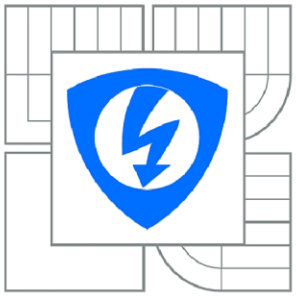
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF RADIO ELECTRONICS

OPTICKÉ SNÍMÁNÍ A ANALÝZA BYTOVÝCH MĚŘIDEL

OPTICAL RECOGNITION AND ANALYSIS FOR HOME METERING

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

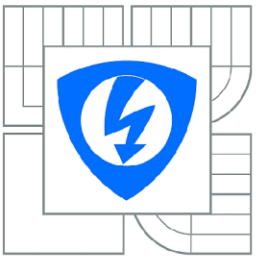
AUTOR PRÁCE — Bc. PETR MACHALA

AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ POVALAČ, Ph.D.

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student:	Bc. Petr Machala	***ID:***	134548
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Optické snímání a analýza bytových měřidel

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s běžnými bytovými měřidly (vodoměr, elektroměr, plynoměr) a zhodnoťte možnost optického snímání počitadla stavu. Navrhněte základní koncepci využívající kameru a mikrokontrolér s jádrem ARM Cortex-M a periferií DCMI. Prostudujte princip optického rozeznávání znaků OCR, především číslic. Na vývojové desce demonstrujte propojení kamery s mikrokontrolérem.

Navrhněte a realizujte prototyp snímacího systému včetně základního plošného spoje a napájení, využijte modul kamery a vývojové desky. Vytvořte firmware pro mikrokontrolér, který bude zpracovávat obraz z kamery, vyhodnocovat počitadla stavu, údaje zaznamenávat a vizualizovat na displeji. Firmware doplňte o ochranné funkce při kontinuálním snímání měřidel - detekce úniku plynu, vytopení vodou apod.

DOPORUČENÁ LITERATURA:

[1] STMicroelectronics [online]. STM32F427xx, STM32F429xx, ARM Cortex-M4 32b MCU+FPU. Datasheet - production data [cit. 7.5.2014]. Dostupné na www: http://goo.gl/7Zb9Xu.

[2] HLAVÁČ, V., ŠONKA, M. Počítačové vidění. Grada, Praha, 1993.

Termín zadání:	9.2.2015	***Termín odevzdání:***	21.5.2015

Vedoucí práce: Ing. Aleš Povalač, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Tomáš Kratochvíl, Ph.D.
Předseda oborové rady

ABSTRAKT

Tématem této diplomové práce je studium a řešení problematiky týkající se optického snímání a analýzy bytových měřidel pro následný záznam a statistické zpracování. Pro tuto požadovanou aplikaci byla využita vývojová deska 32F429IDISCOVERY výrobce *STMicroelectronics* s mikrokontrolérem ARM Cortex-M4 a modul kamery OV7670 výrobce *OmniVision*, kde byl postupně implementován požadovaný firmware. Diplomová práce tedy obecně představuje komplexní řešení od získávání snímků použitým modulem kamery až po výslednou analýzu a zpracování naměřených dat a sestrojení funkčního prototypu navrženého snímacího systému.

KLÍČOVÁ SLOVA

ARM, OCR, 32F429IDISCOVERY, OV7670, DCMI, SCCB

ABSTRACT

The topic of this master's thesis is a study and solution of problems regarding an optical recognition and home metering analysis for the following recording and statistical processing. For this required application a *STMicroelectronics* 32F429IDISCOVERY development board with ARM Cortex-M4 microcontroller and *OmniVision* OV7670 camera module was used, where a required firmware was implemented. This thesis therefore generally represents a complex solution from acquisition of images by used camera module to final analysis and data processing and building a functional prototype of the proposed sensor system.

KEYWORDS

ARM, OCR, 32F429IDISCOVERY, OV7670, DCMI, SCCB

MACHALA, P. *Optické snímání a analýza bytových měřidel*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2015. 56 s., 5 s. příloh. Diplomová práce. Vedoucí práce: Ing. Aleš Povalač, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Optické snímání a analýza bytových měřidel jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Aleši Povalačovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1 ÚVOD

V dnešní moderní automatizované době lze pozorovat rychlý nárůst požadavků kladených na zpracování, analýzu a statistiku různých informačních dat a veličin, které lze ve většině domácností nalézt. Tato skutečnost je dána především velmi rychlým technologickým pokrokem a rozvojem tzv. inteligentních domů.

Jednou z typických veličin, které lze v domácnosti zaznamenávat a zpracovávat je údaj spotřeby elektrické energie, plynu nebo vody. Právě tato informace o dosažené spotřebě je pro majitele většinou jednou z nejdůležitějších, protože je přímo svázána s výslednými náklady na daný objekt a v mnoha případech může i poukázat na možnou závadu v objektu jako je např. nebezpečný únik plynu nebo vody.

Samotná informace o celkové spotřebě domácnosti je zaznamenávána hlavním bytovým měřidlem, jako je např. elektroměr, plynoměr nebo vodoměr a je zobrazena na číselníku měřidla. Odtud lze tuto informaci jednoduše snímat a dále zpracovávat.

Právě optické snímání číselníků a následné zpracování získaných dat je stěžejní částí tohoto dokumentu. Na základě těchto požadavků bude navržen prototyp snímacího systému *OptRec*, který bude přehledně a postupně popsán v dalších kapitolách.

1.1 Druhy bytových měřidel

Než přejdeme k samotnému obecnému popisu funkce prototypu, tak je dobré věnovat určitou pozornost jednotlivých druhům bytových měřidel a jejich dělení.

Na dnešním trhu lze nalézt nepřeberné množství různých bytových měřidel, nicméně obecně lze říct, že všechny tyto druhy mají jeden společný rys a to přítomnost číselníku zobrazující celkovou spotřebu měřidla. Některá modernější měřidla jsou již vybavena i digitálním číselníkem a impulzními výstupy, odkud lze spotřebu velmi jednoduše snímat (např. pomocí převodní konstanty 1000 impulzů/kWh), ovšem tyto měřidla jsou výrazně dražší a stále nejsou součástí mnoha domácností. Obecně lze tedy říct, že v rámci této aplikace se zaměřujeme pouze na starší analogová měřidla.

Pokud tedy pomineme samotou funkci měřidla a zaměříme se pouze na interpretaci údaje o dosažené spotřebě, tak můžeme měřidla dělit např. takto:

- Podle počtu číselníků
  - Jedno-číselníkové (typicky standardní vodoměr)
  - Více-číselníkové (např. dvousazbový elektroměr)
- Podle typu číselníku
  - Digitální
  - Analogový
- Podle počtu zobrazovaných číslic spotřeby atd.

Některé z výše zmíněných vlastností bytových měřidel lze vidět na Obr. 1.1, Obr. 1.2 a Obr. 1.3 níže. Zmíněná rozmanitost v oblasti bytových měřidel představuje nutnost spolupráce uživatele se snímacím systémem pro zajištění korektní funkce snímání. Tento fakt je samozřejmě nutné vzít v potaz při návrhu výsledného ovládacího firmwaru (FW) prototypu. V principu tedy hovoříme o určitém uživatelském nastavení systému pro potřeby konkrétního měřidla.

Obr. 1.1: Zástupce membránového plynoměru pro domácnosti [1]

Obr. 1.2: Zástupce dvousazbového třífázového elektroměru pro domácnosti [2]

Obr. 1.3: Zástupce lopatkového vodoměru pro domácnosti [3]

1.2 Optické rozeznávání znaků

Dále je vhodné i zběžně pohovořit o problematice optického rozeznávání znaků (OCR), které je klíčovou součástí výsledného návrhu prototypu snímacího systému.

Samotné OCR je široce rozšířenou a známou metodou, která umožňuje digitalizaci obrazových textů, s nimiž lze následně pracovat standardně jako s normálním počítačovým textem (např. v ASCII kódu). OCR je samo o sobě velmi rozsáhlou a komplexní problematikou zahrnující odvětví jako jsou rozeznávání vzorců, umělá inteligence nebo počítačové vidění. Proto se této problematice věnují přední počítačové společnosti jako *Adobe*, *Microsoft*, *ABBYY* aj. Metoda je široce využívána např. při digitalizaci starých knih v knihovnách, zálohování důležitých firemních dokumentů nebo při rozpoznávání registračních značek automobilů jak lze vidět na Obr. 1.4 níže.

Obr. 1.4: Příklad aplikace OCR algoritmu v praxi [4]

OCR se obecně skládá, ze třech funkčních kroků. Prvním krokem je fáze pre-processingu, která připravuje získaný obrazový text pro požadované zpracování, následně dochází k aplikaci samotného OCR algoritmu a nakonec provedení určitého post-processingu. Samotné OCR je obecně možné dále dělit do dvou hlavních kategorií a to standardní OCR, kdy jsou rozeznávány jednotlivé znaky podle předlohy nebo tzv. optické rozeznávání slov (OWR), kdy jsou rozeznávány jednotlivé znaky a poté ve zmíněné fázi post-processingu jsou pomocí oddělovačů poskládána jednotlivá slova. Tyto slova jsou samozřejmě následně testována podle slovních předloh, což logicky znamená nutnost dělení OWR pro jednotlivé jazyky a kultury.

V rámci post-processingu je nutné aplikovat i určitý algoritmus korekce, protože OCR algoritmus nemusí ihned rozpoznat všechny skenované znaky nebo slova. Pokud i v případě korekce není daný znak nebo dané slovo rozpoznáno tak je nutné korekci provést ručně uživatelem. Tento přístup lze vidět např. v případě široce rozšířeného webového testu *reCAPTCHA* pro odlišení skutečných uživatelů od robotů. Zde je vždy uvedeno jedno známé kontrolní slovo a jedno neznámé slovo, které uživatel svým správným přepisem vkládá do databáze společně s dalšími uživateli, kde tímto způsobem společně pomáhají při digitalizaci historických archívů.

V našem případě bude ovšem problematika samotné aplikace OCR metody značně zjednodušena, protože v principu vyžadujeme rozpoznávání pouze jednotlivých, v mnoha případech jasně definovaných, znaků a to konkrétně pouze číslic. Zjednodušeně tedy budeme zpracovávat jednotlivé pixely získaného obrazu číselníku a poté rozpoznané tvary porovnávat se známými tvary v databázi, tedy čísly 0-9. Tímto získáme požadovaný stav číselníku připravený pro další zpracování.

Vzhledem ke složitosti problematiky týkající se OCR bude využit již existující procesní algoritmus upraven podle potřeb navrhovaného systému. Problematika nalezení vhodného algoritmu a jeho úprava bude popsána v dalších kapitolách.

1.3 Funkce prototypu

Výsledná funkce navrženého prototypu snímacího systému bude principiálně velmi jednoduchá. Prototyp bude obecně navržen pro práci ve dvou funkčních režimech, nicméně samotné snímání vždy bude probíhat podle následujícího postupu:

1) Získání snímku stavu spotřeby měřidla kamerou
2) Aplikace OCR algoritmu
3) Zpracování dat spotřeby

První funkční režim lze označit jako manuální, který bude sloužit pro okamžité měření na základě požadavku uživatele. Režim je tedy určen pro krátkodobé měření, kdy uživatel drží prototyp v ruce a manuálně snímá údaj o dosažené spotřebě.

Druhý funkční režim bude navržen jako plně automatický, kde bude podle daného nastavení uživatelem cyklicky snímán stav bytového měřidla a získaná data budou následně zpracována a vyhodnocena. Tento funkční režim je tedy určen na dlouhodobé automatické měření, kde se prototyp upevní před bytové měřidlo a postupně pravidelně snímá informace o dosažené spotřebě.

1.4 Zhodnocení funkce prototypu

Na závěr úvodní kapitoly je dobré se pozastavit nad samotnou realizovatelností prototypu a zhodnotit možnosti optického rozeznávání.

Z předchozích kapitol je viditelné, že výsledná aplikace prototypu musí nutně obsahovat rozsáhlejší možnosti nastavení FW uživatelem, což je dáno skutečností, že bytová měřidla nepodléhají žádnému standardu, co se týče optické interpretace údajů měření spotřeby. Jako protiklad můžeme zmínit např. QR kódy, které mají přesně danou strukturu pro zajištění kompatibility mezi jednotlivými zařízeními. Tato fakt představuje tedy určitý problém s kompatibilitou mezi jednotlivými měřidly, kterou bude muset zajistit sám uživatel správným nastavením prototypu.

Obecně lze ovšem říct, že za předpokladu dostatečně kvalitního snímku z kamery je možné následně pomocí aplikace OCR algoritmu získat požadovaná data. Lze ovšem již předem usoudit, že problematickými částmi bude např. ošetření stavu mezi překlápějícími se číslicemi (viz Obr. 1.2) nebo samotný korektní chod OCR algoritmu. Tyto algoritmy jsou totiž většinou optimalizovány pro funkce zmíněné v předchozí kapitole, tedy případ malého textu černé barvy na bílém podkladu apod.

2 HARDWAROVÁ KONCEPCE

Prvním krokem při návrhu libovolného elektronického zařízení je jednoznačně problematika zvolené hardwarové (HW) koncepce, množství použitých periférií a mechanická konstrukce prototypu. Tuto záležitost značně zjednodušuje použitá vývojová deska, která je již vybavena velkým množstvím interních periférií, což omezuje nutnost použití dalších přídavných periférií na minimum.

Hlavními komponentami tohoto prototypu jsou použitý modul kamery OV7670 a vývojová deska 32F429IDISCOVERY. Tyto a další části prototypu jsou usazeny na základové desce plošného spoje (DPS), která představuje jednoduchý spojující element mezi výše zmíněnými komponentami a současně zajišťuje jejich napájení, konektivitu s paměťovým médiem, LED osvětlení atd.

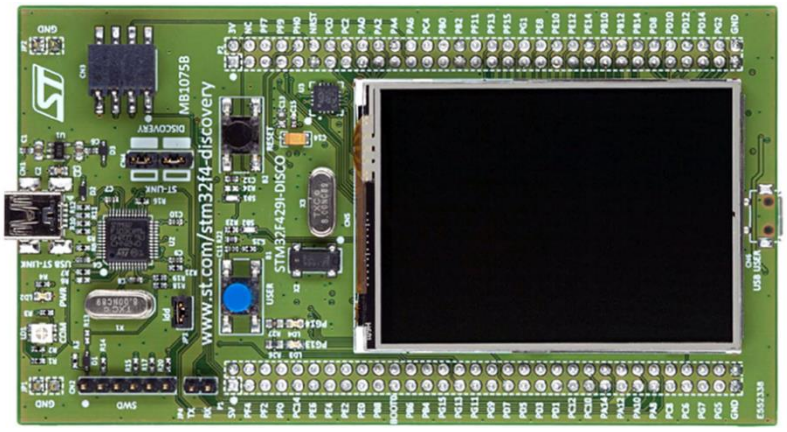
Zjednodušené blokové schéma HW koncepce prototypu snímacího systému lze přehledně vidět na Obr. 2.1 níže.

Obr. 2.1: Blokové schéma HW koncepce prototypu *OptRec*

Konkrétní popis nejdůležitějších používaných částí snímacího systému je uveden v následujících podkapitolách níže, nicméně obecně lze říci, že výsledný prototyp, splňující funkci popsanou v kapitole 1.2, je navrhován tak, aby byl malou, jednoduchou a soběstačnou jednotkou s co nejmenší spotřebou energie.

2.1 Vývojová deska 32F429IDISCOVERY

Základem výše nastíněného prototypu snímacího systému je vývojová deska 32F429IDISCOVERY výrobce *STMicroelectronics*, kterou lze vidět na Obr. 2.2 níže.



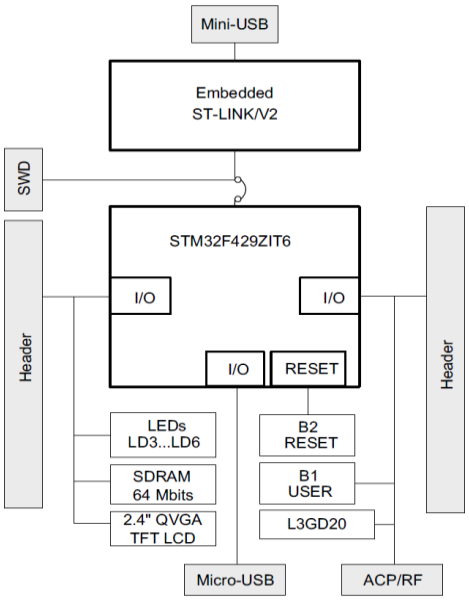
Obr. 2.2: Fotografie vývojové desky 32F429IDISCOVERY [5]

Tato použitá vývojová deska je osazena 32 bitovým mikrokontrolérem (MCU) architektury ARM Cortex-M4 s označením STM32F429ZIT6. Tento MCU je součástí tzv. STM32 F4 série, která disponuje, oproti sérii předchozí, značným množstvím nových periférií, desítkami nových procesorových instrukcí atd. Obecně to tedy znamená zvýšení výkonu, efektivnější využití procesorového času a snížení spotřeby. MCU dále obsahuje i jednotku pro práci s plovoucí desetinnou čárkou (FPU), což výrazně urychluje výpočty v tomto datovém typu. Určitým speciálním rysem této architektury je skutečnost, že zde nenajdeme rozsáhlé množství oddělených vyrovnávacích pamětí a bufferů, jak je u valné většiny MCU zvykem. Tento fakt je dán skutečností, že architektura Cortex-M4 již a priori přepokládá využití řadiče přímého přístupu do paměti (DMA), který je v případě použitého MCU dvojitý.

Z hlediska vybavení vývojové desky zde můžeme nalézt periferie, jako např. tříosý MEMS pohybový senzor, QVGA RGB TFT LCD displej, SDRAM paměť nebo standardní ovládací mikrotlačítka. Tato skutečnost činí tuto vývojovou desku jako velmi univerzální, což představuje velkou výhodu v praxi. Zbývající HW periferie, popř. další informace týkající se použité vývojové desky je možné nalézt standardně na webových stránkách podpory produktu zde [6], popř. přímo v datovém listu MCU zde [7].

Důležitou částí vývojové desky je HW konektivita s okolím a dalšími externími periférie mi. Vývojová deska standardně nabízí velké množství vstupních/výstupních (I/O) pinů a současně dva USB konektory na DPS, jak lze vidět na Obr. 2.2 výše. První USB konektor typu Mini-B slouží pro připojení vývojové desky k PC a zajišťuje propojení s tzv. ST-LINK/V2 převodníkem představující serial wire debug (SWD) standard pro nahrávání a debuggování programového kódu. Tento SWD standard v případě architektury ARM ekonomicky nahrazuje použití standardu joint test action group (JTAG). Druhý USB konektor typu Micro-B je standardně jako uživatelský, což znamená, že může sloužit např. jako vstup pro USB mass storage (UMS) nebo jako virtual COM port (VCP).

Samotnou vývojovou desku lze ilustrovat jako jednoduché HW blokové schéma, které je možné vidět na Obr. 2.3 níže.

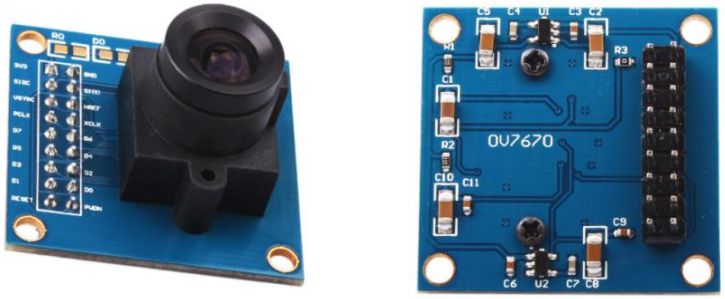


Obr. 2.3: Blokové schéma vývojové desky 32F429IDISCOVERY [5]

Zde můžeme přehledně vidět ilustrační zapojení vybraných částí vývojové desky, jako jsou např. mikrotlačítka, SDRAM nebo LCD displej, nicméně pro korektní identifikaci zapojení příslušných periférií je nutné se opět obrátit na datový list MCU zde [7] nebo přímo na schéma zapojení produktu vývojové desky dostupné na webových stránkách podpory produktu zde [6], popř zde [5].

2.2 Kamera OV7670

Druhou klíčovou komponentou navrhovaného prototypu snímacího systému je kamera OV7670 výrobce *OmniVision*, kterou lze vidět na Obr. 2.4 níže.



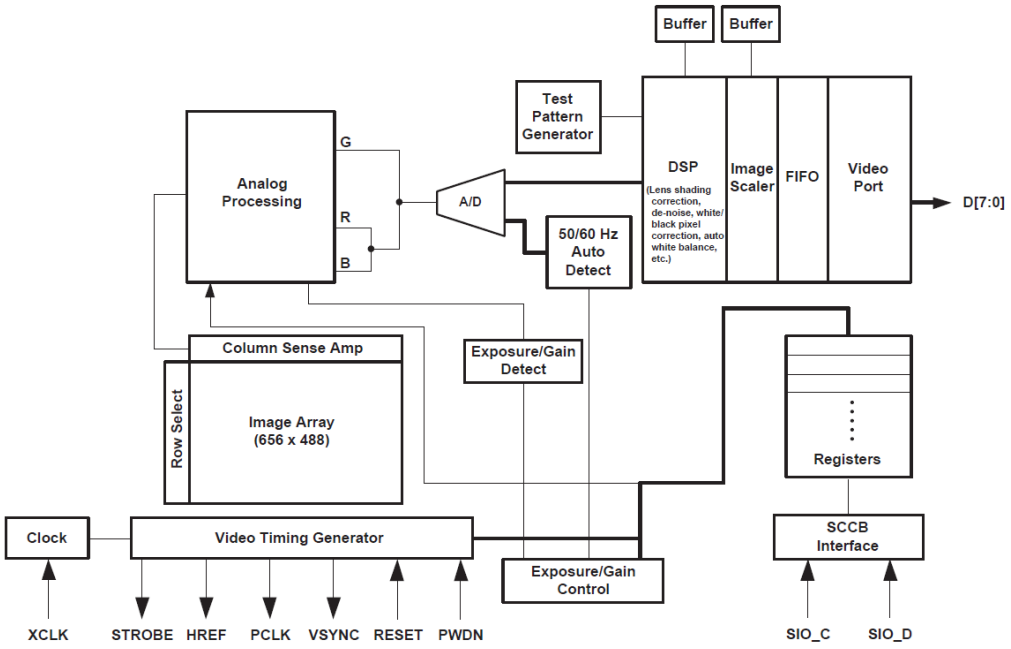
Obr. 2.4: Fotografie kamery OV7670 [9]

Tato kamera standardně disponuje kvalitním CMOS senzorem schopným zaznamenávat rozlišení až 640x480 px (VGA neboli 0.3 Mpx) při obnovovací frekvenci až 30 fps společně s dalšími periferiemi zajišťující obrazové předzpracování a externí konektivitu. Kamera obecně vždy zaznamenává obraz ve výše zmíněném rozlišení a obnovovací frekvenci, který je poté na základě nastavení kamery předzpracováván digitálním signálovým procesorem (DSP) a poté teprve odesílán pomocí digital camera interface (DCMI) sběrnice do koncového zařízení. Kamera tedy na základě nastavení podporuje rozlišení jako zmíněné VGA nebo CIF a jejich alternativy, datové formáty jako RGB565/555 nebo YUV/YCbCr(4:2:2) atd.

Lze říct, že se jedná o velmi malou, kompaktní a relativně levnou alternativu kamery, která i vzhledem ke svému stáří je stále velmi oblíbenou komponentou v různých embedded aplikacích. Tato oblíbenost mezi vývojáři je dána právě především možností uživatelské konfigurace/nastavení kontroly kvality výstupního obrazu, formátování výstupních dat, předzpracování obrazových dat atd. Nutné ovšem zmínit, že hlavní nevýhodou použité kamery je nutnost mechanického ostření, což v určitých aplikacích může být příliš velká překážka.

Programování kamery probíhá pomocí konfigurace rozsáhlého množství interních registrů kamery pomocí serial camera control bus (SCCB) sběrnice. Další informace o použité kameře a soupis všech možných programovatelných registrů lze standardně nalézt v datovém listu kamery zde [10] nebo v aplikačním listu zde [11].

Použitou kameru lze vhodně ilustrovat blokovým schématem, které je možné vidět na Obr. 2.5 níže. Zde můžeme vidět, že většina pinů je spjata s výše uvedenými komunikačními SCCB a DCMI sběrnicemi. Zbývající piny představují podpůrné funkce jako je reset všech ovládacích registrů kamery (RESET) nebo pro uspání modulu kamery (PWDN) a tím snížení její energetické spotřeby.



Obr. 2.5: Blokové schéma kamery OV7670 [10]

2.3 Použité komunikační sběrnice

V této podkapitole se budeme zabývat zjednodušeným popisem funkce dvou nejdůležitějších sběrnic, které lze v navrhovaném prototypu snímacího systému nalézt. Tyto sběrnice zajišťují získání korektních obrazových dat mezi použitým modulem kamery OV7670 a MCU na vývojové desce.

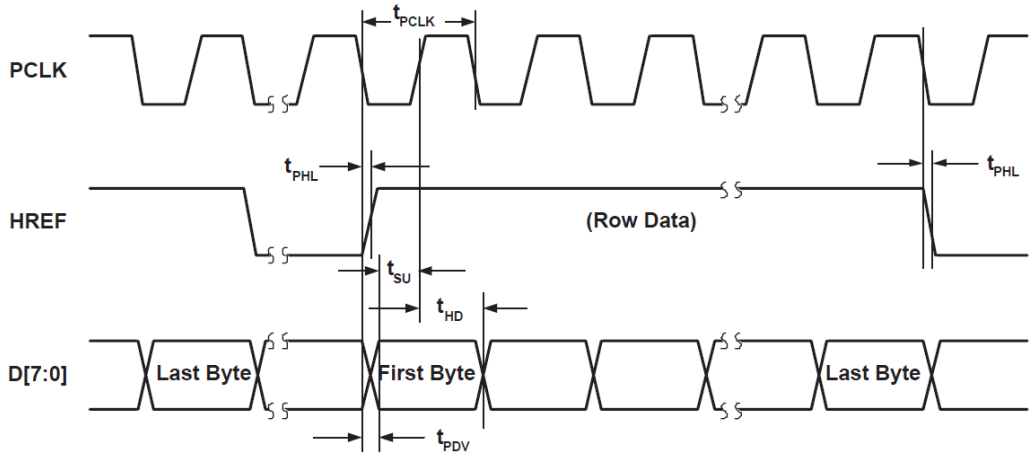
Tato podkapitola obecně popisuje pouze základní HW vlastnosti a informace týkající se dané sběrnice. Samotné nastavení a použití těchto sběrnic bude popsáno v dalších individuálních kapitolách.

2.3.1 DCMI sběrnice

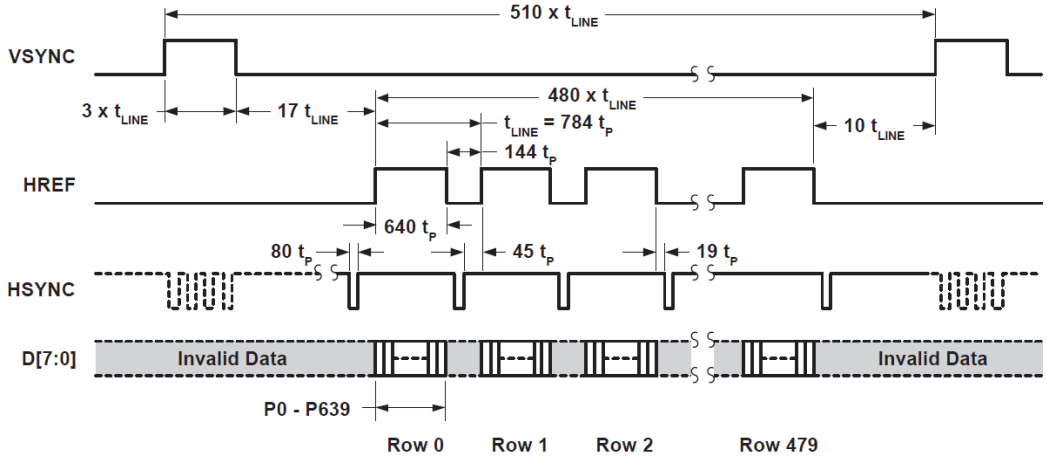
Sběrnice DCMI, jak již její název napovídá, slouží speciálně pro získávání digitálních obrazových dat ve formě snímků popř. přímo videa z kamerových modulů a CMOS senzorů pomocí paralelního rozhraní.

Vzhledem ke skutečnosti, že se jedná o paralelní rozhraní, tak již můžeme předvídat použití určité formy signálové synchronizace. Pokud se tedy na tuto problematiku podíváme z pohledu použité kamery OV7670, tak první klíčovou záležitostí je přivedení systémového hodinového signálu do kamery pomocí pinu XCLK. Bez tohoto hodinového signálu se bude modul kamery jevit jako nefunkční, proto je dobré se o přítomnosti tohoto signálu ujistit pomocí osciloskopu. Optimální hodnotou tohoto signálu je podle datového listu zde [10] hodnota 24 MHz.

Po přivedení hodinového signálu na pin XCLK začne použitá kamera ihned generovat synchronizační (signály PCLK, HREF a VSYNC na stejnojmenných pinech) a datové signály (signály D0 - D7 na stejnojmenných pinech) pro dané koncové zařízení jak lze vidět na Obr. 2.6 a Obr. 2.7 níže.



Obr. 2.6: Časový diagram horizontální synchronizace kamery OV7670 [10]



Obr. 2.7: Časový diagram synchronizace pro VGA snímek kamery OV7670 [10]

Z časových diagramů výše je možné logicky usoudit, že signál PCLK představuje tzv. pixelový hodinový signál, který signalizuje příjem jednoho bajtu obrazových dat. Zde je nutné dodat, že jeden bajt dat nemusí nutně odpovídat jednomu pixelu snímku. Tento fakt přímo souvisí s konfigurací používané kamery a to konkrétně na zvoleném obrazovém formátu. Např. výchozí nastavení kamery je nakonfigurováno pro použití obrazového formátu YCbCr (4:2:2), kde jeden pixel odpovídá dvou bajtům obrazových dat. Signál PCLK je taktéž logicky spjat s obnovovací frekvencí obrazu kamery, kde např. pokud je kmitočet signálu PCLK právě poloviční než signál XCLK, tak poté obnovovací frekvence obrazu kamery bude dosahovat právě 15 fps.

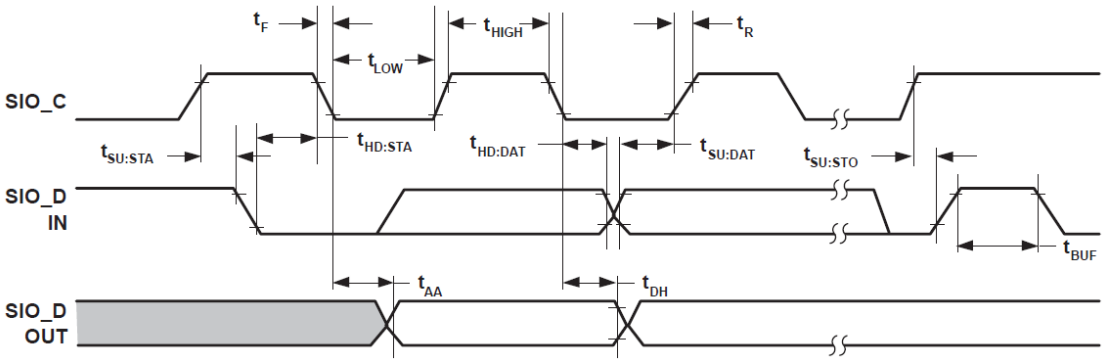
Dále je možné usoudit, že signál HREF představuje horizontální synchronizaci snímku, kde úroveň logické jedničky (H) tohoto signálu představuje právě jeden řádek snímaného obrazu a signál VSYNC představuje vertikální synchronizaci snímku, kde úroveň logické nuly (L) představuje příjem právě jednoho snímku. Toto chování lze velmi přehledně pozorovat na Obr. 2.7 pro příjem obrazu s VGA rozlišením. Zde vidíme, že během aktivní doby signálu HREF/HSYNC proběhne odeslání celkem 640 px ekvivalentní jednomu řádku snímku a během aktivní doby signálu VSYNC proběhne odeslání všech řádků jednoho snímku, tedy celkem 480 řádků.

2.3.2 SCCB sběrnice

Další klíčovou sběrnicí pro zajištění korektní funkce modulu kamery je výše zmíněná sběrnice SCCB. Jedná se o speciální sběrnici používanou právě firmou *OmniVision* pro konfiguraci interních registrů použité kamery.

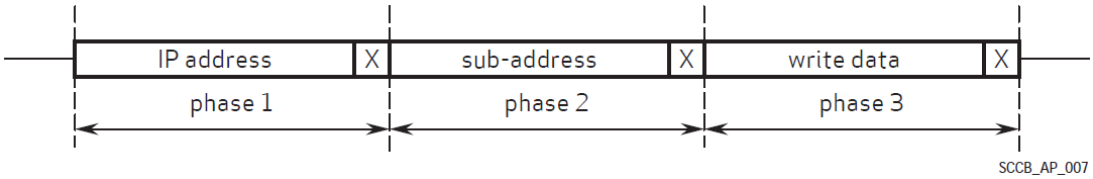
Vzhledem k tomu, že se jedná o standard používaný právě výrobcem kamery, tak můžeme předpokládat, že tato sběrnice obecně není použitým MCU podporována. Nicméně pokud se blíže podíváme do specifikací sběrnice SCCB, kterou udává výrobce zde [12], tak zjistíme, že tato sběrnice se velmi podobá standardně používané inter-integrated circuit (I2C) sběrnici. Obecně tedy můžeme říct, že sběrnici SCCB bude možné emulovat právě standardem I2C.

Z hlediska funkčnosti SCCB sběrnice budeme používat tzv. dvouvodičovou verzi, která zajišťuje komunikaci pouze mezi dvěma koncovými zařízeními (master a slave). V našem případě tedy konkrétně mezi zmíněnou kamerou a MCU. Typický příklad komunikace po této sběrnici lze vidět na Obr. 2.8 níže.



Obr. 2.8: Příklad časového diagramu sběrnice SCCB [10]

Zde můžeme vidět příklad standardní komunikace SCCB sběrnice, kde SIO_C představuje klasický hodinový signál sběrnice a SIO_D představuje signál datový (pro zápis nebo čtení), který se skládá ze START bitu, přenášených dat a STOP bitu. Obecně lze ovšem říci, že pokud chceme konfigurovat kameru pomocí SCCB sběrnice, tak musíme mimo standardů I2C dodržet tři (pro čtení pouze první dvě) komunikační fáze, které lze vidět na Obr. 2.9 níže.



Obr. 2.9: Třífázový cyklus zápisu registrů pomocí SCCB [12]

Tyto tři komunikační fáze představují odeslání ID adresy pro identifikaci zápisu (0x42) nebo čtení (0x43), samotnou adresu požadovaného registru, který chceme modifikovat a hodnotu registru (pouze pro zápis). Pokud je tento komunikační cyklus korektní a současně budou dodrženy všechny ochranné intervaly, tak dojde k úspěšné úpravě požadovaného registru. Pro konkrétnější informace týkající se komunikace pomocí SCCB sběrnice je vhodné se opět obrátit na specifikace zde [12].

2.4 Základová DPS

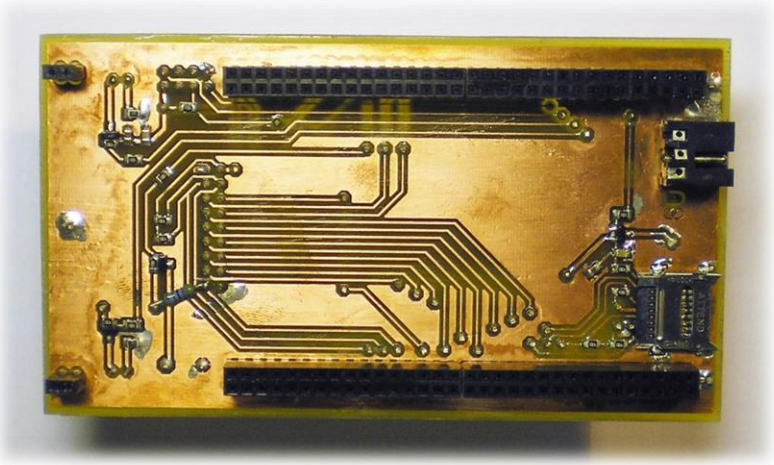
V této chvíli je zaměříme pozornost na návrh základové DPS a tedy částečně i na mechanickou konstrukci výsledného prototypu. Jak již bylo zmíněno v úvodu této kapitoly, základová DPS představuje spojovací element mezi použitými periferiemi a současně zajišťuje napájení prototypu. Z mechanického hlediska je základní představou navrhnout a vytvořit DPS tak, aby mechanicky fixovala použité moduly jako je modul kamery, vývojové desky nebo např. baterie a současně z hlediska rozměrů nepřesáhla rozměry samotné vývojové desky.

Na základě těchto požadavků byla navržena dvouvrstvá DPS, která splňuje všechny výše uvedené požadavky, a to jak z hlediska elektronického zapojení, tak mechanického. Samotná DPS byla navrhována pomocí komerčního editoru plošných spojů *EAGLE v7.2.0* a vyrobena v amatérské laboratoři metodou fotocesty. Zde byla výsledná předloha z editoru plošných spojů nasvícena pomocí UV lampy (2x11 W), na fotocitlivý plošný spoj, který byl poté vyvolán a následně standardně vyleptán pomocí leptacího roztoku.

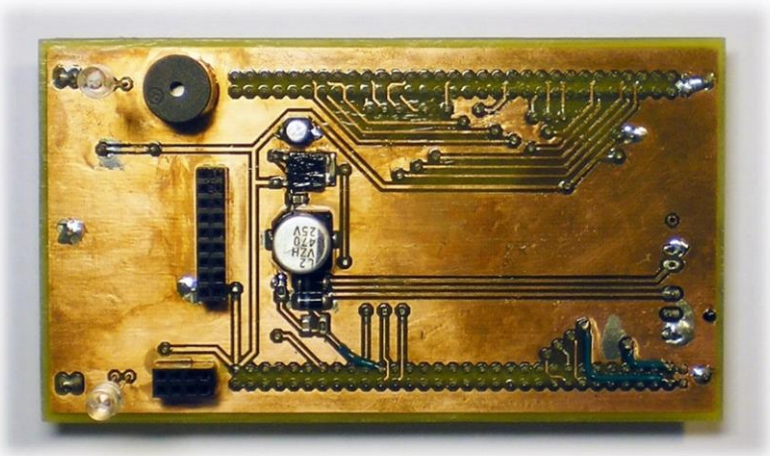
Výsledné zapojení základové DPS lze nalézt v příloze zde A.1 společně s návrhem předlohy obou vrstev DPS zde A.2 a seznamem použitých součástek zde A.3. Z uvedeného schématu můžeme vidět zapojení jednotlivých zmíněných komunikačních sběrnic, které jsou převážně vyvedeny na konektory DPS a používané kontrolní piny. Dále můžeme vidět zapojení akustického měniče pro výstražný systém, osvětlovací LED nebo paměťové médium ve formě standardní microSD karty. Důležitou částí základové DPS je její napájení, kde můžeme vidět, že DPS je možné napájet jak z pevného zdroje (4 až 16 V) jako je např. spínaný zdroj, tak pomocí baterií (3xAA), které jsou uchyceny na zadní straně základové DPS.

Nutno ovšem dodat, že pokud je vývojová deska napájena pomocí USB konektoru typu Mini-B (např. v případě programování), tak je nutné zajistit, aby nebyly přítomny baterie ani napájecí zdroj. Tato skutečnost je dána HW zapojením kombinace vývojové desky a základové DPS. Pro více informací je vhodné se obrátit na schéma zapojení.

Výslednou osazenou základovou DPS bez přídavných modulů je tedy možné vidět na fotografiích Obr. 2.10 a Obr. 2.11 níže.



Obr. 2.10: Fotografie horní strany základové DPS

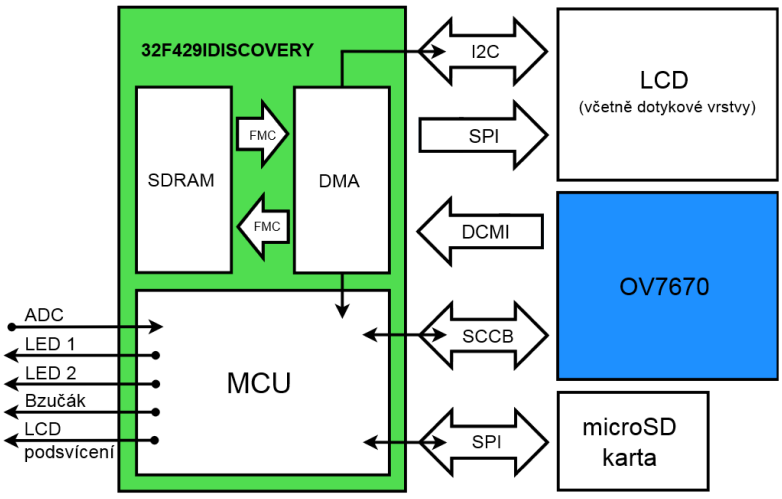


Obr. 2.11: Fotografie spodní strany základového DPS

K fotografiím uvedeným výše je vhodné zmínit, že takto vyrobená DPS musela být po čas vývoje programového kódu několikrát upravována, nicméně tyto HW úpravy nebyly dostatečně dramatické, aby vyžadovaly výrobu nové DPS. Výsledný návrh této DPS v příloze zde A je již samozřejmě opraven podle těchto provedených HW změn. Určitým přídavkem navržené DPS je konektor pro bezdrátový modul ESP8266, který obecně není součástí této diplomové práce, nicméně z praktického hlediska byla základová DPS již navržena s možností připojení tohoto bezdrátového modulu.

2.5 Výsledná konstrukce prototypu

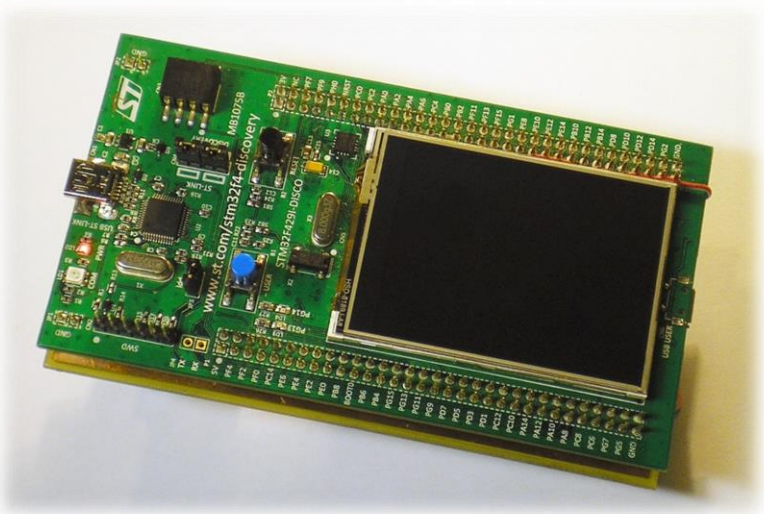
Na závěr této kapitoly tedy zbývá pouze pohovořit o výsledné konstrukci celého prototypu včetně výčtu použitých periférií. Po předchozím popisu je již možné získat určitou představu, jak bude výsledný prototyp vypadat. Samotný prototyp svou konstrukcí odpovídá obecné HW koncepci, jak lze vidět na Obr. 2.1, nicméně kompletnější popis prototypu snímacího systému včetně použitých periférií lze přehledně vidět na Obr. 2.12 níže.



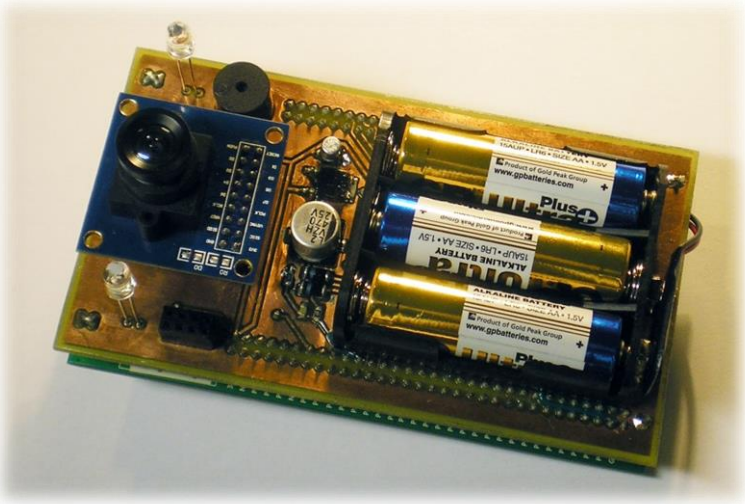
Obr. 2.12: Blokové schéma výsledného prototypu snímacího systému

Z blokového schéma výše je tedy možné vidět další použité periferie, které nebyly zmíněny v předchozím popisu HW částí, protože se ve většině případů jedná o standardní a hojně používané periferie, které lze nalézt ve velkém množství jiných embedded aplikacích. Mezi tyto periferie patří např. standardní serial peripheral interface (SPI) sběrnice pro komunikaci s řadičem LCD displeje popř. SD kartou nebo I2C sběrnice pro komunikaci s řadičem rezistivní dotykové vrstvy displeje. Určitou speciální záležitostí je komunikace s RTC modulem nebo SDRAM, které jsou výhradně problematikou výrobce *STMicroelectronics*. Všechny tyto výše zmíněné periferie budou blíže popsány v další kapitole, kdy se budeme již přímo zabývat návrhem obslužného FW prototypu.

Lze tedy říct, že výsledný prototyp je malý a kompaktní snímací systém, který lze velmi jednoduše držet v ruce pro případ jednorázových měření nebo vhodně ukotvit pro měření dlouhodobá. Výsledný kompletní prototyp včetně přídavných modulů je možné vidět na fotografiích Obr. 2.13 a Obr. 2.14 níže.



Obr. 2.13: Fotografie přední strany prototypu snímacího systému



Obr. 2.14: Fotografie zadní strany prototypu snímacího systému

3 OVLÁDACÍ FIRMWARE

Dalším důležitým krokem úspěšného návrhu libovolného elektronického zařízení je vytvoření samotného ovládacího FW pro dosažení požadované funkce prototypu.

Tato kapitola se tedy bude zabývat konkrétním vývojem programového kódu a jeho implementací na výsledný prototyp snímacího systému. Postupně tedy budou popsány jednotlivé funkční částí zajišťující komunikaci mezi MCU a periferiemi a nakonec bude popsáno samotné jádro prototypu a zpracování dat. Je ovšem nutné vzít v potaz, že samotný programový kód je velmi rozsáhlý a nelze jej tedy celý v tomto dokumentu rozebrat. Kompletní programový kód lze samozřejmě nalézt v digitální příloze.

Nicméně před samotným návrhem FW je dobré se obeznámit se samotným jádrem architektury ARM a s podporou použité vývojové desky jako je například sbírka knihoven, utilit a příkladů, které mohou vývojáři ve výsledné aplikaci napomoci.

Výrobce použité vývojové desky *STMicroelectronics* nabízí všechny výše zmíněné příklady podpory v rámci jediného FW balíčku, který lze nalézt zde [8]. Tento balíček je kompletní platformou pro úspěšný návrh libovolné aplikace a to jak z hlediska podpory samotného jádra MCU, tedy standardu cortex microcontroller software interface standard (CMSIS) pro MCU Cortex-M, tak z hlediska podpory použité vývojové desky 32F429IDISCOVERY. Pro potřeby samotného programování je vhodné se nechat inspirovat oficiálním uživatelským manuálem vývojové desky, který lze nalézt zde [13].

3.1 Vývojové prostředí

Než přejdeme k popisu samotného obslužného FW prototypu, tak je dobré se okrajově zmínit o použitém vývojovém prostředí, protože dobře zvolné vývojové prostředí může programátorovi v mnoha situacích pomoci a vývoj programového kódu ulehčit. Výrobce *STMicroelectronics* v datovém listu použité vývojové desky, který lze nalézt zde [5] popř. zde [13], doporučuje použití komerčních vývojových prostředí jako:

- *Atollic TrueSTUDIO* (založeno na GCC)
- *IAR EWARM*
- *Keil MDK-ARM*

Tyto výše zmíněná vývojová prostředí jsou obecně si sobě velmi podobná a všechna obsahují většinu standardních programátorských nástrojů, nicméně vzhledem k tomu, že se jedná o velmi kvalitní komerční vývojová prostředí, tak je jejich použití vždy omezeno velmi vysokou pořizovací cenou. V určitých případech lze výše zmíněný software (SW) získat jako demoverzi s určitými omezeními jako je např. maximální velikost kompilovaného programového kódu (typicky 32 KB) apod.

Pokud se podíváme na opačnou stranu spektra, tedy na freeware distribuci ARM vývojových prostředí, tak pravděpodobně největšími zástupci jsou *EM::Blocks* a *CooCox CooIDE*, která jsou logicky založena na GCC. Tento volně dostupný SW je obdobně použitelný jako jejich komerční verze, nicméně zde většinou narážíme na problematičtější zajištění kompatibility s použitým HW a obecně problematiku stability jak vývojového prostředí, tak navrženého FW.

Určitou alternativou je taktéž použití oficiálního grafického vývojového prostředí *STM32CubeMX*, nicméně právě přístup grafického programování je velmi neefektivní a značně nepraktický.

Pro potřeby této diplomové práce byl zvolen kompromis mezi komerční a freeware open source sférou a to tak, že pro vývoj kritických částí programového kódu bylo využito vývojové prostředí *Keil MDK-ARM uVision5*, které je omezeno právě zmíněnou velikostí kompilovaného programového kódu a pro kompletaci projektu bylo využito vývojové prostředí *Em::Blocks v2.30* s integrovaným GCC kompilátorem.

3.2 Externí SDRAM paměť

Jako první se v této kapitole zaměříme na problematiku implementace SDRAM paměti, která je součástí použité vývojové desky. Tato SDRAM paměť je totiž klíčovou částí, bez které by realizace prototypu byla nemožná.

Samotné použití této paměti se v principu omezuje na pouhou korektní inicializaci flexible memory control (FMC) periferie a následně použití standardního přístupu do adresovacího prostoru jako by se jednalo o součást programové paměti RAM. Počáteční adresa SDRAM paměti je definována jako 0xD0000000 a její celková šířka je 64 Mbitů popř. 8 MB nebo v hexadecimálním zápisu jako 0x00800000.

Konfigurace FMC periferie je obecně rozsáhlejší problematikou a nebude tedy uvedena v této dokumentaci, nicméně vzhledem k tomu, že se jedná o součást vývojové desky, tak je možné velmi jednoduše použít již hotové řešení dostupné přímo od výrobce, které lze nalézt ve zmíněném FW balíčku zde [8] nebo již upravenou verzi, kterou lze nalézt v digitální příloze.

Nyní je ovšem vhodné pohovořit o interním rozdělení této externí SDRAM paměti, aby byla zajištěna ochrana proti potenciálním kolizím mezi jednotlivými procesy, které budou SDRAM využívat. V našem případě je tato paměť rozdělena do tří oblastí, které jsou reprezentovány jako direktivy preprocesoru a představují počáteční adresy těchto jednotlivých paměťových oblastí. Toto rozdělení poté vypadá jako:

```
// SDRAM addresses
#define SDRAM_MEM_START      (uint32_t)0xD0000000      //6MB
#define SDRAM_LCD_START1     (uint32_t)0xD0600000      //1MB
#define SDRAM_LCD_START2     (uint32_t)0xD0700000      //1MB
```

Samotné vysvětlení, proč byly tyto jednotlivé oblasti zvoleny podle výše uvedeného vzoru, bude rozvedeno dále v textu, protože jsou spjaté s funkcí některých periférií a navrženého systému obecně.

3.3 Získání snímku kamerou

Dále pohovoříme o problematice týkající se získání obrazových dat modulem kamery, která představuje základ pro následnou aplikaci OCR apod. Získáním obrazových dat je obecně myšlen přesun těchto dat odesílaných kamerou do paměti MCU, což bude hlavním podmětem pro tuto podkapitolu. V našem případě budeme konkrétně používat přenos mezi zmíněnou DCMI sběrnicí a externí SDRAM pamětí, která pro naše potřeby nyní představuje datový buffer.

3.3.1 Hodinový signál kamery

Jako první se zaměříme na konfiguraci systémového hodinového signálu pro použitý modul kamery. Jak již bylo uvedeno v kap. 2.3.1, tento hodinový signál musí být inicializován co nejdříve, jinak je modul kamery nepoužitelný.

Samotnou konfiguraci jakéhokoliv hodinového signálu lze provést mnoha způsoby, nicméně v našem případě z důvodu uvolnění několika klíčových kontrolních pinů, byla zvolena alternativa generování hodinového signálu pomocí jednoduchého čítače/časovače v pulse-width modulation (PWM) módu se střídou 50%. Konkrétně byl zvolen čítač/časovač 1 (TIM1), který je vyveden na pin PA9.

Výsledná konfigurace pinu se skládá z korektní inicializace potřebného HW, tedy konfigurace výše uvedeného I/O pinu, nastavení jeho alternativní funkce a konfigurace zmíněného čítače/časovače. V tomto příp ad je důležitá konfigurace pinu jako výstupní typu push-pull a nastavení interního pull-up rezistoru.

Dále již tento čítač/časovač konfigurujeme standardně pro zvolený PWM mód. Výsledná konfigurace poté z hlediska programového kódu vypadá následovně:

```
// Time base configuration
TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_Period = TIM_PERIOD-1;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
TIM_ARRPreloadConfig(TIM1, ENABLE);

// TIM PWM1 Mode configuration
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_Pulse = TIM_DUTY_50;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

// Output Compare PWM Mode configuration
TIM_OC2Init(TIM1, &TIM_OCInitStructure);
TIM_OC2PreloadConfig(TIM1, TIM_OCPreload_Enable);

// Enable TIM
TIM_CtrlPWMOutputs(TIM1, ENABLE);
TIM_Cmd(TIM1, ENABLE);
```

3.3.2 SCCB komunikace

Dále se zaměříme na problematiku emulace SCCB sběrnice pomocí standardně používané sběrnice I2C, protože jak již bylo zmíněno v kap. 2.3.2, tyto sběrnice se sobě v principu velmi podobají.

Prvním krokem je opět standardně inicializace potřebného HW, tedy konfigurace příslušných I/O pinů, které budeme pro sběrnici používat a jejich korektní nastavení. V tomto případě je důležité piny nakonfigurovat jako výstupní typu otevřený kolektor/drain, zrušení použití pull-up nebo pull-down rezistorů, které jsou realizovány HW na základové DPS, a samozřejmě zadání alternativní funkce těchto pinů.

Konkrétně byla vybrána kombinace sběrnice I2C1 na následujících pinech:

- PB8 – I2C1_SCL (SCCB_SIOC)
- PB9 – I2C1_SDA (SCCB_SIOD)

Samotná použitá konfigurace I2C sběrnice, která je součástí inicializace SCCB sběrnice poté z hlediska programového kódu vypadá následovně:

```
// I2C config
I2C_DeInit(I2C1);
I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
I2C_InitStructure.I2C_OwnAddress1 = 0x00;
I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
I2C_InitStructure.I2C_ClockSpeed = 100000;
I2C_Init(I2C1,&I2C_InitStructure);
I2C_Cmd(I2C1, ENABLE);
```

V této fázi již můžeme přejít přímo k problematice emulace SCCB sběrnice. Vzhledem ke skutečnosti, že v naší aplikaci požadujeme pouze konfiguraci použitého modulu kamery, tak se budeme zabývat pouze funkcí zápisu do jednotlivých konfiguračních registrů modulu kamery. Zde je nutné zmínit, že je velmi důležité dodržení tzv. třífázové komunikace, jak bylo popsáno v kap. 2.3.2, jinak může dojít k chybné funkci sběrnice.

Výsledná emulace použité SCCB sběrnice poté z hlediska programového kódu vypadá následovně:

```
uint8_t SCCB_write_reg(uint8_t reg_addr, uint8_t* data){
    uint32_t timeout = SCCB_TIMEOUT;

    // Send start bit
    I2C_GenerateSTART(I2C1, ENABLE);
    // Wait until I2C is busy
    while(!I2C_GetFlagStatus(I2C1, I2C_FLAG_SB)){
        if ((timeout--) == 0)
            return 1;
    }
    while( !I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_MODE_SELECT)){
        if ((timeout--) == 0)
            return 1;
    }
    // Send slave address (OV7670_WRITE_ADDR)
    I2C_Send7bitAddress(I2C1, OV7670_WRITE_ADDR, I2C_Direction_Transmitter);
    while(!I2C_CheckEvent(I2C1,
    I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED)){
        if ((timeout--) == 0)
            return 1;
    }
    // Send register address
    I2C_SendData(I2C1, reg_addr);
    while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_BYTE_TRANSMITTED)){
        if ((timeout--) == 0)
            return 1;
    }
    // Send new register value
    I2C_SendData(I2C1, *data);
    while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_BYTE_TRANSMITTED)){
        if ((timeout--) == 0)
            return 1;
    }
    // Send stop bit
    I2C_GenerateSTOP(I2C1, ENABLE);

    // Write done
    return 0;
}
```

Výše uvedená funkce přestavuje tedy zápis popř. změnu hodnoty pouze jednoho jediného registru kamery, což naznačuje její opakované použití. Kromě standardního odesílání dat a START/STOP bitů můžeme ve výše uvedené funkci vidět i zmíněné odesílání adresy signalizující provedení zápisu apod.

Další důležitou částí emulované funkce jsou výše uvedené čekací smyčky, které zjišťují stav na I2C sběrnici. Tyto smyčky zamezují, aby se algoritmus dostal do nežádoucího nedefinovaného stavu, nicméně v praxi takto může dojít k vypršení časového limitu (např. vlivem rušení), což znamená vyvolání chybového stavu (vrácení hodnoty 1). Zde je poté nutné celou inicializaci kamery provést znova.

3.3.3 Konfigurace modulu kamery

Pokud byla provedena inicializace SCCB sběrnice korektně, tak v této chvíli je možné přejít ke konfiguraci interních řídících registrů DSP modulu kamery, čímž získáme požadovanou funkci tohoto modulu.

Pokud se obecně zaměříme na problematiku konfigurace libovolného HW zařízení, tak lze říct, že se vývojář vždy standardně jako první obrací na datový nebo aplikační list daného produktu, v našem případě tedy modulu použité kamery. Nicméně právě v případě použité kamery OV7670 narážíme na velmi chabou podporu ze strany výrobce *OmniVision*. Oficiální dokumenty jsou z hlediska konfigurace registrů velmi omezené a často nepravdivé. Při vývoji programového kódu můžeme tedy narazit na problémy, kdy výchozí konfigurace modulu v několika případech neodpovídá reálné funkci (např. registr 0x42) nebo skutečnost, že pro dosažení uspokojivých výsledků je nutná konfigurace i zakázaných registrů bez oficiálního popisu atd. Zde je typickým příkladem rezervovaný registr na adrese 0xb0, který ve svém výchozím stavu konfiguruje kameru do barevného schématu, které je velmi podobné černobílému režimu. Pro barevný režim je tedy nutné registr nakonfigurovat na hodnotu 0x84, kterou ovšem není možné v žádné oficiální literatuře nalézt. Další nepříjemností je fakt, že v určitých případech záleží i na pořadí konfigurovaných registrů.

Obecně lze tedy říct, že nejlepším zdrojem informací, co se týče požadovaného nastavení kamery, je uživatelská základna na internetových fórech a volně šířené aplikace této použité kamery. Jako vhodný zdroj informací lze zmínit uživatelskou knihovnu OV7670 zde [15] nebo vývojovou komunitu ARM mbed zde [16]. Nicméně je vhodné kombinovat a konfrontovat informace uživatelů i se zmíněným datovým listem zde [10] nebo aplikačním manuálem zde [11].

Před samotnou konfigurací a testováním modulu kamery je vhodné se přesvědčit, že vzájemná komunikace mezi touto kamerou a MCU je funkční. Pro tuto příležitost je vhodné kameru nakonfigurovat tak, aby byl odesílán pouze zkušební obrazec. Tento obrazec je standardně složen z barevných vertikálních pruhů, na které můžeme být zvyklí z jiných zobrazovacích zařízení. Zkušební obrazec lze aktivovat zapsáním hodnoty 0x08 na adresu registru 0x42.

Kompletní seznam všech provedených úprav registrů kamery jako je např. aplikace automatického vyrovnávání citlivosti (AGC), automatické úpravy doby expozice (AEC) nebo automatické vyvážení bílé (AWB), lze nalézt v digitální příloze, nicméně nyní můžeme pohovořit alespoň o některých základních úpravách týkající se systémových parametrů a formátu jednotlivých snímků, které je nutné pro korektní snímání ošetřit.

Tyto základní úpravy pro výsledný prototyp snímacího systému lze vidět v programovém kódu níže, který je součástí rozsáhlejšího dvourozměrného pole:

```
{0x12, 0x80},      // - Reset registers

// Image format
{0x12, 0x14},      // - QVGA size, RGB mode
{0x40, 0xd0},      // - RGB565
{0xb0, 0x84},      // - Color mode
{0x1e, 0x00},      // - Mirror/VFlip disable

// Hardware window
{0x11, 0xc0},      // - PCLK settings
{0x32, 0x80},      // - HREF
{0x17, 0x17},      // - HSTART
{0x18, 0x05},      // - HSTOP
{0x03, 0x0a},      // - VREF
{0x19, 0x02},      // - VSTART
{0x1a, 0x7a},      // - VSTOP

// Scalling numbers
{0x70, 0x3a},      // - X_SCALING
{0x71, 0x35},      // - Y_SCALING
{0x72, 0x11},      // - DCW_SCALING
{0x73, 0xf0},      // - PCLK_DIV_SCALING
{0xa2, 0x02},      // - PCLK_DELAY_SCALING
```

Z výše uvedeného programového kódu můžeme vidět např. resetování všech registrů do výchozího nastavení nebo nastavení synchronizačních signálů pro zajištění synchronizace na základě zvoleného formátu snímků. Co se týče nastavení odesílaných obrazových dat, tak lze vidět, že byl zvolen již výše zmíněný formát RGB565 a rozlišení 320x240 px (QVGA), který je podporován kamerou i použitým displejem.

3.3.4 DCMI komunikace a DMA přenos

Nakonec tedy zbývá se zaměřit na konfiguraci samotné DCMI sběrnice. Prvním krokem je tedy opět standardní inicializace potřebného HW, tedy konfigurace příslušných I/O pinů, které budeme pro tuto sběrnici používat, jejich korektní nastavení a samozřejmě zadání alternativní funkce těchto pinů. Zde je důležitá konfigurace pinů jako výstupní typu push-pull a nastavení interních pull-up rezistorů. Z rozsáhlého množství alternativ byla vybrána následující kombinace použitých pinů:

- PB7 – DCMI_VSYNC
- PA4 – DCMI_HREF
- PA6 – DCMI_PCLK
- PE6 – DCMI_D7
- PE5 – DCMI_D6
- PD3 – DCMI_D5
- PE4 – DCMI_D4
- PG11 – DCMI_D3
- PC8 – DCMI_D2
- PC7 – DCMI_D1
- PC6 – DCMI_D0

Nyní je možné přejít přímo ke konfiguraci samotné DCMI sběrnice. Zde logicky budeme používat HW synchronizaci a osmibitový přenos, nicméně klíčovou součástí je nastavení právě zmíněné synchronizace vývojové desky a modulu kamery. Touto problematikou jsme se již zabývali v kap. 2.3.1, kde vidíme, že datové signály je nutné vzorkovat v okamžicích nástupné hrany signálu PCLK. Tyto data jsou poté platná v případě H úrovně signálu HREF a L úrovně signálu VSYNC, nicméně v referenčním manuálu MCU zde [14] zjistíme, že nastavení těchto úrovní musí být konfigurováno v invertované podobě vzhledem k úrovním zmíněným výše.

Samotná použitá konfigurace této DCMI sběrnice poté z hlediska programového kódu vypadá následovně:

```
// DCMI config
DCMI_DeInit();
DCMI_InitStructure.DCMI_CaptureMode = DCMI_CaptureMode_Continuous;
DCMI_InitStructure.DCMI_ExtendedDataMode = DCMI_ExtendedDataMode_8b;
DCMI_InitStructure.DCMI_CaptureRate = DCMI_CaptureRate_1of2_Frame;
DCMI_InitStructure.DCMI_PCKPolarity = DCMI_PCKPolarity_Rising;
DCMI_InitStructure.DCMI_HSPolarity = DCMI_HSPolarity_Low;
DCMI_InitStructure.DCMI_VSPolarity = DCMI_VSPolarity_High;
DCMI_InitStructure.DCMI_SynchroMode = DCMI_SynchroMode_Hardware;
DCMI_Init(&DCMI_InitStructure);
DCMI_Cmd(ENABLE);
```

Dále je nutné se zaměřit na konfiguraci a použití DMA přenosu, protože jak již bylo zmíněno v kap. 2.1 popisu vývojové desky, aplikace MCU Cortex-M4 implicitně předpokládá využití DMA přenosu pro manipulaci s daty a převážně s datovými toky. V našem případě budeme tedy využívat jeden kanál jednoho datového toku jediného DMA řadiče, který bude sloužit pro jednoduchý přesun mezi datovým registrem DCMI periferie představující obrazová data a zmíněnou SDRAM pamětí.

Samotná konfigurace DMA kanálu může být z hlediska programového kódu mírně obtížnější záležitostí, takže je vhodné se obrátit na referenční manuál MCU zde [14] pro zajištění korektní konfigurace a ověření dostupnosti používaných periférií na daném DMA řadiči. Námi požadované funkci odpovídá kanál 1 toku 1 řadiče DMA2.

Hlavními parametry této konfigurace bude tedy zvolení správného DMA kanálu, nastavení přenosu mezi periférií a pamětí, zadání zdrojové adresy datového toku, koncové adresy paměti kam mají být data ukládána, nastavení velikosti FIFO bufferu podle velikosti přijímaných obrazových dat atd. V případě použití DMA přenosu je velmi užitečnou funkcí DMA řadiče automatizovaný převod mezi datovými typy (Word, HalfWord, Byte) použité periferie popř. paměti a paměti. Nicméně v našem případě není žádný automatizovaný převod potřeba, protože DMA řadič ukládá data o totožné velikosti celého datového registru DCMI periferie do oblasti externí SDRAM, která představuje první oblast LCD bufferu pro použitý displej.

Samotná použitá konfigurace tohoto DMA přenosu poté z hlediska programového kódu vypadá následovně:

```
// DMA config
DMA_DeInit(DMA2_Stream1);
DMA_InitStructure.DMA_Channel = DMA_Channel_1;
DMA_InitStructure.DMA_PeripheralBaseAddr = DCMI_DR_ADDRESS;
DMA_InitStructure.DMA_Memory0BaseAddr = SDRAM_LCD_START1;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
DMA_InitStructure.DMA_BufferSize = IMG_BYTES/4;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;
DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
DMA_Init(DMA2_Stream1, &DMA_InitStructure);
DMA_Cmd(DMA2_Stream1, ENABLE);
```

Zde je vhodné dodat, že k použití DMA kanálu se váže možnost nastavení přerušení podle zvolené události uživatelem. V našem případě budeme využívat přerušení při přenesení celého snímku kamery do cílové oblasti (DMA_IT_TC). Nutno ovšem dodat, že pro správnou funkci tohoto přerušení je nutné toto nastavení nakonfigurovat i v rámci nested vector interrupt controller (NVIC) řadiče přerušení.

3.4 Zobrazení obrazových dat

Další problematikou, kterou se budeme intenzivně zabývat je zobrazení získaného snímku na dostupný LCD displej použité vývojové desky. Tato záležitost je obecně jednou z nejproblematičtějších částí projektu, a proto jí bylo věnováno značné množství času spojeného s návrhem a testováním výsledného algoritmu zobrazení.

Samotný LCD displej je v případě použité vývojové desky řízen řadičem ILI9341 výrobce *ILITEK*. Zde se pro zobrazování libovolného vizuálního obsahu standardně využívá LCD-TFT display controller (LTDC) periferie, která umožňuje velmi rychlé a kvalitní zobrazování obrazových dat pomocí 18b paralelní sběrnice (RGB666), nicméně tento způsob zobrazování není možné v našem případě nijak využít.

Překážkou je fakt, že některé z požadovaných pinů, které jsou pro funkci LTDC periferie klíčové jsou již využity pro DCMI sběrnici použitého modulu kamery. Konkrétně neproblematičtější je funkce pinu PA4, který je používán pro horizontální synchronizaci DCMI sběrnice a zároveň pro vertikální synchronizaci LTDC periferie. Obecně tento pin nelze žádným způsobem sdílet ani multiplexovat a je tedy nutné nalézt jinou alternativu, která umožní požadovanou komunikaci s řadičem displeje.

Tento problém lze tedy v našem případě řešit pouze jedinou možností a to použitím čistě sériové SPI komunikace, kterou budeme mimo konfigurace a ovládání řadiče používat i pro přenos přímo obrazových dat. Tento způsob komunikace je logicky mnohem pomalejší a přináší jistá další úskalí, nicméně správným algoritmem lze tento přístup značně zefektivnit a ve výsledku jej přizpůsobit pro potřeby této aplikace.

Nutné ovšem dodat, že pouze datová komunikace nestačí pro zajištění korektní komunikace a je tedy nutné dodržet specifikace použitého řadiče ILI9341. V našem konkrétním případě samozřejmě mluvíme o korektní inicializaci řadiče, která je v případě použité vývojové desky značně ulehčena a dále korektním ovládání řídících signálů data/command (označen WRX), reset (označen RDX) a samozřejmě chip select (označen CSX). Více informací o tomto řadiči lze nalézt zde [17] a současně je vhodné věnovat pozornost zapojení vývojové desky zde [5].

3.4.1 SPI komunikace a DMA přenos

Začneme tedy samotnou konfigurací SPI periferie pro komunikaci se zmíněným řadičem displeje. Použitý MCU obecně nabízí celkem šest SPI sběrnic, které lze libovolně využívat, nicméně z hlediska HW zapojení použité vývojové desky je možné použít pouze jedinou kombinaci na následujících pinech:

- PF7 – SPI5_SCK
- PF8 – SPI5_MISO
- PF9 – SPI5_MOSI

Zde je opět nutné se první zaměřit na HW konfigurací I/O pinů, které budeme pro tuto sběrnici používat, jejich korektní nastavení a samozřejmě zadání alternativní funkce těchto pinů. Zde je důležitá konfigurace pinů jako výstupní typu push-pull a zrušení použití pull-up nebo pull-down rezistorů. Poté je již možné přímo přejít ke konfiguraci SPI periferie, která z hlediska programového kódu vypadá následovně:

```
// SPI config
SPI_Cmd(SPI5, DISABLE);
SPI_StructInit(&SPI_InitStruct);
SPI_InitStruct.SPI_BaudRatePrescaler = prescaler;
SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStruct.SPI_Mode = SPI_Mode_Master;
SPI_InitStruct.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStruct.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStruct.SPI_NSS = SPI_NSS_Soft | SPI_NSSInternalSoft_Set;
SPI_Init(SPI5, &SPI_InitStruct);
SPI_Cmd(SPI5, ENABLE);
SPI_I2S_DMACmd(SPI5, SPI_I2S_DMAReq_Tx, ENABLE);
```

Z výše uvedeného programového kódu můžeme vidět nastavení potřebných HW vlastností, které jsou samozřejmě vázány na použitý řadič displeje. Podle takto provedené konfigurace je již možné řadič jednoduše inicializovat a dále podle specifikací přenášet obrazová data. Ovšem tento způsob blokujícího zobrazování, kdy je nutné každý přenášený bajt odesílat manuálně, je velmi neefektivní a pomalý. Pro dosažení vyšších obnovovacích kmitočtů je vhodnější využít již několikrát zmiňovaný DMA řadič. Použití DMA řadiče je již možné vidět i z programového kódu výše, kde můžeme vidět právě povolení DMA přenosu pro vysílací (TX) fázi.

Nyní tedy provedeme konfiguraci dalšího DMA kanálu podobně jako bylo uvedeno v předchozí kapitole. Zde opět je vhodné se obrátit na referenční manuál MCU zde [14]. Námi požadované funkci, konkrétně tedy SPI5_TX, odpovídá kanál 2 toku 4 řadiče DMA2. Konfigurace poté je obdobná jako v předchozí kapitole pouze s rozdílem velikosti FIFO bufferu a změnou směru toku. Nyní tedy potřebujeme přenášet data z paměti na periferii, odkud se odvíjí i zdrojová a cílová adresa datového toku.

Samotná použitá konfigurace tohoto DMA přenosu poté z hlediska programového kódu vypadá následovně:

```
// DMA config
DMA_DeInit(DMA2_Stream4);
DMA_InitStructure.DMA_Channel = DMA_Channel_2;
DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t) p_img_adr;
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t) &(SPI5->DR);
DMA_InitStructure.DMA_DIR = DMA_DIR_MemoryToPeripheral;
DMA_InitStructure.DMA_BufferSize = IMG_BYTES/3;
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
DMA_InitStructure.DMA_Mode = DMA_Mode_Normal;
DMA_InitStructure.DMA_Priority = DMA_Priority_VeryHigh;
DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
DMA_Init(DMA2_Stream4, &DMA_InitStructure);
DMA_Cmd(DMA2_Stream4, ENABLE);
```

Opět je vhodné dodat, že zde budeme používat přerušení při přenesení celého datového objemu do cílové oblasti (DMA_IT_TC), konkrétně v tomto případě při přenesení celého obsahu bufferu. Dále pro správnou funkci tohoto přerušení je nutné jeho nastavení nakonfigurovat i v rámci NVIC řadiče přerušení.

3.4.2 Zpracování DMA přerušení

Nyní je nutné se zaměřit na problematiku, jak výše popsaná přerušení od zvolených DMA kanálů zpracovat. V principu požadujeme pouze, aby se data z použité DCMI sběrnice přesunula do oblasti LCD bufferu odkud se poté pomocí SPI sběrnice postupně odešlou směrem k řadiči displeje.

Jednoduše tedy požadujeme, aby jakmile dojde k vyvolání přerušení od DMA toku 1, tak aby došlo k aktivaci DMA přenosu toku 4. Zde ovšem narážíme na omezení samotného DMA řadiče, který dokáže inkrementovat např. paměťovou oblast pouze v limitu neznaménkové 16 bitové hodnoty, tedy maximálně do hodnoty 65536.

Znamená to tedy, že zatímco v případě DMA toku 1, který obecně operuje pouze s 32 bitovými hodnotami je celkový počet inkrementů roven hodnotě 38400, tak v případě DMA toku 4 operujícím s 8 bitovými hodnotami je celkový počet inkrementů roven hodnotě 153600. Tato hodnota je již nad rámec možností DMA řadiče a je tedy nutné DMA přenos na SPI periferii rozdělit na tři jednotlivé DMA přenosy. Jinak řečeno jednomu přenosu DMA toku 1 odpovídají celkem tři přenosy DMA toku 4. Je tedy jasné, že se přerušení budou zpracovávat podle mírně složitějšího algoritmu.

Výsledná obsluha DMA přerušení toku 1 tedy vypadá z hlediska programového kódu následovně:

```
/* DCMI DMA interrupt */
void DMA2_Stream1_IRQHandler(void){
    uint32_t n, lcd_buffer;
    uint32_t *lcd_sdram1 = (uint32_t*) SDRAM_LCD_START1;
    uint32_t *lcd_sdram2 = (uint32_t*) SDRAM_LCD_START2;

    // DMA complete
    if(DMA_GetITStatus(DMA2_Stream1, DMA_IT_TCIF1) == SET){
        // Convert image to little endian and crop image
        for (n = 0; n < IMG_HALF_PX; n++){
            lcd_buffer = lcd_sdram1[n];
            lcd_sdram2[n] = (((lcd_buffer & 0xff000000) >> 8) |
                             ((lcd_buffer & 0x00ff0000) << 8) |
                             ((lcd_buffer & 0x0000ff00) >> 8) |
                             ((lcd_buffer & 0x000000ff) << 8));
        }

        // Prepare LCD for image
        LCD_ILI9341_SetCursorPosition(0, 0, IMG_HEIGHT-1, IMG_WIDTH-1);
        LCD_ILI9341_SendCommand(ILI9341_GRAM);

        // SPI send data
        ILI9341_WRX_SET;
        ILI9341_CS_RESET;
        SPI_DMA_Init(lcd_sdram2);

        // Clear IRQ flag
        DMA_ClearITPendingBit(DMA2_Stream1, DMA_IT_TCIF1);
    }
}
```

Z výše uvedeného programového kódu je možné vidět několik důležitých částí, jako např. konverze obrazových dat z big-endianu používaného MCU na little-endian používaného řadičem LCD displeje. Pozorný čtenář si může i všimnout, že v rámci této konverze se data zároveň ukládají do třetí oblasti SDRAM paměti, tedy do druhého LCD bufferu. Tato skutečnost souvisí se zpožděním při deaktivaci DCMI sběrnice, kde lze v určitých případech narazit na nežádoucí vykreslení posledního přijatého snímku. Vzhledem k množství dostupné externí paměti tento problém ovšem nepředstavuje žádnou překážku. Pouze je vodné dodat, že v rámci této dokumentace byla pro lepší ilustraci odstraněna část programového kódu zajišťující funkci ořezu získaného snímku, která bude popsána v následující podkapitole.

Následně můžeme vidět přípravu displeje pro příchozí data a následnou inicializaci DMA toku 4 od adresy začátku druhého LCD bufferu. Jak bylo možné vidět v předchozí podkapitole, tak v rámci této inicializace je i aktivace tohoto DMA toku. V této fázi tedy máme vykreslenou první třetinu obrazových data na LCD displej, což vyvolá přerušení od DMA toku 4.

Obsluha tohoto přerušení poté z hlediska programového kódu vypadá následovně:

```
/* LCD SPI interrupt */
void DMA2_Stream4_IRQHandler(void){
    static uint8_t n = 0;
    uint32_t *lcd_sdram2 = (uint32_t*) SDRAM_LCD_START2;
    uint32_t offset_0 = (IMG_BYTES/3)>>2, offset_1 = (IMG_BYTES/3)>>1;

    // DMA_SPI complete
    if(DMA_GetITStatus(DMA2_Stream4, DMA_IT_TCIF4) == SET){
        //Wait for SPI to be ready
        while (SPI_I2S_GetFlagStatus(ILI9341_SPI, SPI_I2S_FLAG_BSY));

        // Reinitialize SPI DMA channel
        if ( n == 0){
            SPI_DMA_Init(lcd_sdram2 + offset_0);
            n++;
        } else if (n == 1){
            SPI_DMA_Init(lcd_sdram2 + offset_1);
            n++;
        } else{
            ILI9341_CS_SET;
            n=0;
        }

        // Clear IRQ flag
        DMA_ClearITPendingBit(DMA2_Stream4, DMA_IT_TCIF4);
    }
}
```

Zde již můžeme vidět velmi jednoduchý způsob posuvu ukazatele na adresu SDRAM paměti podle daného offsetu. Tento posuv musí však být součástí opětovné inicializace DMA toku 4. Tímto způsobem se vykreslí další dvě třetiny obrazových dat na LCD displej a nakonec se provede ukončení datového přenosu.

3.4.3 Funkce ořezu obrazových dat

Nakonec této podkapitoly je vhodné krátce pohovořit o jedné z přídavných funkcí, která je ve výsledném FW prototypu implementována a váže se na problematiku zobrazování obrazových dat. Konkrétně budeme hovořit o funkci jednoduchého ořezu získávaných obrazových dat podle aktuálního nastavení uživatelem.

Tato funkce v principu slouží pouze pro lepší definici aktivní oblasti obrazových dat pro následnou aplikaci OCR algoritmu. V praxi totiž může velmi snadno nastat situace, kdy nelze jednoduše snímat pouze aktivní oblast, v našem případě tedy číselník např. plynoměru a je nutné zabrat mnohem větší oblast. Ovšem tato oblast již může zahrnovat jiné údaje jako např. druhý číselník nebo identifikační číslo apod. a tyto aditivní informace by mohly nežádoucím způsobem ovlivnit výslednou aplikaci OCR.

Pokud se tedy zaměříme na samotnou implementaci algoritmu realizující ořez snímku, tak pravděpodobně nejjednodušším způsobem je aplikace přímo automatického ořezu pomocí DCMI periferie. Ovšem zde narazíme na problematiku nutnosti rekonfigurace horizontální a vertikální synchronizace, pro každou kombinaci ořezu, kterou může uživatel zadat. Tato skutečnost činí použití tohoto automatizovaného ořezu jako nevyhovující, protože jak již může být z předcházejících kapitol jasné, tak navržený algoritmus snímání a zobrazování obrazových dat je velmi závislý na přesné synchronizaci mezi modulem kamery a MCU a proto jej nelze neustále libovolně měnit. Nutno dodat, že ořez lze provádět i přímo pomocí interního DSP použitého modulu kamery, nicméně jak bylo uvedeno v kap. 3.3.3, tak samotná konfigurace kamery je velmi problematickou a obecně nastavení tohoto ořezu pomocí interních registrů kamery je značně obtížné.

Nakonec byla tedy zvolena realizace manuálního ořezu, který je realizován během zpracovávání přerušení od DMA toku 1. Jak již bylo uvedeno výše, v tomto přerušení je realizována konverze dat, takže lze v rámci této konverze přidat ještě samotnou funkci ořezu. Tento způsob je obecně opět velmi neefektivní, nicméně v našem případě lze využít skutečnost rozdílu mezi velmi rychlým zpracováváním dat z DCMI sběrnice a pomalým odesíláním těchto dat do LCD pomocí SPI sběrnice.

Výsledný programový kód, který je umístěn ve zmíněném přerušení DMA toku 1 poté z hlediska programového kódu vypadá následovně:

```
uint32_t data_buffer_d1 = (IMG_WIDTH - crop_width)>>1;
uint32_t data_buffer_d2 = (IMG_HEIGHT - crop_height)>>2;

// Convert image to little endian and crop image
for (n = 0; n < IMG_HALF_PX; n++){
    lcd_buffer = lcd_sdram1[n];

    if (n >= IMG_HALF_HEIGHT*data_buffer_d1 &&
        n <= IMG_HALF_HEIGHT*(data_buffer_d1+crop_width)-1 &&
        n%IMG_HALF_HEIGHT >= data_buffer_d2 &&
        n%IMG_HALF_HEIGHT+1 <= data_buffer_d2+(crop_height>>1))
        // Convert data to little endian
    else
        lcd_sdram2[n] = 0xFFFFFFFF;
}
```

Výše uvedený programový kód ořezu se může zprvu zdát mírně složitější, ovšem ve skutečnosti se MCU pouze opakovaně dotazuje, zda jsou daná data v aktivní oblasti vybrané uživatelem nebo mimo tuto oblast. Pokud se nachází v této oblasti, tak MCU provede zmíněnou konverzi dat a v případě, že je mimo tuto oblast, tak data nahradí bílou barvou. Pozorný čtenář si může všimnout, že tento ořez se v našem případě provádí po jednotlivých 32 bitových registrech, které představují dva obrazové pixely vedle sebe. Je tedy jasné, že tento algoritmus nemůže provádět ořez po jednotkách pixelů, ale po jejich párech. V principu lze tedy říct, že nejmenší možný algoritmem povolený ořez může být celkem 4 px, tedy 2 px na každé straně snímku.

Tímto způsobem se tedy velmi jednoduše upraví již existující obrazová data, která jsou poté opět jako celek zobrazena na LCD displej. Nutno ovšem zmínit, že tento způsob zobrazování spojený s ořezem je velmi závislý na přesném časování jednotlivých událostí, což může představovat určitý problém při přenesení algoritmu na jinou platformu nebo v případě použití jiného systémového taktu. Tento fakt již bylo možné vysledovat např. z konfigurace kamery, která pracuje s relativně vysokým systémovým taktem, ovšem DCMI periferie zpracovává pouze každý druhý snímek.

3.5 OCR algoritmus

Dále pohovoříme o problematice týkající se korektní implementace konkrétního OCR algoritmu a záležitostí s tím spojených jako je jeho stabilita apod.

Předem je vhodné zmínit, že vzhledem k rozsáhlé složitosti OCR algoritmu bude popis konkrétního algoritmu velmi omezen a pozornost bude zaměřena především na jeho korektní implementaci v koncové aplikaci.

3.5.1 Výběr vhodného algoritmu

Jako první se zaměříme na výběr samotného použitého OCR algoritmu. Obecně lze totiž zjistit, že OCR algoritmů ve volné distribuci existuje celá řada, nicméně každý z nich na jiné úrovni co se týče kvality a možností implementace v naší koncové aplikaci.

Do nejužšího výběru se nakonec dostali celkem tři největší zástupci zmíněných volně dostupných OCR platforem a to tedy:

- *Tesseract-ocr v3.02.02*
- *GOCR v0.50*
- *Ocrad v0.24*

Tyto výše uvedené platformy byly následně postupně podrobně odzkoušeny pomocí identických testovacích sekvencí/předloh v PC pomocí příkazové řádky. Samotné výsledky tohoto testování nejsou v této dokumentaci uvedeny, protože obecně nemají žádnou informační hodnotu vztahující se k této práci a současně si lze porovnání provést velmi jednoduše provést na libovolné předloze samostatně. Více informací o použití těchto jednotlivých algoritmů v PC lze nalézt např. zde [18].

Podle výše zmíněného testování s přihlédnutím na možnosti implementace byl nakonec zvolen algoritmus *GOCR v0.50*. Obecně vzato nutno zmínit, že algoritmus *GOCR* nedosáhl nejlepších výsledků z hlediska rozeznávání znaků, nicméně jeho nespornou výhodou je jeho jednoduchost a přehlednost, čemuž napomáhá i fakt, že tento algoritmus je kódován kompletně v programovacím jazyce C. Více informací o tomto algoritmu lze nalézt na oficiálních webových stránkách zde [19].

Na základě testování lze obecně zmínit několik základních faktů, o které se úspěšné optické rozeznávání opírá. Zde je vhodné dodat, že všechny výše uvedené algoritmy jsou navrženy pro podobné účely, jak bylo uvedeno v kap. 1.2, tedy např. digitalizace knih, dokumentů apod. Toto znamená, že algoritmy a priori předpokládají černý text, v našem případě číslice, na bílém pozadí a správnou orientaci textu. Tento fakt je nutné ve výsledné implementaci nekompromisně vzít v potaz a tedy vstupní data patřičně upravit podle aktuálně snímaného textu.

3.5.2 Implementace GOCR algoritmu

V této fázi, když již víme, jaký konkrétní algoritmus budeme v koncové aplikaci používat, tak se můžeme pustit do jeho přímé implementace na používaný MCU.

Algoritmus *GOCR* primárně pracuje jako standardní zásuvný modul, kdy je tomuto modulu zpřístupněn určitý vstupní obsah, v tomto případě určitá obrazová data a jeho výstup je vyveden do příkazové řádky nebo např. do textového souboru. Z hlediska implementace lze tedy obecně říct, že v našem případě se bude jednat převážně o úpravy, které budou zmíněné vstupní a výstupní fáze simulovat.

Prvním krokem je tedy úplně odstranění, popř. pouhé vynechání programového kódu, který je spojen s nahráváním obrazových snímků. V případě algoritmu *GOCR* se jedná o nahrávání libovolného snímku v PNM formátu a jeho dekódování. Po odstranění této části algoritmu je nyní možné předávat vstupní data manuálně jako ukazatel na standardní *char*/*uint8_t* pole. Datový typ tohoto pole již může napovědět, že součástí čtení PNM formátu byla i úprava těchto dat. Je tedy nutné získaná data z kamery ještě patřičně upravit a to konkrétně provést konverzi z formátu RGB565, který je získáván kamerou na formát greyscale.

Programový kód realizující tuto konverzi poté vypadá následovně:

```
// Converting RGB565 to grayscale
i=0;
for (n = 0; n < IMG_PX; n++) {
   // - RGB565
   r = (uint8_t) ((lcd_buffer >> 11) & 0x1F);
   g = (uint8_t) ((lcd_buffer >> 5) & 0x3F);
   b = (uint8_t) (lcd_buffer & 0x1F);

   // - RGB888
   r = ((r * 527) + 23) >> 6;
   g = ((g * 259) + 33) >> 6;
   b = ((b * 527) + 23) >> 6;

   // - Greyscale (luminosity algorithm)
   img_buffer[i]=(char)(0.2126*(float)r+0.7152*(float)g+0.0722*(float)b);
   i++;
}
```

V programovém kódu můžeme vidět typické části spojené s obrazovou konverzí, jako je rozdělení pixelů na jednotlivé barvy (R, G, B), jejich úpravu a následný převod na greyscale formát. Zde je taktéž možné vidět, že je použita tzv. luminosity metoda, která nejlépe zvýrazňuje černou barvu, v našem případě text a potlačuje pozadí snímku. Více informací o této metodě lze nalézt např. zde [20]. Vhodné je taktéž dodat, že v této fázi je nutné zajistit zpětnou konverzi snímku na big-endian formát, opět provést ořez snímku na zvolené rozlišení a přeskládat data podle zvolené orientace snímku.

Pokud tedy pomineme výše uvedenou problematiku natočení textu, jeho barvu apod., tak lze říci, že v této fázi je vše připraveno pro předání ukazatele na pole obrazových dat a výsledné rozeznávání. Nyní tedy můžeme přejít k modifikaci výstupní fáze algoritmu. Zde je opět nutné úplně odstranit, popř. vynechat programovou část realizující zpracování výstupních dat a nahradit jí vlastním zpracováním. Toto zpracování se skládá pouze ze dvou jednoduchých částí, kde první pouze kopíruje obsah výstupních dat na zvoleného libovolného pole představující textový buffer a druhá odstraní všechny irelevantní znaky, jako jsou např. mezery.

Tímto způsobem jsme tedy postupně upravili zvolený algoritmus *GOCR* tak, aby zpracovával námi požadovaná obrazová data a následně své výsledky uložil do uživatelského bufferu. Obsah tohoto bufferu lze následně převést na standardní číslo, které lze poté porovnávat, ukládat apod.

Nakonec je ještě vhodné alespoň okrajově pohovořit o uživatelském nastavení použitého *GOCR* algoritmu. Toto nastavení obecně není přísně nutné provádět, protože velká část zpracování se provádí dynamicky, nicméně pro naše specifické podmínky je vhodné několik úprav provést. Samotné nastavení algoritmu se v případě spouštění z příkazové řádky nastavuje pomocí jednotlivých atributů daného příkazu, nicméně v našem případě je nutné nastavení provést manuálně.

Toto manuální nastavení poté z hlediska programového kódu vypadá následovně:

```
/* init cfg */
job->cfg.cs = 0;
job->cfg.spc = 0;
job->cfg.mode = 32;
job->cfg.dust_size = 1;
job->cfg.only_numbers = 0;
job->cfg.verbose = 0;
job->cfg.out_format = UTF8;
job->cfg.lc = "";
job->cfg.db_path = (char*) NULL;
job->cfg.cfilter = "0123456789";
job->cfg.certainty = 90;
job->cfg.unrec_marker = "";
```

Zde z programového kódu výše lze vidět nejdůležitější atributy jako je např. filtr rozeznávaných znaků, deaktivaci prachového filtru nebo stupeň shody, kdy je znak korektně rozpoznán. Pro lepší porozumění všech atributů je možné se obrátit zde [21].

3.5.3 Rozšíření paměťového prostoru

Závěrem této podkapitoly je nutné pohovořit o problematice týkající se rozšíření systémového paměťového prostoru.

Při aplikaci námi modifikovaného *GOCR* algoritmu totiž velmi rychle zjistíme, že i v případě korektní implementace celého algoritmu dochází v určitých případech fáze rozeznávaní znaků k přetečení paměťové haldy do oblasti zásobníku. Tento fakt vede k ukončení programu, tedy k odskočení do nekonečné smyčky, a je způsoben mnohonásobnými alokacemi paměti uvnitř algoritmu. Jediným způsobem, jak situaci řešit je tedy rozšířením paměťového prostoru.

Tento paměťový prostor se rozšíří právě o určenou oblast v externí SDRAM paměti, která byla zmíněna v kap. 3.2. K systémové paměti se tedy připojí celkem 6 *MB* volného prostoru, který může být algoritmem a obecně systémem využit.

Nicméně jak reálně tuto externí paměť připojit se může stát mírně obtížné. Pouhým přidáním/modifikací paměti ve skriptu pro linker není možné dosáhnout žádného reálného výsledku vlivem hodnot fyzických adres v paměti a systémového zpracování paměťových alokací. Jednoduše řečeno pokud je zásobník umístěn od konce základního paměťového prostoru, tak kdykoliv by došlo k zápisu do externí SDRAM paměti, která je adresou mnohonásobně výše, tak by si systém tuto situaci interpretoval opět jako přetečení. Jedinou alternativou je tedy přímo změna systémového zpracování paměťových alokací, která je volána funkcemi typu alloc apod.

Výsledná změna systémového zpracování paměťových alokací poté z hlediska programového kódu vypadá následovně:

```
caddr_t _sbrk(int incr) {
    static char *heap_end;
    static char *sdram_start = (char*) SDRAM_MEM_START;
    static char *sdram_end = (char*) SDRAM_LCD_START1;
    char *prev_heap_end;

    // Modify system memory management
    if (heap_end == 0) {
        heap_end = sdram_start;
    }
    prev_heap_end = heap_end;

    if (heap_end + incr > sdram_end) {
        return  (caddr_t) -1;
    }

    heap_end += incr;
    return (caddr_t) prev_heap_end;
}
```

Z uvedeného kódu výše můžeme velmi přehledně vidět, že celá oblast haldy je přesunuta do externí SDRAM paměti, kde dojde k přetečení až v případě, že by tato oblast začala zasahovat do oblasti prvního LCD bufferu. V principu se tedy úplně nejedná o rozšíření stávajícího paměťového prostoru, nicméně oblast zásobníku stále zůstává v základní RAM paměti.

3.6 Hodiny reálného času

Dále se zaměříme na funkčnost podpůrných funkcí, které bude výsledný prototyp snímacího systému využívat. Jednou z těchto funkcí jsou standardní hodiny reálného času neboli real time clock (RTC) o kterých bude pojednávat tato podkapitola.

3.6.1 Implementace RTC

Problematiku implementace RTC lze v případě použité vývojové desky jednoduše řešit použitím interního RTC obvodu, který obecně disponuje mnoha funkcemi, nicméně v našem případě nás bude zajímat pouze několik z nich.

Konkrétně budeme v koncové aplikaci využívat standardní funkci hodin pro zachování informace o systémovém čase, schopnost probuzení MCU z módu nízké energetické spotřeby pomocí tzv. RTC alarmu a nakonec záložní RTC registry, které budou popsány dále v textu. Zmíněný alarm obecně pracuje velmi intuitivně a to tak, že pokud dojde k dosažení hodnoty hodin shodné s hodnotou zapsanou v tomto alarmu, tak je vyvoláno přerušení od RTC hodin a v případě, že je toto přerušení napojeno na externí přerušení neboli EXTI, tak dojde i k probuzení MCU.

Samotné použití a implementace RTC periferie vývojové desky je obecně dosti komplikovanou záležitostí a proto je vodné se opět obrátit minimálně na ukázkové příklady uváděné výrobcem nebo na libovolnou podpůrnou knihovnu. Kompletní programový kód navrženého prototypu lze samozřejmě nalézt v digitální příloze, nicméně nyní je vhodné pohovořit o několika dalších záležitostech, které se k použití RTC vážou.

Při implementaci interního RTC obvodu vývojové desky je obecně možné zvolit jeden ze dvou zdrojů oscilátoru. Konkrétně lze využít buď interní kalibrovaný RC oscilátor (LSI) nebo standardní krystalový externí oscilátor (LSE). Oba samozřejmě o výchozím kmitočtu 32768 Hz.

Pokud použijeme primární alternativu interního LSI oscilátoru, tak postupným testováním velmi rychle zjistíme, že i v případě údajné kalibrace tohoto oscilátoru dochází k značným odchylkám RTC hodin, což činí jeho použití jako velmi diskutabilní. Tato nepřesnost může představovat odchylku až několik sekund připadající na každou minutu RTC hodin. Obecně je tedy nutné přistoupit na druhou alternativu, tedy použití externího oscilátoru.

Zde ovšem narazíme na skutečnost, že externí oscilátor není součástí standardní použité vývojové desky 32F429IDISCOVERY. Logickým důvodem je jistě cena tohoto oscilátoru v porovnání s cenou právě interního RC oscilátoru. Nicméně i v případě, že oscilátor není na vývojové desce osazen, je možné velmi rychle zjistit, že vývojová deska je z hlediska HW na tuto alternativu připravena a oscilátor lze tedy velmi jednoduše dodat. Více informací jak externí oscilátor dodat neboli, jak vývojovou desku z hlediska HW upravit je možné nalézt v jejím manuálu zde [5], kde lze nalézt i odpovídající schéma zapojení.

3.6.2 Záložní RTC registry

Velmi důležitou problematikou z hlediska funkce výsledného prototypu je použití záložních RTC registrů. Tyto záložní registry jsou totiž obecně jediným místem, kde je možné uložit libovolná data tak, aby byly dostupná i v případě, že dojde k uvedení MCU do módu nízké energetické spotřeby a následnému probuzení.

Jedná se celkem o 20 záložních 32 bitových registrů od adresy 0x40002850. K těmto registrům lze přistupovat standardně jako k libovolnému paměťovému obsahu, nicméně je doporučeno pro tyto účely využívat uživatelské funkce definované v použitých standard peripheral libraries (SPL) knihovnách, které umožní jednoduchý přístup pomocí logických adres 0 až 19, popř. 0x00 až 0x13.

V naší aplikaci jsou všechny výše zmíněné záložní registry hojně využívány a proto je vhodné je patřičně popsat. Nutno ovšem dodat, že struktura jednotlivých registrů je dána postupným vývojem a nemusí se tedy jednat o rozložení nejvhodnější.

Nejdůležitějším RTC registrem je registr s pořadovým číslem 0, který v naší aplikaci slouží jako systémový zálohovací registr, kde jsou uloženy všechny informace týkající se uživatelského nastavení prototypu apod. Význam jednotlivých bitů tohoto registru lze vidět v tabulce Tab. 1 níže.

Tab. 1: Ilustrace využití jednotlivých bitů registru č. 0

31 30 29 28 27 26 25 24 23 22	21 20 19 18 17 16	15 14 13 12	11 10 9 8	7	6	5	4	3	2	1	0
Maximální inkrement měření	Interval měření	Výška ořezu	Šířka ořezu	x	x	Přetečení měření	OCR chyba	Inicializace měření	Barva písma	Orientace	LED přisvícení

Vysvětlení jednotlivých bitů z Tab. 1 výše lze rozvést jako:

- Bit 0 – nastavení použití LED přisvícení při záznamu kamerou
- Bit 1 – orientace obrazu pro zpracování OCR algoritmem
- Bit 2 – barva písma pro zpracování OCR algoritmem
- Bit 3 – informace o inicializaci automatického měření (první měření není testováno na přetečení s předchozí hodnotou)
- Bit 4 – zaznamenána chyba při minulé aplikaci OCR v automatickém režimu
- Bit 5 – zaznamenáno přetečení hodnoty při minulém měření v automatickém režimu (podle nastaveného maximálního inkrementu)
- Bit 8 až 11 – koeficient ořezu šířky obrazu (představuje násobek hodnoty 20 *px*, který je poté odečítán od šířky QVGA rozlišení)
- Bit 12 až 15 – koeficient ořezu výšky obrazu (představuje násobek hodnoty 20 *px*, který je poté odečítán od výšky QVGA rozlišení)
- Bit 16 až 21 – koeficient intervalu automatického měření (představuje násobek 5 *min*, který je poté přičítán k základové hodnotě 5 *min* a je omezen na maximum 59, tedy max. interval 300 *min*)
- Bit 22 až 31 – maximální možný inkrement hodnoty měření (tato hodnota je přičítána k základové hodnotě 1 a je omezena na max. 999, tedy max. možný inkrement 1000)

Další použité registry jsou již mnohem jednoduššího charakteru. Konkrétně registry s pořadovým číslem 1 až 9 slouží pouze k uložení předchozích hodnot měření manuálního režimu a registry s pořadovým číslem 10 až 18 slouží pro obdobný účel, ovšem pro režim automatický. Každá hodnota, která je tedy v rámci měření uložena je zapsána do individuálního záložního RTC registru. Tento způsob je mírně neefektivní, nicméně nelze předem zaručit, jak velkou hodnotu bude snímací systém snímat, takže je třeba selektovat vždy jeden celý registr.

Nakonec poslední registr s pořadovým číslem 19 slouží jako identifikace inicializace pro samotnou programovou RTC knihovnu. Zde je jednoduše zapsána testovací hodnota, kterou se ověřuje funkčnost přístupu k RTC registrům a zároveň slouží pro identifikaci stavu systému v případě opětovné inicializace RTC periferie po probuzení prototypu apod.

3.7 Datové záznamy

Další podpůrnou funkcí výsledného prototypu je problematika datových záznamů. Z přechozí kapitoly je již jasné, že pokud dojde k úspěšnému měření, tak je hodnota tohoto měření uložena do specifikovaného RTC registru, nicméně právě tento způsob logování přináší celou řadu úskalí.

Zde pouze jednoduše rozvedeme, že tato záloha do RTC registrů probíhá jednoduše jako doplňování naměřených hodnot do paměti (podle daného režimu), kde je vždy poslední zaznamenaná hodnota ztracena. Tento způsob uchovávání hodnot je pro potřeby validace měření, přibližnou vizualizaci výsledků apod. dostačující, nicméně z hlediska dlouhodobé analýzy a obecně z pohledu statistického sběru dat nevyhovující. Nutno dále dodat, že tyto hodnoty jsou zaznamenávány pouze v čisté formě, tedy že k nim není nijak kódován čas, kdy byly zaznamenány apod.

Pro tyto náročnější požadavky byla tedy zvolena vhodnější alternativa datových záznamů a to konkrétně implementace hojně využívaných flash pamětí. Tuto skutečnost již bylo možné vypozorovat podle návrhu základové DPS prototypu, kde lze vidět pouzdro pro použitou microSD paměťovou kartu. Informace ohledně zapojení tohoto pouzdra lze samozřejmě nalézt v příloze zde A.1.

Samotná část zajišťující zpracování dat na microSD, popř. SD kartu je řešena pomocí file allocation table (FAT) souborového modulu pro embedded systémy *FatFs*. Více o tomto modulu lze nalézt na oficiálních webových stránkách zde [22].

3.7.1 Komunikace s microSD kartou

Pokud hovoříme obecně o problematice komunikace se standardní SD kartou, tak lze tuto záležitost řešit několika možnými způsoby. Ovšem pro potřeby této aplikace, kdy se záznamy skládají vždy pouze z několika málo údajů, které jsou zapisovány do textového souboru lze použít způsob nejjednodušší a to tedy opět komunikaci pomocí SPI sběrnice.

Pokud opomeneme konfiguraci chip select (CS) pinu, tak můžeme opět přejít přímo ke konfiguraci SPI sběrnice. Zde byla vybrána jedna kombinace ze šesti dostupných SPI sběrnic podle obecné HW dostupnosti na následujících pinech:

- PC10 – SPI3_SCK
- PC11 – SPI3_MISO
- PC12 – SPI3_MOSI

Další konfigurace již probíhá stejným způsobem jako v případě SPI komunikace s řadičem LCD displeje. První je tedy nutné opět provést HW konfiguraci I/O pinů jako výstupní typu push-pull, zrušení použití pull-up nebo pull-down rezistorů a nastavení alternativní funkce těchto pinů. Poté je již možné přímo přejít ke konfiguraci SPI periferie, která z hlediska programového kódu vypadá následovně:

```
// SPI config
SPI_InitStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
SPI_InitStruct.SPI_Mode = SPI_Mode_Master;
SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;
SPI_InitStruct.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStruct.SPI_CPHA = SPI_CPHA_1Edge;
SPI_InitStruct.SPI_NSS = SPI_NSS_Soft | SPI_NSSInternalSoft_Set;
SPI_InitStruct.SPI_BaudRatePrescaler = prescaler;
SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_Init(SPI3, &SPI_InitStruct);
SPI_Cmd(SPI3, ENABLE);
```

Z výše uvedené konfigurace lze již opět vidět všechny potřebné HW parametry, které jsou pro samotný přenos klíčové, jako je např. pořadí bitů apod. V této fázi je tedy možné přejít přímo k implementaci *FatFs* modulu.

3.7.2 Implementace FatFs modulu

Samotná implementace *FatFs* modulu je obecně jednoduchou záležitostí, protože tento modul/algoritmus je velmi rozšířený, univerzální a spolehlivý nástroj pro libovolnou komunikaci s paměťovými médii. V našem případě se jedná o komunikaci s microSD paměťovou kartou podle souborového systému FAT16 nebo FAT32.

Pro naše potřeby, mimo samotnou inicializaci a řízení komunikace, byla vytvořena pouze jediná funkce, která vykonává jednoduché připojení požadovaného textu/řetězce k stávajícímu textu ve zvoleném textovém souboru. Výsledný programový kód této funkce vypadá následovně:

```
uint8_t FATFS_append_str(const TCHAR* str_path, const TCHAR* str){
    FATFS FatFs;
    FIL fil;
    FRESULT res;

    // Mount SD card
    res = f_mount(&FatFs, "", 1);
    if (res == FR_OK){

        // Open/Create file and append string
        res = open_append(&fil, str_path);
        if (res == FR_OK){
            f_puts(str, &fil);
            // Close opened file
            f_close(&fil);
            return 0;
        }
        // Unmount SD card
        f_mount(0, "", 1);
    }

    // Error
    return 1;
}
```

Zde z programového kódu výše můžeme velmi přehledně vidět všechny kroky, které algoritmus postupně provádí, tedy od připojení karty až po zapsání zvoleného řetězce. V případě, že nastane v některém kroku k chybě, tak je vygenerován chybový stav, což je interpretováno jednoduše jako navrácení hodnoty 1.

Je nutné ovšem dodat, že v případě použitého microSD pouzdra na základové DPS není součástí pin detekující přítomnost paměťové karty. Tento řídící signál je ovšem ve výchozím stavu *FatFs* modulu povolen a nebylo by tedy možné provádět žádnou komunikaci. Tuto část programového kódu stačí pouze jednoduše zakázat, popř. úplně odstranit, aby neblokovala potenciální komunikaci.

3.8 Uživatelské ovládání prototypu

Dále se zaměříme na problematiku týkající se uživatelského ovládání navrženého snímacího systému. Samotná vývojová deska, pokud opomeneme možnosti externích ovládacích periférií, nabízí obecně dvě možnosti, jak prototyp ovládat nebo řídit.

V našem případě využijeme obě tyto periferie, tedy uživatelské tlačítko i rezistivní dotykovou vrstvu LCD displeje. V této podkapitole se tedy zaměříme na jejich korektní nastavení, specifický popis a vysvětlení jakou mají ve výsledné aplikaci funkci.

3.8.1 Uživatelské tlačítko

Jako první se zaměříme na uživatelské tlačítko. Tlačítka obecně lze obsluhovat mnoha způsoby, nicméně každý z nich se hodí na jinou aplikaci. V našem případě bude uživatelské tlačítko sloužit k dvěma důležitým funkcím.

První funkcí je probouzení MCU z režimu nízké spotřeby a druhou funkcí je implementace domovského tlačítka, které při stisku uživatele vrátí vždy na původní domovskou obrazovku. Z výčtu těchto funkcí lze usoudit, že toto tlačítko tedy nebude aktivováno velmi často a není nutné nijak složitě řešit jeho mechanické zákmity apod. Jako obsluhu tlačítka je tedy možné zvolit jednoduché externí přerušení.

Pokud hovoříme o externím přerušení z libovolného HW pinu tak je nejprve nutné si ujasnit, který konkrétní pin bude jako vstup použit. V případě uživatelského tlačítka je tato problematika přímočará, protože toto tlačítko je zapojeno na pin PA0.

Na základě vybraného pinu je poté možné aktivovat korespondující linku externího přerušení, tedy EXTI_Line0. Samozřejmostí je ovšem opět korektní HW konfigurace I/O pinu, ovšem tentokrát jako vstupní a připojením pull-down rezistoru. Následná konfigurace přímo externího přerušení vypadá následovně:

```
// EXTI interrupt config
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
EXTI_InitStruct.EXTI_Line = EXTI_Line0;
EXTI_InitStruct.EXTI_LineCmd = ENABLE;
EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_Init(&EXTI_InitStruct);
```

Samozřejmostí je ovšem opět nastavení tohoto externího přerušení v rámci NVIC řadiče přerušení jinak může být přerušení nefunkční.

3.8.2 Rezistivní dotyková vrstva

Dalším a v případě naší aplikace hlavním ovládacím prvkem navrženého prototypu snímacího systému je rezistivní dotyková vrstva LCD displeje. Tato dotyková vrstva je obsluhována řadičem STMPE811 výrobce *STMicroelectronics*.

Tento řadič obecně zpracovává informace o mechanickém dotyku na standardní čtyřvodičové rezistivní vrstvě a tuto informaci poté odesílá přímo jako souřadnice pomocí I2C nebo SPI sběrnice k dalšímu zpracování. V našem případě opět z důvodu HW zapojení použité vývojové desky lze použít pouze alternativu komunikace pomocí I2C sběrnici a to na následujících pinech:

- PA8 – I2C3_SCL
- PC9 – I2C3_SDA

Prvním krokem je tedy opět již typická HW inicializace těchto použitých I/O pinů a jejich korektní konfigurace jako výstupní typu push-pull, zrušení použití interních pull-up nebo pull-down rezistorů, které jsou již realizovány externě na použité vývojové desce a samozřejmě nastavení alternativní funkce těchto pinů. Poté konfigurace přímo I2C sběrnice z hlediska programového kódu vypadá následovně:

```
// I2C config
I2C_DeInit(I2C3);
I2C_InitStruct.I2C_ClockSpeed = clockSpeed;
I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;
I2C_InitStruct.I2C_OwnAddress1 = 0x00;
I2C_InitStruct.I2C_Ack = I2C_Ack_Disable;
I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2;
I2C_Init(I2C3, &I2C_InitStruct);
I2C_Cmd(I2C3, ENABLE);
```

Zde z programového kódu výše můžeme opět vidět parametry samotného I2C přenosu, nicméně pro požadovanou funkčnost dotykové vrstvy ještě musíme opět dodržet specifikace jejího řadiče STMPE811. Samozřejmě opět mluvíme o korektní inicializaci, která je v tomto případě mírně obsáhlejší v porovnání např. s inicializací řadiče LCD displeje, nicméně opět se dá vycházet z určitých příkladů nebo použít předlohu z FW balíčku zde [8]. Více informací o tomto řadiči lze popř. nalézt zde [23] a současně je vhodné věnovat pozornost zapojení vývojové desky zde [5].

Pouze okrajově se ještě zmíníme o tom, jak bude výsledná funkce dotykové vrstvy v systému obecně organizována. Jedním a samozřejmě nejjednodušším ze způsobů, jak je možné odezvu dotykové vrstvy zjišťovat je kontinuálním dotazováním se, zda byl detekován dotek uživatelem. Tento způsob je ovšem velmi neefektivní a proto je vhodnější využít opět možnost externího přerušení od zmíněného řadiče.

Použitý řadič dotykové vrstvy totiž obsahuje mimo samotnou I2C sběrnici taktéž jeden externí vývod, který právě může informovat MCU o tom, že nastal potenciální dotyk uživatelem. Nutno ovšem dodat, že tuto signalizaci je nutné povolit pomocí korektní konfigurace v rámci inicializace řadiče a současně je nutné zajistit nulování flagů přerušení tohoto řadiče, které tuto externí signalizaci ovládají. Samotný vývod této signalizace je HW napojen na pin PA15 použité vývojové desky. Tímto způsobem lze tedy funkci dotykové vrstvy v systému značně zjednodušit a to konkrétně tak, že pokud dojde k dotyku uživatele na displeji, tak bude vyvoláno externí přerušení, kde se MCU dotáže, na jakých souřadnicích k danému dotyku došlo. Následné zpracování poté záleží na konkrétní situaci.

Nakonec tedy pohovoříme opět o konfiguraci jednoduchého externího přerušení, které se bude samozřejmě podobat konfiguraci externího přerušení v předchozí podkapitole. Jediným rozdílem bude obecně pouze použitá linka externího přerušení podle výše zmíněného pinu, tedy EXTI_Line15 a současně změna detekované hrany na sestupnou, protože jak můžeme ze zapojení použité vývojové desky zde [5] vidět, tak na tento vývod je již připojen pull-up rezistor stejně jako v případě I2C sběrnice. Následná konfigurace přímo externího přerušení vypadá tedy následovně:

```
// EXTI interrupt config
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource15);
EXTI_InitStruct.EXTI_Line = EXTI_Line15;
EXTI_InitStruct.EXTI_LineCmd = ENABLE;
EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
EXTI_Init(&EXTI_InitStruct);
```

Samozřejmostí je ovšem opět nastavení tohoto externího přerušení v rámci NVIC řadiče přerušení jinak může být přerušení nefunkční.

3.9 Energetická správa prototypu

Dále je vhodné se zaměřit na dnes velmi populární problematiku embedded systémů a to konkrétně energetickou správu a nížení spotřeby navrženého prototypu.

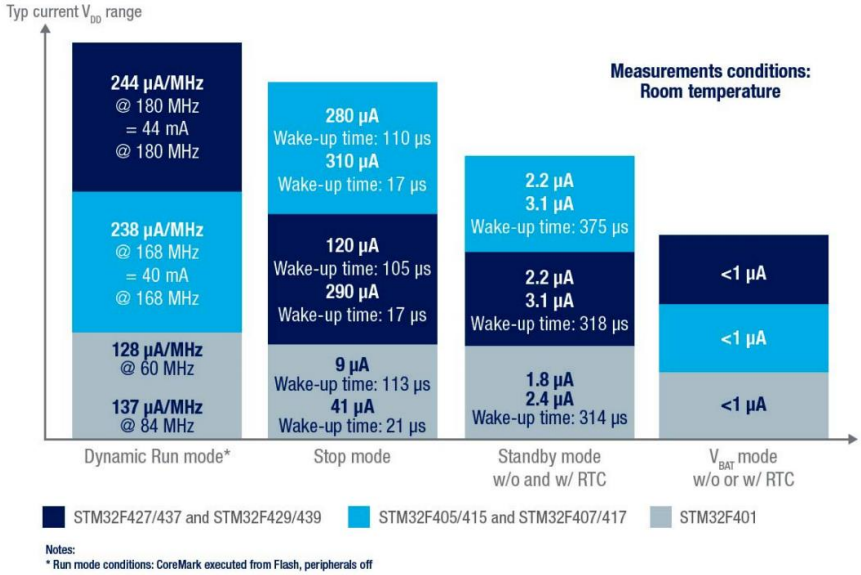
Tato problematika se obecně vždy stává velmi atraktivní až nezbytnou v případě, kdy dané zařízení je nebo může být potenciálně napájeno bateriově. Zde je tedy logické, že nároky na celkovou minimalizaci energetické spotřeby jsou mnohem vyšší.

Obecně lze snížení energetické spotřeby řešit mnoha způsoby, nicméně dominantními přístupy jsou buď podtaktování daného MCU nebo využitím určitých nízkoenergetických módu daného HW. Tyto módy většinou zahrnují podobný způsob podtaktování nebo vypínání, popř. uspání nepoužívaných periferií apod.

V našem případě z hlediska rychlosti zpracovávání obrazových dat, OCR algoritmu a převážně z hlediska stability FW necháme MCU pracovat na taktu 168 MHz, který bude neměnný a samotné omezení spotřeby budeme regulovat pomocí zmíněných nízkoenergetických módů a vypínáním nepoužívaných periférií.

3.9.1 Implementace nízkoenergetických módů

Samotná použitá vývojová deska nabízí celkem tři nízkoenergetické módy, které lze libovolně používat, nicméně každý se obecně hodí na jinou aplikaci. Přehled těchto módu je možné vidět na Obr. 3.1 níže.



Obr. 3.1: Přehled nízkoenergetických módů MCU STM32 řady F4 [24]

Z Obr. 3.1 výše můžeme vidět, že pro naši aplikaci by bylo tedy nejlepší použití tzv. V_{BAT} módu, který obecně zajišťuje napájení pouze RTC obvodu, kde všechny ostatní periferie jsou deaktivovány. Nicméně pokud se obrátíme na libovolnou dokumentaci týkající se MCU STM32 řady F4, tak rychle zjistíme, že MCU přejde do tohoto módu pouze v případě výpadku hlavního napájení, ovšem za předpokladu, že je zachováno napájení na V_{BAT} pinu např. záložní baterií. Obecně lze tedy říct, že tento způsob snížení spotřeby je pro náš případ nevhodný a dokonce nerealizovatelný, protože použitá vývojová deska je z hlediska HW zapojení navržena tak, že má pouze jediné společné napájení, které je spojeno pro celou tuto desku. Více informací o tomto HW zapojení je samozřejmě možné najít zde [5].

Naše problematika se tedy omezila na použití druhého nejúspornějšího módu a to konkrétně na použití Standby módu. Tento mód zahrnuje deaktivaci všech vnitřních hodinových signálu kromě funkce RTC obvodu, deaktivaci všech periferií jako je samotný MCU, SDRAM apod. Jediná data, která lze tedy po přechodu z tohoto módu obnovit jsou data, která byla uložena ve zmiňovaných záložních RTC registrech.

Samotný přechod do tohoto módu je velmi jednoduchý a z hlediska programového kódu vypadá následovně:

```
// Clear standby/wakeup flag
PWR_ClearFlag(PWR_FLAG_SB | PWR_FLAG_WU);

// Enable wakeup pin (BUTTON = PA0)
PWR_WakeUpPinCmd(ENABLE);

// Go to standby mode
PWR_EnterSTANDBYMode();
```

Zde z programového kódu výše můžeme velmi přehledně vidět pouze vynulování flagů signalizující probuzení MCU, aktivaci uživatelského tlačítka jako potenciální zdroj pro budoucí probuzení a nakonec přímo přechod do Standby módu. Následné probuzení z tohoto módu lze poté provést několika libovolnými událostmi, nicméně v našem případě budeme používat buď výše uvedené uživatelské tlačítko nebo RTC alarm, jak bylo zmíněno v kap. 3.6.1.

3.9.2 Deaktivace nepotřebných periferií

Jak již bylo naznačeno v předchozí podkapitole, samotný přechod MCU do Standby módu nezaručuje deaktivaci dalších komponent, kterými je výsledný prototyp osazen a v rámci snižování spotřeby je tedy nutné před vstupem do tohoto módu tyto nepotřebné periferie deaktivovat.

V našem případě se jedná o několik málo periférií, jako jsou řadič displeje, řadič rezistivní dotykové vrstvy, modul kamery apod. Obecně lze ovšem říci, že největší spotřebu lze pozorovat u modulu použité kamery, podsvícení LCD displeje a LED přisvícení. Tyto periferie je tedy nutné před přechodem do Standby módu nekompromisně deaktivovat.

Zde ovšem narazíme na problém u zmiňovaného podsvícení LCD displeje. Toto podsvícení je HW zapojeno přímo na hlavní napájení a kdykoliv je tedy vývojová deska a obecně celý prototyp napájen, tak je displej rozsvícen. Konfigurací řadiče displeje taktéž není možné toto podsvícení displeje deaktivovat, takže jedinou alternativou je tedy HW úprava vývojové desky.

Samotná úprava se jednoduše skládá z přerušení napájecího spoje k anodám LED podsvícení displeje a vyvedením tohoto spoje ven od LCD displeje. Konkrétně se jedná o vodič č. 44 na plochém spojovacím kabelu mezi vývojovou deskou a LCD displejem. Samotné externí vyvedení tohoto vodiče je provedeno jednoduchým spojem vodiče na anodu jedné z LED. Tento externí vývod je následně možné přímo připojit na napájecí napětí, nicméně pro tyto potřeby je již připravena základová DPS, kde lze podsvícení jednoduše ovládat pomocí MCU. Více informací o zapojení lze nalézt ve schématu vývojové desky zde [5] nebo zapojení základové DPS v příloze zde A.1.

3.9.3 Informace o stavu baterie

Další problematikou, která má určitou návaznost na energetickou správu prototypu je část pojednávající o měření aktuálního stavu baterií. Tato informace je z hlediska navrženého prototypu čistě uživatelskou záležitostí, takže pouze informuje uživatele o nutnosti výměny baterie apod.

Samotné baterie, jak již bylo uvedeno v kap. 2.4, jsou upevněny k základové DPS a slouží k napájejí navrženého prototypu v případě, že není dostupný žádný jiný externí napájecí zdroj. Napětí těchto baterií je měřeno standardně pomocí 12 bitového interního analogově digitálního převodníku (ADC), který je součástí MCU.

Vzhledem ke skutečnosti, že baterie mohou, co se týče výše jejich napětí přesahovat napájení použité vývojové desky, které je mj. použito i pro napěťovou referenci ADC, tak je nutné tyto baterie zapojit minimálně přes napěťový dělič. Zde opět narážíme na problematiku energetické spotřeby, protože klasický dělič je ve skutečnosti pouze zdroj kontinuálního svodového proudu, což představuje energetické ztráty. V našem případě ovšem můžeme využít skutečnosti, že nepotřebujeme měřit napětí na bateriích kontinuálně a proto zde můžeme využít děliče s velmi vysokou impedancí v kombinaci s malou paralelní kapacitou a nemusíme mít obavu, že nepřiměřeně dynamicky zatížíme daný ADC kanál. Samotné zapojení děliče je možné vidět standardně ve schématu zapojení v příloze zde A.1.

Je nutné pouze zajistit, aby spoj mezi napěťovým děličem a ADC převodníkem byl přímý bez dalšího aditivního HW, který by naše měření zkresloval. Tato skutečnost může být mírně obtížná právě v případě použité vývojové desky, protože je hustě osazena jinými komponentami. Přesto lze ovšem k tomuto účelu nalézt několik pinů jako např. pin PA5, kam je zmíněný dělič napojen.

Samotná korektní konfigurace ADC opět předpokládá korektní HW konfiguraci použitého pinu a to konkrétně nastavení I/O pinu jako analogový vstup a připojení interního pull-down rezistoru. Následná konfigurace použitého ADC kanálu poté z hlediska programového kódu vypadá následovně:

```
// ADC common config
ADC_CommonInitStruct.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
ADC_CommonInitStruct.ADC_Mode = ADC_Mode_Independent;
ADC_CommonInitStruct.ADC_Prescaler = ADC_Prescaler_Div2;
ADC_CommonInitStruct.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_15Cycles;
ADC_CommonInit(&ADC_CommonInitStruct);

// ADC init config
ADC_InitStruct.ADC_ContinuousConvMode = DISABLE;
ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStruct.ADC_ExternalTrigConv = DISABLE;
ADC_InitStruct.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
ADC_InitStruct.ADC_NbrOfConversion = 1;
ADC_InitStruct.ADC_ScanConvMode = DISABLE;
ADC_InitStruct.ADC_Resolution = ADC_Resolution_12b;
ADC_Init(ADC1, &ADC_InitStruct);
ADC_Cmd(ADC1, ENABLE);
```

Zde z programového kódu výše můžeme vidět všechny důležité parametry převodu jako je jeho rozlišení zvoleného ADC kanálu, zpoždění mezi jednotlivými datovými vzorky atd. Pro detailnější popis funkce a významu jednotlivých parametrů je vhodné se obrátit např. na referenční manuál zde [14].

V této fázi tedy máme převodník připraven k samotnému ADC převodu, nicméně ještě je nutné definovat funkci, která bude tento převod spouštět a zpracovávat jeho výstupní hodnotu. Samotný výstup této funkce je tedy podle zvoleného nastavení výše 12 bitová hodnota. Obecně tedy číselná reprezentace v rozsahu od 0x000 po 0xFFF představující hodnotu napětí od 0 V po zvolenou napěťovou referenci, v našem případě tedy po napájecí napětí 3,3 V.

Výsledná funkce realizující jednorázový převod poté z hlediska programového kódu vypadá následovně:

```
// Config ADC conversion
ADC_RegularChannelConfig(ADC1, ADC_Channel_5, 1, ADC_SampleTime_15Cycles);

// Start ADC conversion
ADC_SoftwareStartConv(ADC1);

// Wait until done
while(ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC)==RESET);

return ADC_GetConversionValue(ADC1);
```

Nakonec této podkapitoly je vhodné zmínit, že pro tuto a podobné aplikace ADC převodu je vhodné konfiguraci provádět experimentálně na známém zdroji napětí současně s multimetrem. Tímto způsobem se vyvarujeme možných chyb spojených s nepřiměřeně velkým dynamickým zatížením ADC vysokou impedancí apod.

3.10 Jádro systému

Nyní po popisu všech prototypem používaných periférií je na závěr možné pohovořit o samotném jádru systému, a jak jsou tedy tyto jednotlivé periferie skupinově ovládány, aby bylo docíleno požadované funkce.

Nutno ovšem dodat, že následující popis je vlivem složitosti celého systému pouze jeho zjednodušenou formou. Pro případný kompletní obraz funkce prototypu je nutné se obrátit na výsledný programový kód v digitální příloze.

3.10.1 Kooperativní multitasking

Pokud pomineme problematiku (re)inicializačních postupů vázaných na přechody mezi Standby a Run módem a obsluhu jednotlivých přerušení, která byla popsána v předchozích kapitolách, tak hlavním ovládacím mechanizmem navrženého prototypu je implementace velmi jednoduchého kooperativního multitaskingu.

Jedná se o standardní implementaci, kdy je aktivován tzv. Systick, tedy systémový čítač/časovač s periodou 1 ms (je možno zvolit i jinak). Tímto způsobem dochází každou milisekundu k odskoku do přerušení, kde inkrementována hodnota tzv. čítače tiků a současně v našem případě je přidána další hodnota sloužící pro přesnou zpožďovací blokující smyčku, která je dekrementována. Aktivace výše zmíněného Systicku je velmi jednoduchá a z hlediska programového kódu vypadá pouze jako jediný řádek následovně:

```
// Enable SysTick timer
SysTick_Config(SystemCoreClock/1000);
```

Následné přerušení poté vypadá z hlediska programového kódu následovně:

```
/* SysTick interrupt */
void SysTick_Handler(void) {
    tick_counter++;

    if (tick_delay != 0) {
        tick_delay--;
    }
}
```

V této fázi je již možné přejít přímo k implementaci samotného kooperativního multitaskingu. Pro potřeby tohoto multitaskingu je obecně nutné vytvořit skupinu lokálních proměnných (např. pole), kde jsou uloženy hodnoty jednotlivých provedených tiků pro každou vytvořenou cyklickou úlohu a současně funkci, která se bude dotazovat, zda již došlo k dovršení zvoleného intervalu, neboli zda má být daná úloha vykonána. Tato funkce z hlediska programového kódu vypadá následovně:

```
uint8_t Time_elapsed(uint32_t interval, volatile uint32_t *task) {
   uint32_t current_tick, current_task;

   current_tick = tick_counter;
   current_task = *task;
   // Variable non-verflow
   if (current_tick >= current_task){
       if ((current_tick - current_task) >= interval) {
           *task = current_tick;
           return 1;
       }
   }
   // Variable overflow
   else {
       if (((0xFFFFFFFF - current_task) + current_tick) >= interval) {
           *task = current_tick;
           return 1;
       }
   }

   return 0;
}
```

Zde z programového kódu můžeme vidět právě výše zmíněné dotazovaní se funkcí, zda došlo k uplynutí nastaveného intervalu pro danou úlohu. Současně je možné vidět ošetření přetečení proměnné čítače tiků, bez kterého by zmíněný multitasking přestal po určité době korektně vykonávat jednotlivé úlohy.

Tímto způsobem je tedy možné implementovat libovolný počet úloh se svým konkrétním cyklickým intervalem, které jsou poté součástí hlavní nekonečné smyčky a jsou vykonávány právě podle zvoleného intervalu. Samozřejmě je nutné vzít v potaz samotný princip funkce kooperativního multitaskingu, tedy pokud řízení programu není úlohou předáno zpět jádru, tak dojde k zamrznutí programu v dané úloze.

3.10.2 Kompletace systému

Nyní ve světle všech předchozích kapitol je již možné pohovořit celkové kompletaci systémového jádra a tím nastínit jak celý prototyp snímacího systému funguje.

Nejprve se tedy podíváme na výše popsanou implementaci multitaskingu a jednotlivé cyklické úlohy. V našem případě používáme celkem 4 jednoduché úlohy, které prakticky zajišťují chod celého programu. Některé z úloh by bylo možné řešit i jiným způsobem, nicméně právě implementace kooperativního multitaskingu mnoho programátorských situací značně zjednodušuje a to ne pouze pro cyklické operace. Samotné úlohy lze poté jednotlivě popsat jako:

- Úloha č. 1 – zobrazení nebo obnova domovské obrazovky
- Úloha č. 2 – obsluha tlačítek rezistivní dotykové vrstvy
- Úloha č. 3 – zobrazení nebo obnova informace hodin
- Úloha č. 4 – zobrazení nebo obnova informace stavu baterie

Abychom správně pochopili funkci některých těchto úloh, tak je nutné se první zmínit o skutečnosti, že některá uživatelská tlačítka LCD displeje jsou v rámci programu používána k několika funkcím zároveň v závislosti na daném aktuálním stavu systému, což znamená, že je nutné nějakým způsobem tento stav systému identifikovat. Pro tuto příležitost byl definován výčtový typ reprezentující jednotlivé systémové stavy, které se postupně na základě vstupu od uživatele mění.

Tyto stavy jsou z hlediska programového kódu definovány takto:

```
// System flowchart enum
enum POSITION_List{
    SYS_INIT,
    SYS_ERR,
    SYS_HOME,
    SYS_MANUAL_MODE,
    SYS_AUTOMATIC_MODE,
    SYS_SETTINGS_MODE,
    SYS_MANUAL_OCR,
    SYS_MANUAL_SAVE,
    SYS_SETTINGS_TIME_DATE,
    SYS_SETTINGS_CROP,
    SYS_END
};
typedef enum POSITION_List SYSTEM_Position;
```

Podobný výčtový typ, nicméně rozsáhlejší je použit i v rámci definice zmíněných uživatelských tlačítek a v principu slouží jako identifikační číslo (ID) pro tyto jednotlivá grafická tlačítka. Toto ID je poté taktéž výstupní hodnotou přerušení od rezistivní dotykové vrstvy, jak bylo přehledně popsáno v kap. 3.8.2, které je následně zpracováno ve výše uvedené úloze č. 2 kooperativního multitaskingu.

Pokud se nyní zaměříme na problematiku zmíněných uživatelských tlačítek, tak v principu je jedná pouze o doplňující knihovnu ke stávající knihovně zajišťující zobrazování obsahu na LCD displej. Tato knihovna slouží pro definici jednotlivých tlačítek, jejich korektní zobrazování a další podpůrné funkce.

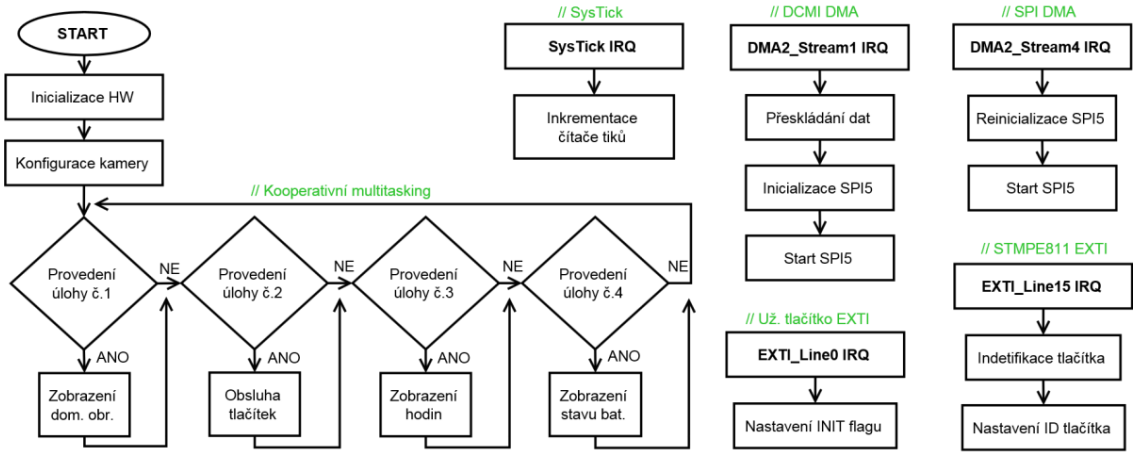
Konkrétně v našem případě jsou všechna uživatelská tlačítka definována současně s inicializací systému a následně jsou v rámci programu tyto tlačítka pouze aktivována nebo deaktivována. Tyto definice jsou obecně součástí mnohaúrovňové struktury, která je součástí výše zmíněné podpůrné knihovny.

Definice této struktury vypadá z hlediska programového kódu následovně:

```
// Button options struct
typedef struct {
    uint16_t x;
    uint16_t y;
    uint16_t width;
    uint16_t height;
    uint16_t background;
    uint16_t borderColor;
    uint16_t flags;
    char* label;
    LCD_FontDef* font;
    uint16_t color;
} LCD_ILI9341_Button;
```

Zde můžeme z programového kódu výše vidět jednotlivé prvky struktury, které představují atributy daného tlačítka podle jeho ID jako např. šířku tlačítka, výšku tlačítka, zvolené písmo štítku apod.

Tímto způsobem získáme podstatou část systémového jádra, které je schopno reagovat na změny a impulzy od uživatele a následně provádět požadované procesy. Výslednou funkci navrženého prototypu snímacího systému lze poté ilustrovat zjednodušeným vývojovým diagramem, který lze vidět na Obr. 3.2 níže.



Obr. 3.2: Vývojový diagram systému navrženého prototypu

Z Obr. 3.2 výše tedy můžeme zjednodušeně vidět všechny klíčové postupy, které zajišťují správný chod navrženého systému. Nejdůležitějšími částmi navrženého FW je mimo zmíněný kooperativní multitasking samotné zpracování obou DMA přerušení popsané v předchozích kapitolách, které zajišťuje korektní zobrazení získaných snímků na LCD displej a poté zpracování úlohy č. 2, která obstarává samotné uživatelské ovládání, které následné řídí celý prototyp. Pro konkrétnější informace ohledně zmíněné funkce systému je vhodné se obrátit na digitální přílohu s kompletním programovým kódem.

4 UŽIVATELSKÝ FIRMWARE

V předchozí kapitole byla popsána funkce navrženého prototypu z hlediska vývoje řídícího FW programátorem, která měla postupně vysvětlit jednotlivé systémové postupy pro snazší pochopení problematiky, nicméně nyní je nutné se zaměřit na popis prototypu i z druhé strany, konkrétně ze strany uživatele.

Postupně tedy budou popsány jednotlivé části výsledného prototypu snímacího systému právě z pohledu uživatele. Převážně se bude jednat o popis navrženého grafického uživatelského rozhraní (GUI) a výsledných měřících funkcí, které jsou uživateli v rámci systému dostupné.

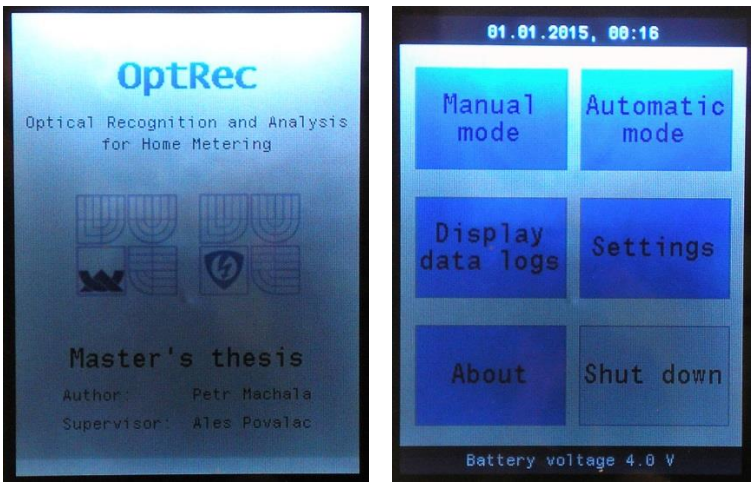
Nutno pouze okrajově dodat, že fotografie v této kapitole jsou reálnými fotografiemi LCD displeje, protože vlivem systémového algoritmu postupného zobrazování nebylo možné tyto obrazová data jednoduše zpracovat v čisté formě. Fotografie se tedy mohou jevit jako mírně nekvalitní.

4.1 Uživatelské GUI

Jako prví tedy pohovoříme obecně o navrženém uživatelském GUI, které zpracovává požadavky od uživatele a následně ho provádí celým systémem.

Pokud budeme postupovat systematicky, tak ihned po přivedení napájení je uživateli zobrazena úvodní obrazovka prototypu, kterou je možné vidět na Obr. 4.1 níže. Samotné pozadí této obrazovky bylo vytvořeno převodem standardního obrázku na jednorozměrné pole délky odpovídající jednotlivým pixelům celého LCD displeje. Tento převod byl proveden pomocí SW *MATLAB*, ovšem nebude nijak dále popisován, protože není pro tuto dokumentaci klíčový, nicméně je součástí digitální přílohy.

Dále se provede konfigurace použitého modulu kamery, kterou je nutné provést vždy, když je prototyp znova zapnut nebo pokud je probuzen z režimu Standby. Tato skutečnost souvisí s minimalizací energetické spotřeby, jak již bylo uvedeno v kap. 3.9.2, protože tento modul je vypínán současně s MCU. Po úspěšné konfiguraci je uživateli zobrazena hlavní domovská stránka, kterou lze vidět na Obr. 4.1 níže.

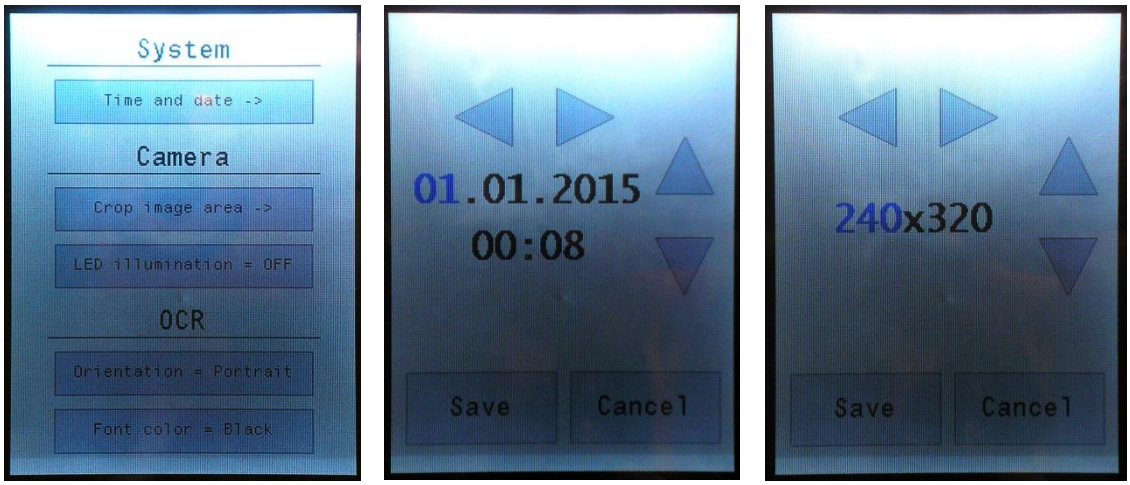


Obr. 4.1: Úvodní a domovská obrazovka uživatelského GUI

Z Obr. 4.1 výše lze přehledně vidět jednotlivé GUI nabídky, které jsou uživateli dostupné a to společně s informací aktuálního data a času v horní části obrazovky a informací aktuálního stavu baterie v dolní části obrazovky. Stav hodin a baterie je pro uživatele dostupný pouze na této domovské obrazovce, kterou lze vždy opakovaně vykreslit pomocí uživatelského tlačítka, jak již bylo popsáno v kap. 3.8.1.

Je vhodné pouze okrajově zmínit, že signalizace stavu baterie začíná nabývat na odchylce se snižující se hodnotou napětí na ADC kanálu. Proto byla zvolena úroveň 3,5 *V*, která představuje limit, kdy je ještě signalizace stavu baterií zobrazována a v opačném případě je zobrazeno hlášení, že baterie nebyla detekována. Tato úroveň byla zvolena na základě experimentálního měření, představující přibližný limit, kdy přestává spolehlivě fungovat lineární stabilizátor napětí na základové DPS. Dalším limitem je hodnota 4 *V*, kdy je aditivně zobrazována zpráva informující uživatele o nízkém napětí baterií.

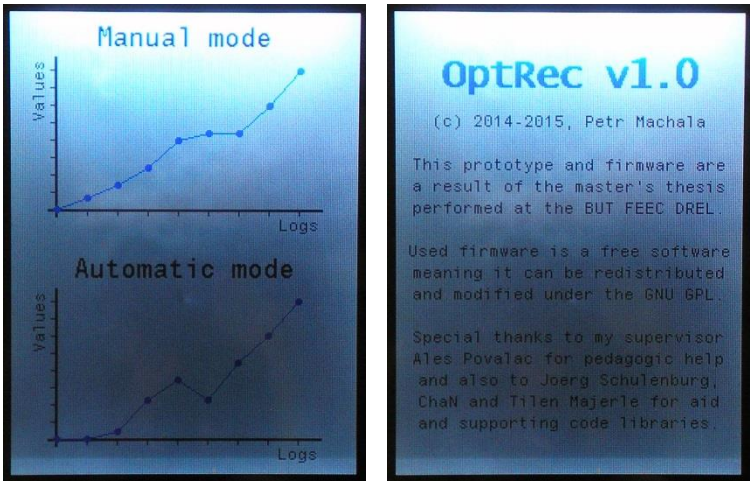
Nyní přejdeme do nabídky nastavení, která ilustruje jednotlivé možnosti z hlediska nastavení prototypu, které jsou uživateli dostupné. Hlavní nabídku nastavení je možné vidět na Obr. 4.2 níže. V této nabídce můžeme vidět jednotlivá nastavení jako je LED přisvícení pro modul kamery, orientaci a barvu písma pro OCR algoritmus a dvě aditivní nabídky představující další nastavovací obrazovky. První aditivní obrazovkou je klasické nastavení data a času a druhou obrazovkou je nastavení ořezu obrazových dat. Obě tyto nabídky lze opět vidět na Obr. 4.2 níže.



Obr. 4.2: Nabídky obecného nastavení, data a času a funkce ořezu

Posledními dvěma nabídkami, kterým se budeme v této podkapitole věnovat, jsou nabídky pro orientační zobrazení naměřených dat a nabídka informativní popisující daný prototyp snímacího systému. Obě tyto nabídky lze pouze zobrazit, takže nelze přes ně aktivovat žádné další funkce. Povolen je pouze návrat na domovskou stránku pomocí uživatelského tlačítka nebo dotyku na libovolnou část displeje. Tyto nabídky je možné vidět na Obr. 4.3 níže.

Nyní je vhodné pouze okrajově zmínit, že nabídka pro zobrazení naměřených dat je pouze orientační záležitostí, protože jak již bylo popsáno v kap. 3.6.2, tak samotná data uložená v RTC registrech obsahují pouze přímo naměřené hodnoty bez jakékoliv informace o čase, kdy byly zaznamenány. Tato skutečnost může výsledné zobrazení, a to převážně v případě manuálního měření, výrazně zkreslit.



Obr. 4.3: Nabídka orientačního zobrazení naměřených dat a informační nabídka

Nakonec samozřejmě zůstává vypínací nabídka, která ovšem slouží pouze pro dlouhodobé uvedení prototypu do Standby módu neboli v našem případě vypnutí. Systém ještě dále obsahuje značné množství oznamovacích obrazovek, které uživatele informují např. o uložení dat na SD kartu apod., nicméně v rámci této dokumentace byly tyto informační obrazovky vynechány.

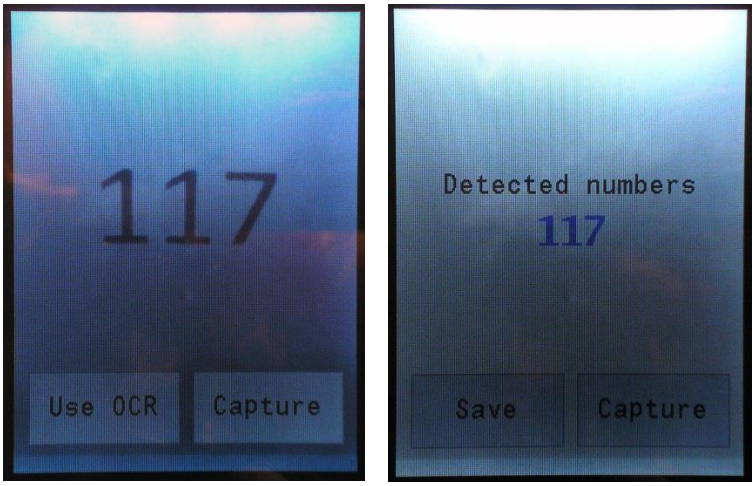
4.2 Manuální režim

Nyní můžeme přejít k hlavnímu popisu prvního uživatelského režimu a to konkrétně k popisu režimu manuálního.

Funkce tohoto režimu je v principu velmi přímočará. Po přechodu do nabídky manuálního režimu se uživateli začne ihned zobrazovat kontinuální přenos obrazových dat kamery na displej a čeká se na impulz od uživatele pro zachycení požadovaného snímku. Tato funkce se dá přirovnat k funkci standardního digitálního fotoaparátu, kdy je uživatelem provedeno směrování kamery, ostření apod. a pořízení snímku. Pořízení snímku je v našem případě prováděno dotykem na libovolnou oblast displeje. Příklad pořízení relativně tmavého testovacího snímku lze vidět na Obr. 4.4 níže.

Po pořízení požadovaného snímku se uživateli nabízejí dvě další možnosti a to buď opětovně zachytit nový snímek v případě, že je získaný snímek nevyhovující (např. rozmazaný) nebo přímo aplikovat rozeznávání číselníku pomocí OCR algoritmu.

V případě, že se uživatel rozhodně pro aplikaci OCR algoritmu, tak se postupně začnou blokujícím způsobem vykonávat všechny procedury, které aplikaci OCR předcházejí, tedy zpětná konverze dat, natočení snímku, ořez apod. a následně je aplikován přímo samotný OCR algoritmus. Takto lze obecně dospět k několika nekorektním výstupům, které mohou zahrnovat rozeznání nepřiměřeného množství znaků, nerozeznání žádného znaku apod. V těchto situacích je vhodné překontrolovat uživatelské nastavení jako je např. orientace snímku pro OCR apod. Nicméně korektní výstup je zobrazení číselné hodnoty, která byla zachycena na vstupním snímku. Příklad aplikace OCR algoritmu lze opět vidět na Obr. 4.4 níže.



Obr. 4.4: Nabídka pořízeného testovacího snímku kamerou a následná aplikace OCR algoritmu v manuálním režimu

V případě úspěšné aplikace OCR algoritmu je uživateli opět zobrazena nabídka dvou možností a to buď zrušení aktuálního výsledku a nutnosti provedení celého procesu záznamu a rozeznávání znova nebo uložení získaných dat do paměti.

Co se týče problematiky ukládání získaných dat do paměti, tak jak již bylo popsáno v předchozích kapitolách, ukládání se obecně provádí na vloženou microSD paměťovou kartu a zároveň do interních RTC registrů pro další systémové potřeby. Z hlediska paměťové karty jsou jednotlivé záznamy ukládány do textového souboru *optrec_manual_mode.txt*, kde jsou vždy postupně připojeny k stávajícímu obsahu. Formát jednotlivých záznamů se skládá pouze z aktuálního nastaveného data a času, kdy byl daný odečet proveden a samotné odečtené hodnoty. Příklad části záznamů podle Obr. 4.3 poté v textovém souboru vypadá následovně:

```
01.01.2015,00:20,10
01.01.2015,00:21,50
01.01.2015,00:21,117
```

4.3 Automatický režim

Nakonec této kapitoly můžeme tedy přejít k popisu druhého uživatelského režimu a to konkrétně režimu automatického.

Funkce tohoto režimu je v principu úplně totožná s režimem manuálním z přechozí podkapitoly, nicméně jak název napovídá, tento režim slouží pro dlouhodobé automatické měření, kde jsou všechny kroky prováděny automaticky. Tento fakt samozřejmě představuje vyšší nároky na korektní snímání kamery a proto je vhodné před aplikací tohoto režimu vždy jako první použít režim manuální, kde se uživatel jednoduše přesvědčí o korektním odečtu apod.

Pokud tedy přejdeme do nabídky automatického režimu, tak můžeme přehledně vidět nastavení tohoto režimu, které musí být vždy provedeno uživatelem podle aktuálního měření. Konkrétně se jedná o interval, jak často má být měření prováděno (min. 5 min a max. 300 min) a jaký je povolený inkrement daného měření (min. 1 a max. 1000). Tuto nabídku je možné vidět na Obr. 4.5 níže.

Obr. 4.5: Nabídka automatického režimu

V případě zahájení automatického měření jsou provedeny všechny inicializační procedury jako např. vynulování ovládacích bitů zálohovacího RTC registru č.0, nastavení RTC alarmu apod. a následně je prototyp uspán. První měření bude následně provedeno podle nastaveného intervalu uživatelem. Tato prodleva umožnuje uživateli dostatečný čas pro umístění prototypu do požadované polohy apod.

Následná funkce prototypu je již plně automatická a uživateli tedy v principu nedostupná, protože během měření jsou vypnuté všechny nepotřebné periferie jako je např. podsvícení LCD displeje, dotykový vrstva apod. Měření je samozřejmě možné v libovolné době mimo měření vypnout pomocí uživatelského tlačítka.

Z hlediska samotného automatického režimu je nyní vhodné zjednodušeně popsat, jak tento vyhodnocovací algoritmus funguje a obecně jak získaná data zpracovává. Pokud vynecháme problematiku probouzení samotného prototypu, s tím spojenou reinicializaci HW a následné měření, tak zpracování dat probíhá následovně.

V případě, že jsou data stále korektní, tedy že mají stále rostoucí tendenci a jsou v dovoleném limitu nastaveném uživatelem, tak jsou data standardně ukládána a měření pokračuje do té doby, než bude přerušeno uživatelem. Obecně ovšem v době měření mohou nastat dvě události, které mohou jeho průběh ovlivnit a to konkrétně, že dojde např. vlivem přetáčením číselníku ke špatnému odečtu OCR algoritmem nebo že dojde k přetečení snímané hodnoty. Tato událost detekce přetečení je jednou ze základních funkcí, pro které je tento uživatelský režim navržen a slouží tedy k detekci úniku plynu, vody apod. při kontinuálním měření.

Co se týče zpracování těchto událostí popsaných výše, tak v rámci automatického měření je obecně povolen jeden špatný OCR odečet a jedno přetečení, kdy FW ještě nedetekuje chybový stav, protože se může potenciálně jednat pouze o planý poplach. Pokud jsou detekovány dva chybové stavy za sebou, tak dojde k vyvolání odpovídajícího alarmu, tedy alarmu špatného OCR odečtu nebo alarmu přetečení. Obecně lze tedy říct, že v reálné situaci by potenciálně mohla nastat situace, kdy by došlo k přetečení měřené hodnoty, následně k chybnému OCR odečtu a nakonec opět k přetečení než by došlo k vyvolání alarmu přetečení. Lze tedy říct, že celkové zpoždění pro detekci přetečení může tedy obecně nabýt maximálně trojnásobku nastaveného intervalu uživatelem.

Z hlediska ukládání získaných dat je systém obecně totožný se systémem manuálního režimu, nicméně v tomto případě jsou záznamy ukládány do samostatného textového souboru *optrec_automatic_mode.txt*, kde jsou vždy postupně připojeny k stávajícímu obsahu. Formát dat je opět totožný, což znamená, že rozlišení mezi jednotlivými režimy je pouze ve formě odlišného koncového textového souboru. Příklad části záznamů podle Obr. 4.3 poté v tomto souboru vypadá následovně:

```
01.01.2015,00:47,50
01.01.2015,00:52,20
01.01.2015,00:57,117
```

Z těchto záznamů uvedených výše můžeme právě vidět jednu z možných situací, které v rámci automatického měření mohou nastat a to konkrétně případ, kdy dojde k snímání nižší hodnoty, než je hodnota předchozí. Tento stav je signalizován systému jako špatný OCR odečet a pokud by následná hodnota nebyla korektní, tak by byl vyvolán právě zmíněný OCR alarm. Pokud je další záznam korektní, tak se v měření jednoduše pokračuje. Nutno ovšem dodat, že v tomto případě není možné porovnávat korektní hodnoty mezi sebou podle nastaveného limitu uživatelem, ale s jeho dvojnásobkem, protože jsou v rozpětí dvojnásobku měřícího intervalu.

5 ZHODNOCENÍ A DEMONSTRACE

Nakonec této dokumentace se zaměříme na problematiku závěrečného zhodnocení navrženého prototypu snímacího systému a postupně rozebereme některé jeho funkce z hlediska HW realizace a navrženého FW.

5.1 Získaná obrazová data

Jako první se zaměříme na zhodnocení problematiky týkající se dosažené kvality získaných obrazových dat pomocí použitého modulu kamery, protože se jedná o klíčovou část této diplomové práce, které byl věnován značný prostor. Pro tuto problematiku byl jeden ze získaných snímků přenesen ve své čisté formě do PC, kde byl následně obnoven a není tedy zatížen kvalitou LCD displeje atd.

Samotná komunikace a přenos dat byl proveden pomocí standardního sériového přenosu přes USB sběrnici mezi vývojovou deskou ve VCP módu a PC, kde poté byla tato data obnovena pomocí SW *MATLAB*. Problematika vzájemné komunikace mezi vývojovou deskou a PC je obecně velmi rozsáhlou záležitostí, nicméně lze říct, že v našem případě je založena na oficiální předloze USB VCP podle výrobce. Samotný popis této vzájemné komunikace je ovšem v rámci této dokumentace vynechán, protože je pro funkci výsledného prototypu irelevantní. Ovšem knihovnu zajišťující konfiguraci použité vývojové desky do VCP módu včetně ovladače pro PC lze pro zajímavost nalézt v digitální příloze.

Výsledný snímek získaný modulem kamery podle použité kompletní konfigurace je poté možné vidět na Obr. 5.1 níže. Kompletní konfiguraci modulu kamery lze samozřejmě nalézt v digitální příloze.

Obr. 5.1: Příklad snímku získaného modulem kamery

Z Obr. 5.1 výše tedy můžeme vidět výslednou kvalitu obrazových dat získaných pomocí použitého modulu kamery OV7670. Obecně lze říct, že tato kamera je svým poměrem mezi cenou a kvalitou dobrým modulem po libovolné embedded aplikace, nicméně odvrácenou stranou je, jak již bylo uvedeno v kap. 3.3.3, velmi složitá konfigurace a špatná kvalita dostupných oficiálních materiálů.

5.2 Použitý OCR algoritmus

Dále pohovoříme o zhodnocení problematiky použitého OCR algoritmu a jeho následné aplikaci v praxi. Nutno zmínit, že tato problematika je velmi rozsáhlou záležitostí, která vyžaduje individuální přístup a značné množství testování v reálných podmínkách, což nebylo možné z časových důvodů při návrhu výsledného prototypu zajistit.

Z těchto důvodů tedy nebude uvedeno žádné reálné měření, které by v principu nemělo žádnou informační hodnotu, protože je vázáno na aktuální podmínky, danou aplikaci a nastavení prototypu. Proto budou uvedeny pouze určité poznatky, které byly během testování pozorovány. Určitým příkladem může být demonstrační zobrazení uživatelského GUI na Obr. 4.4 v kap. 4.2.

Pokud se podíváme na funkci OCR algoritmu pro snímání černých číslic na bílém pozadí, tak je výsledek velmi dobrý a stabilní z hlediska několikanásobného snímání. Pokud se jedná o inverzní situaci, kdy jsou číslice bílé a pozadí černé, tak v určitých případech lze dosáhnout dobrých výsledků i když není nastavena odpovídající barva písma v nastavení prototypu. S odpovídajícím nastavením odlišné barvy od černé lze poté dosáhnout srovnatelně dobrých výsledků jako pro černou barvu textu.

Ve velké části provedených měření bylo možné výslednou detekci vylepšit ořezáním snímku od irelevantních dat, ovšem v určitých případech lze narazit na situace, kdy byl ořezem natolik narušen poměr velikosti mezi ciframi a pozadím, že algoritmus není schopen žádnou cifru detekovat. Obecně lze tedy říct, že je vhodnější snímat číslice z větší dálky a provést pouze nezbytně nutný ořez snímku.

Problematičtější začíná být situace při aplikaci OCR na reálné podmínky ciferníků analogových měřidel. Zde je obecně bez ořezu snímku velmi problematické dosáhnout uspokojivých výsledků z hlediska několikanásobného snímání a to i za předpokladu korektního nastavení prototypu. Velkým problémem je fakt, že v případě analogových číselníků jsou cifry uzavřeny v jednotlivých boxech, které mají velmi ostré hrany. Tyto hrany poté mohou být potenciálním zdrojem pro fantomové cifry, nejčastěji logicky cifry jako je 1 nebo 0.

Za předpokladu korektního OCR nastavení jako je barva písma a natočení lze v určitých případech vylepšit detekci pomocí snížení limitu jistoty rozpoznaných znaků, jak bylo uvedeno v kap. 3.5.2. Zde je ovšem nutné si uvědomit, že snížením tohoto limitu většinou podpoříme schopnost rozeznávat jednotlivé cifry analogového číselníku, nicméně zároveň tím podporujeme i potenciální výskyt fantomových cifer, jak bylo zmíněno výše. V reálné situaci se tak může např. stát, že je někdy detekováno o jednu nebo více cifer navíc, což znamená výrazné přetečení o jeden nebo více řádů. Tato skutečnost může znamenat právě problematický chod automatického režimu, kde není žádná možnost, jak by uživatel mohl jednotlivé výsledky korigovat.

Z textu výše je tedy jasné, že samotné použití prototypu je silně závislé na aktuálních podmínkách, jako je např. typ analogového měřiče, osvětlení místnosti nebo konkrétním nastavením prototypu. Vhodné pouze dodat, že hodnota limitu jistoty rozpoznaných znaků, která v programovém kódu pevně nastavena na 90% je ideálním poměrem pro korektní detekci a je tedy vhodnější aktuální podmínky této skutečnosti přizpůsobit správným nastavením prototypu.

5.3 Výsledná energetická efektivita

Nakonec se podíváme na problematiku výsledné energetické spotřeby celého navrženého prototypu a provedeme orientační výpočet potenciální výdrže v případě, že by prototyp byl dlouhodobě napájen pouze bateriemi.

Jako první se podíváme na celkovou spotřebu prototypu v případě svého maxima, tedy když má prototyp současně aktivovánu většinu svých periferií. Konkrétně se jedná o stav snímání kamerou v manuálním režimu při aktivaci LED přisvícení. Zde bylo měřením určeno, že celková spotřeba prototypu je v tomto stavu cca 280 mA, kde cca 80mA je spotřeba samotného MCU.

Dále se zaměříme na spotřebu v případě, kdy je MCU uveden do Standby módu a tedy z hlediska snímacího systému je celý prototyp vypnut. Zde měřením zjistíme dosti zarážející hodnotu celkové spotřeby a to konkrétně hodnotu ve výši cca 45 mA, kde cca 210 uA je spotřeba samotného MCU. Výsledek tohoto měření není nijak chybný, ale právě poukazuje na skutečnost, že použitá vývojová deska je mj. osazena mnoha dalšími periferiemi, které se nedají jednoduše deaktivovat. Nejdominantnějším zdrojem pro tohoto externího proudu je tzv. *STLINK/V2* interní obvod, který slouží pro debuggování a programování osazeného MCU. Tento obvod, který lze blokově vidět např. na Obr. 2.3 je osazen dalším MCU s označením STM32F103C8 výrobce *STMicroelectronics*. Tento MCU samostatně může podle datového listu zde [25] mít spotřebu až 50 mA. Je tedy jasné, že v případě budoucí energetické optimalizace je nutné tento interní obvod deaktivovat nebo provést návrh samostatné DPS.

Pokud ovšem budeme vycházet z měření uvedeného výše, tak můžeme zmíněný odhad provést velmi jednoduše. V případě použití automatického režimu s nejkratším intervalem (5 min), kde vezmeme v potaz cca 5 - 6 sekund zpoždění, které představuje dobu zpracování měření, tak dostaneme výslednou průměrnou hodnotu spotřeby kolem cca 50 mA. Nutno pouze zmínit, že byla zanedbána skutečnost, že v automatickém režimu je po celou dobu měření vypnut LCD displej a výsledná spotřeba tedy může být menší, nicméně ve výsledném průměru je tato hodnota zanedbatelnou. Pokud tedy vezmeme v potaz typické kapacity prodávaných baterií, tedy cca 700 mAh pro obyčejné NiCd baterie a 2700 mAh pro alkalické baterie, tak v našem případě dostáváme výdrž bateriově napájeného prototypu vždy v horizontu desítek hodin popř. několika dnů. Konkrétně cca pul dne v případě použití obyčejných baterií a cca dva dny při použití alkalických baterií, nicméně tento výsledek je nutné brát pouze jako velmi hrubý odhad.

Z výše uvedeného textu je tedy jasné, že výsledný prototyp není úplně nejlépe navržen, co se týče energetické efektivity, nicméně pozitivní stránkou je skutečnost, že zde můžeme najít mnoho prostoru k vylepšení. Lze tedy říct, že baterie v případě navrženého prototypu slouží pouze jako záložní zdroj pro krátkodobá měření a primárním způsobem napájení je libovolný externí zdroj.

6 ZÁVĚR

V této dokumentaci byla postupně nastíněna a následně popsána problematika týkající se optického snímání číselníků analogových bytových měřidel pro následnou aplikaci OCR algoritmu za účelem získání požadovaných dat spotřeby měřidla. Pro tuto problematiku byl postupně navržen výsledný prototyp snímacího systému *OptRec* skládající se z vývojové desky 32F429IDSICOVERY a použitého modulu kamery OV7670. Tyto jednotlivé části jsou propojeny pomocí navržené základové DPS, která mj. obsahuje i napájení modulů přídavnou flash paměť aj.

První klíčovou problematikou této práce je propojení použité vývojové desky s modulem kamery a korektní konfigurace tohoto modulu. Samotná konfigurace modulu kamery je realizována pomocí speciální SCCB sběrnice, která byla emulována pomocí standardizované sběrnice I2C. Pro potřeby přenosu obrazových dat je použita vestavěná DCMI sběrnice v kontinuálním režimu, kde jsou přijímaná data postupně pomocí DMA řadiče přesouvána do oblasti externí SDRAM paměti, která je součástí použité vývojové desky. Odkud jsou obrazová data následně vizualizována na LCD displej nebo podrobena rozeznávání pomocí OCR algoritmu. Samotná problematika snímání jednotlivých snímků a jejich následné zobrazení na LCD displeji se po čas návrhu ukázala jako velmi problematickou a proto musel být navržen optimalizovaný algoritmus založený několikanásobných DMA přenosech.

Další klíčovou částí je problematika implementace použitého OCR algoritmu, kde byl vybrán algoritmus *GOCR v0.50*, který představuje vhodný poměr mezi kvalitou optického snímání a možností implementace do naší koncové aplikace. Samotná problematika implementace je převážně omezena na použití výše zmíněného algoritmu jako zásuvného modulu a jeho správného nastavení, nicméně za předpokladu rozšíření interního paměťového prostoru MCU, který byl realizován opět pomocí externí SDRAM paměti. Postupným testováním bylo tedy nalezeno optimální nastavení OCR algoritmu pro potřeby snímání analogových číselníků, který byl následně implementován ve výsledném navrženém prototypu.

Poslední klíčovou částí je návrh systémového jádra prototypu, uživatelského ovládacího GUI a zajištění funkce dalších podpůrných periférií jako např. funkce interního RTC obvodu, komunikace s externí flash paměti apod. Samotné jádro se obecně skládá z jednoduchého kooperativního multitaskingu, který obstarává jednotlivé cyklické operace a dalších systémových operací ve formě interních a externích přerušení. Tímto způsobem je zajištěn korektní chod ovládacího FW a komunikace s uživatelem. Z hlediska uživatelského GUI je uživateli umožněno prototyp obecně používat ve dvou funkčních režimech. Tyto režimy lze označit jako manuální pro krátkodobá individuální měření a automatický pro dlouhodobá měření, který je vybaven ochrannými funkcemi proti úniku plynu, vody apod.

Obecně lze tedy říct, že byl na základě zadání postupně navržen kompletní a univerzální systém pro optické snímání a analýzu bytových měřidel, který lze použít v libovolných reálných podmínkách pro jednoduchý záznam bytových měřidel a následné statistické zpracování. Prototyp lze ovšem právě díky své univerzalitě považovat i jako velmi dobrou platformu pro další potenciální aplikace a projekty.

LITERATURA

[1] Elektroměr. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-11-30]. Dostupné z: http://goo.gl/6cZH90

[2] Plynoměr. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-11-30]. Dostupné z: http://goo.gl/LYvCw9

[3] Vattenmätare (mätinstrument). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2014-11-30]. Dostupné z: http://goo.gl/ghM6Ks

[4] License Plate Recognition Software. *Adaptive Recognition Hungary* [online]. 2004 [cit. 2014-12-14]. Dostupné z: http://goo.gl/8iybKD

[5] UM1670 User manual: Discovery kit for STM32F429/439 lines. In: *STMicroelectronics* [online]. 2013 [cit. 2014-11-29]. Dostupné z: http://goo.gl/Xhvmwz

[6] 32F429IDISCOVERY: Discovery kit for STM32 F429/439 lines – withSTM32F429ZI MCU. *STMicroelectronics* [online]. 2014 [cit. 2014-11-29]. Dostupné z: http://goo.gl/J5Iyq9

[7] STM32F427xx, STM32F429xx: ARM Cortex-M4 32b MCU+FPU. In: *STMicroelectronics* [online]. 2007 [cit. 2014-11-29]. Dostupné z: http://goo.gl/nL7vQc

[8] STSW-STM32138: STM32F429 discovery firmware package (UM1662). *STMicroelectronics* [online]. 2011 [cit. 2014-11-29]. Dostupné z: http://goo.gl/H0i9JT

[9] OV7670 Camera Module. *ElecFreaks store* [online]. 2013 [cit. 2014-11-29]. Dostupné z: http://goo.gl/equQ3L

[10] OMNIVISION. *OV7670/OV7171 CMOS VGA (640x480) CameraChip: Preliminary Datasheet*. 2005.

[11] OMNIVISION. *OV7670/OV7171 CMOS VGA (640x480) CameraChip: Implementation Guide*. 2005.

[12] OmniVision Serial Camera Control Bus (SCCB): Functional Specification. In: *OmniVision* [online]. 2007 [cit. 2014-11-29]. Dostupné z: http://goo.gl/HeOPGU

[13] UM1680 User manual: Getting started with STM32F429 Discovery software development tools. In: *STMicroelectronics* [online]. 2013 [cit. 2014-11-29]. Dostupné z: http://goo.gl/ae47zH

[14] RM0090 Reference manual: STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx and STM32F43xxx advanced ARM®-based 32-bit MCUs. In: *STMicroelectronics* [online]. 2014 [cit. 2014-12-06]. Dostupné z: http://goo.gl/3elIMV

[15] A V4L2 driver for OmniVision OV7670 cameras. *Linux Cross Reference* [online]. 2006 [cit. 2014-12-06]. Dostupné z: http://goo.gl/brGCRU

[16] *ARM mbed Developer Site* [online]. 2014 [cit. 2014-12-06]. Dostupné z: http://goo.gl/WVdydG

[17] ILI9341: a-Si TFT LCD Single Chip Driver 240RGBx320 Resolution and 262K color. 2015. In: *Newhaven Display* [online]. [cit. 2015-05-11]. Dostupné z: http://goo.gl/2C8fIY

[18] *How to scan and OCR like a pro with free software* [online]. January 2009. [cit. 2015-05-11]. Dostupné z: http://goo.gl/zQd7GJ

[19] *GOCR* [online]. June 2000 [cit. 2014-12-14]. Dostupné z: http://goo.gl/BfcF3h

[20] Grayscale. 2001-. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2015-05-11]. Dostupné z: http://goo.gl/8tTHvT

[21] *Ubuntu manuals: GOCR* [online]. 2010. [cit. 2015-05-11]. Dostupné z: http://goo.gl/hzrHke

[22] *FatFs: Generic FAT File System Module* [online]. 2015. [cit. 2015-05-11]. Dostupné z: http://goo.gl/eBns7O

[23] STMPE811: 8-bit port expander with advanced touchscreen controller. 2015. In: *Alldatasheet.com* [online]. [cit. 2015-05-11]. Dostupné z: http://goo.gl/uJV2Lg

[24] AN4365 Application note: Using STM32F4 MCU power modes with best dynamic efficiency. 2014. In: *STMicroelectronics* [online]. [cit. 2015-05-11]. Dostupné z: http://goo.gl/3elIMV

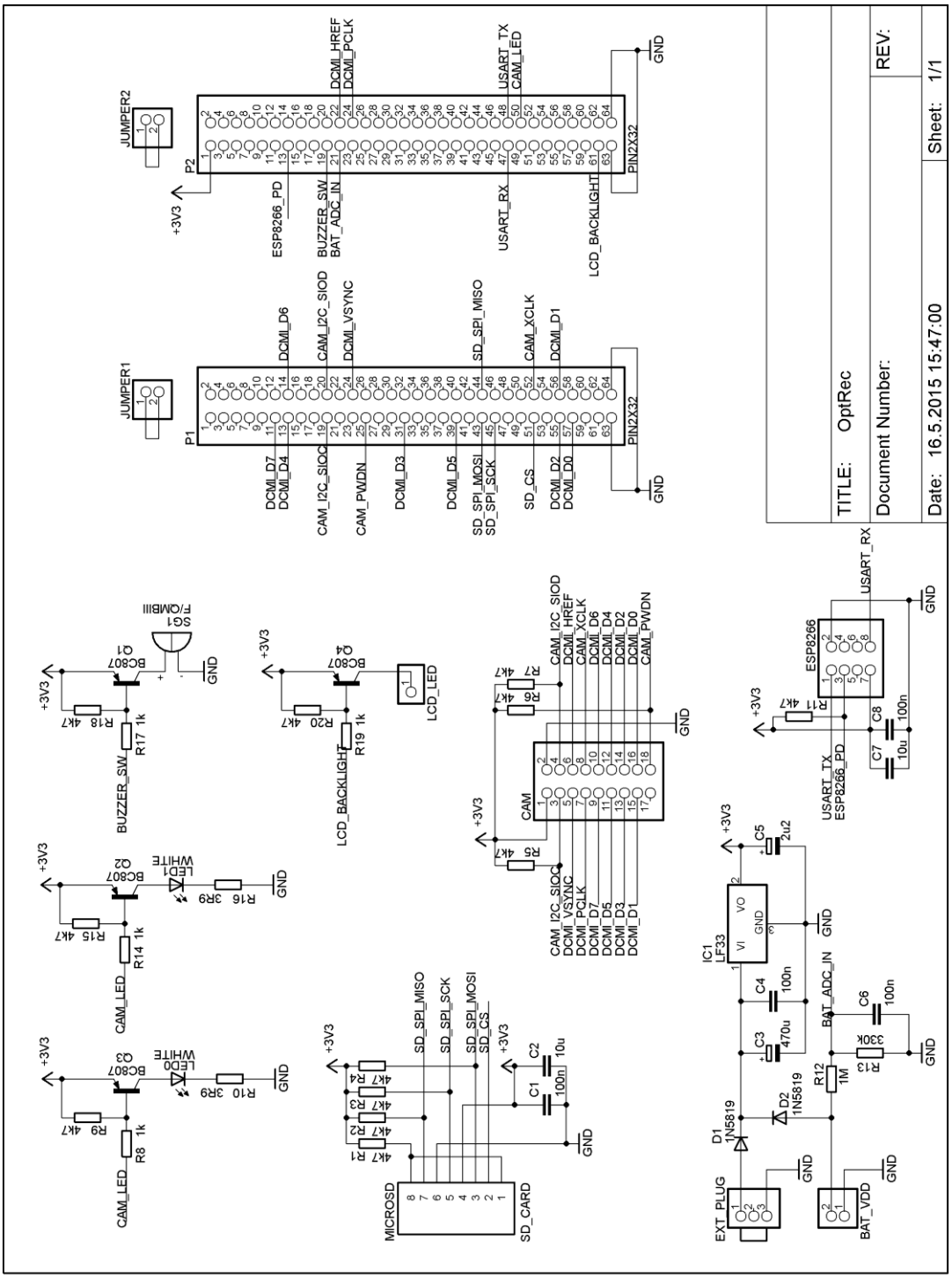
[25] STM32F103x8 STM32F103xB: Medium-density performance line ARM-based 32-bit MCU. In: *STMicroelectronics* [online]. 2013 [cit. 2015-05-18]. Dostupné z: http://goo.gl/RwVNvL

SEZNAM SYMBOLŮ A ZKRATEK

AEC	Automatic exposure control
AGC	Automatic gain control
ASCII	American standard code for information interchange
AWB	Automatic white balance
CIF	Common intermediate format
CMOS	Complementary metal–oxide–semiconductor
CMSIS	Cortex microcontroller software interface standard
COM	Communication port
DCMI	Digital camera interface
DMA	Direct memory access
DPS	Deska plošného spoje
DSP	Digital signal processor
FAT	File allocation table
FIFO	First in, first out
FMC	Flexible memory control
FPU	Floating-point unit
FW	Firmware
GCC	GNU compiler collection
GUI	Graphical user interface
HW	Hardware
I/O	Input/output
I2C	Inter-integrated circuit
ID	Identifier
IRQ	Interrupt request
JTAG	Joint test action group
LCD	Liquid-crystal display
LSE	Low speed external (clock)
LSI	Low speed internal (clock)
MCU	Microcontroller unit
MEMS	Micro-electro-mechanical systems
OCR	Optical character recognition
OWR	Optical word recognition
PNM	Portable anymap format
PWM	Pulse-width modulation
QR	Quick response
QVGA	Quarter video graphics array
RTC	Real-time clock
SCCB	Serial camera control bus
SDRAM	Synchronous dynamic random access memory
SPI	Serial peripheral interface
SPL	Standard peripheral library
SW	Software
SWD	Serial wire debug
TFT	Thin-film-transistor
UMS	USB mass storage
USB	Universal serial bus
VCP	Virtual COM port
VGA	Video graphics array

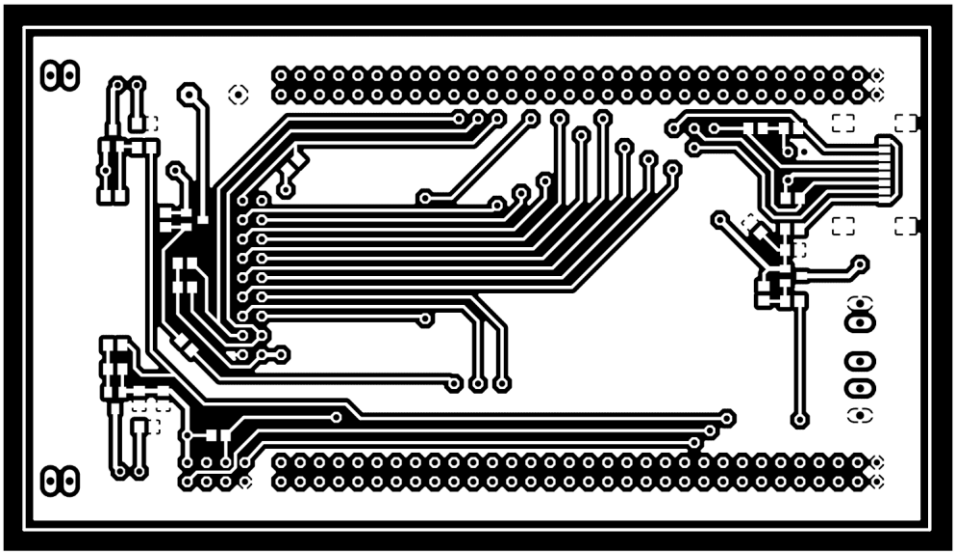
A ZÁKLADOVÁ DPS

A.1 Schéma zapojení



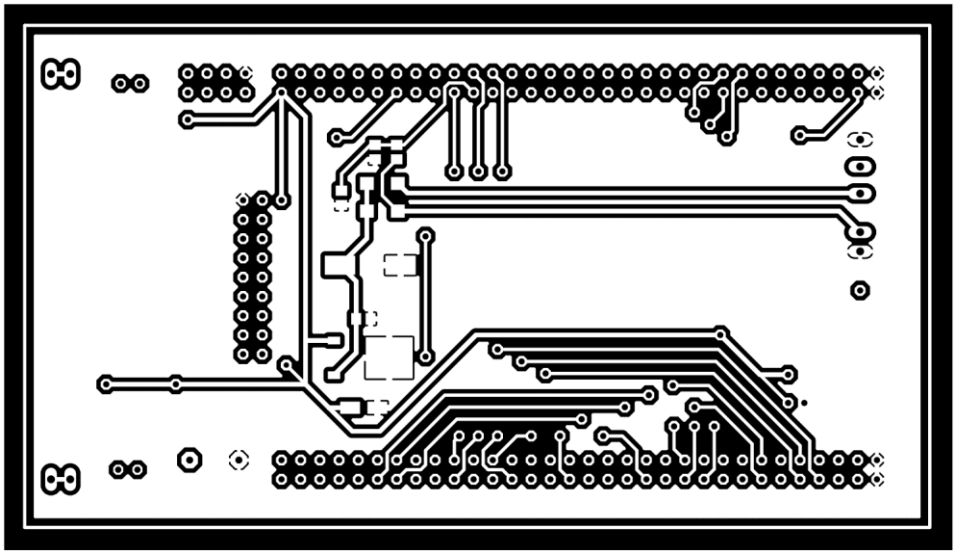
Obr. 6.1: Kompletní schéma zapojení základové DPS

A.2 Předloha DPS



Obr. 6.2: Předloha základové DPS – horní strana

Rozměry předlohy DPS podle Obr. 6.2 výše jsou 66 x 119 mm, měřítko M1:1.



Obr. 6.3: Předloha základové DPS – spodní strana

Rozměry předlohy DPS podle Obr. 6.3 výše jsou 66 x 119 mm, měřítko M1:1.

A.3 Seznam součástek

Tab. 2: Seznam použitých součástek základové DPS

Označení	Hodnota	Pouzdro	Popis
C1, C4, C6, C8	100n	C-0805	Keramický kondenzátor
C2, C7	10u	C-0805	Keramický kondenzátor
C3	470u/25V	CPOL-1010	Elektrolytický kondenzátor
C5	2u2/10V	CPOL-0405	Elektrolytický kondenzátor
D1, D2	1N5819	MELF	Schottkyho dioda
IC1	LF33	DPAK	Lineární regulátor napětí
Q1, Q2, Q3, Q4	BC807-25	SOT23	Bipolární PNP tranzistor
R1, R2, R3, R4, R5, R6, R7, R9, R11, R15, R18, R20	4k7	R-0805	Rezistor SMD
R8, R14, R17, R19	1k	R-0805	Rezistor SMD
R10, R16	3R9	R-0805	Rezistor SMD
R12	1M	R-0805	Rezistor SMD
R13	330k	R-0805	Rezistor SMD
LED0, LED1	WHITE200/150°	LED5MM	LED dioda
SG1		RM=6.5mm, d=12mm	Elektromagnetický bzučák
SD_CARD		MicroSD Socket	Pouzdro microSD karty
EXT_PLUG		d=2.1mm	Napájecí souosý konektor do DPS
P1, P2		RM=2.54mm, PIN2X32	Dutinková lišta 2x32 pinů
CAM		RM=2.54mm, PIN2X9	Dutinková lišta 2x9 pinů
ESP8266		RM=2.54mm, PIN2X4	Dutinková lišta 2x4 piny
JUMPER1, JUMPER2, BAT_VDD		RM=2.54mm, PIN1X2	Dutinková lišta 1x2 piny
LCD_LED		RM=2.54mm, PIN1X1	Dutinková lišta 1x1 pin

A.4 Seznam použitých pinů

Tab. 3: Seznam použitých pinů vývojové desky 32F429IDISCOVERY

Port	Pin	Přiřazená funkce
PORT A	PA0	ON(1)/OFF(0) - uživatelské tlačítko
	PA3	ON(0)/OFF(1) - ovládání bzučáku
	PA4	DCMI_HSYNC - komunikace s modulem kamery
	PA5	ADC1 - měření napětí baterií
	PA6	DCMI_PIXCLK - komunikace s modulem kamery
	PA8	I2C3_SCL - komunikace s dotykovou vrstvou
	PA9	TIM1_XCLK - hodinový signál modulu kamery
	PA10	SPI3_CS - chip select SD karty
	PA15	TP_INT - přerušení řadiče dotykové vrstvy
PORT B	PB4	ON(1)/OFF(0) - deaktivace modulu kamery
	PB5	FMC_SDCKE1 - komunikace s SDRAM
	PB6	FMC_SDNE1 - komunikace s SDRAM
	PB7	DCMI_VSYNC - komunikace s modulem kamery
	PB8	SCCB_SCL - komunikace s modulem kamery
	PB9	SCCB_SDA - komunikace s modulem kamery
	PB10	USART3_TX - komunikace s bezdrátovým modulem
	PB11	USART3_RX - komunikace s bezdrátovým modulem
	PB12	ON(0)/OFF(1) - ovládání LED osvětlení
PORT C	PC0	FMC_SDNWE - komunikace s SDRAM
	PC1	ON(1)/OFF(0) - deaktivace bezdrátového modulu
	PC2	SPI5_CS - chip select LCD displeje
	PC6	DCMI_D0 - komunikace s modulem kamery
	PC7	DCMI_D1 - komunikace s modulem kamery
	PC8	DCMI_D2 - komunikace s modulem kamery
	PC9	I2C3_SDA - komunikace s dotykovou vrstvou
	PC10	SPI3_SCK - komunikace s SD kartou
	PC11	SPI3_MISO - komunikace s SD kartou
	PC12	SPI3_MOSI - komunikace s SD kartou
	PC14	OSC32_IN - externí RTC krystal
	PC15	OSC32_OUT - externí RTC krystal
PORT D	PD0	FMC_D2 - komunikace s SDRAM
	PD1	FMC_D3 - komunikace s SDRAM
	PD3	DCMI_D5 - komunikace s modulem kamery
	PD8	FMC_D13 - komunikace s SDRAM
	PD9	FMC_D14 - komunikace s SDRAM
	PD10	FMC_D15 - komunikace s SDRAM
	PD12	ON(1)/OFF(0) - reset pin LCD displeje
	PD13	ON(1)/OFF(0) - data/command pin LCD displeje
	PD14	FMC_D0 - komunikace s SDRAM
	PD15	FMC_D1 - komunikace s SDRAM

Port	Pin	Přiřazená funkce
PORT E	PE0	FMC_NBL0 - komunikace s SDRAM
	PE1	FMC_NBL1 - komunikace s SDRAM
	PE4	DCMI_D4 - komunikace s modulem kamery
	PE5	DCMI_D6 - komunikace s modulem kamery
	PE6	DCMI_D7 - komunikace s modulem kamery
	PE7	FMC_D4 - komunikace s SDRAM
	PE8	FMC_D5 - komunikace s SDRAM
	PE9	FMC_D6 - komunikace s SDRAM
	PE10	FMC_D7 - komunikace s SDRAM
	PE11	FMC_D8 - komunikace s SDRAM
	PE12	FMC_D9 - komunikace s SDRAM
	PE13	FMC_D10 - komunikace s SDRAM
	PE14	FMC_D11 - komunikace s SDRAM
	PE15	FMC_D12 - komunikace s SDRAM
PORT F	PF0	FMC_A0 - komunikace s SDRAM
	PF1	FMC_A1 - komunikace s SDRAM
	PF2	FMC_A2 - komunikace s SDRAM
	PF3	FMC_A3 - komunikace s SDRAM
	PF4	FMC_A4 - komunikace s SDRAM
	PF5	FMC_A5 - komunikace s SDRAM
	PF7	SPI5_SCK - komunikace s LCD displejem
	PF8	SPI5_MISO - komunikace s LCD displejem
	PF9	SPI5_MOSI - komunikace s LCD displejem
	PF11	FMC_SDNRAS - komunikace s SDRAM
	PF12	FMC_A6 - komunikace s SDRAM
	PF13	FMC_A7 - komunikace s SDRAM
	PF14	FMC_A8 - komunikace s SDRAM
	PF15	FMC_A9 - komunikace s SDRAM
PORT G	PG0	FMC_A10 - komunikace s SDRAM
	PG1	FMC_A11 - komunikace s SDRAM
	PG3	ON(0)/OFF(1) - ovládání podsvícení LCD displeje
	PG4	FMC_BA0 - komunikace s SDRAM
	PG5	FMC_BA1 - komunikace s SDRAM
	PG8	FMC_SDCLK - komunikace s SDRAM
	PG11	DCMI_D3 - komunikace s modulem kamery
	PG15	FMC_SDNCAS - komunikace s SDRAM