interní FLASH a bootovací ROM paměti.

Paměťový DMA kontrolér poskytuje vysokou šířku pásma přenosu dat. Dokáže vykonávat blokové přenosy kódu, či dat mezi interní a externí pamětí.

4.2.5 DMA kanály

Procesor disponuje několika na sobě nezávislými DMA kanály, které podporují autonomní přenos dat s minimální režií pro jádro. DMA přenosy mohou být iniciovány mezi interní pamětí a jakoukoliv periferií, jež DMA podporuje. Dále mezi periferií, jež podporují DMA přenosy a externím zařízením, které je připojené přes EBIU. Mezi periferie podporující

DMA patří: SPORT, SPI, UART, RSI a PPI. Každá periferie má k dispozici příslušný DMA kanál.

DMA kontrolér podporuje jak 1-dimensionální, tak 2-dimensionální přenosy. Inicializace přenosu může probíhat na základě registrů, nebo pomocí deskriptorů.

4.2.6 Watchdog

Procesor obsahuje 32-bitový čítač, který může být softwarově nastaven do funkce Watchdog. Čítač je časován systémovými hodinami (SCLK).

4.2.7 Periferie

Procesor obsahuje následující periferie:

- 2x Čítače podporující rotační enkodér
- 8x Časovačů/čítačů s podporou PWM
- 2x SPI rozhraní
- 2x SPORT rozhraní
- 2x UART rozhraní
- 1x PPI rozhraní
- 2x 3-fázové PWM jednotky
- 1x CAN rozhraní
- 1x RSI(podpora pro paměťové karty)
- 1x TWI rozhraní (I2C..)
- 1x Interní 32 MBit Flash
- 1x ADC řídící modul
- 35x GPIO pinů

Tabulka 4.3: Přehled bootovacích režimů

BMODE[2-0]	**zdroj dat**
000	bez bootování
001	interní FLASH v asynch. módu
010	interní FLASH v synch. módu
011	SPI0 z externí SPI paměti
100	SPI0 v slave režimu
101	PPI v slave režimu
110	rezervováno
111	UART0 v slave režimu

4.2.8 GPIO

Procesor disponuje celkem třemi vstupně-výstupními porty(GPIO) a to: port F(16 pinů), port G(16 pinů) a port H(3 piny). Každý tento port je defaultně nakonfigurován do funkce GPIO jako vstup.

Změna logické hodnoty na pinu, který je nakonfigurován jako vstupní, není jádrem rozeznána okamžitě, nicméně zde dochází k prodlevě 3 SCLK cyklů než je daná změna detekována procesorem. Pokud je navíc vstupní pin nakonfigurován ke generaci přerušení při detekování změny úrovně, dochází k minimální prodlevě 4 SCLK cyklů. Pokud je vstupní pin nakonfigurován ke generaci přerušení při detekci hrany, dochází k minimální prodlevě 5 SCLK cyklů.

4.2.9 Možnosti bootování

Procesor může nabootovat z různých zdrojů dat viz tabulka (4.3) v závislosti na nastavení pinů BMODE[2-0]. Bootování je možné ve dvou režimech a to v režimu SLAVE, či v režimu MASTER. V režimu MASTER se procesor aktivně podílí na vyčítaní dat. V režimu SLAVE, procesor jen přijímá data z nadřazeného procesoru.

Kapitola 5

Obvod FT2232HL

Obvod FT2232HL zprostředkovává komunikaci mezi procesorem a PC přes rozhraní USB. FT2232HL má dva nezávisle na sobě konfigurovatelné porty, které lze nastavit jako převodník z USB na:

- UART (RS232,RS422 či RS485, vše až 12MBaud)
- 245 FIFO (synchronní - přes 25 Mb/s, asynchronní - až 10 Mb/s)
- MPSSE (JTAG, I2C,SPI,Bit-Bang)
- Mód emulace MCU hosta
- Sériové rozhraní podporující optické oddělení

5.1 Komunikace s FT2232HL ze strany PC

Komunikace s obvodem je možná pomocí dvou druhů driverů. Oba drivery je možné stáhnout z webu firmy FTDI.

5.1.1 Driver VCP

VCP(Virtual COM port) driver se chová tak, že po připojení obvodu FT2232HL k PC, dojde k detekci nového hardware a v PC se vytvoří nový COM port. Z pohledu programátora je využití nového COM portu zcela transparentní a neliší se od použití COM portu, kterým je PC hardwarově vybaven. Využití těchto ovladačů je jednodušší oproti

D2XX, protože není třeba psát dodatečný software na straně PC, lze použít jakýkoliv terminálový program.

5.1.2 Ovladače D2XX

Ovladač D2XX je tvořen dynamickou knihovnou FTD2XX.DLL, kterou je možné v projektu implementovat. Veškeré dostupné metody jež lze z této DLL knihovny volat, jsou vypsány v [5]. Této knihovny je využito při návrhu obslužné knihovny SMART kamery.

5.1.3 Volba typu rozhraní

Po resetování obvodu FT2232HL, je typ periferie nakonfigurován dle EEPROM, defaultně na UART. Z tohoto důvodu je použití ovladačů VCP jednoduché, poněvadž není nutné žádné dodatečné nastavení. Zvolení některých periferií je možné čistě softwarovou cestou pomocí funkce `FT_SetBitMode()` volané z DLL knihovny FTD2XX.DLL, nicméně pro některé periferie je nutné mít správně naprogramovanou EEPROM. Rozdělení, které periferie je možné zvolit čistě softwarovou cestou, které naopak jen pomocí EEPROM a které kombinací předchozích je zřejmé z tabulky (5.1).

Pro zvolení synchronního FT245 FIFO módu je nutné využít najednou jak EEPROM tak i softwarové volby. Nejprve je nutné do EEPROM uložit volbu 245 FIFO, čímž se po resetu FT2232HL nastaví do asynchronního FT245 FIFO módu. Dále je nutné pomocí funkce `FT_SetBitMode()` zvolit synchronní FT245 FIFO mód.

5.2 Připojení obvodu FT2232HL k PC

Po napájení obvodu FT2232HL na PCB je vhodné vyzkoušet, zda-li obvod komunikuje s PC. K tomuto účelu lze využít program FT_PROG, který lze nalézt na webu firmy FTDI. Program je třeba spusit a nechat naskenovat všechny obvody FTDI připojené k PC viz obr. (5.1). Pokud je k PC připojen jediný obvod od FTDI, měl by se zobrazit jako `DEVICE:0`. V opačném případě obvod nefunguje tak jak má, a je nutné chybu hledat na PCB.

Tabulka 5.1: Konfigurace typu periferie obvodu FT2232HL pomocí EEPROM či programově

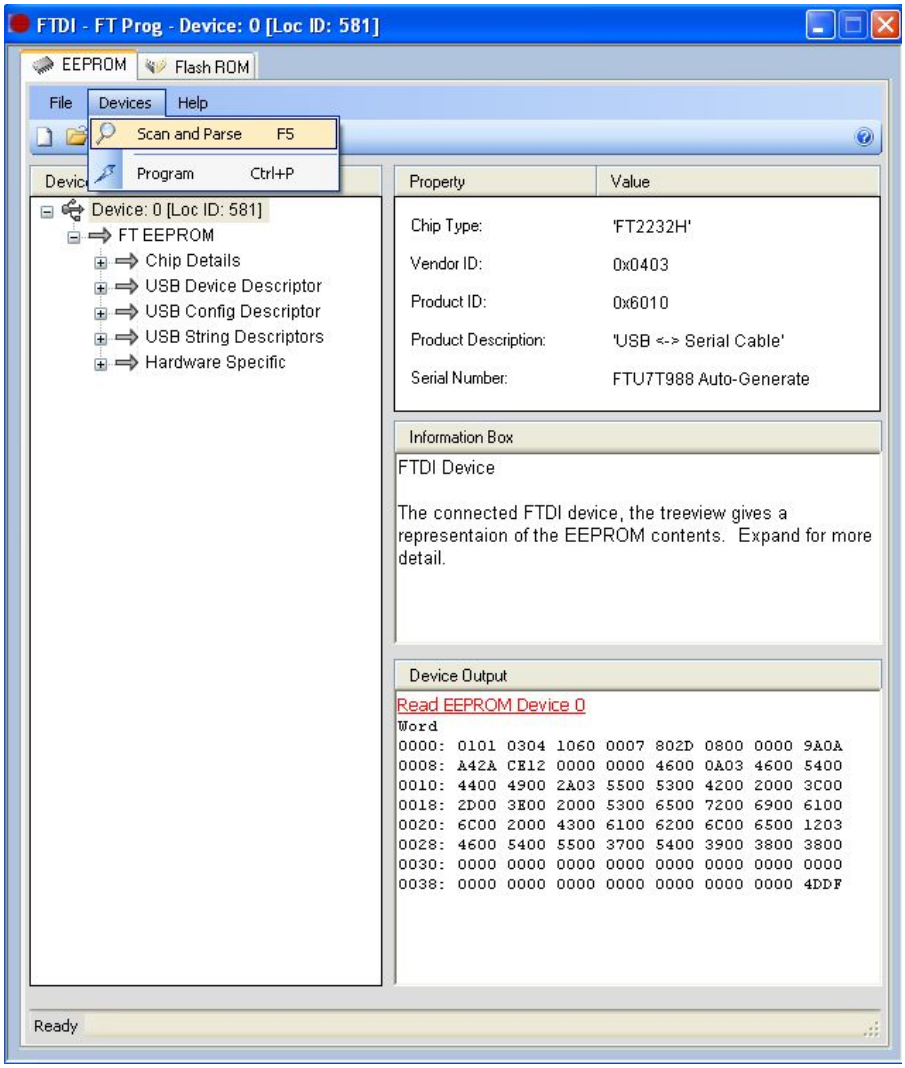
	EEPROM konfig.	softwarově konfig.
UART	√	
asynch.245 FIFO	√	
synch.245 FIFO	√	√
asynch. Bit-Bang		√
synch. Bit-Bang		√
MPSSE		√
Fast Serial Iterface	√	
CPU-Style FIFO	√	
Host Bus Emulation		√

5.3 Využití FT2232HL jako programátoru paměti 25256

Vývojové kity popsané v (6) a (7) obsahují pamět 25256 ze které je možné nabootovat. Program do paměti je možné nahrát jakýmkoliv externím programátorem, nicméně manipulace s pamětí mezi programátorem a kitem je velmi nepohodlná. Náklady na zakoupení externího programátoru jen podtrhují nevhodnost tohoto řešení. U vývojového kitu s BF533 se s možností programování paměti přes FT2232HL již počítá a veškeré potřebné spoje jsou zapojeny již na PCB. Ze schématu (17.1) je zřejmé, že první čtyři piny portu A jsou využity k propojení s pamětí. Pátý vodič slouží k resetování procesoru během programování paměti. U vývojového kitu s BF504 byl kladen důraz nato, aby byly veškeré piny libovolně využitelné, proto je nutné příslušné signálové vodiče propojit pomocí vodičů ručně. Jako programovací software je využit ft2232_prog_25256.exe, vyvinutý v rámci diplomové práce [16]

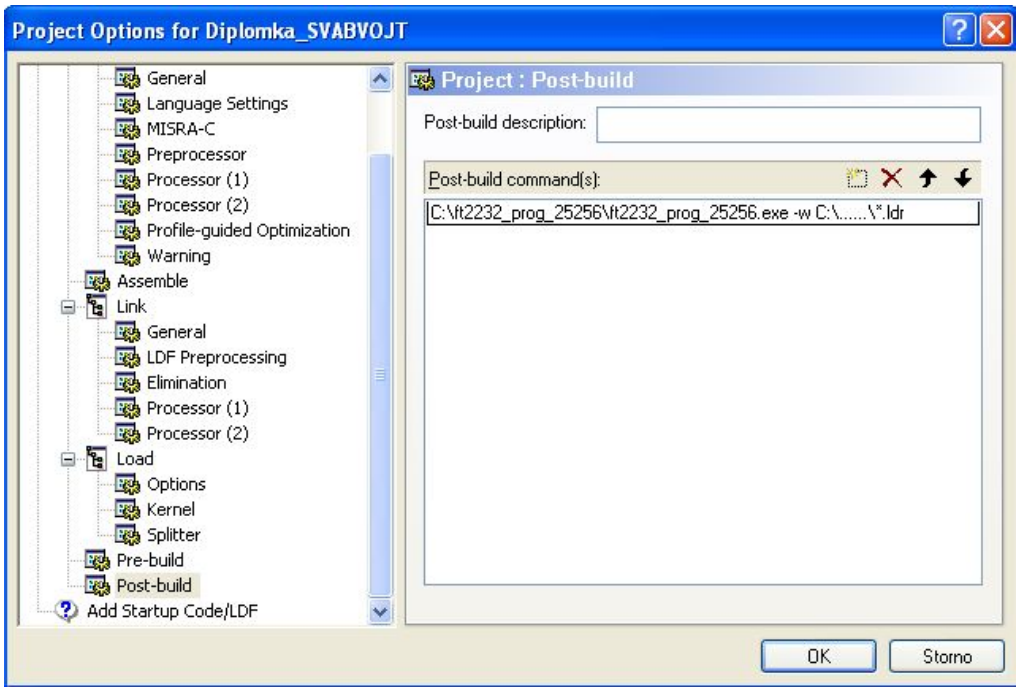
5.3.1 Postup pro nahrání programu do paměti 25256

1. Připojíme kit k napájení
2. Propojíme kit pomocí USB kabelu s PC
3. V příkazové řádce spustíme programátor



Obrázek 5.1: Skenování obvodů od FTDI

4. Výše uvedený bod lze také zautomatizovat a to tak, že příkaz zapíšeme přímo do "Post-Build"v prostředí VDSP viz obr. (5.2). Takto se program do paměti nahraje po každé kompilaci projektu zcela automaticky.



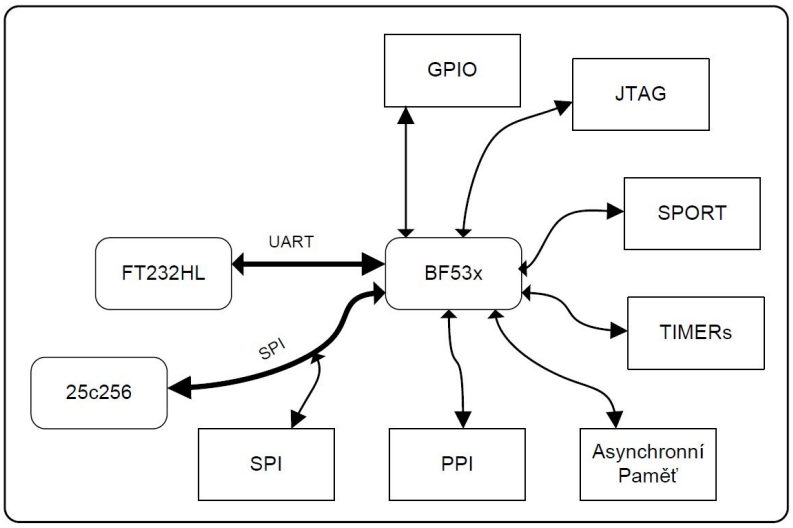
Obrázek 5.2: Nastavení VDSP pro programování paměti

Kapitola 6

Návrh vývojového kitu s DSP BF533

6.1 Blokové schéma kitu

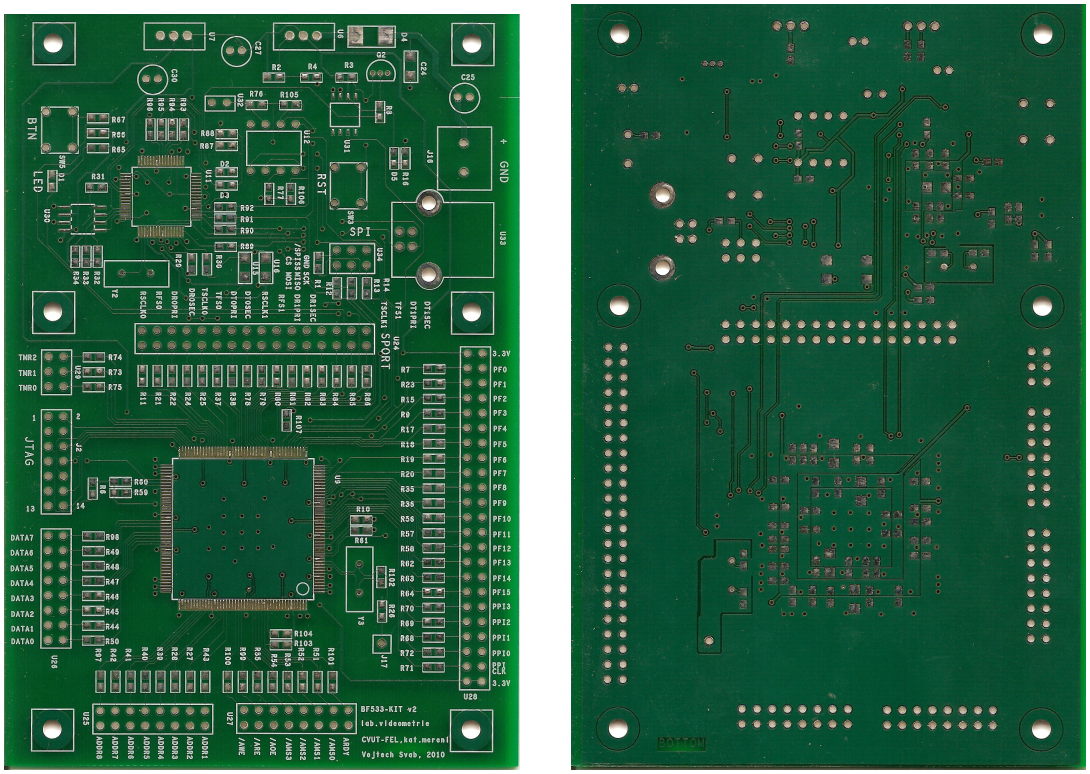
Na obr. (6.1) je vyobrazeno blokové schéma vývojového kitu s BF53x. Z obrázku jsou patrné jednotlivé vyvedené periferie. Konektor na SPI rozhraní je na stejné sběrnici jako bootovací paměť, nicméně je vyveden další pin CS(chip select), tudíž lze připojené zařízení adresovat.



Obrázek 6.1: Blokové schema vývojového kitu s BF53x

6.2 Návrh PCB

Vývojový kit byl navržen z finančních důvodů na 2-vrstvé desce plošných spojů, ačkoliv pro návrh desek s DSP je obvykle využívána min. 4-vrstva. O to důkladnější muselo být provedeno blokování procesoru, jež je taktován na 400 MHz. Vyrobený plošný spoj je vyobrazen na obr. (6.2).

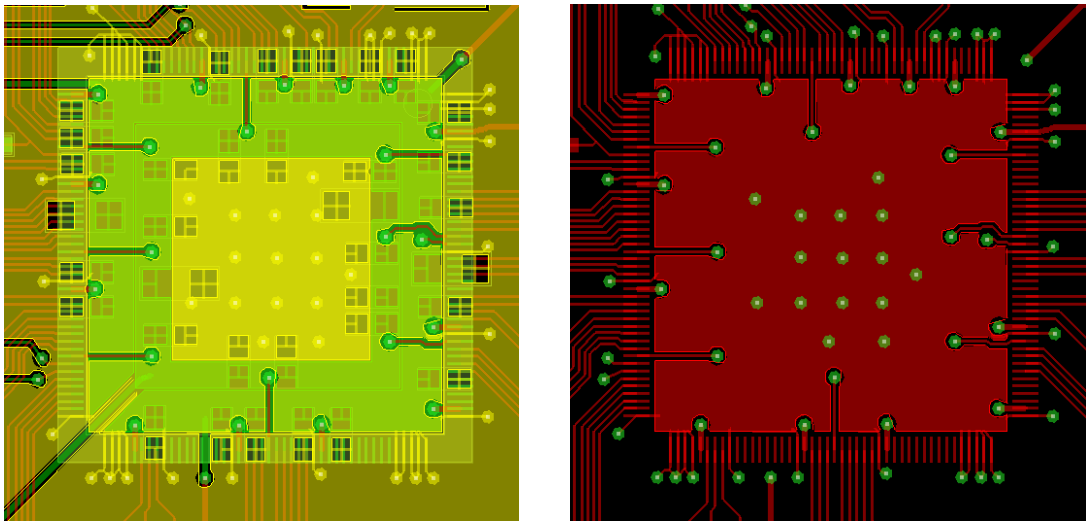


(a) (b)

Obrázek 6.2: PCB vývojového kitu s BF53x

Procesor má zvlášť napájení pro jádro a zvlášť pro periferie, kde obojí je nutné náležitě blokovat. Na obr. (6.3(a)) je znázorněno blokování procesoru. Pro pochopení jednotlivých barev je třeba určité představivosti. Vrstva Top je v barvě červené, vrstva BOTTOM je v barvě zelené a aby lépe vynikly zemnící plochy, tak byly vysvíceny žlutě. Na obr. (6.3(b)) je umístěn stejný pohled, ale je vidět jen vrstva TOP. Porovnáním obou obrázků, si lze udělat celkový nadhled. Napájení jsou rozvedena ve spodní vrstvě ve formě čtvercových rámečků. Největší rámeček tvoří napájení pro periferie, kde blokovací kondenzátory jsou vyvedeny do rozlité měděné plochy vně rámečku, jež tvoří zem. Aby mohla být blokována i větev pro napájení jádra procesoru, byla uvnitř nejmenšího rámečku rozlita měď ve tvaru

čtverce, jež je prokovena do vrchní vrstvy, kde je přes pady uzemněna do spodní rozlité mědi. Podobný přístup byl aplikován i u obvodu FT2232HL.



(a) (b)

Obrázek 6.3: PCB vývojového kitu s BF53x

6.3 Použité integrované obvody

BF53x

Popis obvodu v kap. (4.1).

FT2232HL

Popis obvodu v kap. (5).

ADM708TAR

Obvod s označením ADM708TAR je dohlížecí obvod monitorující napětí 3,3V. V případě poklesu napětí pod danou mez, ADM708TAR resetuje procesor.

93C46

Obvod s označením 93C46 je sériová EEPROM paměť. Na kitu slouží pro uchování konfiguračních dat pro obvod FT2232HL

MC33269T

Obvod s označením MC33269T je lineární "Low-Drop"stabilizátor napětí. Úbytek napětí na stabilizítoru je 1V. Maximální výstupní proud je 800 mA. Obvod je vybaven ochranou proti zkratu a přehřátí. Na kitu reguluje napětí 3,3V a 1,2V.

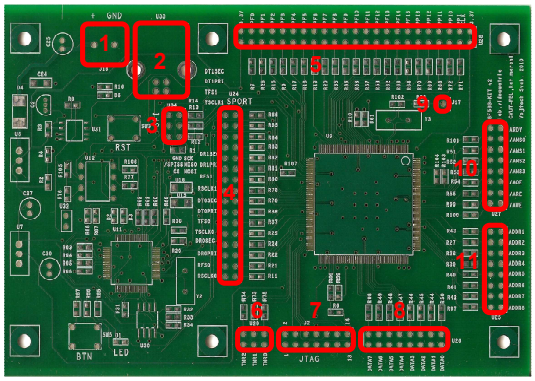
6.4 Popis vyvedených konektorů

Popis jednotlivých konektorů umístěných na vývojovém kitu je vázán na obr. (6.4) a je následující:

1. Konektor napájení
2. Konektor USB
3. SPI
4. SPORT
5. Porty PFx a PPI
6. Časovače TMRx
7. JTAG
8. Data pro asynchronní paměť
9. Testovací pin chodu krystalu
10. Řídící signály pro asynchronní paměť
11. Adresace pro asynchronní paměť

6.5 Bootování procesoru

Procesor BF53x neobsahuje „on-chip“ EEPROM (FLASH) paměť pro uchování programu, ale musí si při každém zapnutí nahrát program z externí paměti (tzv.nabootovat)



Obrázek 6.4: Popis konektorů na kitu s BF53x

do L1 instrukční paměti procesoru.
Nabootováno můž být z:

- 16-bitové externí paměti
- 8/16 bitové externí flash paměti
- přímo z SPI (BF533 je během bootování jako slave)
- 8/16/24 EEPROM/FLASH paměti pomocí rozhraní SPI (BF533 je jako master)

Na kitu je využito poslední možnosti a to bootování z EEPROM 25c256, což je 256-kbytová sériová paměť. Program je samozřejmě nutné také měnit, zde právě přichází na řadu obvod FT2232HL. K nahrání programu vytvořeného v VDSP++ (soubor .ldr) slouží programátor vytvořený v rámci [16], který nastaví port A obvodu FT2232HL do režimu SPI, uvede BF53x do resetu a nahraje program do paměti.

6.6 Oživení kitu

Pokud byly veškeré součástky správně zapájeny, měl by být kit okamžitě připraven k použití. K ověření funkce slouží program "Test_BF533_KIT.ldr"umístěný na přiloženém CD. Po nahrání programu do paměti dle (5.3) a resetování kitu by měla LED blikat frekvencí 1Hz. Pokud vše takto funguje, je tím ověřena funkce jak procesoru BF53x, obvodu FT2232HL tak i bootovací paměti.

6.7 Technické specifikace

Napájecí napětí: 5 [V]

Odebíraný proud: 75 [mA] (při CCLK=160MHz,SCLK=32MHz,žádná zátěž na pinech)

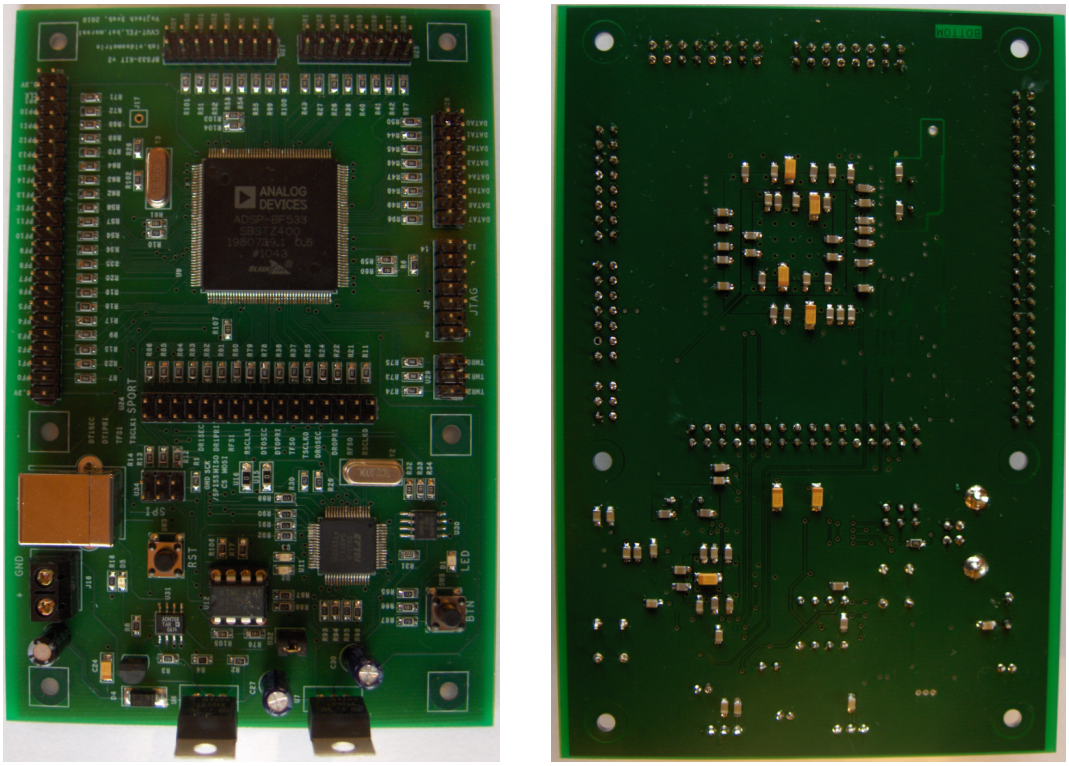
Debuggovací rozhraní: JTAG

Komunikace s PC: USB

Rozměry DPS: 130x85 [mm]

6.8 Fotodokumentace

Na obr.(6.5) je vyobrazen vyrobený vývojový kit.



(a) (b)

Obrázek 6.5: Zhotovený vývojový kit s BF53x

Kapitola 7

Návrh vývojového kitu s DSP BF504F

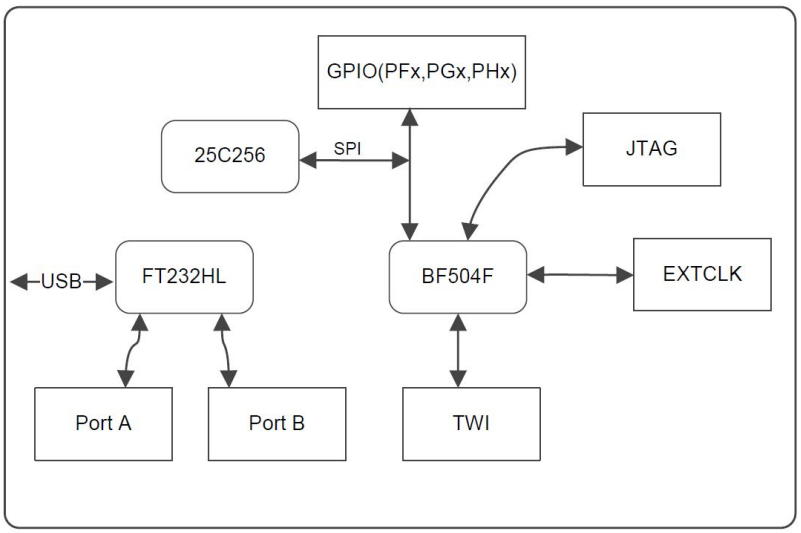
Tento kit byl navržen k ověření vlastností signálového procesoru BF504F. Veškeré možné periferie jsou vyvedeny na headery, které jsou přímo přístupné uživateli. Dále je na plošný spoj umístěn čip FT2232HL, pomocí kterého je možné komunikovat s PC přes rozhraní USB.

Pro možnosti testování byly na desku umístěny tlačítko a indikační dioda připojené k GPIO pinům procesoru. Zapájení veškerých obvodových součástek, ačkoli to zpočátku nebylo zcela jisté, lze ruční metodou.

7.1 Blokové schéma kitu

Na obr. (7.1) je vyobrazeno blokové schéma vývojového kitu s BF504F. Z obrázku jsou patrné jednotlivé vyvedené periferie. Oproti kitu S BF533 je zde rozdíl především v tom, že obvod FT2232HL není s procesorem spojen žádným vodičem, FT2232HL a BF504F zde tedy tvoří dva nezávislé celky, jež mají společné jen napájecí obvody a resetovací obvod.

Většina periferií, kterými procesor disponuje jsou sdílené s GPIO piny, proto nejsou v blokovém schematu uvedeny. Patice pro bootovací paměť 25c256, je také připojena k GPIO pinům. V případě použití pinů na kterých je bootovací paměť pro jiné účely než-li bootování, je vhodné uvážit vyndání paměti z patice a nastavení bootování z interní paměti.



Obrázek 7.1: Blokové schema vývojového kitu s BF504F

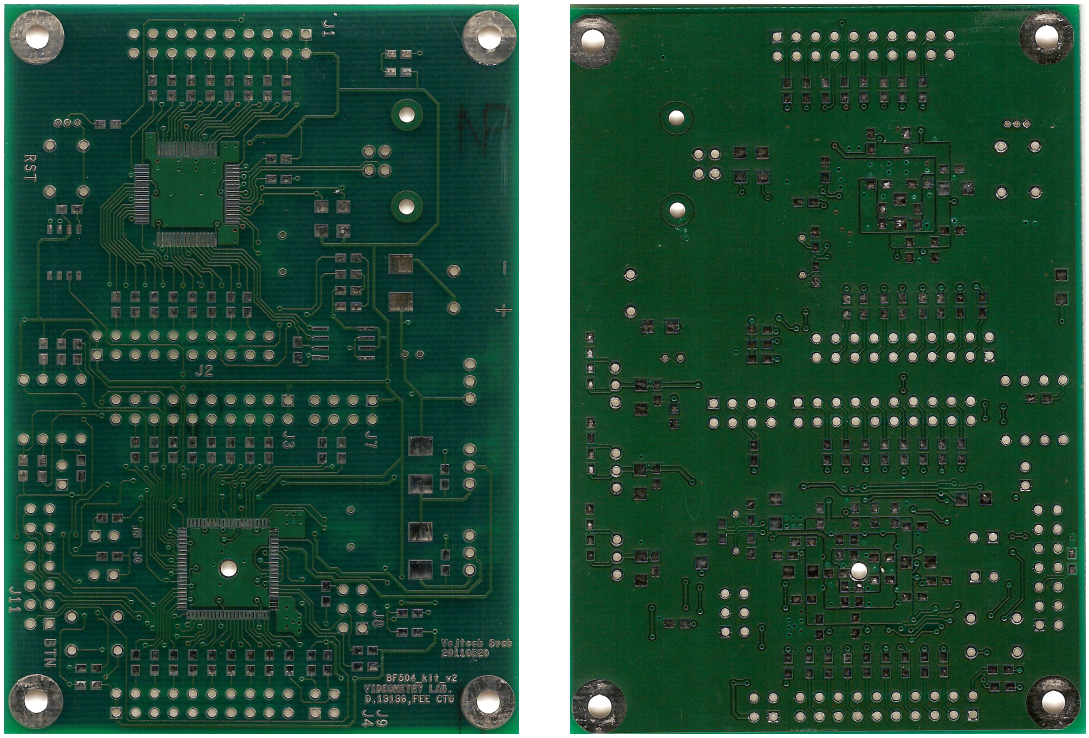
7.2 Návrh PCB

Vývojový kit byl navržen z finančních důvodů stejně jako kit s BF533 na 2-vrstvé desce plošných spojů, ačkoliv pro návrh desek s DSP je standardně využívána 4-vrstva. O to důkladnější muselo být provedeno blokování procesoru jež je taktován na 400 MHz.

Veškeré piny, jak z BF504 tak i z FT2232HL jsou vyvedeny na headery, které jsou umístěny tak, aby spoje mezi procesorem a headery byly co nejkratší.

Celkem procesor obsahuje 5 zemnících pinů, 4 jsou umístěny po obvodu pouzdra a jeden, který je využit zároveň jako termální pad pro odvod tepla je umístěn zespoda (na způsob BGA pouzdra). Aby bylo možné ruční zapájení procesoru, bylo využito triku, kdy je pod procesorem vyvrtán 2mm prokov, propojený z jedné strany přes pájecí plošky k zemi. Skrze tento prokov je možné zapájet termální pad ručně. Pokud by prokov nebyl vyvrtán, bylo by ruční pájení neuskutečnitelné.

Procesor má 3 různá napájení a to zvlášť pro jádro, periferie a zvlášť pro FLASH paměť. Kombinace třech různých napájení spolu s vyvrtaným otvorem pod procesorem a využití jen 2-vrstvé PCB značně stěžuje návrh plošného spoje v místě umístění procesoru. Na obr. (7.3(a)) je znázorněno blokování a uzemění procesoru. Pro pochopení jednotlivých barev je třeba určité představivosti. Vrstva TOP je v barvě červené, vrstva BOTTOM je v barvě zelené a aby lépe vynikly zemnící plochy, tak byly vysvíceny žlutě. Na obr. (7.3(b)) je umístěn stejný pohled, kde je vidět jen vrstva TOP. Porovnáním obou obrázků, si lze udělat celkový nadhled. Pod procesorem je vyvrtán 2mm prokov, jež zároveň spojuje zemnící plochy a umožňuje přípájení termálního padu procesoru, jež



(a) (b)

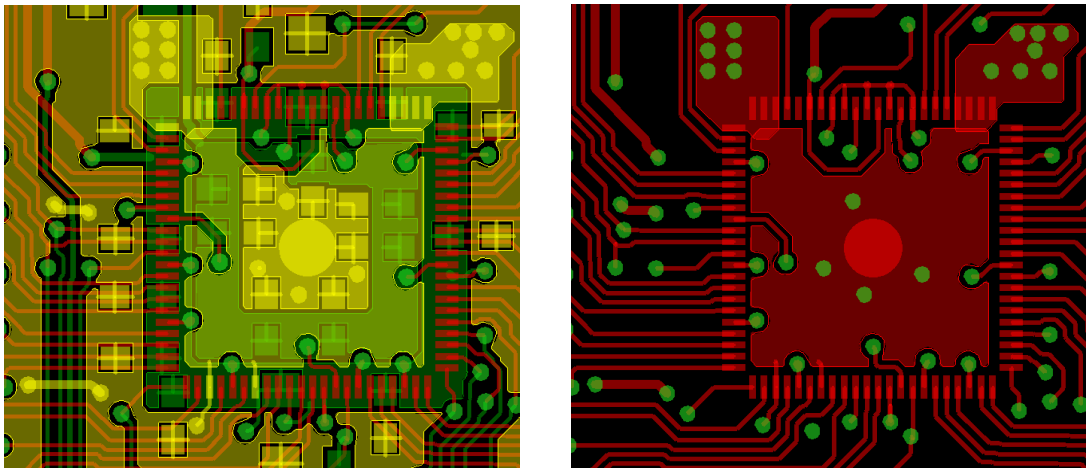
Obrázek 7.2: PCB vývojového kitu s BF504F

je zároveň zemnící. Ve vrstvě BOT je okolo prokovu rozlitá měď sloužící k uzemnění blokovacích kondenzátoru pro jádro (1,4V), jež jsou připájeny na čtvercovém rámečku okolo. Na větším rámečku je již napětí pro periferie (3.3V), z kterého jsou kondenzátory uzemněny do vnější rozlité mědi kolem procesoru. Vnitřní zemnící plochy pod procesorem jsou spojeny s vnější zemnící plochou na několika místech. Procesor má několik pinů, které nejsou uvnitř pouzdra k ničemu připojeny. Pod těmito piny jsou vytvořeny dvě rozlité měděné plochy(na obr. (7.3(b)) na procesoru nahoře tvoří takové "uši"), sloužící k propojení zemnících ploch ve vrstvě TOP a BOTTOM. Třetím bodem, kde jsou zemnící plochy spojeny, je zemnící pin umístěný dole.

Je zřejmé, že rozlitá měď okolo procesoru ve vrstvě BOTTOM, která by měla primárně sloužit jen k zemnění, je značně přerušovaná vodiči. Ve snaze co nejvíce snížit impedanci zemnících smyček, jsou na několika místech vytvořeny propojky přes vrchni vrstvu TOP. Dvě tyto propojky jsou vidět v levé části obrázku7.3(a).

Na obr.(7.5) je vyobrazen celkový 3D pohled na navržený vývojový kit, pro přehlednost v něm není vyobrazena rozlitá měď.

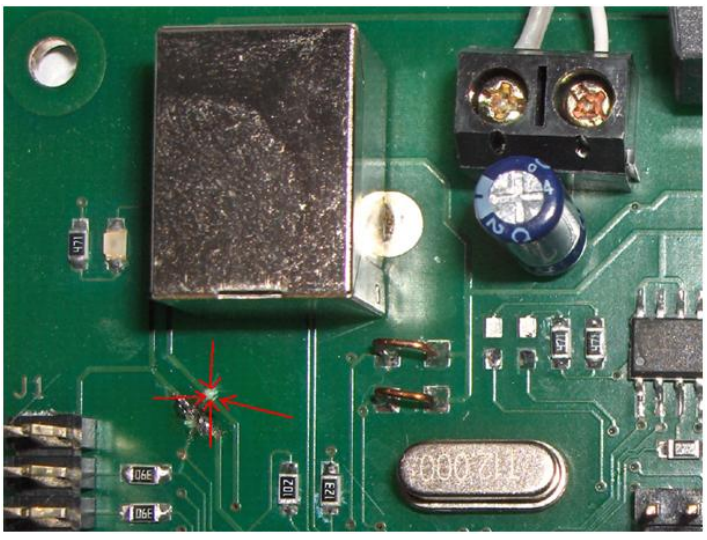
Plošné spoje byly vyrobeny ve dvou verzích. V první verzi byly nalezeny některé chyby.



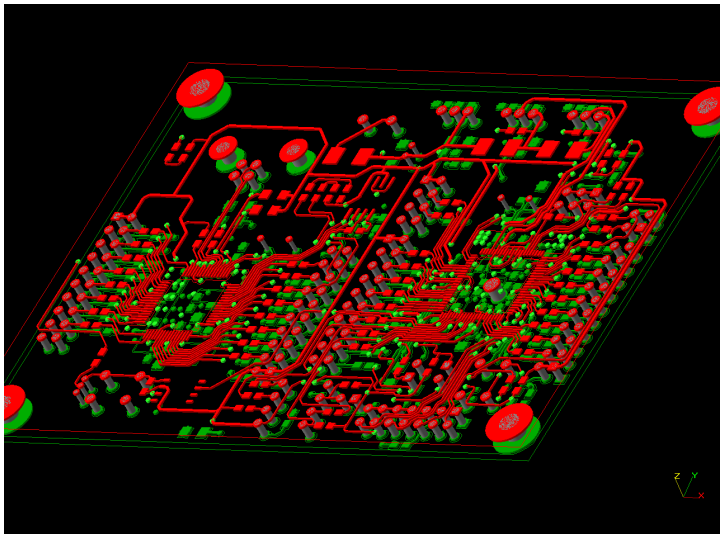
(a) (b)

Obrázek 7.3: Blokování procesoru BF504F na vývojovém kitu

Obvod FT2232HL si interně vytváří napětí 1,8V, které je vyvedeno na pin, z kterého je dále rozvedeno k dalším pinům již na plošném spoji. V první verzi vývojové desky byl tento pin spojen s napětím 1,8V z regulátoru MC33269T. Proto musí být u desek verze 1 tento spoj přerušen dle. (7.4).



Obrázek 7.4: Spoj ve verzi PCB V1.0, který je nutné přerušit



Obrázek 7.5: 3D pohled na vývojový kit s BF504F

7.3 Použité integrované obvody

BF504F

Popis obvodu v (4.2).

Ostatní integrované obvody

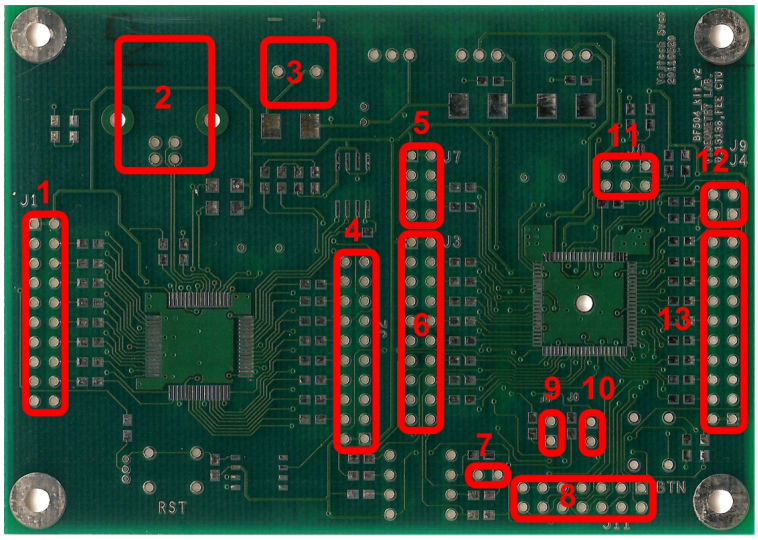
Ostatní použité integrované obvody i jejich počet je shodný jako u vývojového kitu s BF53x a jsou již popsány v (6.3)

7.4 Popis vyvedených konektorů

Popis jednotlivých konektorů umístěných na vývojovém kitu je vázán na obr. (7.6) a je následující:

1. PORT A (FT2232HL)
2. Konektor USB
3. Napájecí svorkovnice

4. PORT B (FT2232HL)
5. Porty PHx
6. Porty PFx
7. WP# (25c256)
8. JTAG
9. EXTCLK
10. NMI#
11. BMODE
12. TWI
13. Porty PGx



Obrázek 7.6: Popis konektorů na vývojovém kitu s BF504F

7.5 Bootování procesoru

Procesor je schopný bootovat z více zdrojů, což je nastaveno pomocí pinů BMODE [2-0] , jež jsou pomocí pull-up rezistorů připojeny na napájecí napětí. Pomocí jumperů je možné

piny uzemnit, čímž je možné vyzkoušet různé bootovací režimy. Patice pro paměť 25c256 je připojena na SPI0. Pro bootování z jiného zdroje, než-li z paměti 25c256, je vhodné mít tuto paměť z patice odstraněnou, vzhledem k tomu, že SPI0 sdílí GPIO piny portu PFx. Nabootováno můž být z:

- Žádné bootování BMODE [000]
- Interní FLASH v asynchronním módu BMODE [001]
- Interní FLASH v synchronním módu BMODE [010]
- SPI0 (BF504F jako master) BMODE [011]
- SPI0 (BF504F jako slave) BMODE [100]
- PPI (BF504F jako slave) BMODE [101]
- Rezervováno
- UART0 (BF504F jako slave) BMODE [111]

Nahrání nového programu do paměti 25c256 je shodné jako v (6.5), ale příslušné signálové vodiče se musejí na headerech ručně propojit.

7.6 Oživení kitu

Pokud byly veškeré součástky správně zapájeny, měl by být kit okamžitě připraven k použití. Pomocí jumperu je třeba zvolit na headeru J8 bootování z SPI paměti dle tab. (4.3). Na lab. zdroji nastavit napětí 5V a max. dovolený proud 100 mA a připojit kit k PC. Ověřit funkčnost obvodu FT2232HL viz (5.2). K ověření funkce procesoru slouží program "Test_BF504_KIT.ldr"umístěný na přiloženém CD. Po nahrání programu do paměti dle (5.3) a resetování kitu by měla LED blikat frekvencí 1Hz. Pokud vše takto funguje, je tím ověřena funkce jak procesoru BF504F, obvodu FT2232HL tak i bootovací paměti.

7.6.1 Pájení BF504F

Procesor je svým pouzdrem určen spíše k přetavení v peci, či jiným průmyslovým metodám. Na obr.(7.7) je procesor zobrazen spolu s jeho rozměry. Z obr.(7.7(b)) je zřejmé, že pouzdro nemá žádné "nožičky" jako pouzdro LQFP u procesoru BF53x, ale přístupné jsou jen čelní plošky o rozměrech 0,2 x 0,15 mm. Ruční pájení se proto z počátku zdálo jako neuskutečnitelné, proto byl první pokus o připájení procesoru uskutečněn v průběžné peci Mistral 260 metodou přetavení. Z finančních důvodů nebyly vyráběny pastovací šablony, proto muselo být nanášení pájecí pasty na pady prováděno ručně, což bylo velmi pracné, nerovnoměrné a časově náročné. Procesor se povedlo připájet až na čtvrtý průjezd pecí, v každém mezikroku musela být na některé piny opět nanesena pájecí pasta. Pokud by byly k dispozici pastovací šablony, byla by tato metoda nejrychlejším řešením, v opačném případě se ukázala jako nevhodná. Nakonec bylo zpětně přistoupeno k ručnímu pájení, kde se osvědčil následující postup:

1. Na PCB nanést dostatečné množství JELLY pájecího gelu(jak na pady, tak i na velký termální pad)

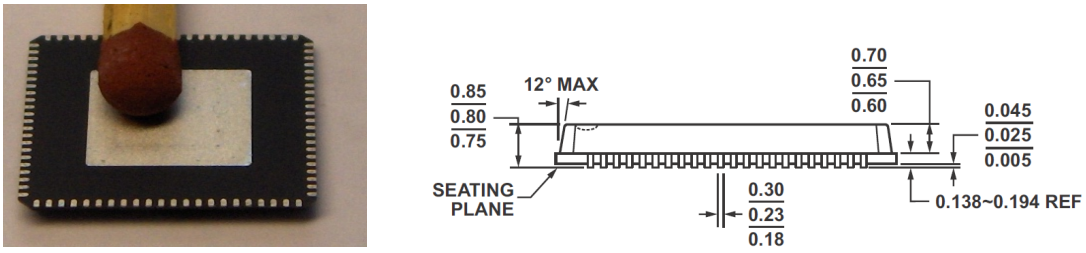
2. Procesor umístit na PCB a ve dvou protiběžných rozích zapájet. Zkratování několika padů v tuto chvíli ničemu nevadí.

3. PCB s uchyceným procesorem umístit pod mikroskop.

4. Pomocí páječky s velmi ostrým hrotem a s minimálním množstvím cínu postupně zapájet veškeré piny.

5. Otočit PCB a do vyvrtaného otvoru pod pouzdrem nanést malé množství pájecího gelu a pad důkladně propájet. Vzhledem k tomu, že se jedná o hlavní uzemňovací pin, musí být jeho připájení velmi důkladné. Úspěšné zapájení padu je možné uvažovat tehdy, pokud se cín vtáhne dovnitř otvoru a vytvoří "kalíšek".

6. Jednotlivé zapájené piny důkladně pod mikroskopem zkontrolovat !



(a) (b)

Obrázek 7.7: Pouzdro BF504F

7.7 Technické specifikace

Napájecí napětí: 5 [V]

Odebíraný proud: 38 [mA] (při CCLK=400MHz,SCLK=100MHz,žádná zátěž na pinech)

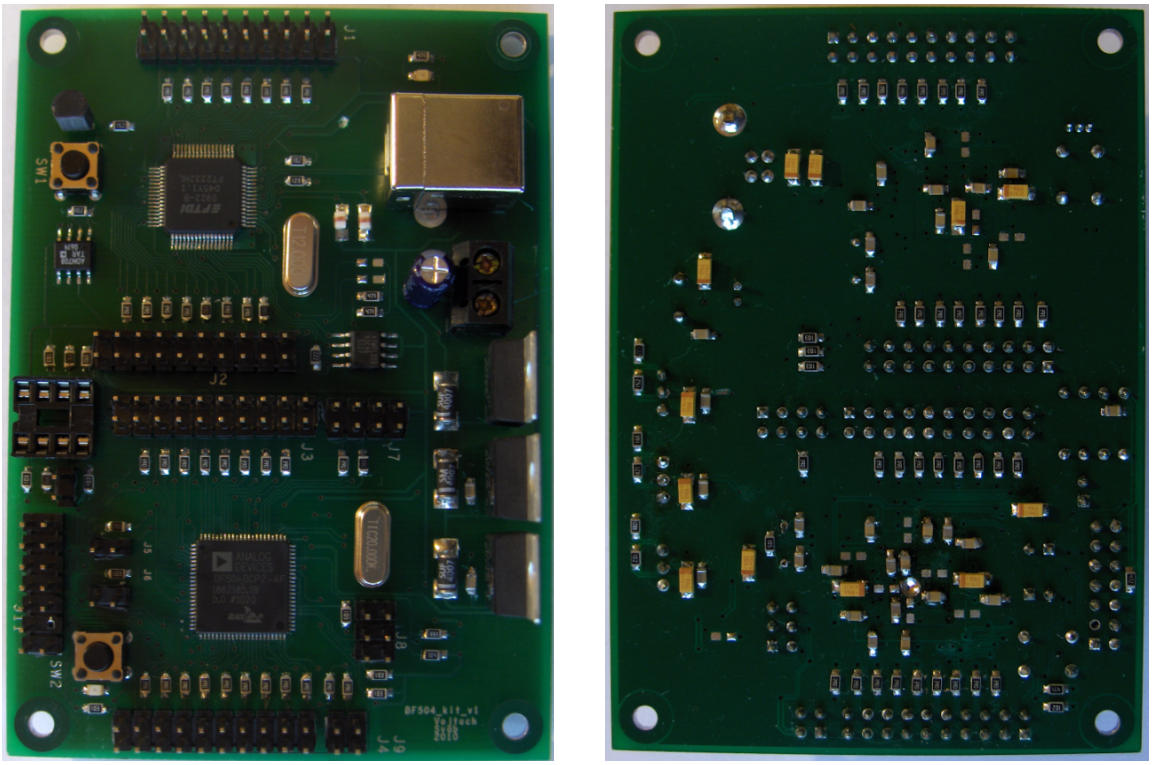
Debuggovací rozhraní: JTAG

Komunikace s PC: USB

Rozměry DPS: 70x97 [mm]

7.8 Fotodokumentace

Na obr.(7.8) je vyobrazen vyrobený vývojový kit.



(a) (b)

Obrázek 7.8: Zhotovený vývojový kit s BF504F

Kapitola 8

Návrh HW SMART kamery

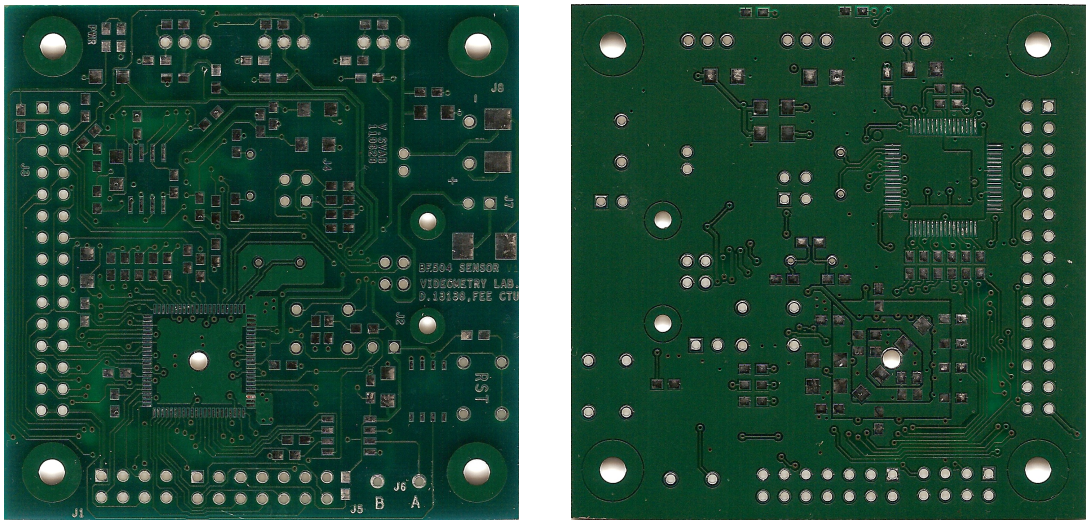
Po úspěšném ověření funkcí procesoru BF504F na vývojovém kitu, mohlo být přistoupeno k samotnému návrhu modulu SMART kamery. Během návrhu byly využity zkušenosti při návrhu kitu, zejména pak návrh plošného spoje v okolí procesoru. Na vývojovém kitu byla odladěna komunikace mezi BF504F na bázi FIFO rozhraní, jejíž signálové vodiče jsou na PCB SMART kamery rozvedeny. SMART kamera disponuje konektorem pro přímé připojení modulu s CMOS sensorem, jež byl vyvinut na katedře měření. Do CMOS sensoru je nutné přivádět vstupní hodinový signál. Pro tento účel je uplatněn pin EXTCLK procesoru BF504F. Tento pin může poskytovat, v závislosti na jeho konfiguraci, frekvenci periferií(SCLK), či jen frekvenci krystalu. Pin je tedy nakonfigurován tak, aby na něm byla frekvence krystalu, čímž může být přímo použit k buzení CMOS sensoru. Z tohoto důvodu je použit 16MHz krystal. Procesor BF53x tímto pinem nedisponuje, proto by k taktování CMOS sensoru muselo být použito externího hradla buzeného přímo z krystalu, jak bylo navrženo v [16].

8.1 Návrh PCB

Obecně je při návrhu PCB vhodné umisťovat veškeré integrované obvody na jednu stranu plošného spoje. Procesor BF504F a obvod FT2232HL bohužel mají opačně indexované datové vodiče. Propojení obou obvodů umístěných na stejné straně plošného spoje by sice bylo možné, nicméně by zabíralo mnoho místa. Další možností by bylo jejich přímé propojení s tím, že data by se orotovala softwarově uvnitř procesoru. Tato možnost by nicméně vedla k nadbytečné softwarové režii, proto jsou oba obvody umístěny na rozdílné

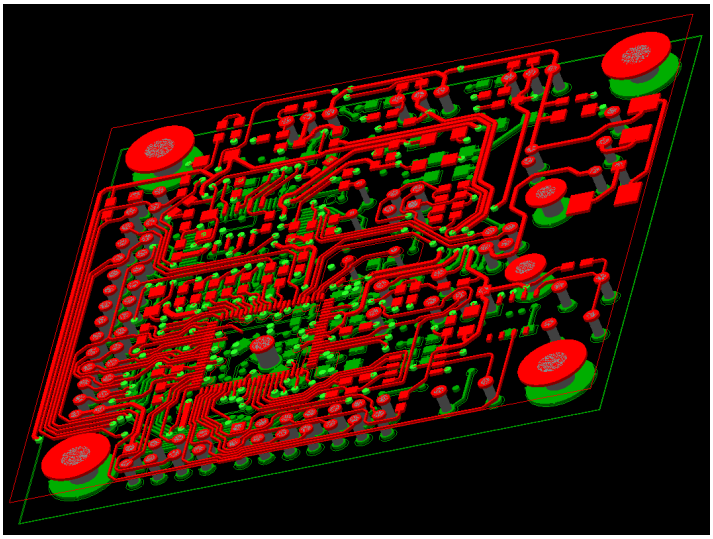
straně plošného spoje. Při umísťování procesoru BF504 a jeho blokování již bylo čerpáno ze zkušeností získaných při návrhu vývojového kitu z kap. (7).

Na obr. jsou umístěny pohledy na navržený plošný spoj. Na obr.(8.2) je vyobrazen celkový 3D pohled na navrženou SMART kameru, pro přehlednost v něm není vyobrazena rozlitá měď.



(a) (b)

Obrázek 8.1: PCB SMART kamery

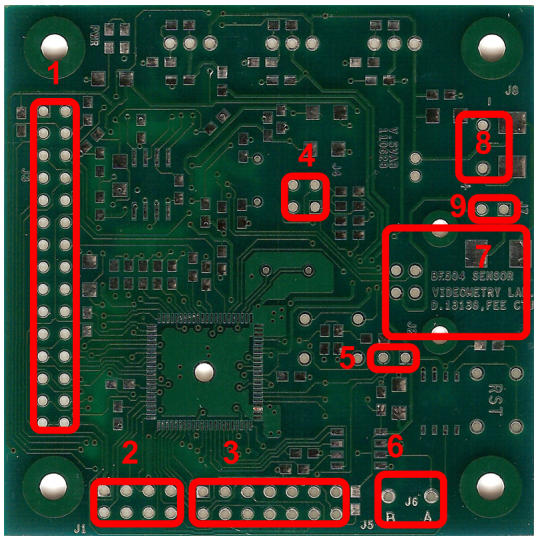


Obrázek 8.2: 3D pohled na PCB SMART kamery

8.2 Popis vyvedených konektorů

Popis jednotlivých konektorů umístěných na vývojovém kitu je vázán na obr. (8.3) a je následující:

1. CMOS senzor
2. SPI
3. JTAG
4. Port H [1-0]
5. Jumper pro volbu bootování
6. Svorkovnice RS-485
7. USB
8. Napájecí svorkovnice
9. Propojka pro umožnění napájení přes USB



Obrázek 8.3: Popis vyvedených konektorů SMART kamery

8.3 Postup zapájení a oživení SMART kamery

Procesor je vhodné zapájet dle postupu (7.6.1). Piny stabilizátorů je nutné před zapájením ohnout o 90° dle obr.(8.4(c)).

8.4 Technické specifikace

Napájecí napětí: 5 [V]

Odebíraný proud: 80 [mA] (při CCLK=400MHz,SCLK=100MHz)

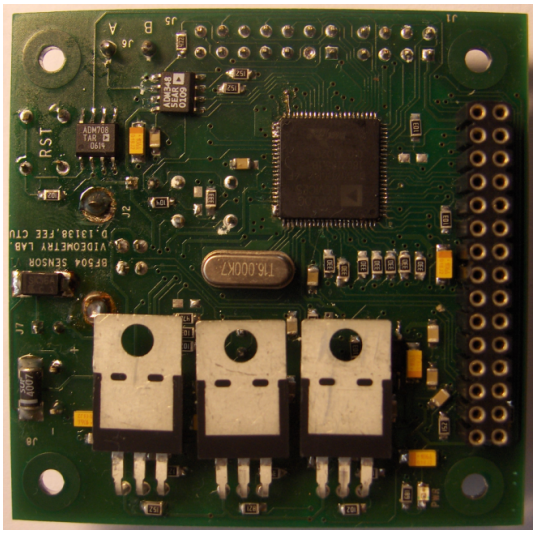
Debuggovací rozhraní: JTAG

Komunikační rozhraní: USB, RS-45

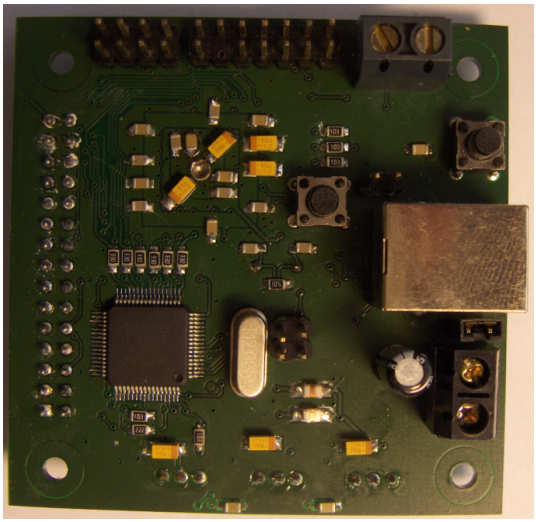
Rozměry DPS: 60x60 [mm]

8.5 Fotodokumentace

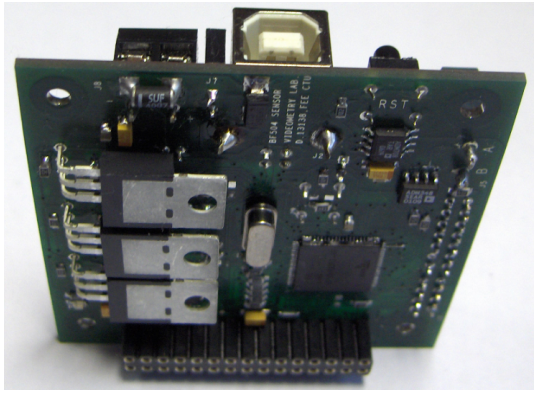
Na obr.(8.4) je vyobrazena vyrobená SMART kamera.



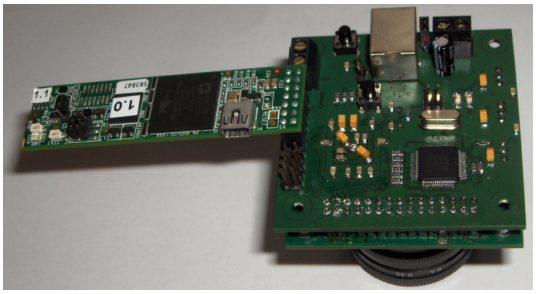
(a)



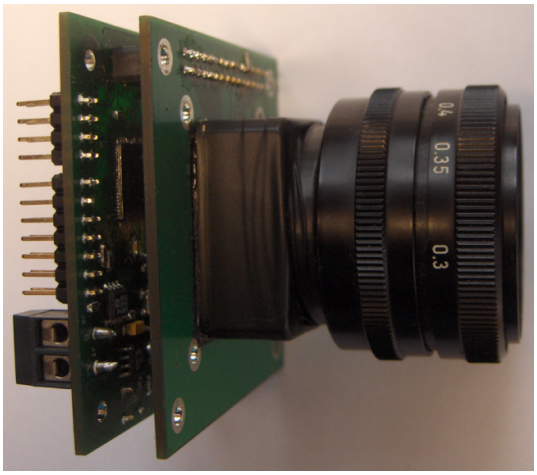
(b)



(c)



(d)



(e)

Obrázek 8.4: Zhotovená SMART kamera

Kapitola 9

Návrh firmware SMART kamery

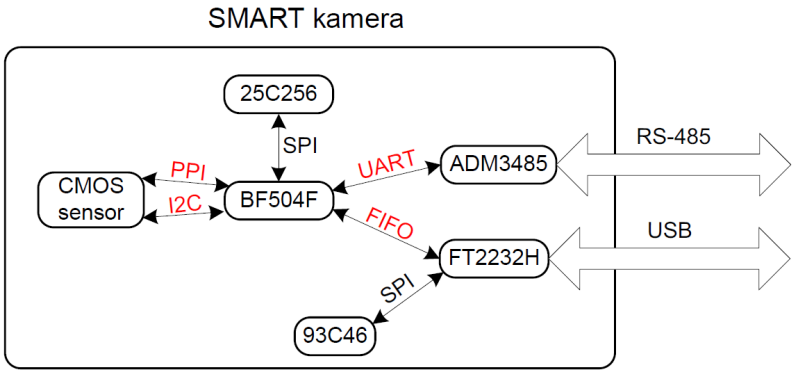
K modulu SMART kamery je možné připojit CMOS sensor MT9M001 typu rolling shutter. V případě, že by byl připojen jiný typ CMOS snímače, je nutné jej ručně inicializovat. Inicializace je možná funkcemi pro komunikaci přes I2C, dostupné přes DLL knihovnu. Potřebné registry, jež je nutné inicializovat je nutné vyhledat v příslušné dokumentaci daného snímače.

9.1 Komunikační sběrnice

V SMART kameře je použito několik druhů sběrnic, jak je zřejmé z obr.(9.1). Komunikace po některých sběrnicích je zcela v režii příslušných integrovaných obvodů a nelze jejich chod ovlivňovat, značeny uvnitř nákresu SMART kamery černě. Některé sběrnice musely být zcela naprogramovány, či muselo být využito hardwarových modulů uvnitř procesoru, značeno červeně.

9.1.1 Komunikace s CMOS sensorem

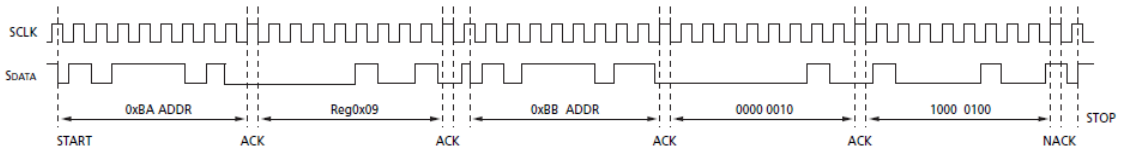
Přenos konfiguračních dat s CMOS sensorem probíhá na sběrnici I2C, která je procesorem hardwarově podporována. Přenos dat je uskutečněn pomocí PPI, z kterého jsou data odebírána pomocí DMA kanálu.



Obrázek 9.1: Použité sběrnice v SMART kameře

Vyčítání konfiguračních dat

Pro vyčítání konfiguračních dat po I2C je třeba využít přenosu s tzv. opakovaným startem. Přenos s opakovaným startem se obecně využívá k vyčítání registrů z různých obvodů. V praxi to znamená, že nejdříve je masterem zaslán start bit a následně adresa registru, z kterého mají být data vyčtena. Dále, aniž by byl zaslán stop bit, se master přepne do přijímacího módu a opět zašle start bit, načež slave zašle obsah požadovaného registru a až nyní je zaslán stop bit. Časování komunikace je zřejmé z obr.(9.2). Ve zdrojovém kódu (9.1) je implementace názorně uvedena. Na řádku 6 je iniciováno odeslání dat. Vzhledem k tomu, že je I2C v tomto místě nakonfigurováno do restart módu, nebude automaticky vložen stop bit. Na řádku 23 je restart mód deaktivován, což znamená, že po úspěšném příjmu požadovaného počtu bytů bude stop bit vložen.



Obrázek 9.2: Vyčítání dat z CMOS sensoru

Zápis konfiguračních dat

Zápis do CMOS sensoru probíhá standardním způsobem, kde je nejdříve zaslána adresa zapisovaného registru a následně příslušná data.

Zdrojový kód 9.1: Příjem dat po I2C

```
1         //---------- odeslani adresy registru --------
2         unsigned short Bytes_Transfer_Count = 1;
3     *pTWI_MASTER_CTL |= (Bytes_Transfer_Count<<6); // Pocet bytu k odeslani
4     *pTWI_XMT_DATA8 = CMOS_Reg; // Odeslani adresy registru
5     asm("ssync;");
6     *pTWI_MASTER_CTL |= RSTART | MEN; // Aktivace restart modu a odeslani dat
7         asm("ssync;");
8           int i=0;
9     while(!(*pTWI_INT_STAT &= XMTSERV)){
10        Wait_10ns(I2C_DELAY_10ns);
11        i++;
12        if(i == 3) break;
13    }
14    i=0;
15    while(!(*pTWI_INT_STAT &= MCOMP)){ // Master transfer complete ?
16        Wait_10ns(I2C_DELAY_10ns);
17        i++;
18        if(i == 3) break;
19    }
20    *pTWI_MASTER_CTL |= MDIR; // I2C jako Rx
21    *pTWI_INT_STAT = MCOMP; // Nulovani status bitu
22    Bytes_Transfer_Count = 2; // Pocet bytu k prijmu
23    *pTWI_MASTER_CTL &= (~RSTART); // Vypnuti restart modu
24    *pTWI_MASTER_CTL |= (Bytes_Transfer_Count<<6);
25    asm("ssync;");
26    i=0;
27    while(!(*pTWI_INT_STAT &= MCOMP)){ // Cekani na dokonceni prijmu
28        Wait_10ns(I2C_DELAY_10ns);
29        i++;
30        if(i == 3) break;
31    }
32    *CMOS_Reg_Data = *pTWI_RCV_DATA8; // Ulozeni prijmutych dat
33    asm("ssync;");
34    *CMOS_Reg_Data = (*CMOS_Reg_Data<<8) | *pTWI_RCV_DATA8;
```

9.1.2 Návrh komunikace na bázi FIFO mezi BF504F a FT2232H

Pro maximalizaci komunikační rychlosti mezi procesorem BF504F a obvodem FT2232HL bylo zvoleno rozhraní asynchronního FIFO módu. Procesor BF504F oproti BF53x bohužel nedisponuje EBIU rozhraním, proto musela být komunikace mezi obvody řešena softwarově. Časování komunikace bylo navrhováno dle Lit. [4]. Program pro komunikaci byl nejdříve sestaven v jazyce C, přesně dle vývojového diagramu na obr. (9.3). Bohužel výsledná rychlost přenosu dat byla nedostatečná a možnost optimalizace kódu byla minimální. Pro přenos obrazových dat, je nicméně třeba dosáhnout co možná největší přenosové rychlosti(cca 8 MB/s). Při návrhu sotwarového FIFO rozhraní jsou kladeny vysoké požadavky na precizní časování, které je pomocí jazyka C nedosažitelné. Z tohoto důvodu bylo přistoupeno k napsání komunikace v assembleru. Komentovaný zdrojový kód je k nahlédnutí v (9.2). Funkce je taktéž navržena dle vývojového diagramu (9.3), nicméně v assembleru je možná "předpříprava"dat, což je zřejmé na řádku 60, kde se následně v požadovaný okamžik data jen "překlopí"na výstupní piny. Tím, že jsou data takto předpřipraveny, je ještě zvýšena komunikační rychlost.

Zdrojový kód 9.2: Softwarový asynchronní přenos v assembleru

```
1  //*************************************************************************
2  // _FT2232HL_ASM_SEND(unsigned char *Data_To_Send, unsigned short Data_Tx_Count);
3  //*************************************************************************
4  .align 4;
5  .global __FT2232HL_ASM_SEND;
6  #define FT2232HL_ASM_SEND_STACK 38
7  __FT2232HL_ASM_SEND:
8      LINK FT2232HL_ASM_SEND_STACK;
9      //Ulozeni Preserved registru
10     [--SP] = (R7:4, P5:3);
11     // Ulozeni adresy prvniho prvku pole do P2
12     P2 = R0;
13     //Ulozeni poctu byte k odeslani
14     [FP - 4] = R1;
15     // Pokud Data_Tx_Count == 0, nic neposilej
16     R0 = 0;
17     CC = R1 ==R0;
18     IF CC JUMP LABEL_END;
```

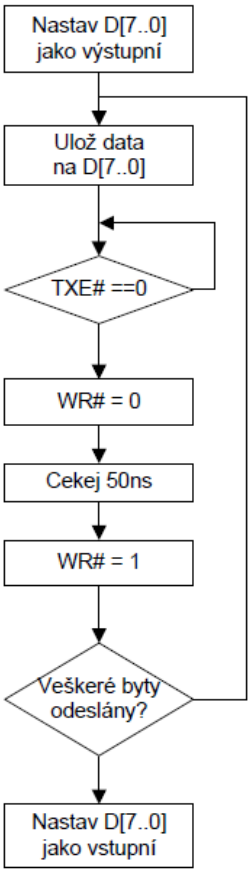
```
19      // Nastaveni datovych pinu jako vystupni
20      R0.L = ~(FT_DATA_PINS);
21      P0.L = lo(PORTFIO_INEN); // v P0 je adresa PORTFIO_INEN
22      P0.H = hi(PORTFIO_INEN);
23      R1 = [P0]; // v R1 je hodnota PORTFIO_INEN
24      R2 = R0 & R1;
25      W[P0] = R2;
26      R0.L = FT_DATA_PINS;
27      P0.L = lo(PORTFIO_DIR); // v P0 je adresa PORTFIO_DIR
28      P0.H = hi(PORTFIO_DIR);
29      R1 = [P0]; // v R1 je hodnota PORTFIO_DIR
30      R2 = R0 | R1;
31      W[P0] = R2;
32      // Data sa na datove vodice predpripravi jiz nyni
33      // *pPORTFIO = Data_To_Send[0];
34      P0.L = LO(PORTFIO);
35      P0.H = HI(PORTFIO);
36      R0 =B[P2++];
37      W[ P0 ] = R0;
38      // Do P0 se ulozi pozadovany pocet bytu k odeslani
39      P0 = [FP - 4];
40      LOOP loop_name LC1 = P0 ; // Autoinicializace LC1
41      // Kvuli rychlosti bude PORTG stale v P4 => P4 se nesmi jiz nikde prepsat!!
42      P4.h = hi(PORTGIO);
43      P4.l = lo(PORTGIO);
44      //kvuli rychlosti bude PORTG_CLEAR stale v P3 => P3 se nesmi jiz nikde prepsat!!
45      P3.h = hi(PORTGIO_CLEAR);
46      P3.l = lo(PORTGIO_CLEAR);
47      //kvuli rychlosti bude FT_WR_non stale v R3 => R3 se nesmi jiz nikde prepsat!!
48      R3 = FT_WR_non;
49      // Zacatek vysilaci smycky
50      LOOP_BEGIN loop_name;
51      FT2232HL_SEND_JUMP_1:
52      // Test zda-li TXE# == 0, pokud neni, skoc zpet na FT2232HL_SEND_JUMP_1:
53      R0.L = W[P4];
54      CC = BITTST ( R0, FT_TXE_non_bit_pos);
55      IF CC JUMP FT2232HL_SEND_JUMP_1;
```

```
56      // #WR na 0
57      W[P3] = R3;
58      // wait min. 50ns
59      nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;nop;
60      // Predpripraveni dat na vystupni piny, part 1
61      P1.L = LO(PORTFIO);
62      P1.H = HI(PORTFIO);
63      R1 = B[P2++];
64      // #WR na 1
65      P0.h = hi(PORTGIO_SET);
66      P0.l = lo(PORTGIO_SET);
67      W[P0] = R3;
68      // Preklopeni dat na vystupni piny, part 2
69      W[ P1 ] = R1;
70      // Konec smycky
71      LOOP_END loop_name;
72      // Nastaveni datovych pinu jako vstupnich
73      R0.L = ~(FT_DATA_PINS);
74      P0.L = lo(PORTFIO_DIR); // v P0 je adresa PORTFIO_DIR
75      P0.H = hi(PORTFIO_DIR);
76      R1 = [P0]; // v R1 je hodnota PORTFIO_DIR
77      R2 = R0 & R1;
78      W[P0] = R2;
79  LABEL_END:
80      /*Navraceni Preserved registru*/
81      (R7:4, P5:3) = [SP++];
82      UNLINK;
83      RTS;
84  __FT2232HL_ASM_SEND.end:
```

S taktou navrženou komunikací lze dosahovat rychlosti kolem 7,180 MB/s.

9.1.3 Implementace RS-485

Rozhraní RS-485 je založeno na využití UART modulu, jež je v procesoru k dispozici. Na obr. (9.4) je vnitřní struktura obvodu ADM3485 a je zřejmé, že zde jsou čtyři signálové piny které je třeba řídit. Signály Tx a Rx UARTu jsou přímo propojeny s piny DI a RO,



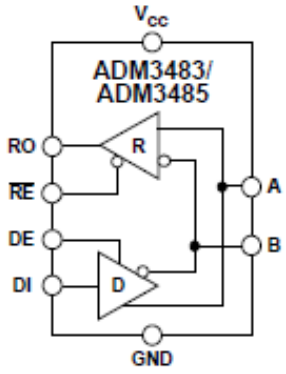
Obrázek 9.3: Vývojový diagram komunikace s FT2232H

což je zřejmé ze schema SMART kamery (17.5). K ovládání pinů RE# a DE, jsou využity GPIO piny.

Obvod ADM3485 je možné z důvodu šetření energie přepnout do sleep módu, nicméně tato funkcionalita není využita. Probuzení ze sleep módu trvá téměř 3 μs, což je při požadavku na co nejmenší odezvu SMART kamery poměrně dlouhá doba. Pokud by byl ADM3485 ve sleep módu, nemohl by ani přijímat data ze sběrnice.

9.1.4 Návrh packetové komunikace

Packety mohou být zasílány jak po USB, tak přes RS-485. Z pohledu programátora firmware je použití jednotlivých rozhraní totožné. Metody pro odeslání packetů přes USB, či RS-485 byly napsány tak, aby měly veškeré argumenty funkcí shodné. Struktura packetu je vždy dodržována dle obr. (11.1).



Obrázek 9.4: Vnitřní struktura ADM3485

Vysílání packetů po USB

Zaslání packetu je iniciováno funkcí `FT2232HL_Send_Packet(..)` v které je packet sestaven cyklickým voláním funkce `_FT2232HL_ASM_SEND(..)` dle (9.3).

Zdrojový kód 9.3: Metoda pro zaslání packetu po USB

```
1 unsigned char FT2232HL_Send_Packet(int Packet_Head,char Packet_ID_Cam,
2             char Packet_ID_Message,int Packet_Data_Lenght,char * Packet_Data){
3         _FT2232HL_ASM_SEND((char*)&Packet_Head,sizeof(Packet_Head));
4         _FT2232HL_ASM_SEND(&Packet_ID_Cam,1);
5         _FT2232HL_ASM_SEND(&Packet_ID_Message,1);
6         _FT2232HL_ASM_SEND((char*)&Packet_Data_Lenght,sizeof(Packet_Data_Lenght));
7         _FT2232HL_ASM_SEND(Packet_Data,Packet_Data_Lenght);
8         return 0;
9 }
```

Vysílání packetů po RS-485

Pro vysílání packetů po RS-485 je využito autonomního přenosu dat přes DMA, čímž nedochází k zatěžování jádra. Packety se odesílají pomocí metody definované v (9.4)

Zdrojový kód 9.4: Metoda pro zaslání packetu přes DMA po RS-485

```
1 RS_485_Send_Packet_DMA(int Packet_Head,char Packet_ID_Cam,char Packet_ID_Message,
2 int Packet_Data_Lenght,char * Packet_Data);
```

Jednotlivé argumenty jsou uvnitř metody překopírovány do bufferu, který je následně předán DMA kanálu k odeslání.

Zpracování příchozích packetů

Aby nedocházelo ke ztrátě příchozích dat po USB, je v maximální možné míře využíván hardwarový buffer v obvodu FT2232HL. Ve vyhrazeném čase je jen buffer vyčten metodou pollingu. V případě rozhraní RS-485, již hardwarový buffer není k dispozici. K tomuto účelu je vytvořen softwarový kruhový buffer pro příchozí data z RS-485. V paměti je k tomuto účelu alokována paměť o velikosti 2048 kB. K automatickému přenosu dat z RS-485 do paměti je nakonfigurován DMA kanál a to do "autobuffer"módu. V tomto módu se při zaplnění bufferu, přepíše adresový pointer `DMAx_CURR_ADDR` hodnotou z `DMAx_START_ADDR`. V konečném důsledku to znamená, že při zaplnění bufferu, jsou přepisována data opět od začátku.

K vyčítání dat z bufferu byla vytvořena metoda dle (9.5). Globální proměnná `UART_Rx_Buffer_Position` odkazuje na aktuální pozici v bufferu, z které lze data vyčítat.

Zdrojový kód 9.5: Vyčtení 1 byte z RS-485 bufferu

```
1  unsigned char RS_485_READ_BYTE_From_Buffer(unsigned char * Data_Read){
2    // Pocet nevyctenych bytu z bufferu
3          unsigned short DMA8_Count_Difference = *pDMA8_X_COUNT - *pDMA8_CURR_X_COUNT;
4          // Pokud TRUE, nebyl prijat zadny novy byte do bufferu
5          if((DMA8_Count_Difference == UART_Rx_Buffer_Position) || (*pDMA8_CURR_ADDR == 0)){
6                  return 1;
7              }else{
8                      *Data_Read = UART_Rx_Buffer[UART_Rx_Buffer_Position];
9                          UART_Rx_Buffer_Position++;
10                         // Cekej, aby se stihl buffer pred pristim volanim teto fce doplnit
11                         Wait_10ns(RS485_10ns_BETWEEN_BYTE_READ);
12                         if(UART_Rx_Buffer_Position == RS485_RX_BUFFER_SIZE){
13                                 UART_Rx_Buffer_Position = 0;
14                                 asm("nop;");
15                         }
16                 return 0;
17         }
18 }
```

Pro vyčítání celého packetu z bufferu je vytvořena metoda `RS_485_Read_Packet()`, která cyklicky žádá `RS_485_READ_BYTE_From_Buffer(..)` o jednotlivé byty a utváří celý packet. Pro vyčítání packetů z USB je vytvořena metoda `FT2232HL_Read_Packet()`, fun-

gující na stejném principu, s tím rozdílem, že jednotlivé byty jsou vyžádány pomocí `FT2232HL_READ_BYTE(..)`.

Packetová komunikace je nastavena tak, že pokud přijde packet přes rohraní USB, budou požadovaná data po USB taktéž odeslána, stejné platí i pro RS-485. Pokud je daná SMART kamera nakonfigurována jako bridge USB/RS-485, budou veškeré přijaté packety z USB, které nejsou určeny pro tuto kameru přeposlány na RS-485. Veškeré přijaté packety po RS-485, které nejsou určeny pro tuto kameru, budou přeposlány po USB do PC. Pokud SMART kamera jako bridge USB/RS-485 nakonfigurována není, budou veškeré packety, které nejsou určeny pro danou kameru ignorovány. Veškeré přijaté packety, nezávisle na rozhraní přes které přišly, a které jsou určeny pro danou SMART kameru jsou rozkódovány a dále zpracovány.

9.2 Zachycení snímku CMOS sensorem

Propojení CMOS sensoru s procesorem je řešeno přes PPI rozhraní, jež je procesorem hardwarově podporováno. Procesor nedisponuje dostatečně velikou pamětí k tomu, aby se do ní mohl uložit kompletní snímek z CMOS sensoru o velikosti 1310720 bytů. Snímek se tedy musí do PC posílat po řádcích a to následujícím způsobem. Pro práci se snímkem jsou staticky alokovány dva buffery (A a B) o velikosti jednoho řádku (1280 bytů). Z CMOS sensoru se přes DMA kanál vyčte první řádek a uloží se do bufferu A. Ihned po příjmu prvního řádku je DMA kanál překonfigurován pro ukládání dat do bufferu B. Během ukládání druhého řádku do bufferu B, je buffer A zasílán do PC. Po příjmu druhého řádku z CMOS sensoru je DMA opět překonfigurován pro ukládání dat do bufferu A. Následně je do PC zaslán buffer B. Tímto způsobem přepínáním bufferů jsou do PC zaslány veškeré řádky z CMOS sensoru.

Z předchozího odstavce je zřejmé, že přenos jednoho řádku do PC musí bezpodmínečně trvat méně, než-li příjem jednoho řádku z CMOS sensoru DMA kanálem. Pokud by tomu tak nebylo, byly by některé řádky ztraceny, čímž by došlo ke znehodnocení výsledné fotografie. Toto je důvod, proč bylo v kapitole (9.1.2) cílem dosáhnout co možná nejvyšší přenosové rychlosti mezi BF504F a FT2232H.

CMOS sensor je v daném zapojení taktován 16 Mhz, nicméně z kapitoly (9.1.2) je zřejmé, že maximální přenosová rychlost mezi BF504F a FT2232H je 7,180 MB/s. Aby mohla být data z CMOS sensoru přímo vyčítána, musela by tedy být přenosová rychlost

minimálně 16 MB/s. Z tohoto důvodu je nutné rychlost toku dat z CMOS sensoru o více jak polovinu zpomalit, čehož je dosahováno prodloužením horizontálního zatemňovacího intervalu v CMOS sensoru.

9.3 Algoritmus vyhledání jednoho markeru

Pokud je třeba v prostoru detekovat pozici libovolného předmětu, je nutné daný předmět vhodným způsobem odlišit od pozadí. V případě, že bude na daném předmětu nějaký bod světlejší než-li pozadí, lze pomocí níže uvedené metody určit střed světlého bodu, čímž je určena pozice předmětu.

9.3.1 Princip metody

Každé tuhé těleso má hmotný bod, na který pokud působíme určitou silou, tak vyvolá stejné silové účinky, jakoby jsme působili na celé těleso. Tento bod se nazývá těžiště. Dále lze říci, že těžiště tuhého tělesa je působiště tíhové síly, která působí na těleso v homogenním tíhovém poli. Těžiště tělesa lze v jedné souřadnici vypočítat dle 9.1.

$$x_T = \frac{1}{m} \int x dm \tag{9.1}$$

Pro výpočet těžiště z n diskrétních hmotných bodů lze integrál nahradit sumou dle 9.2.

$$x_T = \frac{\sum_{i=1}^{n} x_i m_i}{m} \tag{9.2}$$

Při snímání markeru CMOS senzorem, nelze uvažovat žádné hmotné body. Nicméně nahrazením jednotlivých tíhových hmotných bodů u tělesa, za jasovou intenzitu jednotlivých bodů v markeru, lze určit jasový střed markeru dle 9.3.

$$x_T = \frac{\sum_{i=1}^{n} x_i I_i}{\sum_{i=1}^{n} I_i} \tag{9.3}$$

Tato metoda již byla publikována např. v Lit.[18] nebo [16], proto zde již o podrobné implementaci nebude psáno.

9.4 Algoritmus vyhledání více markerů

Při hledání více markerů v obraze se v tomto algoritmu taktéž využívá teoretického základu popisovaného v předchozí kapitole. Zatímco se v předešlém algoritmu počítalo těžiště nad celým obrazem, je zde těžiště počítano nad každým markerem zvlášť. Toto přináší do výpočtu hned několik problémů. Při nalezení pixelu, jehož hodnota přesahuje hodnotu THRESHOLD, se již nelze vrátit k předchozímu řádku a zjistit, zda-li se jedná o nově nalezený marker, či již marker nalezený v předchozím řádku. Z tohoto plyne, že se musejí určitým způsobem uchovávat informace o již nalezených markerech.

9.4.1 Ukládání informací o nalezených markerech

Při tvorbě datové struktury vhodné k uchovávání informací o nalezených markerech, byly uvažovány dva přístupy jež, nebyly realizovány:

1. Informace uchovávat v jednorozměrném poli, kde by každý prvek tvořila struktura o několika proměnných. Procházení pole by bylo založeno na klasickém indexování. Prohledávání pole by bylo velmi neefektivní, vzhledem k tomu, že některé markery jsou během výpočtu již nalezeny, tudíž je zbytečné jejich strukturu procházet. Uchovávání indexů struktur markerů, které se mají upravovat a které nikoliv, by bylo problematické.

2. Vzhledem k tomu, že není známo kolik markerů bude nalezeno, by bylo vhodné využít datové struktury typu obousměrný spojový seznam. V obousměrném spojovém seznamu si každý prvek uchovává pointer na předchozí a po něm následující prvek. Při nalezení nového markeru by se pro něj dynamicky alokovala paměť a prvek by se pomocí pointerů připojil ke stávajícímu seznamu. Nevýhodou tohoto přístupu je fakt, že hledání markerů musí být velmi rychlé a dynamickou alokací paměti vzniká zbytečná režie. Dále by došlo vyjmutím struktury nalezeného markeru ze seznamu ke ztrátě adresy na danou strukturu.

Skloubením předchozích dvou přístupů vznikl model, jež se v programu osvědčil. Data o každém nalezeném markeru jsou uložena ve struktuře ve tvaru dle (9.6).

Zdrojový kód 9.6: Struktura pro nalezené markery

```
typedef struct{
        unsigned int I_Sum_All; // Celkový součet intenzit pixelů
        signed int X_Left_Border; // Souřadnice prvního pixelu markeru na daném řádku..
        signed int X_Right_Border; // .. doplněná o offset
        unsigned int I_Sum_Row; // Součet intenzit pixelů v jednom řádku
        unsigned int XI; // Suma součinů intenzit pixelů a jejich souřadnice X
        unsigned int YI; // Suma součinů intenzit pixelů a jejich souřadnice Y
        unsigned char Next; // Odkaz na následující strukturu
        unsigned char Prev; // Odkaz na předchozí strukturu
        unsigned char Updated; // Logická hodnota, zda-li bylo během řádku do struktury
                               //zapisováno
        }TCenter;
```

Paměť pro uchování dat o nalezených markerech je alokována staticky jako pole struktur v jazyce C. Do metody "Compute_Centers", kde k popisovanému vyhledávání markerů dochází, je předán jen pointer na první strukturu pole. Pohyb v poli je nicméně výhradně pomocí pointerů, jako ve spojovém seznamu. Při každém nově nalezeném markeru se mu jen přiřadí nová struktura, jež se připojí ke stávajícímu seznamu.

9.4.2 Princip algoritmu

Vývojový diagram algoritmu je znázorněn na obr. (9.5). Daný vývojový diagram přesně odpovídá zápisu kódu v assembleru. Diagram byl koncipován tak, aby i člověk dříve neseznámený s daným algoritmem mohl v kódu provádět případné změny i přesto, že samotný kód má v assembleru přes 600 řádek. V diagramu jsou zřejmé labely, tvořené čtyřčíselnou kombinací čísel. Daná čísla tvoří v kódu návěstí ve tvaru `LABEL_xxxx:` a lze pomocí těchto čísel v kódu přehledně vyhledávat potřebné úseky. Dříve, než-li bude přistoupeno k vysvětlení samotného principu, je vhodné se seznámit s jednotlivými názvy proměnných, které se v algoritmu využívají.

- **Found_Marks_Count:** Celkový počet nalezených markerů. Proměnná je aktualizována při každém nalezení pixelu jehož intenzita přesahuje THRESHOLD a není z předchozího řádku známo, že by patřil již k jinému nalezenému markeru.
- **Actual_row_mark_count:** Počet zpracovávaných markerů na aktuálním řádku. Na konci každého řádku je hodnota vynulována.

- **Prev_row_mark_count:** Počet zpracovávaných markerů na předešlém řádku.
- **Top_Index:** Index od kterého se prochází struktury v paměti(kde ještě není známé těžiště) viz. obr. (9.8).
- **Active_Marks_Count:** Počet markerů, jejichž výpočet ještě není dokončen.
- **Threshold_Under_Count:** Čítač pixelů nedosahujících hranice THRESHOLD, od pixelu který THRESHOLD dosahoval.
- **X_pos:** Aktuální pozice na řádku.
- **Y_pos:** Aktuální pozice ve sloupci.
- **Active_Mark:** Pomocná logická hodnota, která se nastaví při nalezení pixelu dosahující THRESHOLD.
- **PIXEL_OFFSET:** Definice offsetu pro rozpoznání markeru z předchozího řádku.
- **THRESHOLD_UNDER_COUNT:** Maximální počet pixelů s intenzitou menší než Threshold, které budou během procházení markeru uvažovány jakoby Threshold dosahovaly. Ztmavnutí některých pixelů může být způsobeno špinavou optikou, či šumem CMOS čipu. Příliš nízké hodnoty (např. jen 1), můžou vést k tomu, že jeden marker může být rozpoznán jako dva a více markerů. Příklad tmavého pixelů je zřejmý z obr. (9.6).
- **MARKERS_MAX_COUNT_COUNT:** Maximální počet markerů, které mohou být nalezeny.

Na obr. (9.7) je uveden příklad vyhledání několika markerů. Červená čára značí vertikální a modrá značí horizontální pozici. Průnik čar značí aktuální vyčítaný pixel. Zeleně jsou vypsány některé proměnné tak, jak by vypadaly na daném pixelu. Jednotlivé markery jsou označeny indexy tak, jak by byly v paměti uchovány. Na obr. (9.8) je znázorněné, jak by které struktury v daný okamžik na sebe odkazovaly.

Nyní bude vysvětlena idea algoritmu při vyhledávání od prvního pixelu. Vysvětlení bude vázáno na vývojový diagram obr. (9.5) a na nalezené markery dle obr. (9.7). Obraz se prochází od prvního pixelu na prvním řádku do doby, než-li je splněna podmínka Ix>THRESHOLD. Pokud Ix>THRESHOLD splněno není, dojde ke skoku na LABEL_3891, kde se provede pro tuto chvíli několik nepodstatných operací a běh programu se opět vrátí nad LABEL_2658, kde se jen inkrementuje aktuální pozice na řádku. Takto

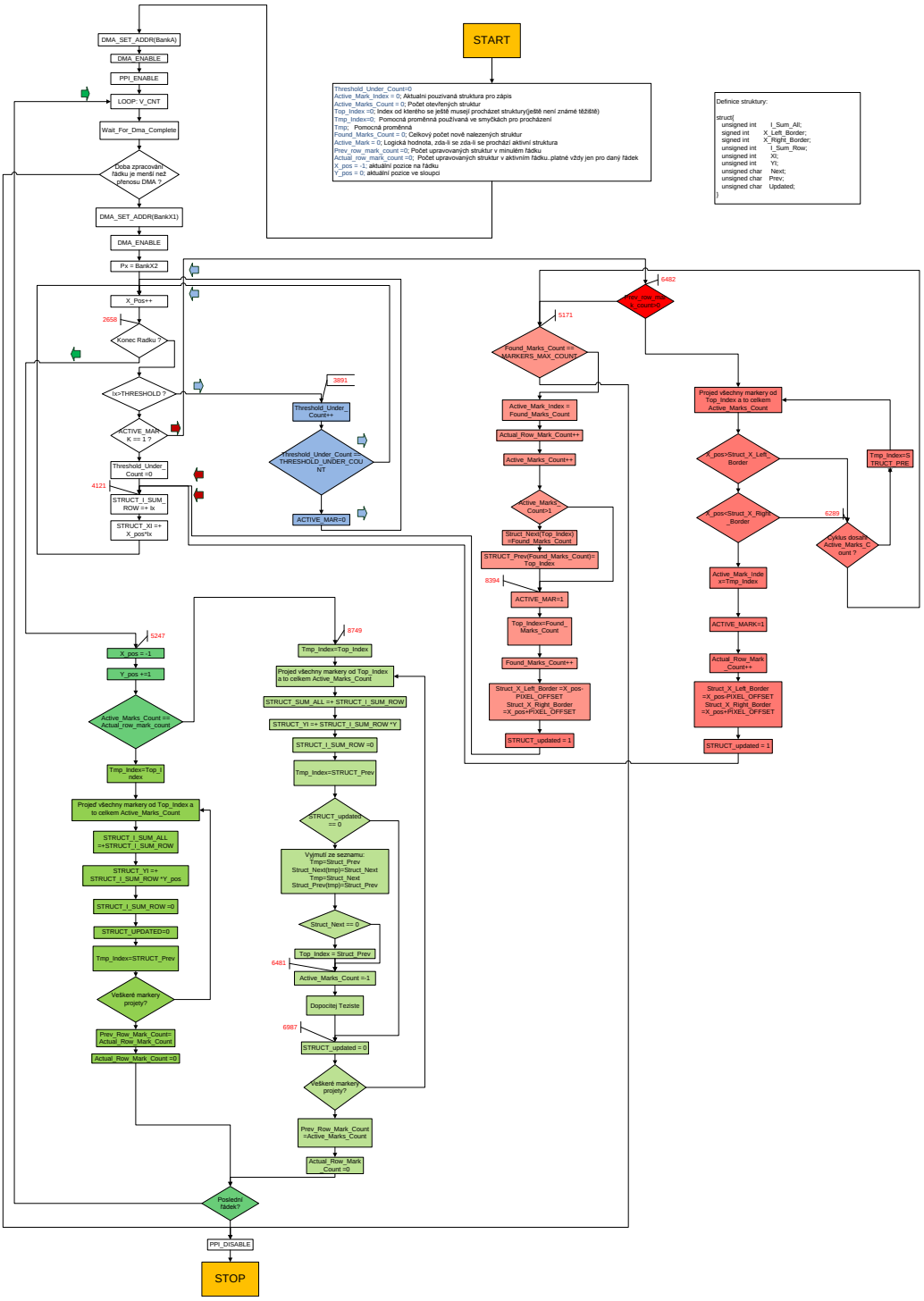
bude postupováno, dokud se nedosáhne konce řádku, čímž se skočí na LABEL_5247. Podmínka "Active_Marks_Count == Actual_row_mark_count" bude splněna, vzhledem k tomu, že žádný marker nebyl nalezen. Níže uvedená zelená sekvence je pro tento okamžik nepodstatná.

Při nalezení prvního pixelu markeru s indexem 0 viz obr. (9.7) již bude splněna podmínka Ix>THRESHOLD. Active_Mark je rovno 0, čímž se skočí na LABEL_6482. Vzhledem k tomo, že v předchozím řádku nebyl nalezen žádný marker, přesune se program na LABEL_5171, kde se kontroluje, zda-li již nebylo nalezeno více markerů, než je povoleno. Bez ošetření maximálního počtu nalezených markerů, by se přistupovalo do paměti jež není pro uchování struktur alokována. Následně se do "Active_Mark_Index " uloží "Found_Marks_Count", což je nyní nula. Znamená to tedy, že při procházení pixelů tohoto markeru se budou veškeré informace zapisovat do struktury s indexem 0. Podmínka "Active_Marks_Count>1" nebude splněna, což má za následek, že struktura bude odkazovat jak v proměnné Next tak v proměnné Prev na index 0(tudíž sama na sebe). V předposledním červeném bloku se uloží pozice nalezeného pixelu rozšířená o offset na obě strany. Následně se program přesune na LABEL_4121. Zde se provedou potřebné mezivýpočty pro výpočet těžiště.

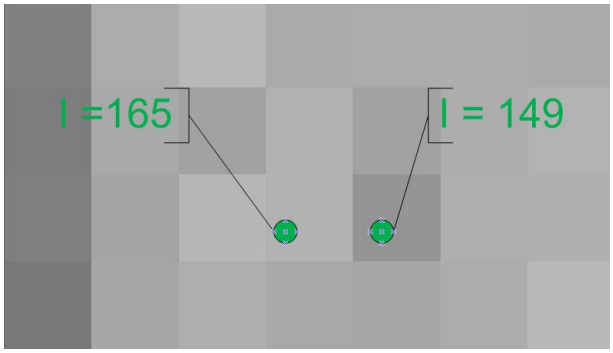
Nyní se program opět vrátí nad label 2658, kde bude opět inkrementována pozice. Dále bude opět splněna podmínka "Ix>THRESHOLD". Vzhledem k tomu, že nyní je procházen marker s indexem 0 bude podmínka "Active_Mark == 1" taktéž splněna. Následně se opět provedou mezivýpočty pro určení těžiště a program se opět přesune na label LABEL_2658. Takto se bude postupovat do té doby, dokud budou nalézány pixely markeru 0, přesahující Threshold. Na daném řádku, již žádné markery nejsou, tudíž bude zpracováván další řádek. Na novém řádku bude postupováno tak dlouho, dokud nebude splněna podmínka "Ix>THRESHOLD". Následně se program přesune na LABEL_6482. Vzhledem k tomu, že na předchozím řádku se již s nějakým markerem pracovalo, bude projit celý seznam veškerých právě zpracovávaných markerů. Cílem procházení seznamu je nalezení takového markeru, který by svou pozicí na předcházejícím řádku odpovídal pozici daného pixelu. V tomto případě se v seznamu nachází jen marker 0.

V předchozím textu již tedy bylo vysvětleno nalezení a procházení jednotlivých markerů. Poslední případ který může nastat je situace, kdy bude na řádku pracováno s n markery a v předchozím řádku s n+1 či více markery. Takovéto markery jsou již tedy kompletně zpracovány a je třeba jejich odstranění ze seznamu prohledávaných markerů, aby nedocházelo ke zbytečnému prodlužování doby vyhledání v seznamu markerů. Případ odstraněného markeru ze seznamu je například marker 2 viz. obr. (9.8). Odstranění

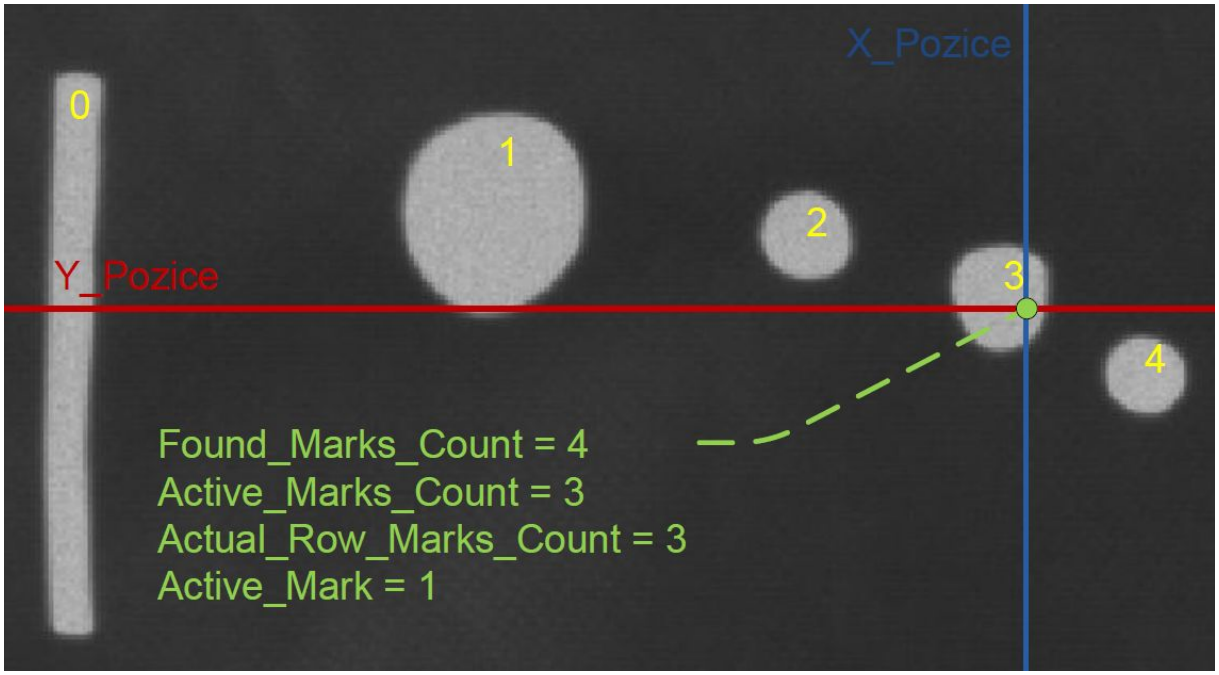
markeru(-ů) ze seznamu probíhá pod LABEL_8749, kde je mnoho příkazů shodných s větví pod labelem 5247, kromě vyjmutí markeru ze seznamu, které nastane pokud je splněna podmínka "Struct_updated == 0".



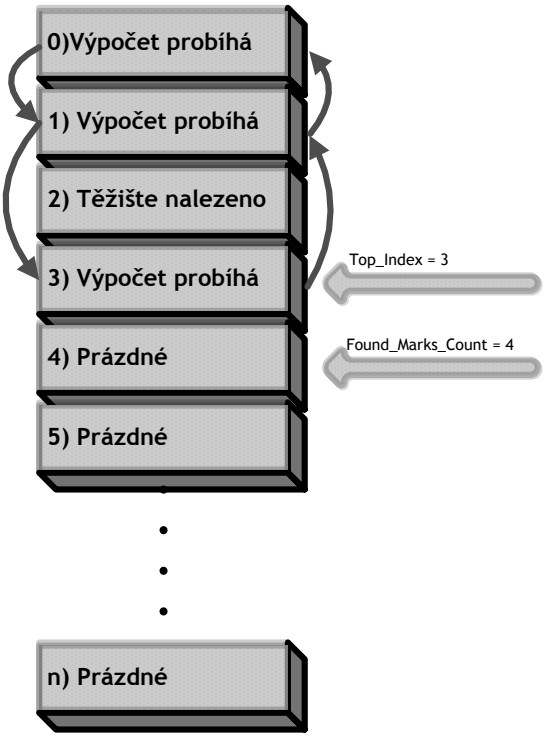
Obrázek 9.5: Vývojový digram algoritmu rozpoznání více stop



Obrázek 9.6: Vliv chyby jednoho pixelu



Obrázek 9.7: Průběh vyhledání markeru



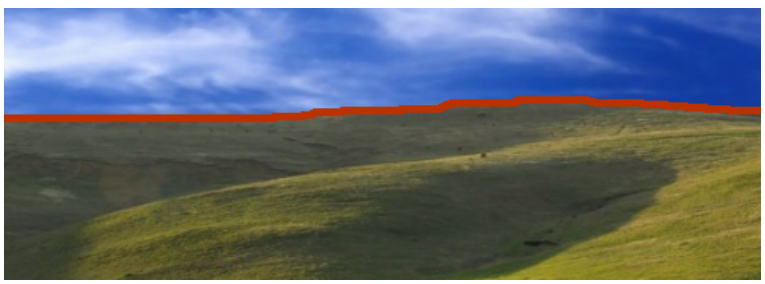
Obrázek 9.8: Procházení paměťi při hledání markerů

9.5 Algoritmus určení úhlu náklonu od horizontu

Je mnoho oblastí lidských činností, kde je třeba znalosti úhlu měřeného předmětu vůči povrchu země. Ve stavebnictví se využívá vodováha, stavící své principy na různých hustotách kapalin(plynu) a zemské gravitaci. Nicméně vodováha není primárně určena k elektronickému odečtu dat, ale slouží jen jako pomocné měřidlo. Prvek, jež umožňuje převod úhlu do elektronické podoby je například akcelerometr. Pokud je akcelerometr využit jako detektor úhlu náklonu, využívá ke své funkci gravitační zrychlení, jež je měřeno v několika osách. Ze znalosti grav. zrychlení v několika osách, lze vypočítat přesnou polohu senzoru v prostoru. Nicméně pokud by byl akcelerometr umístěn na pohybující se předmět, nerozezná dopředné zrychlení od gravitačního a dochází k nepřesnosti měření. Další možností určení úhlu náklonu je použití gyroskopu, časovou integrací úhlové rychlosti otáčení. Nevýhodou gyroskopu je nutná kalibrace pro určení vodorovné polohy. Další možností je využití kamery, kde je úhel náklonu určen ze snímané scény. O tomto přístupu bude pojednáno dále.

9.5.1 Princip metody

Úhel náklonu, který lze změřit kamerou je relativní a je vztažen k obrazovému horizontu. Aby bylo možné určit úhel náklonu, je třeba znát nějaké apriorní znalosti o měřené scéně. Apriorní znalostí v tomto případě je tvrzení: "Nebe je světlejší než krajina". Pokud budeme vycházet z tohoto tvrzení, je možné v obraze vytyčit přesnou hranici mezi krajinou a nebem, což je zřejmé z obr. (9.9). Pokud je takováto hranice proložena přímkou, je již snadné ze znalosti parametrizace přímky určit úhel.



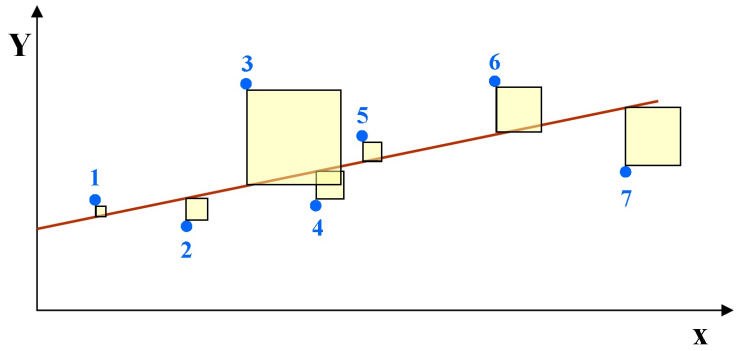
Obrázek 9.9: Hranice mezi světlou a tmavou částí snímku

9.5.2 Metoda nejmenších čtverců

Metoda nejmenších čtverců je metodou, jež se v technické praxi často využívá. Cílem metody je aproximace nalezených bodů, nějakou předem známou matematickou funkcí. V tomto případě se jedná o parametrizaci přímky dle (9.4).

$$y_{(x)} = ax + b \tag{9.4}$$

Cílem metody je minimalizace obsahu čtverců dle. obr. (9.10).



Obrázek 9.10: Metoda nejmenších čtverců

Obsah každého čtverce je uměrný kvadrátu vzdálenosti daného bodu(modré tečky 1-7) od hledané přímky a je dán rovnicí (9.5).

$$S_{(a,b,i)} = (ax_i + b - y_i)^2 \tag{9.5}$$

Součet obsahů n čtverců je tedy dán dle rov. (9.6).

$$S_{(a,b)} = \sum_{i=1}^{n} (ax_i + b - y_i)^2 \tag{9.6}$$

Cílem je minimalizovat danou funkci, čehož lze dosáhnout výpočtem parciálních derivací dle rov. (9.7) a (9.8)

$$\frac{\partial S}{\partial a} = 2\sum_{i=1}^{n}(ax_i + b - y_i)x_i = 2\left(a\sum_{i=1}^{n}x_i^2 + b\sum_{i=1}^{n}x_i - \sum_{i=1}^{n}x_i y_i\right) \tag{9.7}$$

$$\frac{\partial S}{\partial b} = 2\sum_{i=1}^{n}(ax_i + b - y_i) = 2\left(a\sum_{i=1}^{n}x_i + b\sum_{i=1}^{n}1 - \sum_{i=1}^{n}y_i\right) \tag{9.8}$$

Minimum dané funkce musí nastat ve stacionárním bodě, položením tedy rovnic (9.7) a (9.8) nule a následnému vytknutí členů "a"a "b"vzniknou rovnice (9.9) a (9.10)

$$a = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \tag{9.9}$$

$$b = \frac{\left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i\right) - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} x_i y_i\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \tag{9.10}$$

Tyto rovnice je již možno přímo implementovat do kódu, nicméně takovýto postup by byl v tomto případě značně neefektivní. Souřadnice nalezených bodů na "y-ose"jsou zcela závislé na snímaném obrazu, nicméně souřadnice na "x-ose"jsou stále stejné, resp. se vždy jedná o sekvenci hodnot 1,2,3-(H_COUNT). Rovnice lze tedy ještě po dosazení přepsat do tvaru (9.11) a (9.12), kde konstanty jsou určeny dle (9.13).

$$a = \frac{H\_COUNT\sum_{i=1}^{H\_COUNT} x_i y_i - k_1\left(\sum_{i=1}^{H\_COUNT} y_i\right)}{k_3} \tag{9.11}$$

$$b = \frac{k_2\left(\sum_{i=1}^{H\_COUNT} y_i\right) - k_1\left(\sum_{i=1}^{H\_COUNT} x_i y_i\right)}{k_3} \tag{9.12}$$

$$k_1 = \sum_{i=1}^{H\_COUNT} i; k_2 = \sum_{i=1}^{H\_COUNT} i^2; k_3 = H\_COUNT * k_2 - k_1^2 \tag{9.13}$$

9.5.3 Implementace

Celkem byly pro výpočet úhlu vytvořeny dvě metody viz. kód (9.7)

Zdrojový kód 9.7: Hlavičky metod pro výpočet úhlu

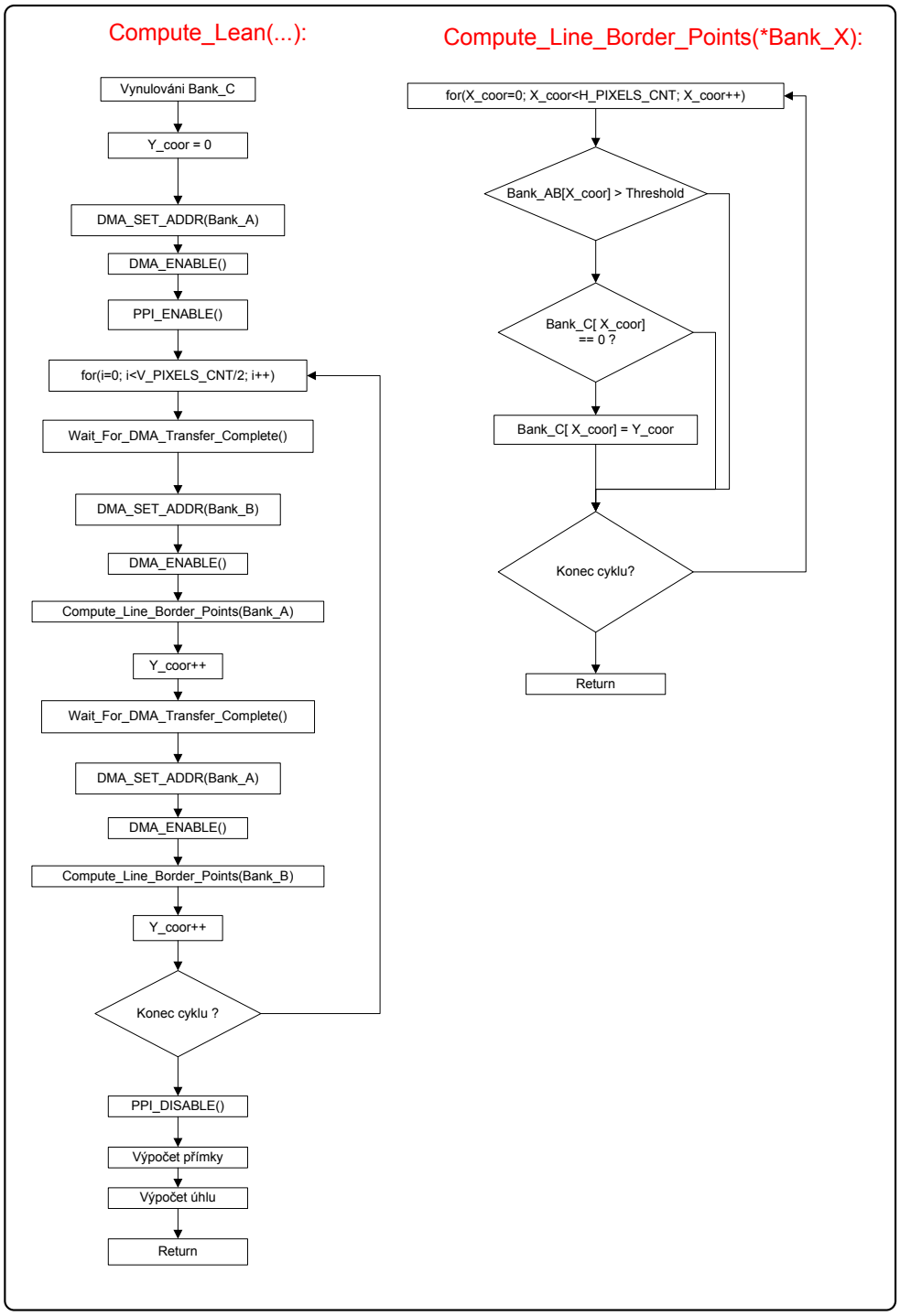
```
1 Compute_Lean(signed short *Angle); // metoda pro výpočet úhlu
2 Compute_Line_Border_Points(unsigned char *sramBankx); // pomocná metoda
```

Obsah metod je znázorněn na vývojovém diagramu na obr. (9.11). Vyčítaní řádku z CMOS senzoru je založeno stejně jako v kapitole (9.2) na přepínání dvou bufferů, s tím rozdílem, že v mezičasech nejsou řádky zasílány do PC, ale je počítán úhel. Dále

je vytvořen další buffer nazvaný Bank_C, do kterého se průběžně ukládají nalezené Y souřadnice. Z vývojového diagramu je patrné, že po každém vyčtení jednoho řádku z CMOS senzoru je volána metoda `Compute_Line_Border_Points(..)`. Zde se projde celý řádek a každý pixel, jehož intenzita je větší než `THRESHOLD` úroveň a zároveň se jedná o první nalezený pixel na souřadnici X, je Y souřadnice uložena do `BANK_C` na souřadnici X. Tímto postupem se projde celý snímek, na jehož konci jsou v `BANK_C` k dispozici nalezené body tvořící hranici mezi nebem a krajinou. Nyní je již možno přistoupit k samotnému výpočtu úhlu, výňatek kódu výpočtu je uveden v (9.8). Nejdříve se vypočítají potřebné sumy a následně čitatel zlomku. Operace "dělení"je definována jen pro kladná čísla, proto je nutné v případě záporného čitatele určit jeho absolutní hodnotu a znaménko přiřadit později. Po výpočtu úhlu je nutný jeho přepočet na stupně.

Zdrojový kód 9.8: Výňatek části kódu výpočtu úhlu náklonu

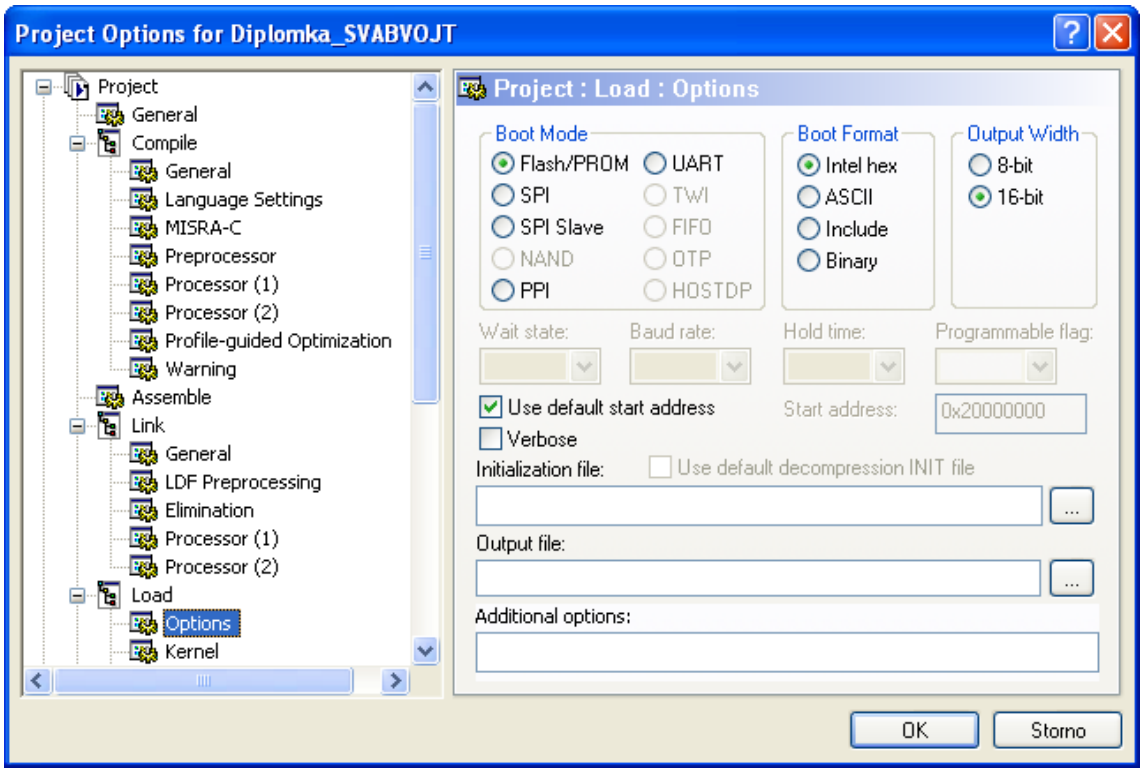
```
1  unsigned int Sum_Yi = 0;
2  unsigned int Sum_XiYi = 0;
3  for(i=0; i<H_COUNT;i++){
4    Sum_Yi += sramBankC[i];
5    Sum_XiYi += (i+1)*sramBankC[i];
6  }
7  unsigned long long a_numerator = ((unsigned long long)H_COUNT*
8  (unsigned long long)Sum_XiYi-(unsigned long long)K1*
9  (unsigned long long)Sum_Yi);
10 // Test, zda-li je citatel kladny
11 if(a_numerator > 0x7FFFFFFFFFFFFFFF ){
12       // ziskani absolutni hodnoty
13       a_numerator = llabs(a_numerator);
14       // vypocet smernice a
15   a = -(float)a_numerator / (float)K3;
16 }else{
17       // vypocet smernice a
18   a = (float)a_numerator / (float)K3;
19 }
20 // vypocet offsetu b
21 b = (K2*Sum_Yi-K1*Sum_XiYi)/K3;
22 // vypocet uhlu, vystup z atan() je v radianech -> prevod na stupne
23 *Angle = atan(a)*57;
```



Obrázek 9.11: Vývojový diagram algoritmu rozpoznání úhlu náklonu

9.6 Bootování z interní FLASH paměti

SMART kamera je navržena tak, aby bylo možné nabootovat z externí paměti přes SPI, či v asynchronním módu z interní FLASH paměti. Bootování z SPI paměti je určeno jen k nahrání bootloaderu a je popsáno v kapitole (9.9), bootování z FLASH paměti je tedy uvažováno jako primární. Při kompilaci programu v prostředi VDSP++ je třeba mít nastavení dle obr.(9.12). Aby SMART kamera z dané FLASH paměti nabootovala, nesmí být vložen jumper pro volbu bootování dle (8.2).



Obrázek 9.12: Nastavení VDSP++ pro bootování z FLASH

9.7 Test FLASH paměti pro běh programu

Již z kapitoly (4.2.4) je zřejmé, že procesory Blackfin mají určitou hierarchii paměti. Nejrychlejší paměti jsou umístěny na stejném čipu jako procesor(interní paměti) a můžeme je rozdělit na části L1 (SRAM) a L2. Paměti, které jsou výrazně pomalejší oproti předchozím, značíme L3(FLASH,SDRAM, ROM..). Pro dosažení maximálního výkonu je vhodné, aby byl celý program uložen v L1 interní paměti. Při větších velikostech programu již tohoto

Zdrojový kód 9.9: Příkazy pro umístění kódu do paměti

```
1  // kód k umístění do FLASH paměti
2  #pragma section("FLASH_code")
3  // kód k umístění do L1 paměti
4  #pragma section("L1_code")
```

ideálního stavu nelze dosáhnout a program musí být uložen také v externí paměti. Volání částí programu z pomalejší externí paměti, zbytečně zpomaluje rychlost vykonávání instrukcí procesorem. Pro dosažení optimálního výkonu lze použít následující triky:

1. Části programu, jejichž rychlost vykonání není důležitá(inicializační rutiny, pomocné funkce atd.) jsou ponechány v externí paměti a naopak časově kritické sekce programu jsou umístěny v interní paměti. Rozhodnutí, který kód má být kde uložen je realizováno pomocí direktiv dle (9.9). Veškeré funkce jsou defaultně umisťovány do L1 paměti. Direktiva musí být uvedena pro každou funkci, či proměnnou zvlášť.

2. Nakonfigurovat procesor tak, aby využíval instrukční a datovou cache. Pokud je část SRAM paměťi nakonfigurována jako cache, nelze již do ní svévolně zapisovat. Veškeré zápisy do cache jsou řízeny autonomně pomocí "cache-kontroléru". "Cache-kontrolér" umožňuje uchovávat části programu a dat, které nejsou často používány v externí paměti. Pokud je volán kód programu, umístěný ve FLASH paměti, který není do cache ještě překopírován z předchozí relace, tak "cache kontrolér"tento kód automaticky do cache překopíruje. V konečném důsledku je to stejné, jako by byl daný kód v L1 paměti již od počátku, nicméně je zde určitá režie pro překopírování kódu. Z obr.(4.4) je zřejmé, kde se jednotlivé cache paměti v paměťovém prostoru nachází. Podrobné informace o použití cache je možné nalézt v Lit.[14]

3. Využívat tzv. overlays, což znamená že program a data jsou uloženy v externí paměti a ve chvíli kdy mají být použity, se manuálně přesunou do interní L1 paměti. V architekturách jež nepodporují cache, je toto jediný způsob jak spustit program z interní paměti. Tato technika je nicméně využívána i na systémech s podporou cache, protože zde má programátor plnou kontrolu nad tím, kdy se který blok dat či programu do L1 paměti přenese. Podrobné informace o použití overlays je možné nalézt v Lit.[15].

Tabulka 9.1: Rychlost vykonání kódu v záv. na použité paměti

Asynchronní mód FLASH					
	bez cache		s cache		
	L1[us]	FLASH[us]	L1[us]	FLASH[us] (1.běh)	FLASH[us] (n-tý běh)
sekvence	2,605	230,962	2,605	223,445	2,610 [1]
smyčka	2,623	362,780	2,623	8,770	2,628
Synchronní burst mód FLASH					
	bez cache		s cache		
	L1[us]	FLASH[us]	L1[us]	FLASH[us] (1.běh)	FLASH[us] (n-tý běh)
sekvence	2,605	63,323	2,605	30,968	2,610
smyčka	2,623	102,808	2,623	3,493	2,628

Níže je popsán test, který je určen k porovnání rychlosti vykonání instrukcí z FLASH, či L1 paměti. FLASH paměť byla zkoušena jak v asynchronním, tak v synchronním módu, jež by měl být rychlejší. Test byl proveden tak, že byl měřen čas vykonání metody "_Test_ASM" viz (9.10). Metoda byla spuštěna jednou a to tak, že bylo vykonáno 1000 instrukcí "nop" a jednou taktéž 1000 instrukcí "nop", nicméně za pomoci smyčky s nulovou režií. Je zřejmé, že smyčka je vykonávána 200-krát, kde při každém průchodu je vykonáno 5 instrukcí "nop". V případě vykonání smyčky 1000-krát s jednou "nop" instrukcí, z FLASH paměti bez použití cache dochází k tomu, že instrukce jsou brány ze zásobníku pipeline a časy jsou mnohokrát menší. Nicméně takovéto hodnoty nelze porovnávat se sekvencí 1000 "nop", proto byla smyčka upravena jen na 200 průchodů. Při načtení 5-té "nop" instrukce ve smyčce se pipeline zásobník zaplní a FLASH paměť musí být znovu zaadresována. Z tabulky výsledků měření (9.1) je zřejmé, že pokud je metoda "_Test_ASM" volána z FLASH paměti s aktivovanou cache, tak se liší doba jejího vykonání v závislosti na tom, zda-li již byla jednou volána. Daný rozdíl je způsoben tím, že při prvním volání metody je celý její kód překopírován z pomalé FLASH paměti do L1 paměti a až následně vykonán. Pokud je metoda volána již po druhé, je rovnou využita její kopie v L1 paměti a doba vykonání je tedy téměř totožná jakoby byla v L1 paměti uložena již od nabootování.

[1] U 2. běhu je čas roven 3,560 us, dále je již čas shodný s výše uvedeným

Zdrojový kód 9.10: Metoda pro test rychlosti vykonání programu z FLASH

```
1  __Test_ASM:
2  /*
3  //--------- SMYCKA ktera probehne 200-krat, v tele ma instrukci "nop"---------
4      R0 = 200;
5      P0 = R0;
6      LOOP loop_name LC1 = P0 ; // autoinitialize LC1 (b)
7      LOOP_BEGIN loop_name; // zacatek smycky
8      nop;nop;nop;nop;nop;
9      LOOP_END loop_name ; // define the last instruction of the loop
10 */
11 //------------------ SEKVENCE nop ------------------------
12     /// *
13     nop;nop;nop;..... nop; // 1000 "nop" instrukci, pro prehlednost nebudou vsechny ''nop''
14     vypsany
15     //*/
16 //-------------------------------------------------------
17     ssync;
18 RTS;
19 __Test_ASM.end:
```

9.8 Čekací smyčka

V mnoha případech je třeba čekat v určitých částech kódu požadovaný čas. Ve vyšších programovacích jazycích, lze používat známé metody "Sleep(..)", při programování DSP je již situace složitější. Pokud se nemá vykonávat během čekání, žádná jiná část kódu, je využívání hardwarových časovačů zbytečné, a zároveň má jejich inicializace velkou časovou režii. Výhodnější je v jazyce C využití for cyklu, nicméně zde velmi záleží na nastavení kompilátoru a for cyklus může být přeložen nevhodně. Pro striktní dodržení časů je v tomto případě výhodné využíti assembleru. Z tohoto důvodu byla vytvořena funkce pozastavující chod programu o požadovaný počet strojových cyklů viz. (9.11). Ve funkci je využito harwarové smyčky, jež má nulový "overhead"(nulová režie na vykonání smyčky). Pokud je tedy frekvence jádra nastavena na 400 MHz, trvá jedno projití smyčky 1/400MHz = 2,5 ns.

Zdrojový kód 9.11: Funkce pozastavující chod programu o x cyklů jádra

```
1  //void _Wait_Tics(int Tics_To_Wait);
2  __Wait_Tics:
3      // 1.argument je umisten v R0(pocet ticku)
4      P0 = R0;
5      LOOP Loop_1 LC0 = P0 ; // autoinicilizace LC0
6      //smycka
7      LOOP_BEGIN Loop_1;
8      nop;
9      LOOP_END Loop_1;
10         RTS;
11 __Wait_Tics.end:
```

Vzhledem k tomu, že frekvence jádra může být během vykonávání programu dynamicky přenastavena, není vhodné čekat předdefinovaný počet strojových cyklů vázaných na určitou frekvenci. Výhodnější je tedy nepožadovat čekání určitého počtu strojových cyklů, ale rovnou času v nanosekundách. Vzhledem k tomu, že jeden instrukční cyklus trvá 2,5 ns, byla vytvořena funkce umožňující čekání v desítkách nanosekund dle (9.12).

Zdrojový kód 9.12: Funkce pro čekání desítek ns

```
1  void Wait_10ns(unsigned int T_10ns_Delay){
2          // Pocet strojovych cyklu behem 10ns
3          unsigned int Tics_Per_10ns = F_CCLK/100000000;
4          _Wait_Tics((unsigned int)(T_10ns_Delay*Tics_Per_10ns));
5          return;
6  }
```

Takto navrženou čekací smyčku lze využít při softwarové implementaci různých sběrnic, kde je nutné dodržovat přesné časování.

9.9 Aktualizace firmware SMART kamery

Jedinou non-volatilní pamětí z které je procesor BF504F schopný nabootovat, je v navrženém uspořádání, interní FLASH paměť. Níže jsou uvedeny dvě možnosti jak lze do FLASH paměti nahrát nový firmware.

9.9.1 Vygenerování firmware do *.LDR souboru

Soubor jež je třeba do FLASH paměti nahrát je vytvořen v prostředí VDSP a má příponu *.ldr. Pro správnou funkci je třeba prostředí následovně nakonfigurovat:

1. Otevřít "Project Options"a vyplnit záložku dle obr. (9.12)
2. Na záložce "Load/Splitter"nesmí být zaškrtnuto "Enable ROM splitter"
3. Potvrdit tlačítkem "OK"
4. Přeložit projekt

9.9.2 Náhrání nového firmware pomocí JTAG rozhraní

Použití JTAG rozhraní k nahrání nového firmware do FLASH paměti je asi tou nejrychlejší cestou. Jako JTAG adaptér je možné využít emulátor ICE-100B. Pro vlastní nahrání programu do FLASH paměti v prostředí VDSP je třeba následujících kroků:

1. Otevřít vlastní FLASH programátor přes "Tools/Flash Programmer"
2. V záložce Drivertab nalézt ovladač pro programátor v VisualDSP instalačním adresáři\Blackfin\Examples\ADSP-BF506F EZ-KITLite\FlashProgrammer\ BF50x4MBFlash\BF506FEzFlashDriver_BF50x4MBFlash.dxe
3. Kliknout na tlačítko "Load Driver"
4. Na záložce "Programming"najít požadovaný *.LDR určený k nahrání do FLASH paměti
5. Kliknout na tlačítko "Program", tím je vše hotovo

9.9.3 Náhrání nového firmware pomocí Bootloaderu přes USB

Program lze do paměti přenést dle kap. (9.9.2), nicméně vlastnictví emulátoru není samozřejmostí. Z tohoto důvodu je vhodné vytvoření BOOTLOADERu, který interní FLASH paměť naprogramuje. Zde se opět naskýtá problém nahrání samotného BOOTLOADERu do FLASH paměti. K nahrání je možné využít SPI EEPROM paměť 25c256, z které by SMART kamera nabootovala a překopírovala BOOTLOADER, který by v ní byl také uložen, do FLASH paměti. Nicméně ve schematu SMART kamery je

naznačeno, že na piny SPI rozhraní je připojena datová sběrnice CMOS sensoru. Pro připojení SPI paměti je tedy nutné CMOS sensor od SMART kamery mechanicky odpojit. Postup nahrání BOOTLOADERu a následné jeho využití k nahrání samotného firmware SMART kamery do FLASH paměti je tedy následující:

1. Pomocí programátoru uložíme do paměti 25c256 vytvořený BOOTLOADER. Jako programátor lze využít vývojové kity dle kap. (5.3).

2. Paměť propojíme s konektorem J3 (konektor J3 definován v (17.5)) dle nákresu (9.13). Dále je třeba k paměti připojit potřebné pull-rezistory a blokovací kondenzátor, stejně jako je tomu v (17.3). Pro propojení je vhodné dle předchozích instrukcí zhotovit redukci.

3. Propojíme propojku J2 dle (17.5), která aktivuje bootování přes SPI rozhraní a připojíme SMART kameru k napájení

4. Nyní již by měl být v FLASH paměti BOOTLOADER nahrán, je tedy možné odpojit SPI paměť od SMART kamery.

5. Odstraníme propojku J2 dle (17.5), čímž se nastaví bootování z FLASH paměti.

6. Propojíme USB konektor s počítačem

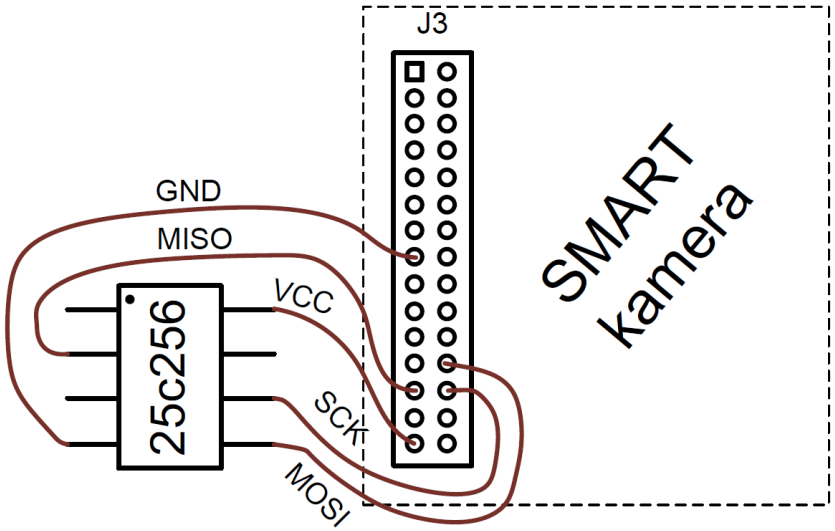
7. Naprogramujeme interní FLASH paměť.

Pokud je již jednou BOOTLOADER v paměti nahrán, lze pro aktualizaci nového firmwaru projít poslední dva kroky dle předchozího postupu.

Z výše uvedeného je zřejmé, že nahrání BOOTLOADERU do FLASH paměti je z časového hlediska poměrně náročná činnost, o nutnosti mechanického oddělení CMOS sensoru od modulu SMART kamery nemluvě. Nicméně je nutné si uvědomit, že BOOTLOADER je do paměti nahráván jen jednou po napájení modulu SMART kamery a dále je již BOOTLOADER jen využíván k nahrávání nového firmware, což je činnost podstatně rychlejší a jednodušší.

9.9.4 Návrh BOOTLOADERu

Úkolem BOOTLOADERu je nahrání firmware do interní FLASH paměti, z tohoto je tedy zřejmé že musí data z PC určitým způsobem přenést. SMART kamera je vybavena dvěmi



Obrázek 9.13: Připojení externí SPI paměti k Smart kameře

komunikačními rozhraními a to USB či RS-485. Vzhledem k tomu, že každý počítač je USB rozhraním vybaven, je jeho použití jednoznačně vhodné. Bude tedy využit obvod FT2232HL, jež je připojen s procesorem BF504F propojen pomocí FIFO rozhraní. Ke komunikaci s FT2232HL ze strany procesoru je vhodné využít již vytvořené softwarové vybavení použité v firmware SMART kamery. Ke komunikaci s interní FLASH pamětí je vhodné využívat knihovny od Analog Devices, které je možné nalézt v instalačním adresáři vývojového prostředí VDSP. Samotný BOOTLOADER je třeba v paměťovém prostoru překládat již od adresy 0x20000000 tak, aby z něj bylo nabootováno.

Jediným úkolem BOOTLOADERU, je tedy uložení firmware do FLASH paměti od adresy například 0x20000400(v záv. na velikosti BOOTLOADERU). Následně je třeba provést kontrolní CRC součet nad uloženým firmwarem, aby byla ověřena konzistence přenesených dat. V případě úspěšného přenosu BOOTLOADERU do paměti (může být opět signalizováno LED), je třeba přenastavit programový čítač na adresu 0x20000400 tak, aby byl již vykonáván samotný kód SMART kamery.

9.9.5 Aktivace BOOTLOADERu

Po připojení SMART kamery k napájení musí existovat mechanismus, jak daný bootloader spustit. K tomuto účelu je vhodné využít uživatelského tlačítka SW2 viz. schema SMART kamery (17.5), které by muselo být po resetování SMART kamery sepnuto. Následně je možné vytvořit spolu s LED určitou spouštěcí proceduru, aby se zabránilo

nechtěnému spuštění BOOTLOADERU. Aktivace BOOTLOADERU může být signalizována blikáním LED. V případě, že nebude po resetování SMART kamery dodržena spouštěcí procedura, musí BOOTLOADER přenastavit programový čítač na adresu, kde se nachází firmware samotné SMART kamery.

Kapitola 10

Návrh obslužného softwaru pro PC

10.0.6 Zobrazení obrazu v prostředí MATLAB

Ve fázi návrhu softwaru, kdy není k dispozici program pro zobrazování obrazu na PC, je využití MATLABu asi tou nejrychlejší cestou. Z kamery lze vyčíst jen holá obrazová data, jež se uloží například do "imagefile.txt". Takto uložená data lze vyzualizovat dle (10.1).

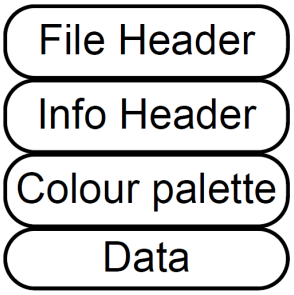
Zdrojový kód 10.1: Vizualizace dat v Matlabu

```
1  V_count =1024;
2  H_count =1280;
3  fid=fopen('imagefile.txt','r');
4  if (fid==-1)
5      error('can not open imput image filem press CTRL-C to exit \n');
6      pause;
7  end;
8  pixel=fread(fid,inf, 'uchar');
9  fclose(fid);
10 tmp1 = pixel';
11 tmp2 = tmp1(1:(V_count*H_count));
12 im = reshape(tmp2,H_count,V_count)';
13 imshow(im,[]); %zobrazeni upraveneho obrazku
```

10.1 Vytvoření BMP obrázku z přijatých dat

Obrazová data, která jsou přijata ze SMART kamery, nejsou vybavena žádnými informačními daty, jedná se jen o "matici" jasových hodnot. Aby mohly být jednotlivé snímky zobrazeny v jakékoliv nadřazené aplikaci, je třeba data převést na nějaký unifikovaný obrazový formát. Vzhledem k tomu, že je vhodné aby převod trval co nejkratší čas, byl vybrán formát obrázku BMP, bez použití komprese.

BMP formát obrazových dat je tvořen hlavičkami, kde následuje barvová paleta a nakonec obrazovými daty data s jasovými hodnotami, bez jakékoliv komprese viz obr. (10.1). Podrobné informace o BMP formátu lze nalézt v [20]. Ze zdrojového kódu (10.2) je zřejmé, jak byly nadefinovány jednotlivé hlavičky. Za povšimnutí stojí příkaz "pack(..)", který zabezpečuje zarovnání jednotlivývch záznamů hlaviček v paměti počítače po bytech. Při absenci tohoto příkazu, by se data zarovnávala na 4-bytový formát, čímž by nebylo dodrženo správné uložení hlaviček a výsledný BMP soubor by nebylo možné otevřít.



Obrázek 10.1: Struktura BMP obrázku

10.2 Tvorba DLL knihovny pro implementaci SMART kamery v nadřazených programech

Veškerý software pro PC, jež byl navržen pro komunikaci se SMART kamerou je ve výsledku zkompilován do DLL knihovny. Jednotlivé dostupné metody a jejich popis je uveden v kap. (17.1).

Zdrojový kód 10.2: Struktury pro tvorbu BMP

```
1  #pragma pack(push, 1)
2  struct BITMAPFILEHEADER_ {
3    __int16 bfType;
4    unsigned int bfSize;
5    __int16 bfReserved1;
6    __int16 bfReserved2;
7    unsigned int bfOffBits;
8  };
9  #pragma pack(pop)
10
11 #pragma pack(push, 1)
12 struct BITMAPINFOHEADER_{
13   unsigned int biSize;
14   unsigned int biWidth;
15   unsigned int biHeight;
16   __int16 biPlanes;
17   __int16 biBitCount;
18   unsigned int biCompression;
19   unsigned int biSizeImage;
20   unsigned int biXPelsPerMeter;
21   unsigned int biYPelsPerMeter;
22   unsigned int biClrUsed;
23   unsigned int biClrImportant;
24 };
25 #pragma pack(pop)
```

Kapitola 11

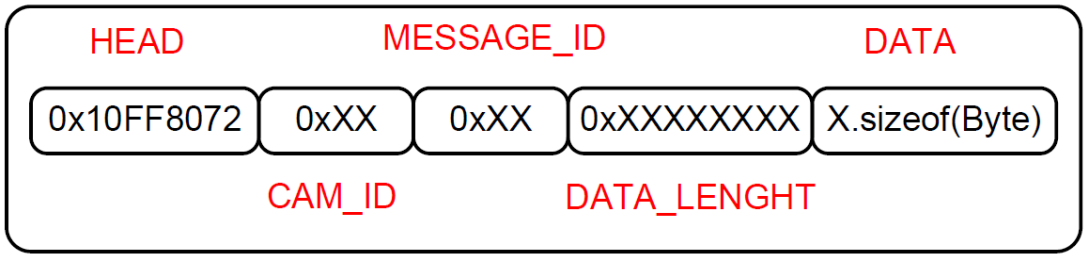
Síťová spolupráce SMART kamery

Smartkamera je navržena tak, aby podporovala práci v síti přes sběrnice USB a RS485. Na obr. (11.2) je znázorněno, jak mohou být jednotlivé kamery připojeny k nadřazenému PC. Dohromady může být v síti připojeno až 32 SMART kamer.

11.1 Složení packetu

Komunikace mezi SMART kamerami a PC probíhá na základě packetové komunikace. Každý packet posílaný jak po USB, tak i po RS-485 má strukturu dle obr. (11.1). Vzhledem k tomu, že může dojít z různých příčin k nedoručení celého packetu, je nutné nějakým způsobem v přijatých datech rozpoznat začátek dalšího packetu. K tomuto účelu slouží hlavička packetu "HEAD", jež svojí kombinací bytů umožňuje bezpečné nalezení packetu v přijímacím bufferu [1]. Po hlavičce následuje "CAM_ID", jež slouží jako identifikátor dané kamery(1-32), popřípadě hodnota 0xFE, pokud je zpráva zasílana všem SMART kamerám najednou. Následuje "MESSAGE_ID", sloužící k identifikaci požadovaného úkolu (zápis do sensoru po I2C, sejmutí snímku, určení těžiště atd.). Nakonec jsou připojeny data, zkládající se z jejich počtu v bytech a vlastních dat. Packet který je SMART kamerou přijmut, je ve stejném formátu po zpracování výsledků a jejich uložení do datové části packetu zaslán zpět do PC. Tímto způsobem nadřazená aplikace v PC pozná, z které SMART kamery daný packet dorazil a co je jeho obsahem.

[1] Podobný přístup již byl na katedře měření úspěšně ověřen.



Obrázek 11.1: Struktura packetu

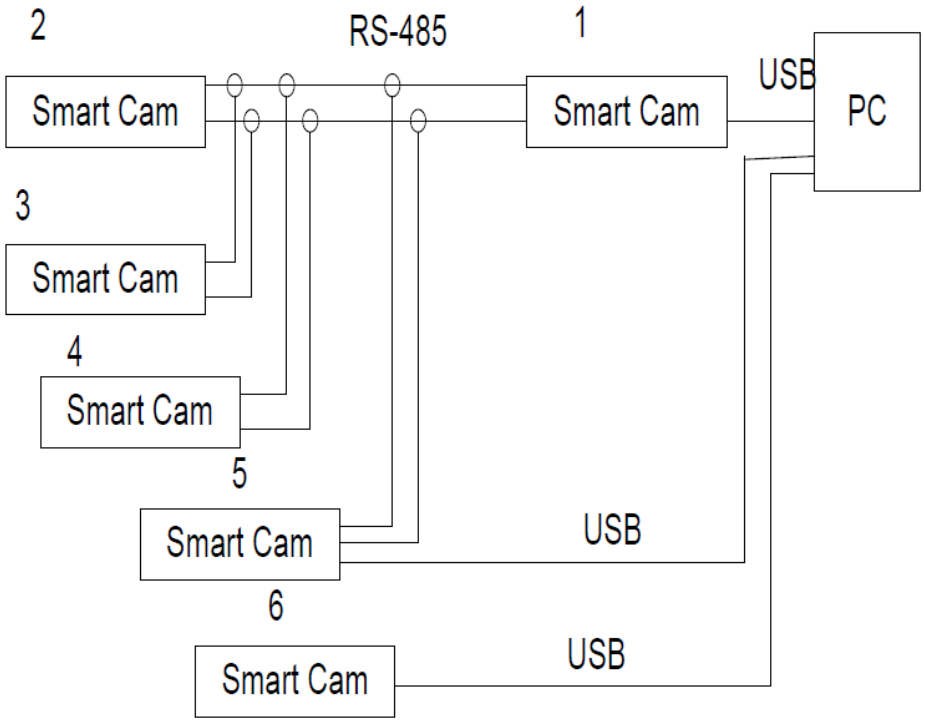
11.2 Možné zapojení sitě

Jak již bylo dříve psáno, k propojení SMART kamer je možné využít USB a RS-485 rozhraní. SMART kamery mohou být vzájemně propojeny téměř libovolně. Mohou být propojeny jen pomocí USB, či jen pomocí RS-485, ale i oběma rozhraními najednou viz (11.2). Nicméně při využití jen rozhraní RS-485, musí být minimálně jedna SMART kamera připojena i pomocí USB(na obr. (11.2) např. SMART kamera 1). Takováto SMART kamera se při inicilizaci sítě automaticky nakonfiguruje do módu bridge USB/RS-485. V případě, že je SMART kamera nakonfigurována jako bridge, stále poskytuje plnou funkcionalitu jako ostatní SMART kamery. Jediné omezení v zapojení sítě tkví v tom, že nesmí vzniknout více oddělených RS-485 sítí, než-li jedna. Pokud by vznikly dvě vzájemně nepropojené RS-485 sítě, vždy bude komunikováno jen s jednou sítí.

Pokud je to možné, je vždy výhodnější připojení SMART kamer přes USB. Komunikace po RS-485 je oproti USB pomalejší a je ponechána zcela bez arbitráže. Z tohoto důvodu, vždy pokud je nějaká SMART kamera připojena pomocí USB, je s kamerou přednostně komunikováno pomocí USB. Volba rozhraní probíhá zcela automaticky a nelze ji voláním funkcí z DLL knihovny ovlivnit. Pokud by tedy uživatel chtěl se SMART kamerou komunikovat jen pomocí RS-485, musí být odpojen USB kabel a následně opět volána funkce `Find_All_Connected_Cameras(..)` z DLL knihovny.

11.3 Uložení informací o jednotlivých kamerách

Vzhledem k tomu, že k PC může být připojeno až 32 SMART kamer, je třeba uchovávat určité informace o každé kameře zvlášť. Informace o každé kameře jsou uloženy v struktuře definované v (11.1). Vysvětlení jednotlivých proměnných je zřejmé z kódu.



Obrázek 11.2: Možné zapojení sítě SMART kamer

Zdrojový kód 11.1: Ukládané informace o každé SMART kameře v PC

```
1 struct TCAM_INFO {
2         bool USB_Connection; // Pokud TRUE, kamera je pripojena k PC pomoci USB
3         bool RS485_Connection; // Pokud TRUE, kamera je pripojena k PC pomoci RS-485
4         FT_HANDLE FT_handle; // Handle na FT2232HL
5         unsigned int Cam_ID; // ID smartkamery
6     };
```

Počet SMART kamer připojených k PC není předem znám, proto je vhodné dynamicky alokovat paměť pro jednotlivé struktury nalezených SMART kamer. Z tohoto důvodu je použit datový kontejner typu "vector" viz. (11.2).

Zdrojový kód 11.2: Využití datové struktury vector

```
1 vector<TCAM_INFO> ALL_Connected_Cameras_Vector;
```

V případě nalezení nové kamery se alokuje paměť pro novou strukturu, která se jen předá do vectoru dle (11.3).

Zdrojový kód 11.3: Přidání nalezené SMART kamery do vectoru

```
1 TCAM_INFO Tcam_Info;
2 ALL_Connected_Cameras_Vector.push_back(Tcam_Info);
```

11.4 SMART kamera jako BRIDGE USB/RS-485

Obvykle žádné PC není standardně vybaveno rozhraním RS-485. Pokud je tedy třeba komunikovat po RS-485, je nutné využít nějaký převodník. Vzhledem k tomu, že SMART kamery disponují jak USB, tak i RS-485 rozhraním, naskýtá se možnost využít přímo danou kameru jako bridge USB/RS-485. Není možné, aby veškeré kamery přeposílaly packety, které pro ně nejsou určené automaticky na druhý typ rozhraní. V tomto případě by došlo ke kolapsu sítě. V každý okamžik může být nastavena právě jedna SMART kamera jako bridge USB/RS-485. Na obr. (11.2) k tomuto účelu mohou posloužit např. kamery 1 a 6.

11.5 Automatické vyhledání připojených smartkamer

Vyhledání veškerých smartkamer je iniciováno pomocí metody **Find _All _Connected _Cameras(..)** dostupné z DLL. V této metodě se vytvoří seznam veškerých připojených čipů připojených přes USB od firmy FTDI. Vzhledem k tomu, že tyto čipy se využívají i v mnoha jiných zařízeních (např. log. analyzátor ASIX SIGMA), nelze uvažovat počet nalezených čipů jako počet nalezených SMART kamer. Z tohoto důvodu je nutné, nějakým způsobem odlišit FTDI čipy na SMART kamerách od jiných zařízení. Jako vhodný způsob se osvědčilo změnit "Product Description" čipu FT2232HL na "SCAM_USB". U věškerých nalezených čipů FTDI je tedy porovnán jejich "Product Description" s "SCAM_USB A" a pokud se shodují, lze předpokládat že se jedná o SMART kameru. Písmeno "A"na konci deskriptou je uvedeno proto, že obvod FT2232HL má dva nezávislé porty, port A a port B. Písmeno určující typ brány je za deskriptor přidáno automaticky driverem obvodu FT2232HL. Dle schema SMART kamery (17.5), je využit pro komunikaci s procesorem port A. Tímto jsou nalezeny veškeré kamery připojené pomocí USB a nyní

musí následovat vyhledání veškerých SMART kamer připojených přes RS485. Vzhledem k tomu, že není předem známo, kterou kameru lze využít jako bridge USB/RS485, je tedy každá SMART kamera připojená přes USB zpočátku uvažována jako případný bridge. Pokud jsou například nalezeny 3 kamery připojené přes USB, je kamera 1 nastavena jako bridge USB/RS485. Následně jsou přes ní na RS485 volány jednotlivé kamery s ID 01-32. Volání probíhá tak, že se zavolá funkce `Request_Echo(..)`, následně se čeké 100ms, a zavolá se funkce `GET_Echo(..)`. V případě úspěšného vyčtení echa, bude právě nakonfigurovaná SMART kamera jako bridge pro tento účel dále používána. V opačném případě bude stejným způsobem postupováno i u kamer 2 a 3.

V počátku návrhu sítě byla uvažována možnost rozlišení jednotlivých SMART kamer pouze jedinečným deskriptorem u FTDI, nicméně tento přístup přinášel hned několik nevýhod. Pokud by ve fázi vyhledávání SMART kamer, byl z nějakého důvodu poškozen procesor BF504F, ale obvod FT2232HL by poskytoval platný deskriptor, byla by daná SMART kamera zahrnuta do seznamu připojených SMART kamer i přesto, že by de facto nefungovala. Další nevýhodou je nemožnost změny deskriptoru ze strany procesoru, pokud by se některé deskriptory překrývaly.

11.6 Komunikace s jednotlivými kamerami

Při návrhu metody získávání požadovaných dat z jednotlivých SMART kamer bylo uvažováno několik přístupů. Pro názornost bude vše uváděno na funkci požadující snímek, tedy `Request_Image(..)`.

1. Při volání funkce `Request_Image(..)` by se v dané funkci čekalo, dokud by kamera požadovaný snímek nezaslala. Funkce `Request_Image(..)` by sama navracela pointer na vytvořený snímek.

   Výhody: rychlé

   Nevýhody: V případě poruchy volané kamery, nebo jen poškození packetu, požadujícím zaslání dat, by došlo k uvíznutí celého programu.

2. Při volání funkce `Request_Image(..)` by se v dané funkci čekalo, dokud by kamera požadovaný snímek nezaslala, ale po stanoveném čase by byla funkce násilně ukončena. Funkce `Request_Image(..)` by sama navracela pointer na vytvořený snímek.

Výhody: V bezporuchovém stavu rychlé.

Nevýhody: V případě poruchy volané kamery, nebo jen poškození packetu, by se zbytečně čekalo na uplynutí "time-out".

3. Voláním funkce `Request_Image(..)` by se jen zažádalo SMART kameru o vyfocení scény. Funkce `Request_Image(..)` by tedy nevracela pointer na vytvořený snímek.

   Výhody: Porucha jedné kamery neovlivní celý systém.

Ačkoliv by první možnost byla nejrychlejší, je možnost zamrznutí celého programu zcela neakceptovatelná. Tento postup byl nicméně zpočátku implemetován a použit při debuggování. Vzhledem k tomu, že postup dle ad 2) by mohl při poruše přenosu packetu zanášet zbytečné prodlevy, byl finálně implementován přístup dle ad 3). Pokud tedy funkce `Request_Image(..)` jen zažádá o zaslání dat, je nutné tyto data nějakým způsobem vyčítat. K tomuto účelu slouží dvě vlákna a to `Decode_Packet_Thread` a `Read_Packet_Thread`.

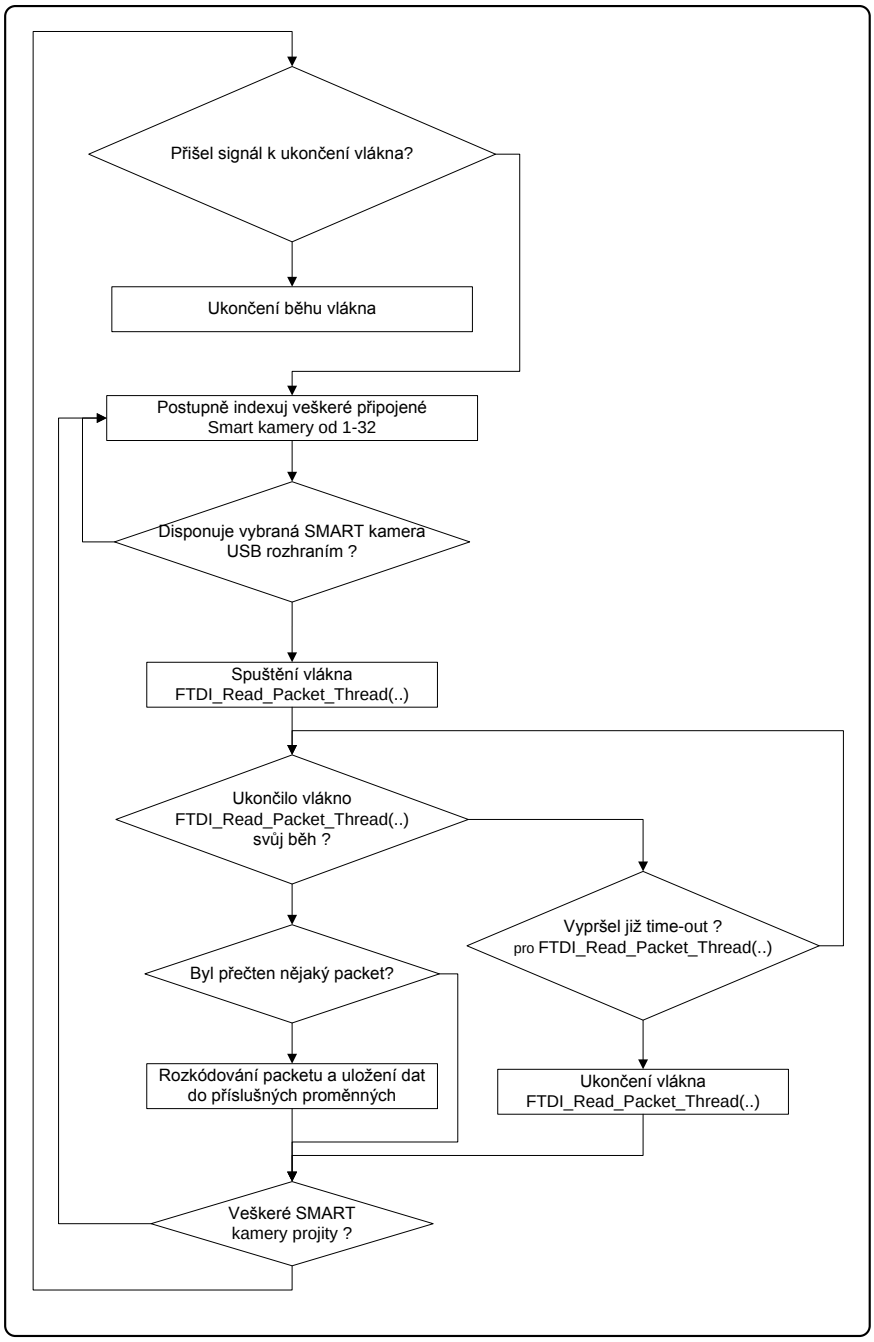
11.6.1 Vlákno Decode_Packet_Thread

Vlákno je vždy aktivováno uvnitř funkce `Find_All_Connected_Cameras(..)`, ihned po nalezení všech SMART kamer připojených k PC.

Úkolem tohoto vlákna je vyčítání příchozích packetů cyklickým voláním vlákna `FTDI_Read_Packet_Thread(..)`. Vzhledem k tomu, že vlákno `FTDI_Read_Packet_Thread(..)` již přímo komunikuje s hardwarem, je mu dán na jeho vykonání jen určitý čas. V případě překročení maximálního času, je vláknem `Decode_Packet_Thread(..)` vyslán signál k okamžitému ukončení `FTDI_Read_Packet_Thread(..)`. Ve výjovém diagramu (11.3) je zachycen princip vlákna.

11.6.2 Vlákno FTDI_Read_Packet_Thread

Voláním tohoto vlákna dochází ke kompletnímu vyčtení jednoho packetu ze zvolené SMART kamery. Volba z které SMART kamery se má packet vyčíst do vlákna vstupuje jako argument. Vždy je vyčteno jen tolik bytů z bufferu obvodu FT2232HL, kolik je třeba. Ve zdrojovém kódu (11.4) je nastíněno, jak je vyčítána hlavička packetu. Nejdříve je zjištěno, zda-li jsou vůbec nějaká data v bufferu, pokud nejsou, vlákno končí svou činnost. Následně se vyčte první byte z bufferu a porovná se s prvním bytem hlavičky.



Obrázek 11.3: Vývojový diagram vlákna `Decode_Packet_Thread(..)`

Pokud jsou shodné, další vyčítaný byte se porovná s druhým bytem hlavičky, takto se postupuje až ke čtvrtému bytu hlavičky. Vyčítání hlavičky z bufferu trvá tak dlouho,

Zdrojový kód 11.4: Vyčtení hlavičky packetu

```
1  // Test, zsa−li jsou v bufferu nejaka data
2  ftStatus = FT_GetStatus(FT_handle,&FTDI_Buffer_Rx_Bytes_Count,&TxBytes,&Event_Status);
3  if(ftStatus != 0) ExitThread(FT_METHOD_ERROR);
4  if(FTDI_Buffer_Rx_Bytes_Count == 0) ExitThread(FT_READ_0_BYTES);
5  // Nalezeni a precteni Packet_Head
6  while(1){
7          // vycteni 1. byte hlavicky
8          while(1){
9           if(FT_Read(FT_handle,&Read_Byte,1,&Bytes_Received) != 0) ExitThread(1);
10          if(Bytes_Received == 1) break;
11         }
12         tmp = (unsigned char)((int)(PACKET_HEAD));
13         if(Read_Byte == tmp){
14         }else continue;
15         .....Zde je ještě vyčten 2. a 3. byte hlavičky.....
16         // vycteni 4. byte hlavicky
17         while(1){
18          if(FT_Read(FT_handle,&Read_Byte,1,&Bytes_Received) != 0) ExitThread(1);
19           if(Bytes_Received == 1) break;
20          }
21          tmp = (unsigned char)((int)(PACKET_HEAD)>>24);
22          if(Read_Byte == tmp){
23                  break;
24          }else continue;
25 }
```

dokud neni hlavička nalezena, nebo dokud nedojde k vyčtení celého bufferu. Podobným způsobem dochází k následnému vyčtení ostatních konfiguračních dat packetu.

Pokud jsou vyčteny veškeré potřebné údaje o packetu, můžou být vyčtena samotná data, což je vyobrazeno ve zdrojovém kódu (11.5). Vyčtená data z bufferu FT2232HL jsou uložena do `Rx_Buffer` a následně překopírována do `Packet_Data`.

Zdrojový kód 11.5: Vyčtení dat packetu

```
1  while(1) {
2          Sleep(1);
3          ftStatus = FT_GetStatus(FT_handle,&FTDI_Buffer_Rx_Bytes_Count,
4                                  &TxBytes,&Event_Status);
5          //pokud je v FTDI bufferu mene dat nez je treba, nebo prave tolik kolik je treba,
6           vsechny je vycti
7          if(FTDI_Buffer_Rx_Bytes_Count <= Data_Still_To_Read_Count){
8                  if (FT_Read(FT_handle,Rx_Buffer,FTDI_Buffer_Rx_Bytes_Count,
9                                      &Bytes_Received) != 0) ExitThread(1);
10                 Data_Still_To_Read_Count -= Bytes_Received;
11                 // Vycti jen tolik, kolik je treba
12                 }else{
13                         if (FT_Read(FT_handle,Rx_Buffer,Data_Still_To_Read_Count,
14                                             &Bytes_Received) != 0) ExitThread(1);
15                         Data_Still_To_Read_Count -= Bytes_Received;
16                 }
17                 for (Count_i=0; Count_i<Bytes_Received; Count_i++ ){
18                         Packet_Data[Actual_Block_Count+Count_i] = Rx_Buffer[Count_i];
19                 }
20                 Actual_Block_Count += Bytes_Received;
21                 if( Data_Still_To_Read_Count == 0 ){
22                         break;
23                 }
24         }
```

11.6.3 Komunikace s DLL funkcemi

Při volání knihovních funkcí je třeba dodržovat určitá pravidla.

1. V paměti PC je alokována paměť jen pro uložení jedněch dat z packetu pro danou funkci. V praxi to znamená, že při volání např. funkce `Request_Global_Center(..)` musí následovat vyčtení dat pomocí `GET_Global_Center(..)`. Pokud by byla volána funkce `Request_Global_Center(..)` dvakrát po sobě, budou data která přijdou s prvním packetem přemazána daty z druhého packetu.

2. Sběrnice RS-485 nemá arbitráž. Pokud bude tedy příliš vysoký tok dat, může docházet ke ztrátě nekterých packetů. Ztrátě packetů lze zabránit tím, že před voláním dalšího požadavku, budou vyčtenu data z předchozího požadavku.

Níže jsou uvedeny dva příklady nevhodného volání knihovních funkcí a k čemu v daném případě dochází:

1. Dle (11.6) je zažádáno o vyfocení scény ze dvou různých SMART kamer.

   Zdrojový kód 11.6: Nevhodné volání knihovní funkce

   ```
   1    Request_Image(Cam_1);
   2    Request_Image(Cam_2);
   ```

   Po přijetí požadavků SMART kamerami, začnou obě vysílat packet se snímkem. Vlákno `FTDI_Read_Packet_Thread(..)` vždy může v jeden okamžik komunikovat jen s jednou SMART kamerou. Pokud se tedy začne vyčítat packet z Cam_1, bude celý vyčten. Nicméně po dobu vyčítání packetu z Cam_1 dojde k zaplnění bufferu v čipu FT2232HL na kameře Cam_2. Do bufferu se uloží kompletní hlavička packetu a prvních několik desítek KB dat. V okamžiku, kdy `FTDI_Read_Packet_Thread(..)` začne vyčítat data z Cam_2, úspěšně vyčte hlavičku a pokusí se vyčíst celý packet, nicméně data nebudou k dispozici a `FTDI_Read_Packet_Thread(..)` zamrzne ve vyčítací smyčce. K ukončení vlákna `FTDI_Read_Packet_Thread(..)` dojde až na základě vyslání signálu k ukončení z vlákna `Decode_Packet_Thread(..)`.

2. Dle (11.6) bude zažádáno o zjištění těžiště ze dvou různých SMART kamer.

   Zdrojový kód 11.7: Nevhodné volání knihovní funkce

   ```
   1    Request_Global_Center(Cam_1);
   2    Request_Global_Center(Cam_2);
   ```

   Pokud jsou obě SMART kamery připojeny přes USB, budou první přijatá data z prvního packetu, přepsány daty z druhého packetu. V případě propojení jen přes RS-485 může nastat i situace, že packety se na sběrnici vzájemně poničí a nebude vyčten ani jeden packet.

Z výše uvedených důvodů je třeba při komunikace využít příklad vyčítání dle (11.8). Tento příklad je vhodný při volání jedné funkce z jedné kamery.

V případě, že je třeba vyčítat data z vícero SMART kamer, je možné několikrát po sobě opakovat přístup (11.8), nicméně pokud by jedna SMART kamera přestala fungovat, dojde k uvíznutí programu. Proto je vhodnější využít přístup dle (11.9). Pokud by v tomto případě došlo k selhání jedné SMART kamery, data z ní nebudou vyčteny, ale data z ostatních kamer vyčítána budou.

Zdrojový kód 11.8: Příklad volání DLL funkce

```
1  while(1){
2          Request_Global_Center(Cam_1);
3          Sleep(Request_Delay_ms);
4          if(GET_Global_Center(&Cam_ID,&x,&y) == 0){
5                  if(cam_1 == *Cam_ID){
6                          //---------- Zde se zpracuji prijata data
7                          break;
8                  }
9          }
10 }
```

Zdrojový kód 11.9: Příklad volání DLL funkce

```
1  while(1){
2          Request_Global_Center(Cam_1);
3          Sleep(Request_Delay_ms);
4          if(GET_Global_Center(&Cam_ID,&x,&y)==0){
5                  if(cam_1 == *Cam_ID){
6                          //---------- Zde se zpracuji prijata data
7                  }
8          }
9          -.-.-.-.-
10         Request_Global_Center(Cam_n);
11         Sleep(Request_Delay_ms);
12         if(GET_Global_Center(&Cam_ID,&x,&y)==0){
13                 if(Cam_n == *Cam_ID){
14                         //---------- Zde se zpracuji prijata data
15                 }
16         }
17 }
```

Kapitola 12

Postřehy při programování BLACKFIN

12.1 Uložení disassembly kódu do souboru

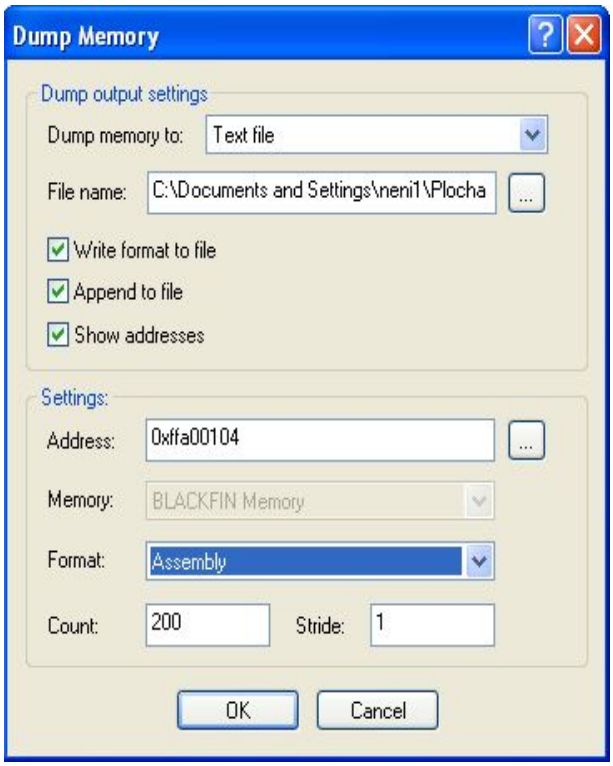
Někdy je vhodné mít uložen výpis assembly kódu v souboru, nicméně daný postup není v prostředí VDSP++ zcela přímočarý. Nejdříve klikneme pravým tlačítkem na "Disassembly window"a z nabídky vybereme příkaz DUMP. načež se zobrazí okno viz obr. 12.1(a), které vyplníme následovně:

1. Vybereme cílovou složku
2. Zaklikneme zobrazení adres v paměti "Show adresses"
3. Zvolíme počáteční adresu, odkud chceme kód zobrazovat
4. V Položce "Format"zvolíme "Assembly"výstup (je táké možný binární výstup, hexadecimální..)
5. V Položce "Count"zvolíme požadovanou délku výpisu kódu

Uložený kód bude vypadat viz obr. 12.1(b)

12.2 Měření času vykonávaného kódu

V mnoha případech je třeba změřit délku vykonávání částí kódu. Někdy je třeba znát počet instrukčních cyklů procesoru, nicméně vhodnější je znalost doby přímo v sekundách.



(a)

```
[FFA00112] R0 = 1 ;
[FFA00114] CALL metoda ;
[FFA00118] NOP ;
[FFA0011A] UNLINK ;
[FFA0011E] RTS ;
[FFA00120] R0 = 0 ;
[FFA00122] IDLE ;
[FFA00124] SSYNC ;
[FFA00126] JUMP.S ___lib_prog_term ;
[FFA00128] R0 = 9 ;
[FFA0012A] R1 = SEQSTAT ;
[FFA0012C] R1 = R1 << 0x1a ;
[FFA00130] R1 = R1 >> 26 ;
[FFA00134] R2 = 0 ;
[FFA00136] JUMP.S _adi_fatal_exception ;
[FFA00138] [ -- SP ] = ( R7:0 , P5:0 ) ;
[FFA0013A] [ -- SP ] = ASTAT ;
[FFA0013C] [ -- SP ] = I0 ;
[FFA0013E] [ -- SP ] = I1 ;
[FFA00140] [ -- SP ] = I2 ;
[FFA00142] [ -- SP ] = I3 ;
[FFA00144] [ -- SP ] = LT0 ;
[FFA00146] [ -- SP ] = LB0 ;
[FFA00148] [ -- SP ] = LC0 ;
[FFA0014A] [ -- SP ] = LT1 ;
```

(b)

Obrázek 12.1: Uložení disassembly kódu do souboru

V kódu (12.1) je znázorněno, jak je možné úsek kódu změřit, čerpáno bylo z [7].

12.3 Volba typu použité paměti

Pokud je požadováno umístění programu či dat do FLASH paměti, je třeba před daný kód vložit speciální direktivu viz (12.2). Dané direktivy se musí vložit vždy před každou metodu, či data zvlášť.

12.4 Změny v power managementu

Změnami v power managementu se rozumí různé změny parametrů fázového závěsu, či uvádění procosoru do módu spánku atp. Jakékoliv změny by neměly být prováděny

Zdrojový kód 12.1: Zjištění doby vykonání části kódu

```
#include <time.h>
#include <stdio.h>
extern int main(void){
volatile clock_t clock_start;
volatile clock_t clock_stop;
double secs;
clock_start = clock();
Some_Function_Or_Code_To_Measure();
clock_stop = clock();
secs = ((double) (clock_stop - clock_start)) / CLOCKS_PER_SEC;
printf("Time taken is %e seconds\n",secs);
}
```

Zdrojový kód 12.2: Volba typu použité paměti

```
        #pragma section("FLASH_code")
        #pragma section("L1_code") // v jazyce C
        .section L1_code; //v assembleru
```

přímým zápisem do registrů power managementu, ale je doporučeno využívat funkci bfrom_SysControl(). Tato funkce je umístěna v ROM paměti procesoru a lze ji volat standardním C způsobem.

12.5 Použití DMA kanálu

Pokud je k nějaké periferii aktivován DMA kanál, veškeré přerušení z dané periferie jsou obslouženy DMA požadavky. Pokud je DMA kanál deaktivován, DMA jednotka ignoruje přerušení z periferie, která jsou následně předána přímo kontroléru přerušení. Aby se zabránilo neočekávanému chování procesoru, je třeba aktivovat DMA kanál před aktivací periferie a naopak deaktivovat periferii před deaktivací DMA kanálu. Čerpáno z [8], strana 378.

Zdrojový kód 12.3: Hlavičkový soubor pro přerušení

```
1 #include <sys\exception.h>
```

Zdrojový kód 12.4: Příklad vytvoření přerušení

```
1 #include <sys\exception.h>
2 EX_INTERRUPT_HANDLER(DMA_UART_RX); // vytvoreni handleru
3 void main(void){
4         register_handler(ik_ivg10, DMA_UART_RX); // prirazeni handleru
5         *pSIC_IMASK = 0x00004000; // povoleni preruseni od DMA6 (UART RX)
6 }
7 EX_INTERRUPT_HANDLER(DMA_UART_RX){ // obsluzna metoda preruseni
8 }
```

12.6 Využití GPIO

Pokud je nějakému GPIO pinu aktivován příslušný driver, daný pin již nesmí být použit jako výstupní. V žádném případě se nesmí aktivovat vstupní driver nastavením registru PORTxIO_INEN a zároveň nastavit daný pin jako výstupní nastavením PORTxIO_DIR registru.

12.7 Využití přerušení

Aby bylo možné využívat přerušení od zvolené periferie, je třeba inkludovat hlavičkový soubor dle (12.3).

Následující kód přiřazuje pro lepší pochopení přerušení k DMA UART-RX. viz (12.4).

Název handleru, v tomto případě "DMA_UART_RX", je možné si zvolit jakýkoliv, nicméně takový aby bylo názorné k čemu patří.

12.8 Optimalizace kódu

Při zapnuté optimalizaci v prostředí VDSP++ může být zpracování kódu rychlejší až 20-krát. Proto je vhodné optimalizaci vždy používat. Čerpáno z [12] s.464.

12.9 Debuggování programu v FLASH

Debuggování programu v FLASH paměti je téměř shodné s debuggováním programu v L1 paměti. Jediné omezení tkví v tom, že při debuggování kódu z FLASH paměti je nutné využívat hardwarové breakpointy. Hardwarových breakpointů je pouze 6, naproti tomu softwarových breakpointů může být libovolný počet. Z tohoto důvodu je vhodné do FLASH paměti umisťovat již stabilní kód a program, který je třeba debuggovat umisťovat v L1.

12.10 Zjištění rozložení kódu v paměti

V mnoha případech je třeba znát, jaké paměti a kolik paměti bylo obsazeno po kompilaci kódu. Nejlepší řešení jak dané rozložení zjistit, je vytvoření MAP souboru. O vytvoření MAP souboru lze VDSP++ zažádat následovně: Project Options/Link/Generate symbol map. Vygenerovaný MAP soubor lze následně otevřít ve webovém prohlížeči. Příklad MAP souboru, kde je zobrazena jen úvodní hlavička je zobrazen na obr.(12.2).

Memory map of link project *p0*

Memory	Start address	End address	Qualifier	Width	Used words
MEM_L1_SCRATCH	0xffb00000	0xffb00fff	RAM	0x8	0x4c
MEM_L1_CODE_CACHE	0xffa04000	0xffa07fff	RAM	0x8	0x0
MEM_L1_CODE	0xffa00000	0xffa03fff	RAM	0x8	0x4000
MEM_L1_DATA_A_CACHE	0xff804000	0xff807fff	RAM	0x8	0x0
MEM_ARGV	0xff800000	0xff8000ff	RAM	0x8	0x1
MEM_L1_DATA_A	0xff800100	0xff803fff	RAM	0x8	0x3f00
MEM_FLASH_BOOT	0x20000000	0x2001ffff	ROM	0x8	0x0
MEM_FLASH	0x20020000	0x203fffff	ROM	0x8	0x1750

Obrázek 12.2: Rozložení využité paměti

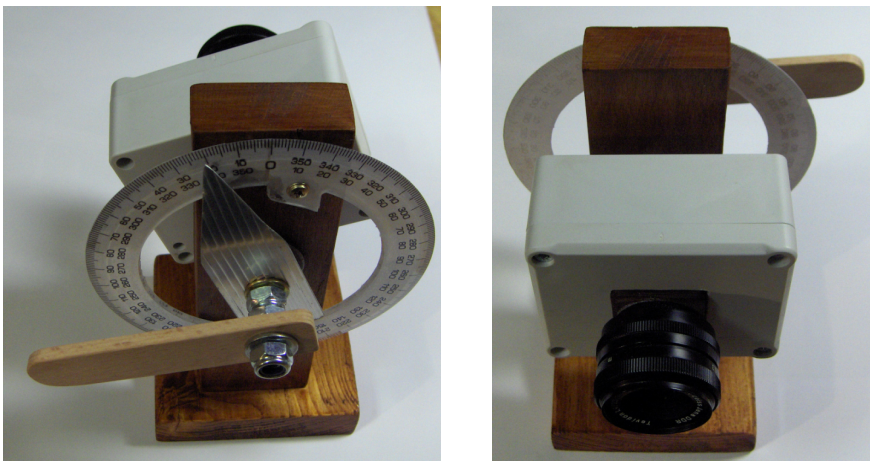
12.11 Eliminace nepotřebných metod

Pokud jsou v projektu použity runtime knihovny, nelze předem určit, kolik paměti bude po kompilaci programu vyžadováno. Jedinou možností je následná kontrola použité paměti v MAP souboru dle (12.10). Dále je vhodné eliminovat nepotřebných metody pomocí: Project Options/Link/Eliminate unused objects.

Kapitola 13

Návrh mechanické části

Navrženou SMART kameru je řeba testovat, k tomuto účelu je nutné její zapouzdření v krabičce. Veškeré navržené algoritmy je možné testovat pomocí držáku obrazců dle obr.(13.3). Navržený algoritmus určení úhlu náklonu je možné testovat přímo pohledem na horizont, nicméně této možnosti nelze využít pokud je nevhodné počasí, či je kolem SMART kamery četná zástavba. Z tohoto důvodu je stojan SMART kamery vybaven úhloměrem viz obr.(13.1). Do držáku obrazců je možné vložit motiv dle obr. (13.2(b)), popřípadě jiný, pomocí kterých lze zcela simulovat venkovní prostředí. Algoritmy určení těžiště je možné testovat vložením černých papírových proužků s bílými markery viz. (13.2(a)).



(a) (b)

Obrázek 13.1: Zapouzdření SMART kamery

(a) (b)

Obrázek 13.2: Motivy pro testy algoritmů

Obrázek 13.3: Měřící pracoviště

Kapitola 14

Seznam použitých zkratek a symbolů

GPIO	General purpose input/output
CCLK ...	Core Clock
SCLK	System Clock
MPSSE ..	Multi-Protocol Synchronous Serial Engine
EBIU	External bus interace unit
ALU	Arithmetic logic unit
MAC	Multiply and accumulate
EBIU	External bus interface unit
DMA	Direct memory acces
RSI	Removable storage interface
DSP	Digital signal processor
PPI	Parallel peripheral interface
HCOUNT	Počet pixelů v řádku CMOS snímače
VCOUNT	Počet řádků na snímek CMOS snímače

Kapitola 15

Zhodnocení dosažených výsledků

V rámci této diplomové práce byla zkonstruována SMART kamera založená na procesoru BF504F a příslušné programové vybavení. Dále byly zkonstruovány dva vývojové kity s procesory BF53x a BF504F.

V počátku realizace této práce bylo cílem navrhnout SMART kameru s procesorem BF53x, proto byl nejdříve zkonstruován vývojový kit s tímto procesorem, kde by se mohlo vyzkoušet obvodové zapojení. V druhé polovině roku 2010 firma Analog Devices nabídla k otestování prototypovou sérii procesorů BF504F s revizí 0.0. Dané procesory byly shledány jako případná alternativa k BF53x, proto bylo několik kusů Doc.Fischerem na katedru měření získáno. Následně byl navržen další vývojový kit tentokrát s procesorem BF504F, jež byl nakonec vybrán pro použití ve SMART kameře.

Procesor BF504F, byl prvním nasazeným procesorem v České republice. Protože se jednalo přímo o vzorky z prototypové výroby bylo možné očekávat interní chyby procesoru, jež výrobce odstraní až v revizi 0.1. Odhalené chyby v procesoru(tzv. anomálie), přiznané výrobcem jsou uvedeny v [11]. Celkem bylo ke dni 18.02.2011 odhaleno celkem 26 anomálií, z toho například:

- Anomálie č.2
  Pro testování konce běhu DMA kanálu nelze číst status bit DMA_RUN. Místo toho je nutné využívat status bit DMA_DONE.

- Anomálie č.13
  Pokud je VDDINT napájeno dříve než-li VDDEXT, může dojít k nesprávnému nastavení frekvence. Nutné v inicializační rutině přenastavit. Tento problém se projevuje jen tehdy, pokud je ponecháváno defaultní nastavení fázového závěsu.

- Anomálie č.14
  Defaultní hodnota MSEL v PLL_CTL registru je nesprávně nastavena na 0x5.

Před každou prací s tímto procesorem je vhodné si tento dokument prostudovat. V konečném důsledku to může být úspora mnoha hodin, v případě, že by nějaká anomálie způsobovala nepředvídatelné chování procesoru.

Práce s procesorem BF504F měla také zodpovědět otázku, zda-li je vhodné tento procesor uvažovat pro další návrhy SMART kamer či jiných zařizení. Samotný výkon procesoru a velké množství podporovaných periferií a nízká cena jasně hovoří pro jeho nasazení. Nicméně je zde také jistý deficit a tím je malá SRAM paměť(32kB pro data a 32kB pro program) bez možnosti jejího dalšího rozšíření. Například u procesoru BF53x je možné přes EBIU rozhraní připojit externí RAM paměť, naproti tomu BF504F využívá EBIU rozhraní výhradně ke komunikaci s FLASH pamětí umístěnou ve stejném pouzdru. Z výše uvedeného je tedy zřejmé, že z L1 paměti lze vykonávat jen programy do velikosti 32 kB. Pro účely jednodušších úloh zpracování obrazu bez použití operačních systémů je tato velikost RAM paměti zcela dostačující. Nicméně pro komplexnější algoritmy již dostačovat nemusí. Jistým východiskem z tohoto problému, je využití integrované FLASH paměti, z které lze přímo vykonávat program. Z kapitoly (9.7) je nicméně zřejmé, že přímé vykonání programu z FLASH paměti je časově mnohonásobně delší oproti užití L1. Tímto způsobem lze FLASH pamět využít jen pro inicializační rutiny a časově nenáročné algoritmy. Stále zde nicméně existuje možnost rychlého vykonání náročných algoritmů, jež jsou náročné i paměťově. Lze využít metody "overlays" či cache kontroléru k překopírování potřebných algoritmů z FLASH do SRAM paměti. Při využití tohoto přístupu je tedy možné vykonávat rozsáhlé programy v rychlé L1 paměti. Nicméně zde stále bude platit omezení, že v L1 nelze mít umístěny větší bloky programu než 32 kB, což tedy vede k náročnější správě instrukčního toku. V případě použití cache se v SRAM paměti vyčlení 16 kB jen pro účely cache úkonů, a tuto pamět již bude spravovat jen cache kontrolér. Uživatelská SRAM paměť bude tedy zmenšena na 16 kB. Zhodnocením výše uvedeného lze tedy konstatovat, že z paměťového hlediska lze procesor bez potíží využívat pro programy jejichž velikost nepřesahuje 32 kB. Větší programy, za jistých omezení, lze nicméně spouštět také.

V hardwarové části návrhu byly navrženy dva vývojové kity, jeden s procesorem BF53x a druhý s procesorem BF504F. Oba kity byly zhotoveny celkem ve dvou verzích, kde ve druhé verzi byly vždy odstraněny některé chyby na plošném spoji. Vývojové kity se osvědčily při testování vývojových aplikací a nyní již aktivně slouží při výuce mikroproce-

sorové techniky na katedře měření. Dále byla navržena a zkonstruována SMART kamera s procesorem BF504F. SMART kamera byla navržena tak, aby k ní mohly být připojovány různé moduly s CMOS snímači. Disponuje napájením jak přes USB, tak z externího zdroje. SMART kamera je navržena jako modulární tak, aby bylo možné připojovat další moduly přes sběrnici SPI. Dále disponuje jak rozhraním USB, tak i rozhraním RS-485.

V dalších krocích vývoje bude možné SMART kameru, při použití procesoru BF504F, dále miniaturizovat i když na úkor některých funkcionalit. Právě ve fázi miniaturizace bude možné těžit z velmi malého pouzdra procesoru. Při požadavku na co nejvyšší miniaturizaci SMART kamery by mohly být podniknuty následující kroky:

1. Odebrání USB rozhraní
   Pro podporu sběrnice USB je třeba obvod FT2232HL, paměť 23c46, konektor USB a další diskrétní součástky. Celkem tedy podpora USB rozhraní vyžaduje poměrně velký prostor na PCB. Komunikace by mohla být realizována jen pomocí RS-485.

2. Umístění CMOS sensoru přímo na PCB SMART kamery
   V nynější podobě se CMOS sensor připojuje jako externí modul k modulu SMART kamery pomocí 30-ti pinového konektoru. Délka tohoto konektoru (4 cm) je klíčovým prvkem, jež určuje výslednou velikost celé SMART kamery. Pokud by byl CMOS sensor umístěn přímo na PCB SMART kamery, propojovací konektor by již nebyl třeba. Nevýhodou tohoto řešení by byl fakt, že by již nebylo možné připojovat různé druhy CMOS sensorů, ale jen takový, na který by SMART kamera byla zkonstruována.

3. Odebrání JTAG konektoru
   JTAG rozhraní hraje nepostradatelnou roli při návrhu softwaru SMART kamery. Nicméně JTAG konektor je 14-pinový header, vyžadující 2x0,5 cm plochy na PCB. Absence JTAG rozhraní by nicméně mohla být řešena tak, že software by byl navržen a vyzkoušen v SMART kameře navržené v této diplomové práci a do miniaturizované verze by již byl nahrán ověřený kód. V případě, že by se výsledný hardware v něčem lišil, musel by být vytvořen softwarový monitor, jež by zasílal obsahy požadovaných registrů po RS-485, čímž by se částečně nahradilo JTAG rozhraní.

Verze miniaturizované SMART kamery, s přihlédnutím k výše popsaným omezením, by s daným procesorem mohla mít rozměry přibližně 4x3 cm.

Při práci s vývojovým kitem a samotnou SMART kamerou s procesory BF504F, nastávaly technické komplikace, jež několikrát vedly k destrukci procesoru. Proto muselo být několikrát přistoupeno k jeho výměně. Při prvním pokusu o odpájení pouzdra bylo postupováno tak, že na pouzdro byl zvrchu přiváděn horký vzduch o teplotě 400°C a ze spodu za pomocí páječky byl odpajován termální pad. Při dalších pokusech se ukázalo, že odpájení pouzdra procesoru je možné jen za pomoci horkého vzduchu. Pouzdro má takovou dostatečnou tepelnou vodivost, že dojde i k tavení pájky pod velkým uzemňovacím padem.

Při návrhu komunikačního kanálu mezi BF504F a obvodem FT2232HL, bylo cílem dosažení co nejvyšší komunikační rychlosti. Komunikační rozhraní bylo zprvu napsáno v jazyce C, nicméně tento jazyk neumožňoval striktní dodržování časů, jež jsou na sběrnici požadovány. Následně bylo tedy přistoupeno k napsání komunikace v assembleru, kde bylo dosaženo přenosové rychlosti 7,180 MB/s. Obvod FT2232HL sice umožňuje ještě vyšší přenosovou rychlost, nicméně procesor již nikoliv.

Tato práce měla také odpovědět na otázku, zda-li je vhodné využívat assembleru při návrhu rychlých algoritmů zpracování obrazu. Ačkoliv při práci s assemblerem je zcela jasné, kdy a která instrukce bude vykonána, nelze obecně říci, že kód napsaný v assembleru bude vždy rychlejší. Daná skutečnost je také podtržena tím, že vývojové prostředí VDSP++ využívá výkonný kompilátor s různými možnými nastaveními optimalizace. V algoritmu rozpoznání více markerů, jež je kompletně napsán v assembleru na téměř 600 řádcích dochází v několika místech ke ztrátě vždy 4 instrukčních cyklů zbytečným čekáním. Toto čekání je zapříčiněno tím, že je do registru, který slouží jako odkaz do paměti zapsána adresa určitého prvku pole a hned v další instrukci je do daného místa zapisováno. Aby nedocházelo k dané prodlevě, musely by tedy být tyto dvě instrukce od sebe odděleny nějakými mezivýpočty využívajícími minimálně dané 4 instrukční cykly, nicméně takto se kód stává velmi nepřehledný. Pokud je funkce napsána v jazyce C, jež je následně zkompilována, již k daným prodlevám nedochází. Zkompilovaný kód je sice nepřehledný, nícméně tím, že je upravován v jazyce C, to ničemu nevadí. Místem, kde je assembler zcela nenahraditelný je psaní funkcí, kde je požadováno striktní dodržování určitých časů, například softwarové implementace komunikačních rozhraní. Pokud je frekvence jádra nastavena na 400 Mhz, lze dodržovat časování po 2,5 ns, což je pro dané účely více než dostačující. V jazyce C by takovýchto časů nebylo možné nikdy dosáhnout.

Celkem byly implementovány 3 algoritmy zpracování obrazu. První algoritmus je schopný určit pozici markeru v obraze, pomocí metody určení těžiště světelné stopy. Další algoritmus je matematicky založený na stejném principu jako algoritmus určení

pozice jednoho markeru, nicméně dokáže určit až 15 nezávislých středů různých markerů najednou. Poslední implementovaný algoritmus je schopný určit úhel náklonu SMART kamery od horizontu. U všech navržených algoritmů bylo cílem vyzkoušet zejména jejich možnost implementace, určování přesnosti chyb tedy nebylo realizováno.

SMART kamera je schopna přenosu snímku do nadřazeného PC pomocí sběrnice USB, kde dochází k zapouzdření snímku do formátu BMP, jež je posléze uložen na pevný disk počítače. SMART kameru je tedy možné využívat i jako jednoduchý fotoaparát.

Veškerá funkcionalita SMART kamery je přístupná pomocí navržených DLL knihoven. První navržená DLL knihovna je napsána v jazyce C++ a její implementace je možná v jakémkoliv programovacím jazyce. Cílem práce nicméně bylo, aby bylo možné SMART kameru implementovat i ve vývojovém prostředí LABVIEW. Pro snažší práci v tomto prostředí byla navržena ještě jedna knihovna v jazyce C#, která jen přejímá kompletní funkcionalitu předešlé knihovny. Přímá implementace DLL knihovny do LABVIEW psané v C++ je taktéž možná, nicméně zdaleka neumožnuje takovy komfort obsluhy jako DLL knihovna napsaná v C#.

Kapitola 16

Závěr

V této práci byla navržena SMART kamera jak po hardwarové, tak po softwarové stránce. Dále byly navrženy dva vývojové kity s procesory BF53x A BF504F. Navržené plošné spoje a mechanické uspořádání SMART kamery jsou z přiložených fotografií jasně viditelné. Softwarová část je poměrně rozsáhlá, zdrojové kódy jsou na několika tisících řádcích a nelze se v tomto textu zabývat každou navrženou metodou. Nicméně softwarové projekty jak pro firmware SMART kamery, tak pro návrh aplikací pro PC jsou psány strukturovaně po blocích, jež náleží dané funkcionalitě. Pro bližší seznámení se s danou implementací je tedy vhodné nahlédnout přímo do zdrojových kódů, jež jsou nedílnou součástí této diplomové práce.

Literatura

[1] *Fischer J.* **Optoelektronické senzory a videometrie**
Praha 2002

[2] *Záhlava V.* **Návrh a konstrukce DPS**
Praha 2010

[3] *Matoušek D.* **USB prakticky s obvody FTDI**
Praha 2003

[4] *FTDI Ltd.* **Katalogový list: FT2232HL**
`http://www.ftdichip.com`

[5] *FTDI Ltd.* **Katalogový list: Software Application Development D2XX Programmer's Guide**
`http://www.ftdichip.com`

[6] *ANALOG DEVICES* **Katalogový list: ADSP-BF531, BF532, BF533**

[7] *ANALOG DEVICES* **Katalogový list: Cycle Counting and Profiling EE-332**

[8] *ANALOG DEVICES* **Katalogový list: ADSP-BF533 Blackfin Processor Hardware Reference**

[9] *ANALOG DEVICES* **Katalogový list: ADSP-BF504/ADSP-BF504F/ADSP-BF506F**

[10] *ANALOG DEVICES* **Katalogový list: ADSP-BF50x Blackfin Processor Hardware Reference**

[11] *Analog Devices* **Katalogový list: ADSP-BF504/BF504F/BF506F Silicon Anomaly List**

[12] *Analog Devices* **C/C++ Compiler and Library Manual for Blackfin Processors**

[13] *Analog Devices* **Katalogový list: Engineer-to-Engineer Note VTG-001**

[14] *Analog Devices* **Katalogový list: Using Cache Memory on Blackfin® Processors EE-271**

[15] *Analog Devices* **Visual DSP++ 5.0 Linker and Utilities Manual**

[16] *Bělík P.* **Snímač polohy využívající obrazové informace**
Praha 2011

[17] *Pavlíček T.* **Metody průběžného zpracování obrazu a jejich implementace do signálového procesoru Blackfin ADSP-BF532**
Praha 2008

[18] *Pribula O.* **Vývojový modul pre spracovanie obrazu so signálovým procesorom ADSP-Blackfin**
Praha 2007

[19] *Grepl L.* **Modul zpracování obrazu se signálovým procesorem Blackfin ADSP-BF53x**
Praha 2008

[20] **Tvorba BMP**
`http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html`

Kapitola 17

Přílohy

17.1 DLL funkce

17.1.1 Find_All_Connected_Cameras

Vlastnosti funkce

Tato funkce musí být volána jako první v pořadí. Zde dochází k alokaci paměti pro veškeré ostatní funkce. Dále zde dochází k vyhledání veškerých připojených kamer do sítě a uložení informací o nich do paměti.

Definice funkce

```
int Find_All_Connected_Cameras(int *Cameras_Count)
```

Parametry funkce

`int *Cameras_Count`..... V tomto pointeru je předán počet nalezených SMART kamer

Návratová hodnota funkce

0..... Úspěšné volání

17.1.2 Close_All_Connected_Cameras

Vlastnosti funkce

Tato funkce musí být volána pro ukončení práce se SMART kamery. Jsou zde ukončeny komunikační kanály s veškerými obvody FTDI. Dále zde jsou dealokována veškerá paměť, jež byla dynamicky alokována ve funkci Find_All_Connected_Cameras.

Definice funkce

```
int Close_All_Connected_Cameras(void)
```

Parametry funkce

`bez parametrů`.....

Návratová hodnota funkce

0..... Úspěšné volání

17.1.3 Request_Image

Vlastnosti funkce

Tato funkce zažáda SMART kameru o vytvoření snímku. Vytvořený snímek se následně uloží do souboru, kde je uložena DLL knihovna ve tvaru "image.bmp". Volat Request_Image lze jen u takových SMART kamer, které jsou připojeny pomocí USB rozhraní.

Definice funkce

```
int Request_Image(char Cam_ID)
```

Parametry funkce

`char Cam_ID`.....ID kamery, která má být volána(1-32).

Návratová hodnota funkce

0..... Žádost byla úspěšně odeslána

1..... Kamera s daným Cam_ID není připojena

17.1.4 Get_Image

Vlastnosti funkce

V případě, že některá kamera odeslala do PC snímek, tak tato metoda určí o kterou kameru se jedná.

Definice funkce

```
int Get_Image(char *Cam_ID);
```

Parametry funkce

`char *Cam_ID`.....Pointer na kameru, která zaslala snímek

Návratová hodnota funkce

0..... Snímek nalezen
1..... Žádný snímek nepřišel

17.1.5 Request_Global_Center

Vlastnosti funkce

Funkce zažádá kameru o výpočet těžiště nad celým obrazem.

Definice funkce

```
int Request_Global_Center(char Cam_ID)
```

Parametry funkce

`char Cam_ID`.....ID kamery, která má být volána(1-32).

Návratová hodnota funkce

0..... Žádost byla úspěšně odeslána
1..... Kamera s daným Cam_ID není připojena

17.1.6 GET_Global_Center

Vlastnosti funkce

Funkce vrací souřadnice vypočítaného těžiště.

Definice funkce

```
int GET_Global_Center(char *Cam_ID,int *x,int *y)
```

Parametry funkce

`char *Cam_ID`.....Pointer na kameru, která zaslala data
`int *x`..... Pointer na nalezenou x souřadnici těžiště
`int *y`..... Pointer na nalezenou y souřadnici těžiště

Návratová hodnota funkce

0..... Funkce vrací platná data.
1..... Data nejsou k dispozici.

17.1.7 Request_Centers

Vlastnosti funkce

Funkce zažádá kameru o nalezení markerů. Maximální počet nalezených markerů je 15. Hodnoty jěž budou vypočteny:

- Počet nalezených markerů
- Optické těžiště každého markeru
- Optická hmotnost každého markeru

Definice funkce

```
Request_Centers(char Cam_ID)
```

Parametry funkce

`char Cam_ID`.....ID kamery, která má být volána(1-32).

Návratová hodnota funkce

0..... Žádost byla úspěšně odeslána

1..... Kamera s daným Cam_ID není připojena

17.1.8 GET_Centers

Vlastnosti funkce

Funkce vrací data o nalezených markerech. Před voláním této funkce musí být alokována pamět pro pole(x,y, Weights) o délce 15 pro každé.

Definice funkce

```
GET_Centers(char *Cam_ID,unsigned char *Found_Marks_Count,unsigned int *x,
          unsigned int *y,unsigned int *Weights);
```

Parametry funkce

`char* Cam_ID`.....Pointer na kameru, která zaslala data

`char *Found_Marks_Count`..... Pointer na počet nalezených markerů

`int *x`.....Pointer na pole x souřadnic (paměť musí být alokována již v uživatelském programu)

`int *y`.....Pointer na pole x souřadnic (paměť musí být alokována již v uživatelském programu)

`int *Weights`..... Pointer na pole optických hmotností daných markerů (paměť musí být alokována již v uživatelském programu)

Návratová hodnota funkce

0..... Funkce vrací platná data.

1..... Data nejsou k dispozici.

17.1.9 Request_Lean

Vlastnosti funkce

Funkce zažádá kameru o výpočet úhlu náklonu. Zvyšující se úhel proti směru hodinových ručiček od horizontály je značen záporným znaménkem.

Definice funkce

```
int Request_Lean(char Cam_ID)
```

Parametry funkce

`char Cam_ID`.....ID kamery, která má být volána(1-32).

Návratová hodnota funkce

0..... Žádost byla úspěšně odeslána
1..... Kamera s daným Cam_ID není připojena

17.1.10 GET_Lean

Vlastnosti funkce

Funkce vrací úhel náklonu.

Definice funkce

```
int GET_Lean(char *Cam_ID,short *Angle)
```

Parametry funkce

`char *Cam_ID`.....Pointer na kameru, která zaslala data
`short *Angle`..... Pointer na úhel

Návratová hodnota funkce

0..... Funkce vrací platná data.
1..... Data nejsou k dispozici.

17.1.11 CMOS_Sensor_Set_Reg

Vlastnosti funkce

Funkce zapisuje data do registrů CMOS snímače připojeného ke SMART kameře. Data jsou do sensoru zapisovány přes sběrnici I2C.

Definice funkce

```
CMOS_Sensor_Set_Reg(char Cam_ID,char CMOS_reg_addr,short CMOS_reg_data)
```

Parametry funkce

`char Cam_ID`.....ID kamery, která má být volána(1-32)
`char CMOS_reg_addr`.....Adresa registru CMOS snímače
`short CMOS_reg_data`.....Data která mají být zapsána

Návratová hodnota funkce

0..... Packet požadující zápis do registru byl úspěšně zaslán. Tímto nicméně není ověřeno, že data byla do CMOS sensoru skutečně zapsána. Vhodné ověřit vyčtením hodnot pomocí Request_CMOS_Sensor_Read_Reg.

17.1.12 Request_CMOS_Sensor_Read_Reg

Vlastnosti funkce

Funkce zažádá o vyčtení registru z CMOS sensoru.

Definice funkce

```
Request_CMOS_Sensor_Read_Reg(char Cam_ID, char CMOS_reg_addr)
```

Parametry funkce

`char Cam_ID`.....ID kamery, z které mají být data vyčteny `char CMOS_reg_addr`.....Adresa registru CMOS snímače, z kterého mají být vyčtena data

Návratová hodnota funkce

0..... OK
1..... Daná SMART kamera není připojena.

17.1.13 Get_CMOS_Sensor_Read_Reg

Vlastnosti funkce

V případě, že některá kamera odeslala do PC data z registru z CMOS snímače,tato metoda určí o kterou kameru se jedná a dodá potřebná data.

Definice funkce

```
Get_CMOS_Sensor_Read_Reg(char * Cam_ID, char * CMOS_reg_addr, short * CMOS_
reg_data)
```

Parametry funkce

`char *Cam_ID`.....Pointer na kameru, která zaslala data
`char *CMOS_reg_addr`.....
`short * CMOS_reg_data`.....

Návratová hodnota funkce

0..... Funkce vrací platná data
1..... Žádná data nejsou k dispozici

17.1.14 FTDI_Reset_Buffer

Vlastnosti funkce

Funkce resetuje hardwarové buffery na obvodu FT2232HL.

Definice funkce

```
int FTDI_Reset_Buffer(char Cam_ID, int uMask)
```

Parametry funkce

`char Cam_ID`.....ID SMART kamery, na které mají být buffery resetovány

`int uMask`..... 1 pro reset RX bufferu, 2 pro reset TX bufferu, 3 pro reset obou bufferů

Návratová hodnota funkce

0..... Reset bufferu úspěšně proběhl

17.2 Vývojový kit s BF533

Tabulka 17.1: Seznam součástek vývojového kitu s BF53x

Pořadí	**Počet**	**Reference**	**Hodnota**
1	1	C4	3.3 uF
2	4	C6,C7,C19,C20	27 pF
3	26	C15,C16,C21,C26,C28,C31, C34,C36,C38,C39,C40,C41, C42,C43,C44,C45,C46,C49, C51,C53,C69,C70,C71,C99, C100,C101	100 nF
4	3	C17,C18,C24	4.7 uF
5	1	C25	100 uF
6	6	C27,C30,C72,C73,C74,C75	10 uF
7	28	C58,C59,C60,C61,C62,C63, C64,C65,C76,C77,C78,C79, C80,C81,C84,C85,C86,C87, C88,C89,C90,C91,C92,C93, C94,C95,C96,C97	330 pF
8	2	C103,C104	330 nF
9	2	D1,D2	5988170107F
10	1	D3	5988140107F
11	1	D4	SK36A
12	1	D5	5988110107F
13	1	J2	S2G7
14	1	J16	ARK550/2EX
15	1	J17	S1G1
16	1	Q2	2N3906
17	69	R1,R5,R7,R9,R11,R12,R13, R14,R15,R17,R18,R19,R20, R21,R22,R23,R24,R25,R27, R28,R35,R36,R37,R38,R39, R40,R41,R42,R43,R44,R45,	33

Pokračování tabulky na další stránce...

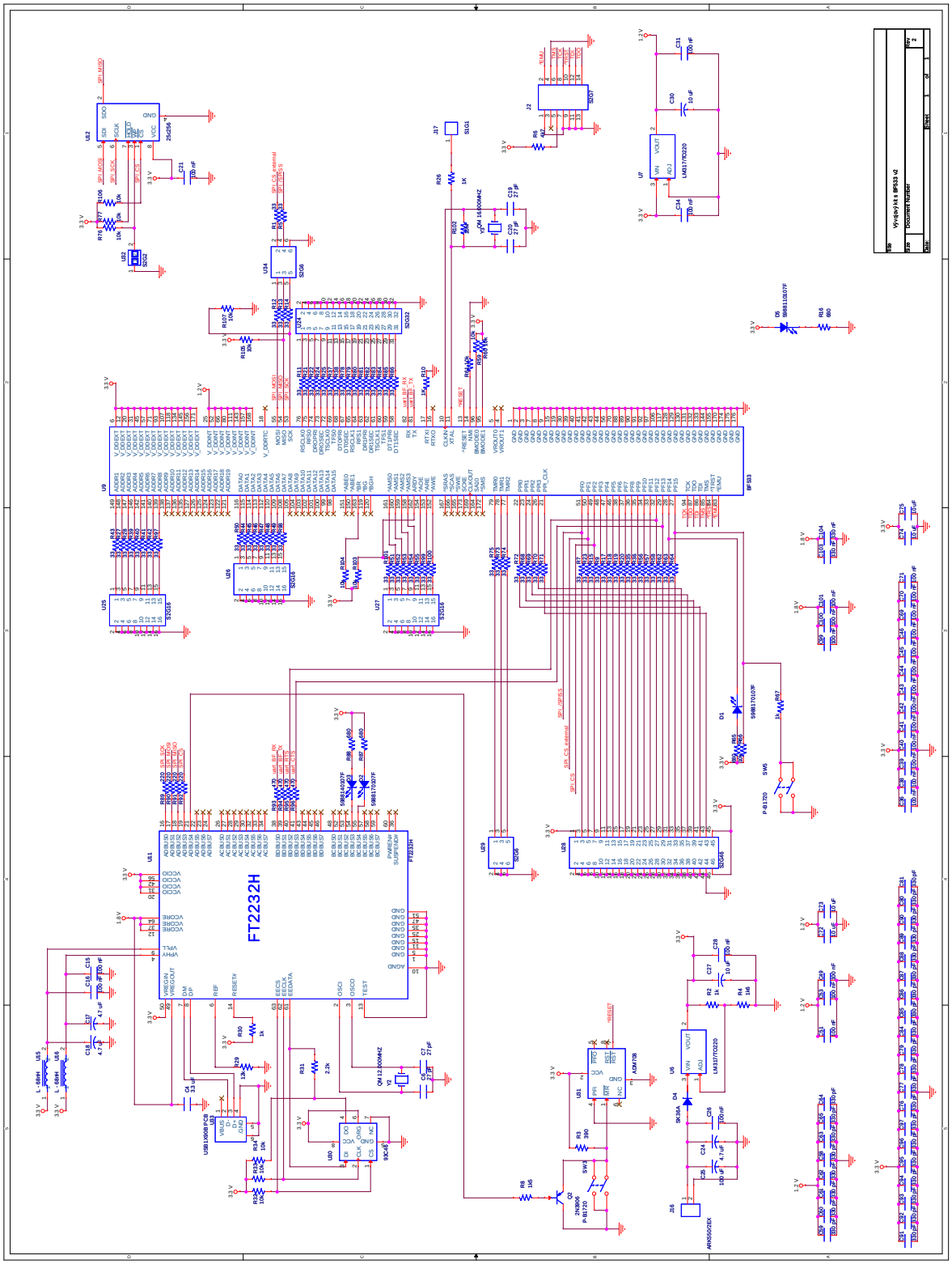
Tabulka 17.1 – Pokračování

Pořadí	**Počet**	**Reference**	**Hodnota**
		R46,R47,R48,R49,R50,R51,	
		R52,R53,R54,R55,R56,R57,	
		R58,R62,R63,R64,R68,R69,	
		R70,R71,R72,R73,R74,R75,	
		R78,R79,R80,R81,R82,R83,	
		R84,R85,R86,R97,R98,R99,	
		R100,R101	
18	5	R2,R10,R26,R30,R67	1k
19	1	R3	390
20	1	R4	1k6
21	1	R6	4k7
22	1	R8	1k5
23	4	R16,R65,R87,R88	680
24	1	R29	12k
25	1	R31	2.2k
26	14	R32,R33,R34,R59,R60,R61,	10k
		R66,R76,R77,R103,R104,	
		R105,R106,R107	
27	4	R89,R90,R91,R92	220
28	4	R93,R94,R95,R96	470
29	1	R102	10M
30	2	SW3,SW5	P-B1720
31	2	U6,U7	LM317/TO220
32	1	U9	BF533
33	1	U11	FT2232H
34	1	U12	25c256
35	2	U15,U16	L - 68nH
36	1	U24	S2G32
37	3	U25,U26,U27	S2G16
38	1	U28	S2G46
39	2	U29,U34	S2G6

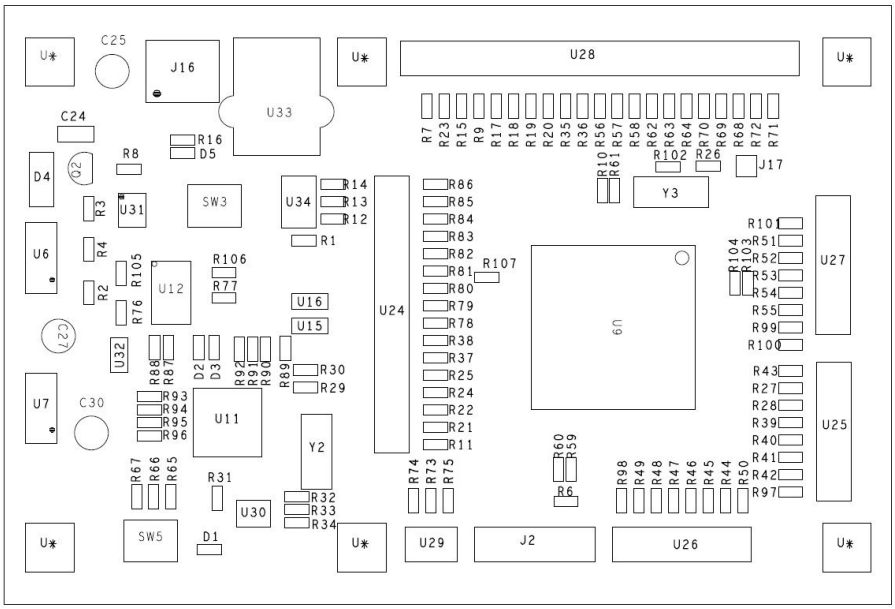
Pokračování tabulky na další stránce...

Tabulka 17.1 – Pokračování

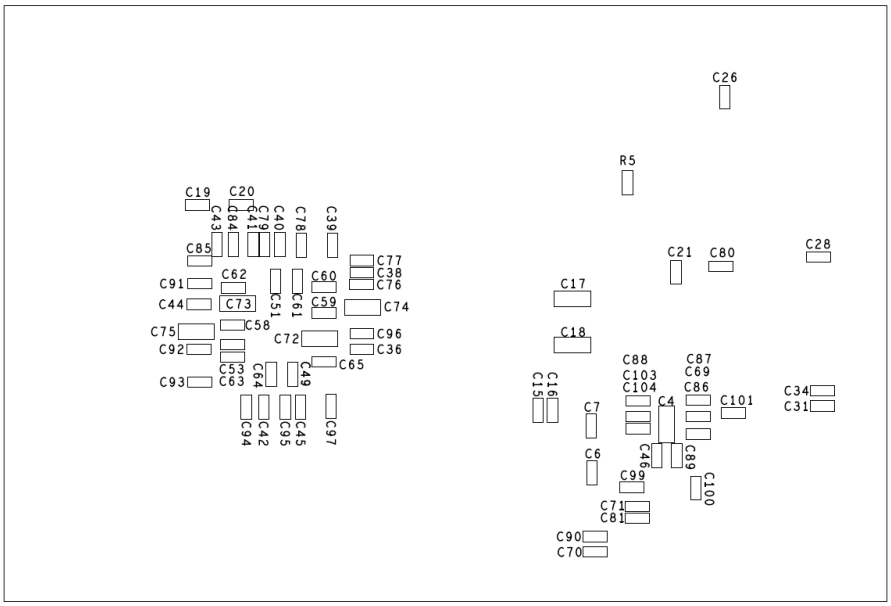
Pořadí	**Počet**	**Reference**	**Hodnota**
40	1	U30	93C46
41	1	U31	ADM708
42	1	U32	S2G2
43	1	U33	USB1X90B PCB
44	1	Y2	QM 12.000MHZ
45	1	Y3	QM 16.000MHZ



Obrázek 17.1: Schéma vývojového kitu s BF533



(a)



(b)

Obrázek 17.2: Osazovací výkresy ke kitu s BF533

17.3 Vývojový kit s BF504F

Tabulka 17.2: Seznam součástek vývojového kitu s BF504F

Pořadí	**Počet**	**Reference**	**Hodnota**
1	14	C1,C2,C5,C13,C15,C18,C21, C23,C30,C31,C46,C50,C59, C60	10 uF
2	25	C3,C4,C8,C14,C16,C17,C19, C20,C22,C24,C25,C26,C32, C33,C34,C35,C36,C42,C43, C47,C48,C51,C52,C53,C54	100 nF
3	4	C6,C7,C9,C10	27 pF
4	1	C11	220 pF
5	1	C12	100 uF
6	13	C27,C28,C29,C37,C38,C39, C40,C41,C44,C45,C55,C56, C57	330 pF
7	1	D1	5988140107F
8	2	D2,D6	5988170107F
9	2	D3,D4	1N4007 SMD
10	1	D5	SK36A
11	1	D7	5988110107F
12	4	H1,H2,H3,H4	Mount Hole
13	4	J1,J2,J3,J4	S2G20
14	3	J5,J6,U8	S2G2
15	1	J7	S2G8
16	1	J8	S2G6
17	1	J9	S2G4
18	1	J10	ARK550/2EX
19	1	J11	S2G8
20	1	MCU1	BF504
21	1	Q1	2N3906
22	73	R1,R2,R3,R5,R6,R7,R8,R9,	33

Pokračování tabulky na další stránce...

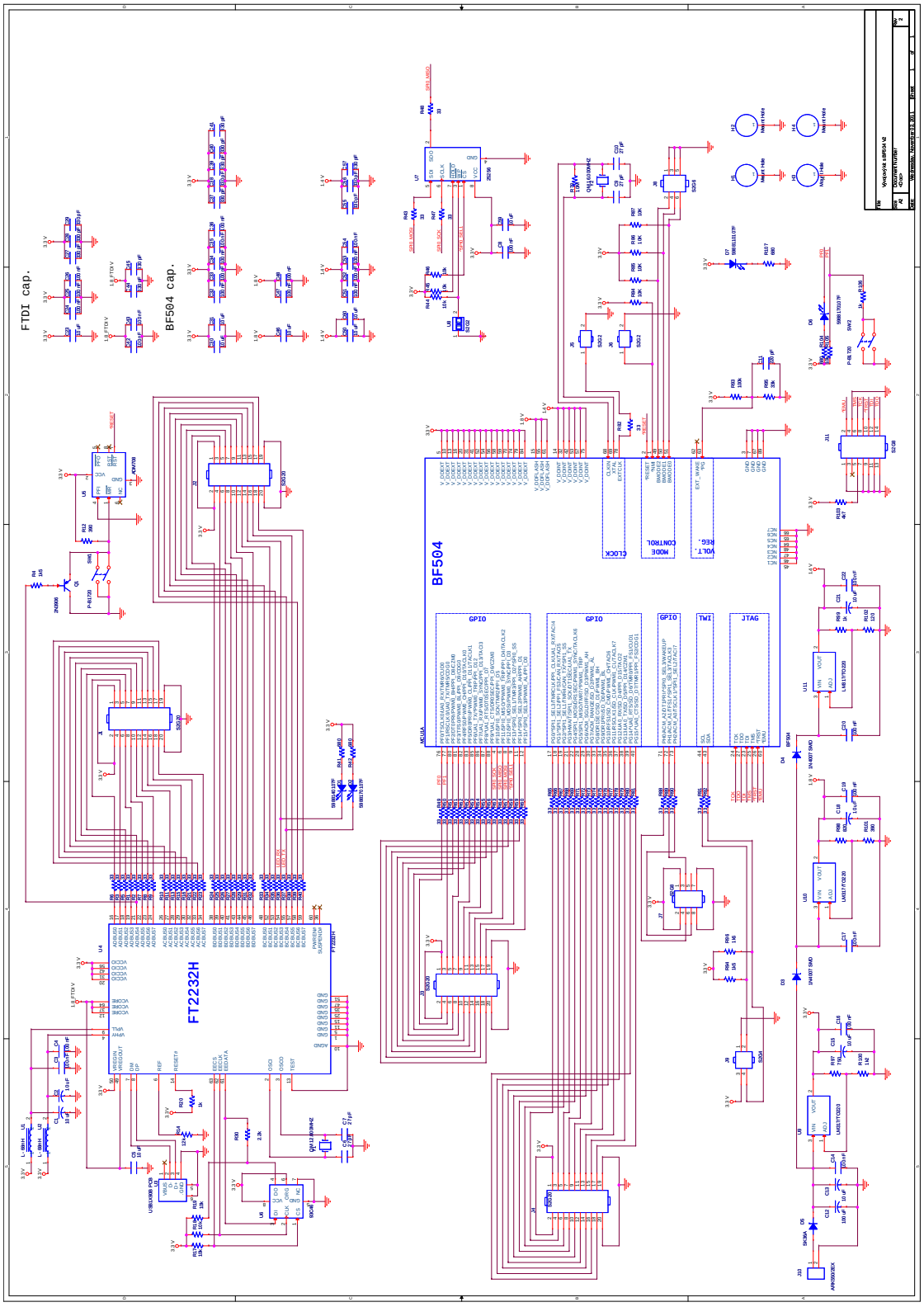
Tabulka 17.2 – Pokračování

Pořadí	**Počet**	**Reference**	**Hodnota**
		R10,R11,R13,R15,R16,R21,	
		R22,R23,R24,R25,R26,R27,	
		R28,R29,R31,R32,R33,R34,	
		R35,R36,R37,R38,R39,R40,	
		R43,R47,R48,R49,R50,R51,	
		R52,R53,R54,R55,R56,R57,	
		R58,R59,R60,R61,R62,R63,	
		R64,R65,R66,R67,R68,R69,	
		R71,R72,R73,R74,R75,R76,	
		R77,R78,R79,R80,R81,R82,	
		R88,R89,R90,R91,R92	
23	3	R4,R94,R96	1k5
24	2	R12,R101	390
25	1	R14	12k
26	11	R17,R18,R19,R44,R45,R46,	10k
		R84,R85,R86,R87,R105	
27	3	R20,R99,R106	1k
28	1	R30	2.2k
29	4	R41,R42,R104,R107	680
30	1	R70	10M
31	1	R93	100k
32	1	R95	33k
33	1	R97	750
34	1	R98	820
35	1	R100	1k2
36	1	R102	120
37	1	R103	4k7
38	2	SW1,SW2	P-B1720
39	2	U1,U2	L - 68nH
40	1	U3	USB1X90B PCB
41	1	U4	FT2232H

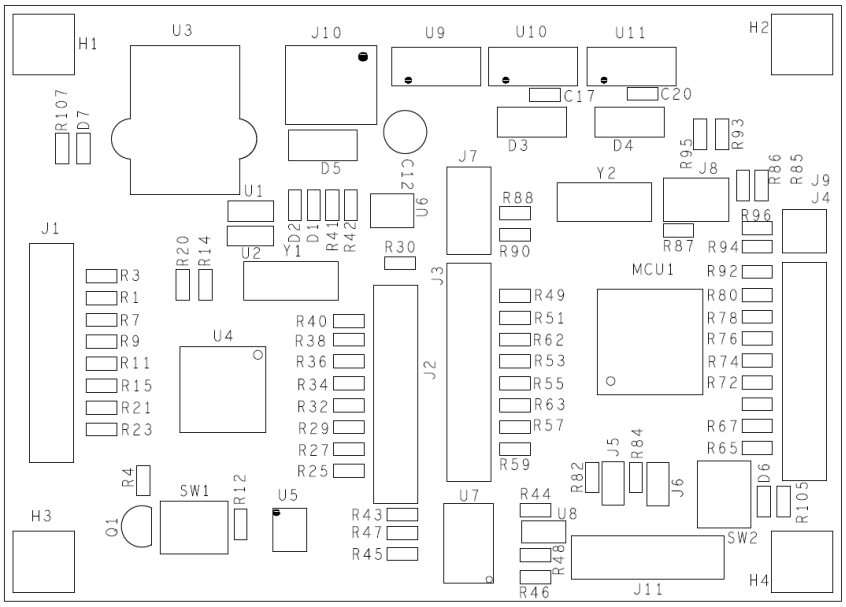
Pokračování tabulky na další stránce...

Tabulka 17.2 – Pokračování

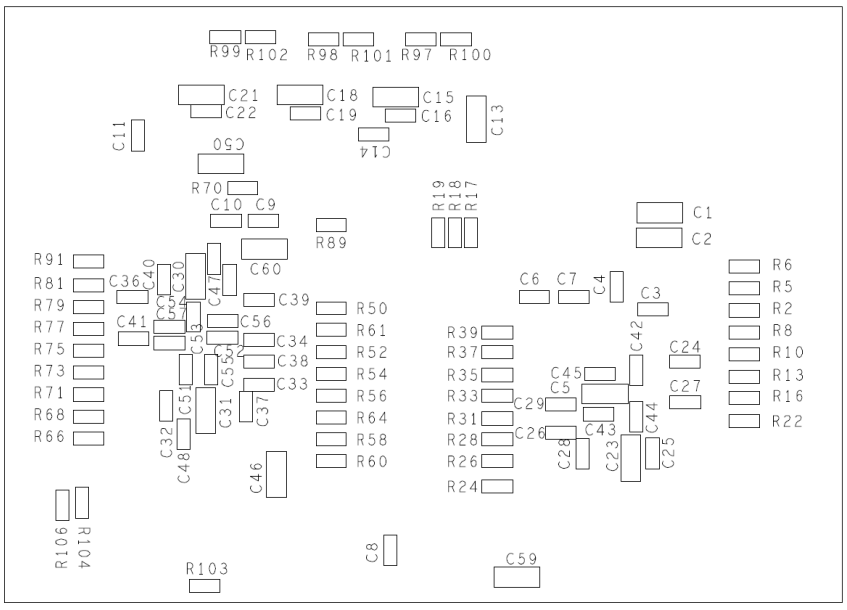
Pořadí	**Počet**	**Reference**	**Hodnota**
42	1	U5	ADM708
43	1	U6	93C46
44	1	U7	25256
45	3	U9,U10,U11	LM317/TO220
46	1	Y1	QM 12.000MHZ
47	1	Y2	QM 16.000MHZ



Obrázek 17.3: Schéma vývojového kitu s BF504F



(a)



(b)

Obrázek 17.4: Osazovací výkresy ke kitu s BF504F

17.4 SMART kamera

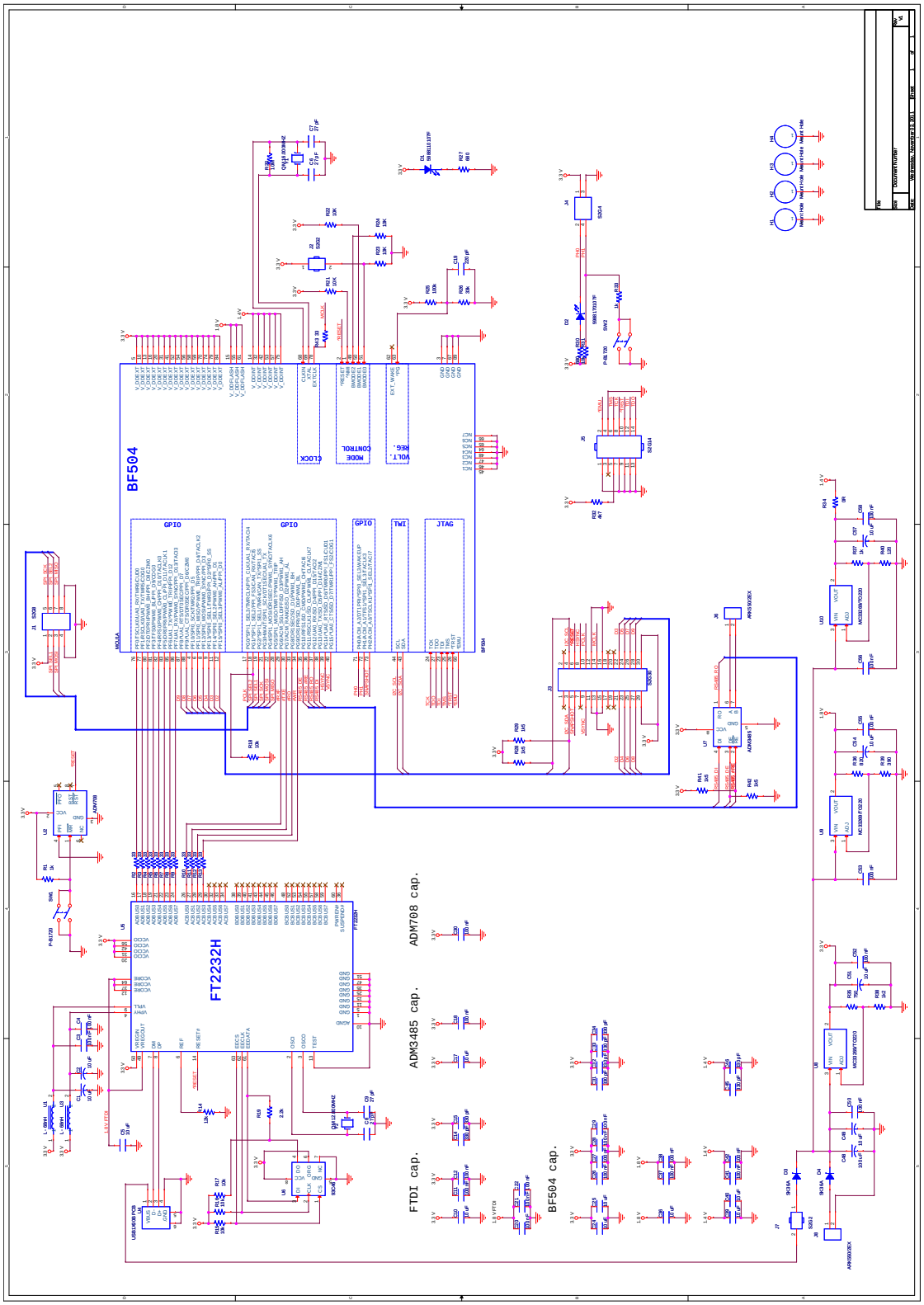
Tabulka 17.3: Seznam součástek SMART kamery

Pořadí	**Počet**	**Reference**	**Hodnota**
1	14	C1,C2,C5,C10,C17,C24,C25, C36,C39,C40,C49,C51,C54, C57	10 uF
2	23	C3,C4,C11,C12,C18,C20, C21,C22,C26,C27,C28,C29, C30,C37,C38,C41,C42,C50, C52,C53,C55,C56,C58	100 nF
3	4	C6,C7,C8,C9	27 pF
4	8	C14,C15,C31,C32,C33,C34, C45,C46	330 pF
5	1	C19	220 pF
6	1	C48	100 uF
7	1	D1	5988110107F
8	1	D2	5988170107F
9	2	D3,D4	SK36A
10	4	H1,H2,H3,H4	Mount Hole
11	1	J1	S2G8
12	2	J2,J7	S2G2
13	1	J3	S2G30
14	1	J4	S2G4
15	1	J5	S2G14
16	2	J6,J8	ARK550/2EX
17	1	MCU1	BF504
18	3	R1,R33,R37	1k
19	13	R2,R3,R4,R5,R6,R7,R8,R9, R10,R11,R12,R13,R43	33
20	1	R14	12k
21	9	R15,R16,R17,R18,R21,R22, R23,R24,R31	10K

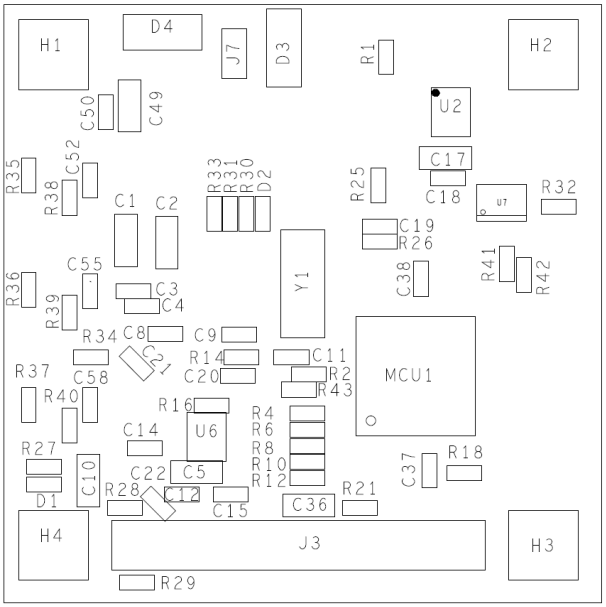
Pokračování tabulky na další stránce...

Tabulka 17.3 – Pokračování

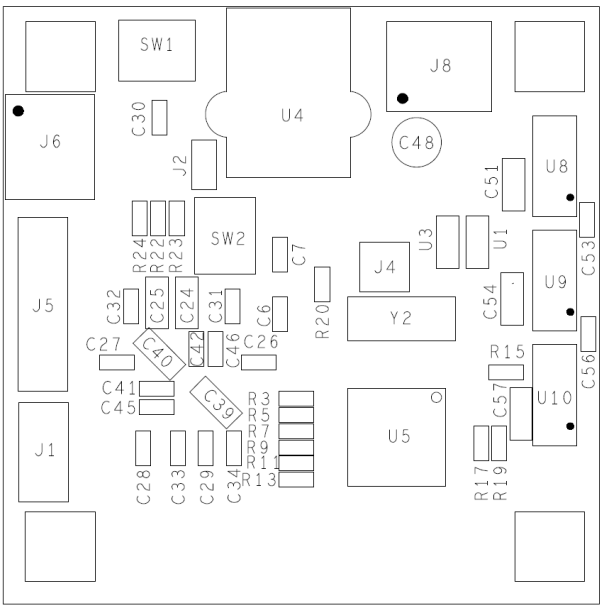
Pořadí	**Počet**	**Reference**	**Hodnota**
22	1	R19	2.2k
23	1	R20	10M
24	1	R25	100k
25	1	R26	33k
26	2	R27,R30	680
27	4	R28,R29,R41,R42	1k5
28	1	R32	4k7
29	1	R34	0R
30	1	R35	750
31	1	R36	820
32	1	R38	1k2
33	1	R39	390
34	1	R40	120
35	2	SW1,SW2	P-B1720
36	2	U1,U3	L - 68nH
37	1	U2	ADM708
38	1	U4	USB1X90B PCB
39	1	U5	FT2232H
40	1	U6	93C46
41	1	U7	ADM3485
42	3	U8,U9,U10	MC33269/TO220
43	1	Y1	QM 16.000MHZ
44	1	Y2	QM 12.000MHZ



Obrázek 17.5: Schéma SMART kamery



(a)



(b)

Obrázek 17.6: Osazovací výkresy modulu SMART kamery

17.5 Obsah přiloženého CD

- Tato diplomová práce v PDF formátu
- Technologická data pro výrobu PCB vývojových kitů a SMART kamery
- Testovací programy pro ověření funkce vývojových kitů
  - BF53x_KIT_LED_BLIK (*.ldr soubor a zdrojový kód)
  - BF504_KIT_LED_BLIK (*.ldr soubor a zdrojový kód)
- Firmware a zdrojové kódy SMART kamery
- Driver SMART kamery ve formě DLL, včetně zdrojových kódů (v C++)
- Pomocná DLL knihovna pro import driveru do prostředí LABVIEW, včetně zdrojového kódu (v C#)
- Ukázka implementace SMART kamery v prostředí LABVIEW
- Použité datasheety použitých součástek a různé pomocné materiály