ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
katedra teorie obvodů

BAKALÁŘSKÁ PRÁCE

2007 **Pavel Košík**

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

katedra teorie obvodů

Periferní deska pro řízení displejů a vstup dat

bakalářská práce

Srpen 2007

Vypracoval: Pavel Košík

Vedoucí práce: Ing. Václav Hanžl, Csc.

Čestné prohlášení

Prohlašuji, že jsem zadanou bakalářskou práci vypracoval samostatně. Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé bakalářské práce nebo její části se souhlasem katedry.

V Praze dne: . . . . . Podpis: . . . . . .

……..Zadání…….
(Originál v originálu bakalářské práce,
oboustranná kopie v kopii bakalářské práce)

Anotace:
Tato práce se zabývá návrhem desky, která je určena pro řízení displejů a je vestavitelná do různých zařízení. Její součástí je zvukový kodek, analogový vstup, logický vstup a výstup, konektor pro SD-kartu, rozhraní RS232 a RS485, paměť SDRAM. Hlavní částí práce je výběr součástek a návrh desky. Součástí návrhu je vytvoření ukázkového programu pro hradlové pole a pro řídicí mikrokontrolér.

Summary:
This project solves design of board, which is designed for driving LC displays and is suitable for built-in. It's parts are sound codec, analog input, logic input and output, connector for SD card, RS232 and RS485 port, SDRAM. Essential part is choice of suitable electronic devices and creation of board's layout. Part of this project is creation of demonstration software for FPGA and microcontroller.

Obsah

Seznam použitých zkratek a symbolů

! - vstup (výstup) aktivní v logické nule
AVR - procesor rodiny AVR výrobce Atmel, v textu míněno ATMEGA 128L použité na desce
BRAM - Block RAM, dvoubránová paměť v hradlovém poli, konfigurovatelná pro různou šířku datové sběrnice a různou hloubku
SDRAM - Synchronous DRAM, synchronní dynamická paměť RAM
DDR SDRAM - Double Data Rate SDRAM, paměť SDRAM s dvojnásobným tokem dat
FPGA - hradlové pole, v textu míněn konkrétní typ XC3S500E použitý na desce
I/O - vstupně-výstupní
LUT - look-up table, základní funkční element v hradlovém poli, vytvářející logickou funkci, v řadě XC3SxxxE je to rychlá čtyřvstupová SRAM
SD-karta - secure digital card, paměťová karta určená do digitálních fotoaparátů, MP3 přehrávačů apod.
VCC - napájecí napětí +3,3V
VCCINT - napájecí napětí +1,2V
VCCIO - napájecí napětí +2,5V
VDD - napájecí napětí +5V

Seznam příloh

Obrázkové přílohy

Ukázky zdrojového kódu

Obsah přiloženého DVD

Tato bakalářská práce ve formátu Pdf

Schéma desky, návrh plošného spoje desky ve formátu programu Eagle

Zdrojové kódy pro AVR

Zdrojové kódy pro FPGA

Katalogové listy obvodů

Kapitola 1

Úvod

Cílem mé práce není navrhnout desku, která by se výpočetním výkonem vyrovnala PC nebo podobným zařízením, ale desku která by především byla schopna samostatně řídit různé typy displejů, byla vhodná k zástavbě do zařízení, byla by schopna komunikovat s jinými deskami a byla by vhodná pro připojení na sběrnici různých procesorů.
Dále mojí snahou je aby deska uměla zpracovat analogové nebo digitální signály, uměla je poslat do PC, zapsat na SD-kartu nebo je uměla zobrazit srozumitelným způsobem.

Hlavní částí práce byl výběr vhodných součástek, návrh samotné desky v návrhovém prostředí pro plošné spoje (Eagle) a samotné osazení a oživení desky. Neméně důležitou částí bylo vytvoření ukázkového programu pro AVR a FPGA, protože bez programového vybavení by deska byla jen „kusem mrtvého železa“.

Výsledkem celé práce je deska, která má dostatečné parametry pro řízení, a některé výpočty a může se tak stát srdcem různých zařízení.
Mým cílem v budoucnu je připojit k desce digitální obrazový snímač a s pomocí desky uložit jeho obraz do SD-karty, nejlépe jako video sekvenci se zvukem, nebo jej rozhraním RS485 poslat do jiné desky.

Kapitola 2

Požadavky pro volbu obvodů a jejich volba

Mojí snahou bylo použít co nejlépe dostupné součástky a to takové, které se dají koupit přímo v některé z prodejen v Praze a nebo na objednávku, ale ne s dodací lhůtou řádově měsíců nebo s minimálním objednacím množstvím stovek kusů. I přes tento požadavek jsem na desce použil některé exotičtější součástky z důvodu jejich speciálních parametrů. Konstrukce desky je modulární, deska na sobě nemá stabilizátory napětí, ale má pro ně otvory pro připájení. Je to proto, že deska má hustotu páté konstrukční třídy, ale zdroj lze navrhnout v nižší třídě, která je levnější. Dále je to také proto, že se mi nezdálo nejvhodnější navrhnout desku závislou na konkrétním stabilizátoru a zvláště proto, že výrobci inovují a časem nabídnou novější stabilizátory s lepšími parametry a vyšší účinností, než jsou k dispozici dnes.

Vzhledem k tomu, že různí výrobci požívají na svých displejích různé konektory, vybral jsem univerzální konektor pro propojení s displejem s rozumnou roztečí, který se dá bez problémů zapájet a pro který se dá v domácích podmínkách vyrobit konektorová redukce pro příslušný displej. Na tento konektor lze také připojit i jiná zařízení, např. digitální obrazový senzor apod.
Pokud by bylo potřeba k desce připojit standardní LCD monitor, lze vytvořit desku, která na sobě bude mít DA převodníky a oddělovací obvody.

Výběr výrobce hradlového pole byl jednoduchý, protože v zaměstnání programuji hradlová pole firmy Xilinx a mám je v oblibě a také protože navrhový systém je pro vybrané obvody zdarma ke stažení na internetu, zvolil jsem právě výrobce Xilinx. Dalším krokem bylo vybrat vhodný typ hradlového pole. Kritériem pro výběr byla jeho velikost (počet LUT, BRAM apod.), počet I/O vývodů, typ konfigurační paměti, typ pouzdra a cena. Snahou bylo použít obvod, který má konfigurační data uložena v paměti, kterou lze jednoduše programovat z připojeného procesoru. Protože použití paměti programované prostřednictvím rozhraní JTAG se mi zdálo poněkud těžkopádné, zvolil jsem obvod, který má konfigurační data uložena v SPI paměti.

Dalším krokem bylo vybrat vhodný řídicí procesor, kritériem pro jeho výběr byl počet I/O vývodů, velikost jeho pamětí, dostupnost vývojových prostředků a jeho rozšířenost mezi konstruktéry elektronických zařízení. Zpočátku jsem uvažoval o rychlejším procesoru typu ARM, ale ten není tolik rozšířený a seznamování se s vývojovým prostředím pro tento procesor by zabralo příliš mnoho času. Proto jsem zvolil známější mikrokontrolér rodiny AVR.

U zvukového kodeku jsem chtěl, aby to nebyl pouze AD a DA převodník, ale aby měl také mikrofonní zesilovač. Vzhledem k tomu že v mém okolí se již jeden typ kodeku objevil, použil jsem stejný typ, ale s napájecím napětím 3V. Tento typ zvukového kodeku se ukázal výhodným také proto, že má zabudovány vnitřní filtry a nastavitelné vstupní a výstupní zesilovače.

Výstupní zesilovače pro reproduktory jsem chtěl co nejúčinnější a proto jsem vybral nejdostupnější obvody ve třídě D.

Dále jsem chtěl, aby deska obsahovala nějakou paměť RAM, ve které by mohla být uložena data nejen pro zobrazení. Statickou RAM jsem zavrhl z důvodu velkého počtu adresových vývodů i při její relativně malé kapacitě a také kvůli její velké proudové spotřebě. Lepší paměť se jevila buď paměť SDRAM nebo DDR SDRAM, ale vzhledem k tomu, že paměť DDR SDRAM má jiné napájecí napětí než ostatní obvody na desce, zvolil jsem paměť SDRAM.

Protože při oživování a používání desky jsou potřeba ovládací prvky jako tlačítka nebo klávesnice, umístil jsem na desku expander portů.

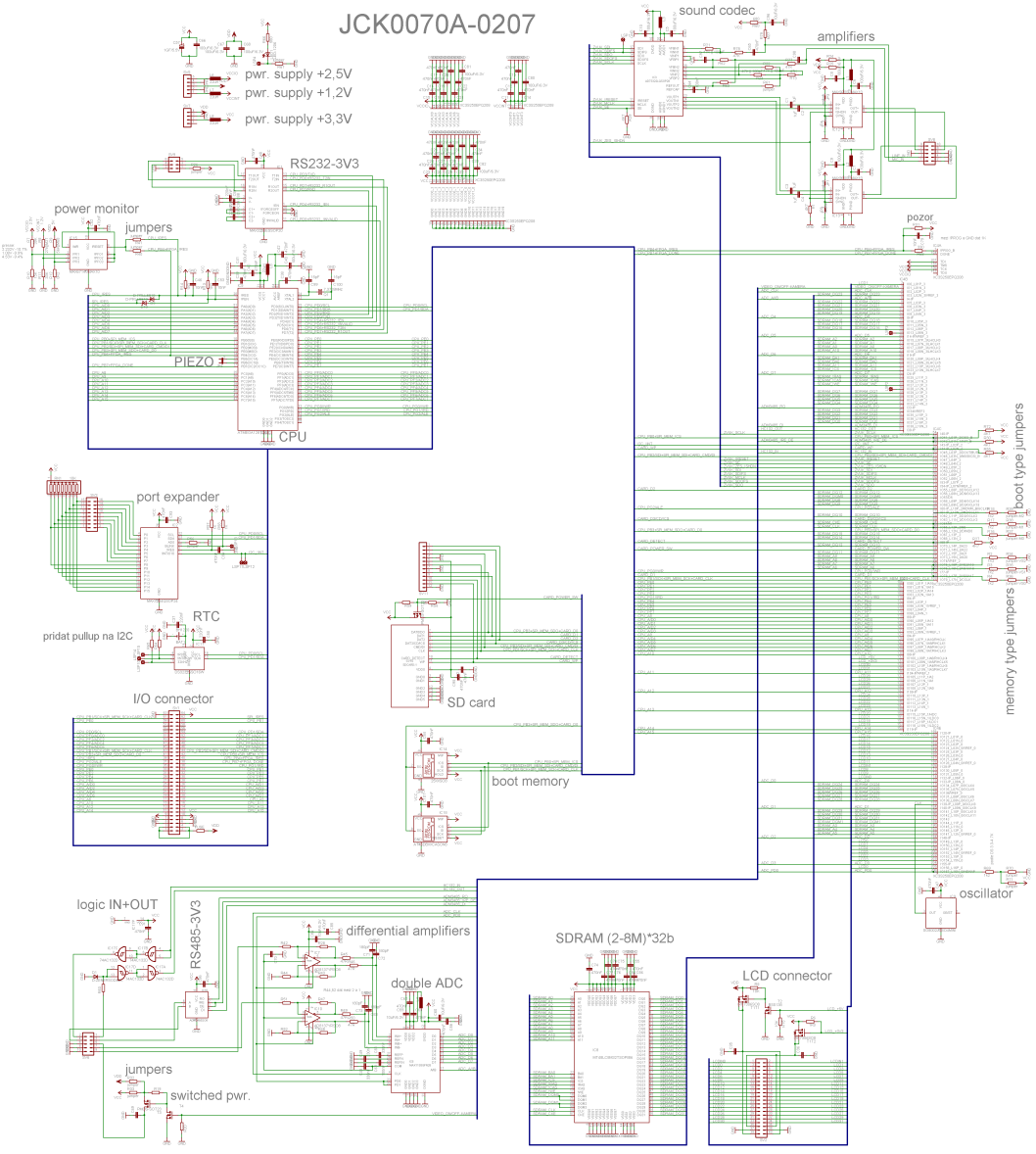
Dále jsem chtěl, aby deska měla obvod reálného času a proto jsem vybral nejdostupnější a nejpřesnější, který byl v době návrhu na trhu.

Dalším požadavkem bylo, aby deska měla analogový vstup pro připojení analogového signálu, např. ČB kamery. Vybral jsem obvod s nejmenším počtem vývodů a s malou spotřebou.

Kapitola 3

Popis funkčních částí desky

V této kapitole jsou popsány jednotlivé funkční části desky.
Značení konektorů je převzato z katalogu firmy GM electronics.
V textu, kde jsou uvedeny hodnoty součástek, jsou vybrány typické hodnoty z katalogového listu.
Ve schématu je FPGA (IC4) typ XC3S250E místo XC3S500E, protože v knihovně Eagle mám pouze XC3S250E. Tato skutečnost nevadí, protože oba typy mají stejné rozložení vývodů.
Pro lepší orientaci jsou v částech schématu umístěny obrázky konektorů při pohledu zhora, dále jsou zde některé pomocné texty napsány anglicky, protože české označení je oproti anglickému poněkud těžkopádné a anglické označení bývá známější nebo výstižnější.



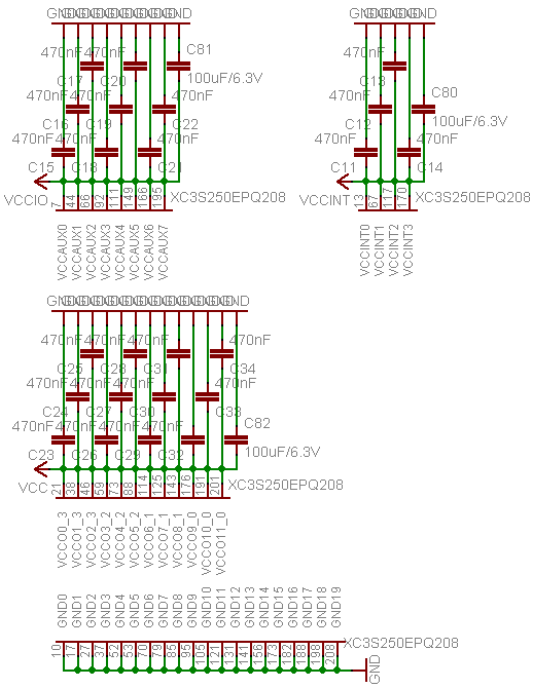
Obrázek 3.1: celkové schéma

3.1 FPGA (IC4)

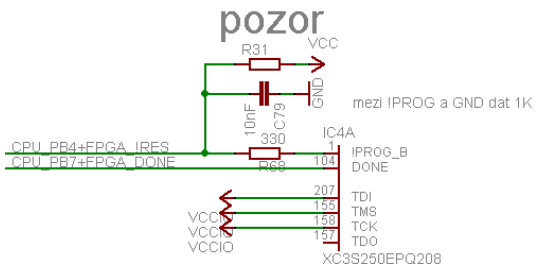
Použité FPGA je typ XC3S500E-5PQG208C v pouzdru PQFP208, výrobce Xilinx. Tento obvod je srdcem celé desky a je připojen na I/O a LCD konektor, AVR, SPI a SDRAM paměť, AD převodník, logický vstup a výstup, ADM3485 a zvukový kodek. Jeho (vybrané) parametry jsou:
ekvivalentní počet hradel - 500 000
počet LUT - 9312
počet BRAM - 20 (kapacita jedné BRAM je 18Kbit)
počet I/O vývodů - 126
počet vstupních vývodů - 32

Propojky R37-R39 určují způsob konfigurace FPGA.
Propojky R34-R36 vybírají typ konfigurační SPI paměti pro FPGA.
Propojka R70 zapíná pullup rezistory na vývodech FPGA během konfigurace, propojka R71 je vypíná.
Rezistory R2 - R4, R16 - R18 a R69 oddělují vývody FPGA od propojek a umožňují tak FPGA, po nakonfigurování, měnit stav na těchto vývodech, nezávisle na propojkách.
R27 je pullup rezistor detekce přítomnosti SD-karty v konektoru.
R30 je pullup rezistor pro deaktivaci ADM3485 při konfigurování FPGA.
R55 je pullup rezistor pro detekci WP (zákaz zápisu - spínač na SD-kartě) SD-karty.
R72 je pullup rezistor pro deaktivaci SPI paměti při konfigurování FPGA.
Rezistor R68 převádí 3,3V signál z AVR na 2,5V signál pro FPGA.
Kondenzátor C79 slouží k odfiltrování případného rušení.

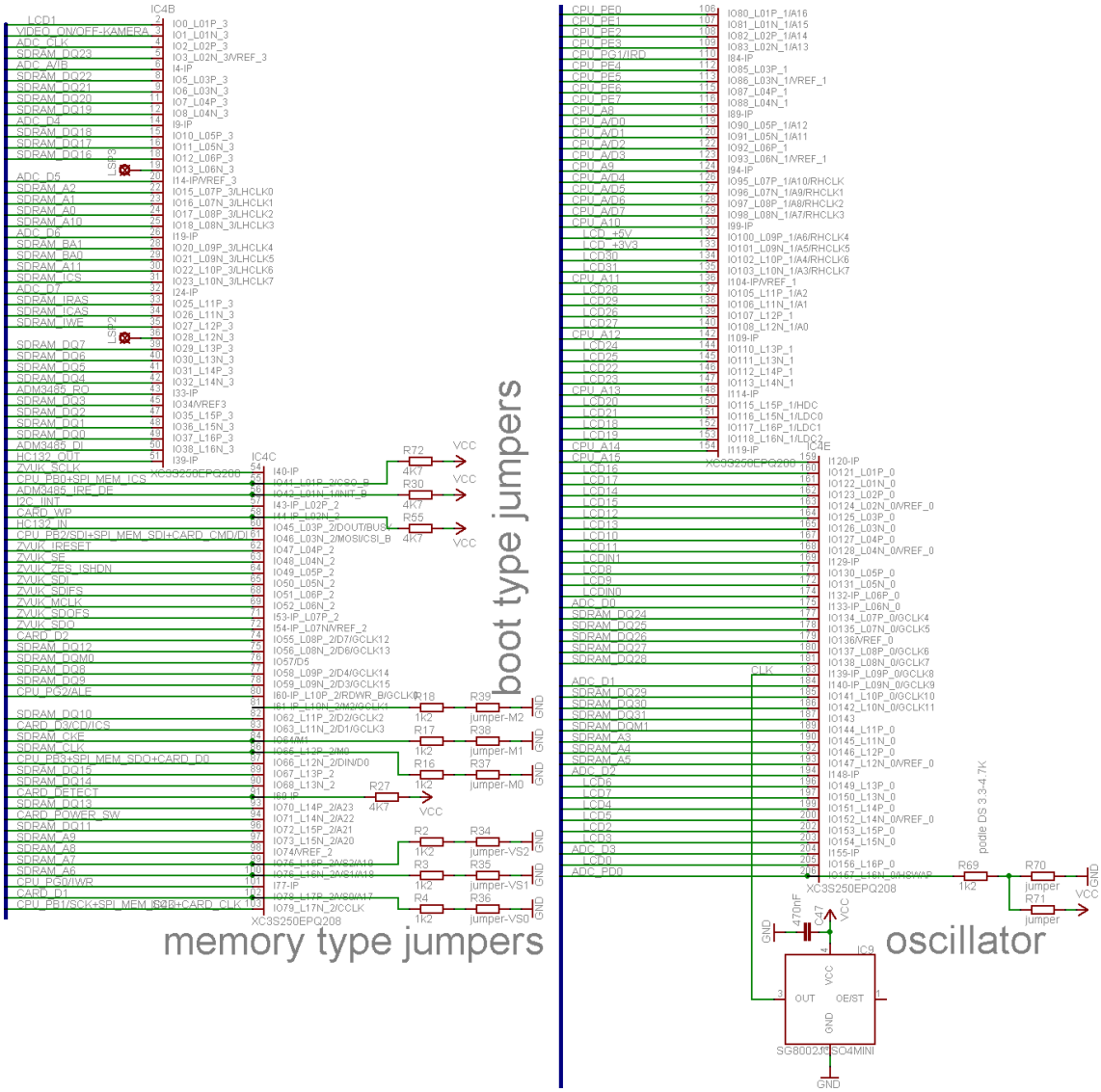
Dva I/O vývody FPGA jsou vyvedeny na prokovy LSP2 a LSP3 a jsou použity při oživování desky a nebo pro libovolný účel.
Jako zdroj hodinového taktu pro FPGA je použit oscilátor SG8002, firmy EPSON, pro napájecí napětí 3,3V a naprogramovaný na výstupní frekvenci 50 MHz.



Obrázek 3.2: zapojení FPGA, část A - napájení



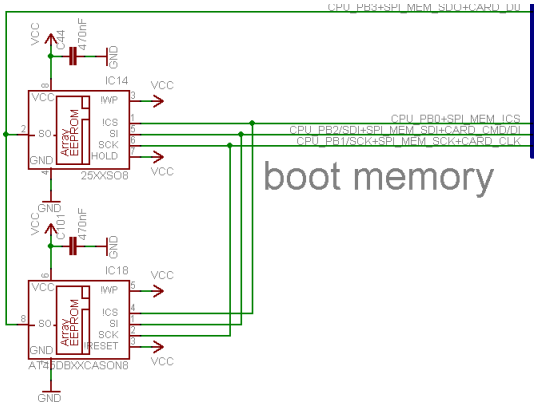
Obrázek 3.3: zapojení FPGA, část B - inicializace, JTAG vývody



Obrázek 3.4: zapojení FPGA, část C - I/O vývody, oscilátor, propojky apod.

3.2 Konfigurační paměť (IC14 nebo IC18) pro FPGA

Použitá SPI paměť je řady 25XX nebo řady AT45DBXX výrobce Atmel (pouze jedna nebo druhá, nikdy ne zároveň obě !!!) o kapacitě 4Mbity (pro XC3S250 je dostatečná kapacita 2Mbity), obě v pouzdru SO8 nebo se stejnou roztečí vývodů.
Je připojena k AVR tak, aby do (z) ní mohlo AVR zapisovat (číst) data prostřednictvím SPI rozhraní a je také připojena na I/O konektor, jehož prostřednictvím ji lze programovat (číst).

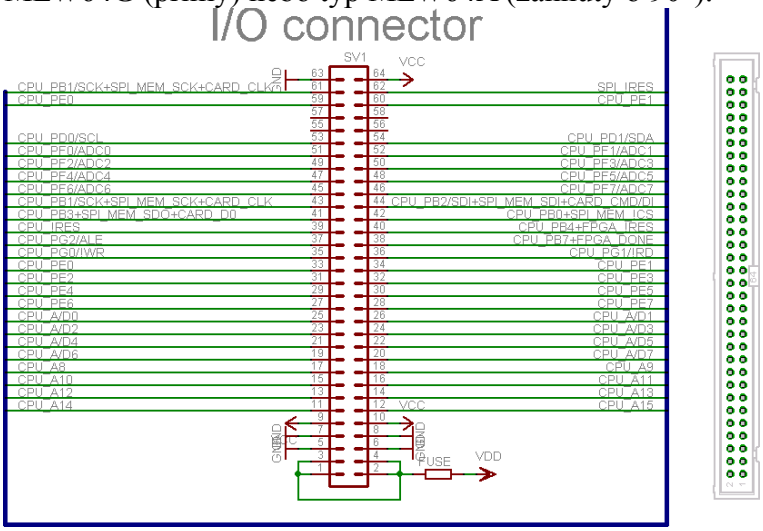


Obrázek 3.5: zapojení konfigurační paměti

3.3 I/O konektor (SV1)

Tímto konektorem je napájena celá deska a je na něj přivedena datová / adresová sběrnice AVR, některé další porty AVR, některé vývody FPGA, SPI rozhraní pro programování AVR a konfigurační paměti FPGA, I^2C rozhraní. Jeho čtyři piny nejsou zapojeny, a lze na ně v případě potřeby připojit potřebné signály.

Konektor je typ MLW64G (přímý) nebo typ MLW64A (zahnutý o 90°).



Obrázek 3.6: zapojení I/O konektoru

3.4 Mikrokontrolér (IC5)

Použitý mikrokontrolér je typ ATMEGA128L rodiny AVR v pouzdru TQFP64, výrobce Atmel.
Suffix L znamená, že mikrokontrolér je určen pro napájecí napětí 2,7V - 5,5V a frekvenci hodinového taktu max. 8 MHz.
Jeho (vybrané) parametry jsou:
frekvence vnitřního taktu 1/2/4/8 MHz / externího 0-8 MHz (pro komunikaci rozhraním RS232 je nejvhodnější použít externí krystal s frekvencí rovnou násobku baudové rychlosti, např. 7,3728 MHz)
vnitřní programová a datová FLASH - 128 kB
vnitřní datová EEPROM - 4 kB
vnitřní datová SRAM - 4 kB
externí paměťový prostor až do 64 kB

I/O porty AVR jsou použity následovně:
PORTA - LSB část adresové / datové sběrnice, připojen na FPGA a na I/O konektor
PA0-PA7 = AD0-AD7

PORTB - SPI rozhraní, piezo, apod., část připojena na I/O konektor
PB0 = !CS pro konfigurační SPI paměť, připojen na I/O konektor
PB1 = SCK pro SPI rozhraní a SD-kartu, připojen na I/O konektor
PB2 = SDI pro SPI rozhraní a DI/CMD SD-karty, připojen na I/O konektor
PB3 = SDO pro SPI rozhraní a DO SD-karty, připojen na I/O konektor
PB4 = !PROGRAM pro FPGA - vymazání aktuálního obsahu FPGA a jeho rekonfigurace, připojen na I/O konektor
PB5 = PWM výstup OC1A pro piezoměnič
PB6 = PWM výstup OC1B pro piezoměnič
PB7 = signál DONE z FPGA - informace, že FPGA načetlo konfigurační data z SPI paměti, připojen na I/O konektor

PORTC - MSB část adresové sběrnice, připojen na FPGA a na I/O konektor
PC0-PC7 = A8-A15

PORTD - rozhraní I^2C a RS232
PD0 = SCL rozhraní I^2C
PD1 = SDA rozhraní I^2C
PD2 = RxD rozhraní RS232
PD3 = TxD rozhraní RS232
PD4 = !EN obvodu MAX3223E
PD5 = !INVALID z obvodu MAX3223E
PD6 = výstup rozhraní RS232, pro libovolné použití, např. řízení toku dat
PD7 = vstup rozhraní RS232, pro libovolné použití, např. řízení toku dat

PORTE - připojen na FPGA a na I/O konektor, s aktuálním programem je použit jako MSB část datové sběrnice (D8-D15)

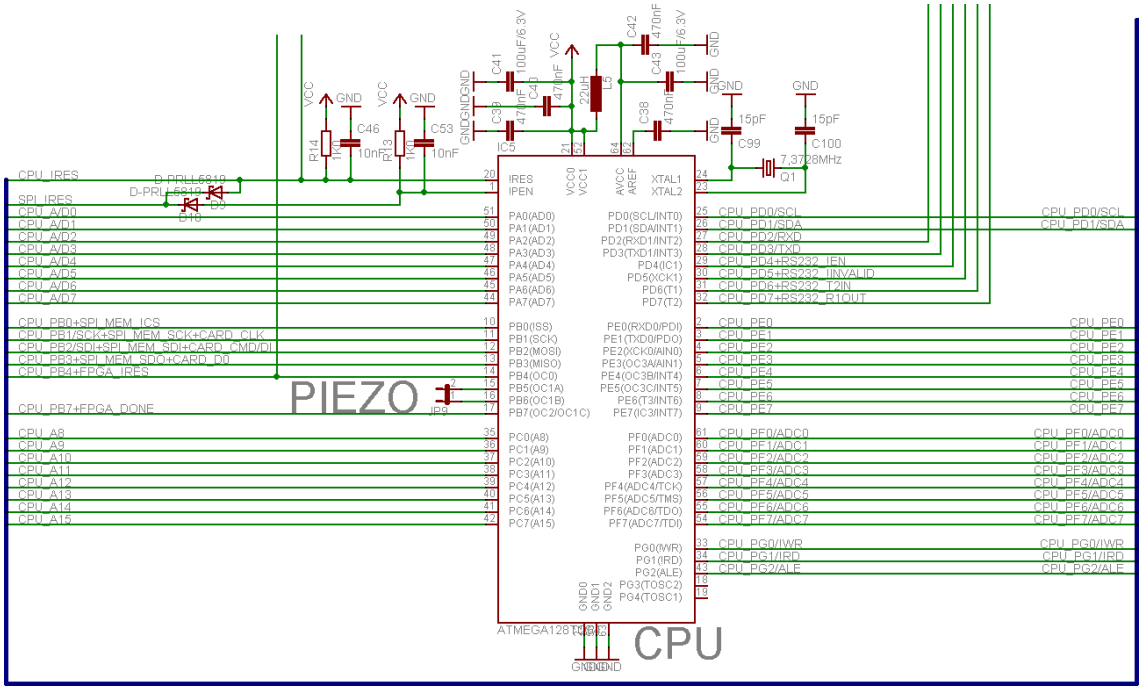
PORTF - libovolné použití, také zastává funkci analogových vstupů, připojen na I/O konektor

PORTG - řídicí signály, část připojena na I/O konektor

PG0 = !WR, připojen na I/O konektor
PG1 = !RD, připojen na I/O konektor
PG2 = ALE, připojen na I/O konektor
PG1 = nezapojen
PG2 = nezapojen

Piezoměnič je připojen mezi dva PWM výstupy, které s aktuálním programem běží v protifázi, a má proto dvakrát vyšší hlasitost, než při běžném připojení mezi výstup a GND (VCC).

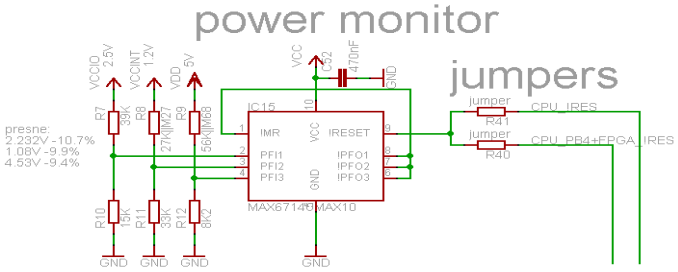
Signál SPI_!RES z I/O konektoru slouží k resetování a uvedení AVR do programovacího módu. Pokud je tento signál držen na log. nule při připojení desky na napájecí napětí, pak se AVR pomocí signálu !PEN uvede do programovacího módu a zůstává v něm trvale až do odpojení napájecího napětí. Když není osazena dioda D10, stejnou funkci zastává signál !RES, ale po deaktivaci SPI_!RES se ihned spustí běh programu AVR.
Dioda D9 od sebe odděluje AVR vstupy !RES od !PEN a také vytváří z push-pull signálu SPI_!RES signál s vlastnostmi otevřeného kolektoru, aby se dalo AVR resetovat signálem z MAX6714.
Kondenzátory C46 a C53 slouží k odfiltrování případného rušení.



Obrázek 3.7: zapojení AVR

3.5 Resetovací obvod (sledovač napájecích napětí - IC15)

Použitý obvod je typ MAX6714 v pouzdru µMAX10, výrobce Maxim.
Tento obvod sleduje velikost napájecích napětí VCC, VDD, VCCINT, VCCIO a při jejich poklesu resetuje AVR nebo FPGA výstupem !RESET (otevřený kolektor). Výběr, který z obvodů bude resetován, se provádí propojkami R40/R41.
Odporové děliče R7/R10, R8/R11, R9/R12 určují velikost rozhodovací úrovně sledovaného napájecího napětí. Poklesem napětí pod rozhodovací úroveň (podle kat. listu 0,62V) na vstupu PFI1 (PFI2 nebo PFI3) se aktivuje !PFO1 (!PFO2 nebo !PFO3.
Výstupy !PFOx mají otevřený kolektor a jsou připojeny na !MR (převzato z kat. listu).
Jeho log. nulou se aktivuje výstup !RESET na min. 140ms. Výstup !RESET se také aktivuje poklesem VCC pod úroveň danou použitým obvodem (MAX6714CUB 3.3V-10%, MAX6714DUB 3.3V-5%).
Napájecí napětí VCC se sleduje vždy (je to hlavní napájecí napětí), sledování ostatních se dá vyřadit vhodnou volbou odporových děličů, nebo přeškrábnutím příslušného spoje !PFOx do !MR na desce.

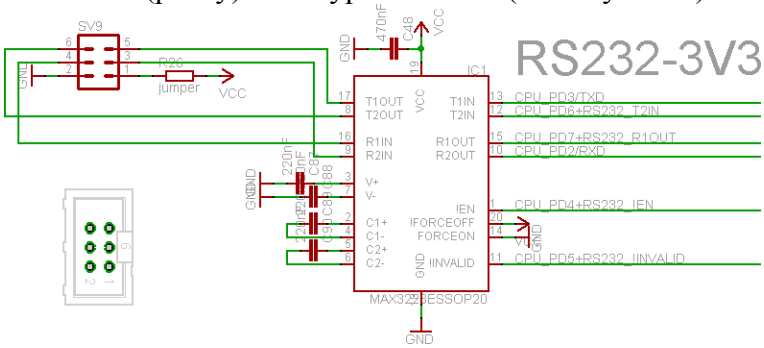


Obrázek 3.8: zapojení resetovacího obvodu

3.6 Rozhraní RS232 (IC1 + SV9)

Použitý obvod je typ MAX3223E v pouzdru SSOP20, výrobce Maxim.
Je určen pro napájecí napětí 3 - 5,5V (na této desce má napájení 3,3V) a převádí signály TTL (LVTTL) na signály o napěťových úrovních RS232.
Tento obvod automaticky vypíná RS232 výstupy, když na jeho RS232 vstupy není připojena platná logická úroveň, tuto skutečnost signalizuje výstupem !INVALID.
Také má vestavěnou vnitřní ochranu proti elektrostatickému výboji až do 15kV (Human Body Model).

Konektor je typ MLW06G (přímý) nebo typ MLW06A (zahnutý o 90°).

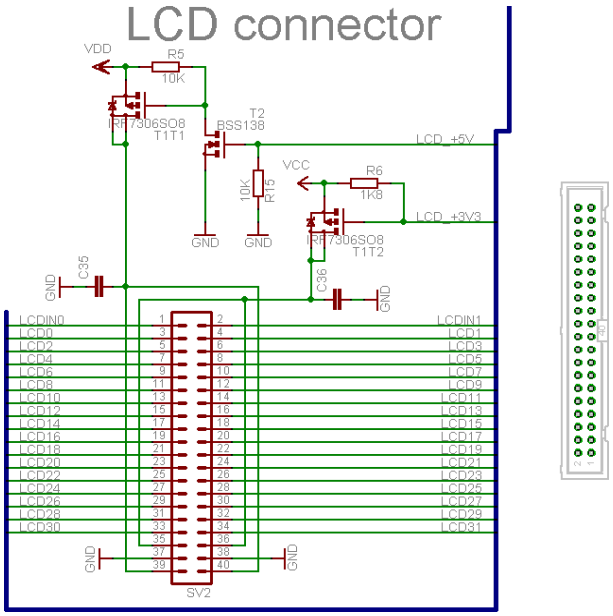


Obrázek 3.9: zapojení RS232

3.7 LCD konektor (SV2)

Tento konektor je připojen na 2 vstupní a 32 I/O vývodů FPGA, je jím napájen a řízen připojený displej, lze ho ale použít i pro jiné účely.
Napájení displeje se zapíná pomocí tranzistorů T1T1, T1T2 a T2, které jsou řízeny pomocí FPGA. T1T1 a T1T2 jsou dva tranzistory PMOS v jednom pouzdru SO8, mohou být i jiný typ než je ve schématu, ale s co nejmenším odporem v sepnutém stavu (R_{DSON}) a spínající při malém napětí (5 V). T2 je libovolný, pinově kompatibilní, NMOS tranzistor spínající při malém napětí (3,3 V).
Rezistory R5, R6 a R15 drží tranzistory zavřené během načítání konfiguračních dat obvodem FPGA.

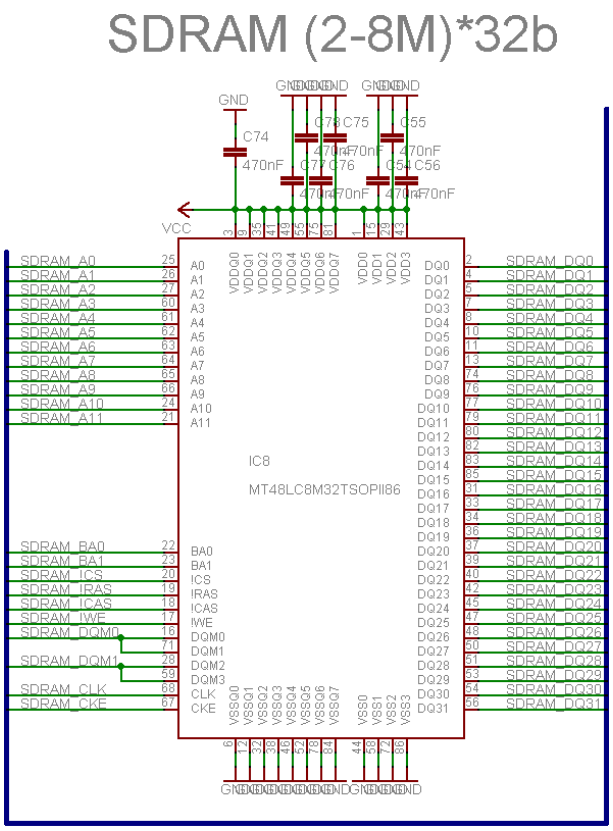
Konektor je typ MLW40G (přímý) nebo typ MLW40A (zahnutý o 90°).



Obrázek 3.10: zapojení LCD konektoru

3.8 Paměť SDRAM (IC8)

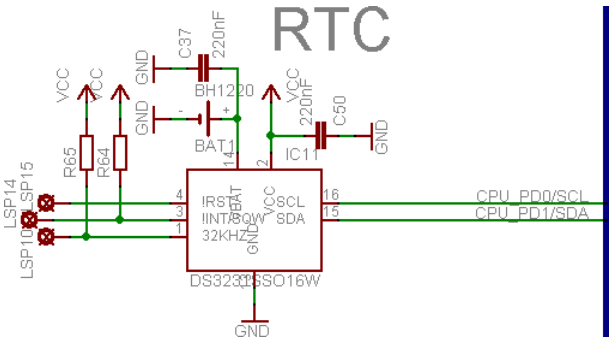
Deska je navržena pro paměť MT48LC8M32 v pouzdru TSOP 86, výrobce Micron.
Je to paměť typu SDRAM o kapacitě 8M * 32b, pro frekvenci až do 166 MHz.
Lze použít i paměti s menší kapacitou, jiných výrobců i jiných rychlostí, měly by být pinově kompatibilní, ale vždy je to třeba před osazením zkontrolovat.
Šířka datové sběrnice byla zvolena 32 bitů z důvodu rychlejšího přístupu do paměti např. při použití displeje s vyšším počtem barev než 16 bitů.
Čtyři vstupy DQM0-3 pro povolení jednotlivých bytů jsou, kvůli nedostatku I/O vývodů FPGA, po dvojicích spolu propojeny. Všechny ostatní řídicí, adresové a datové vývody paměti jsou připojeny na FPGA.



Obrázek 3.11: zapojení paměti SDRAM

3.9 RTC (obvod reálného času - IC11)

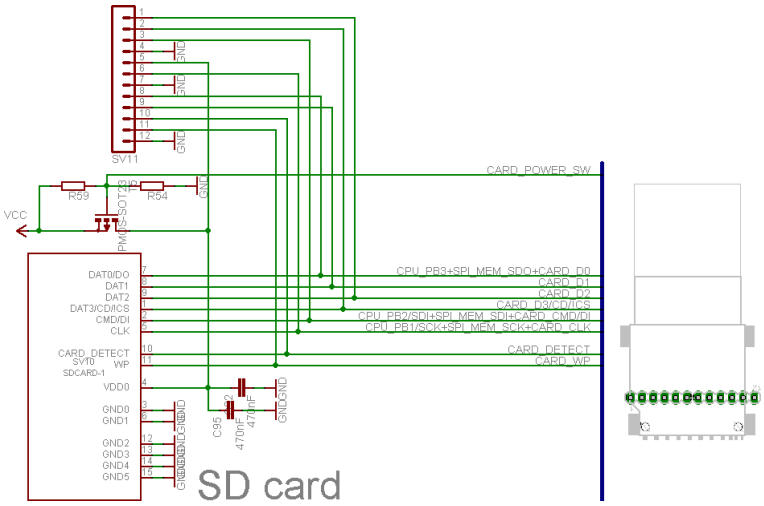
Použitý obvod je typ DS3231 v pouzdru SO16W, výrobce Maxim.
Tento obvod se sám přepíná na záložní baterii při poklesu napájecího napětí, má vnitřní teplotně kompenzovaný krystal a pravidelně provádí kalibraci jeho frekvence.
Je připojen na rozhraní I^2C a některé jeho vývody jsou vyvedeny na prokovy LSP 10, LSP14 a LSP15.



Obrázek 3.12: zapojení obvodu reálného času (RTC)

3.10 Paměťová SD-karta (SV10 nebo SV11)

SD-karta je připojena na SPI rozhraní (z důvodu úspory vývodů FPGA) a na FPGA.
Napájení pro kartu se zapíná tranzistorem T5 spínajícím při malém napětí (3,3V), ten je při konfiguraci FPGA možno držet zavřený rezistorem R59, nebo sepnutý rezistorem R54.
Pro případ nedostupnosti konektoru SV10 (konektor pro SD-kartu), je na desce umístěn konektor SV11 pro připojení dostupnějšího konektoru. Proto jsou konektory SV10 a SV11 umístěny na desce přes sebe.
Paměťová SD-karta je sice připojena na SPI rozhraní, ale na této desce se s ní dá komunikovat pouze pomocí FPGA. Může se to jevit jako nevýhoda, ale je třeba si uvědomit, že paměťová karta zvládá komunikaci na mnohem vyšší frekvenci (až 25 MHz) než běží AVR a než je vůbec schopno programově „mávat“ piny. Proto by byla škoda zpomalovat si s ní komunikaci připojením na samotné SPI rozhraní, a také proto, že karta zvládá komunikaci po čtyřech bitech, na rozdíl od jednobitového SPI rozhraní.



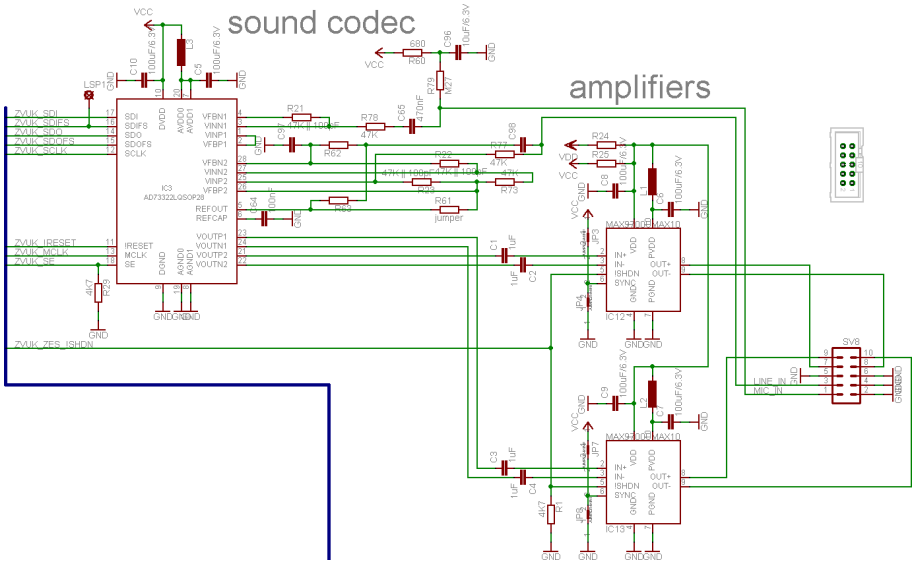
Obrázek 3.13: zapojení SD-karty

3.11 Zvuková část (IC3 + IC12 + IC13 + SV8)

Použitý kodek je typ AD73322L v pouzdru QSOP20, výrobce Analog Devices.
Obsahuje vstupní a výstupní nastavitelné zesilovače, filtry a dva 16-ti bitové AD a DA převodníky se vzorkovací frekvencí do 64kHz.
Rezistory a kondenzátory, na které je přiveden linkový vstup, jsou propojeny univerzálně tak aby se dala použít různá zapojení podle kat. listu.
Ve schématu, kde je u rezistoru napsáno např. 47K||100pF, je na rezistoru o hodnotě 47KΩ umístěn kondenzátor s kapacitou 100pF.

Výstupní zesilovače jsou typ MAX9700 v pouzdru µMAX10, výrobce Maxim.
Jsou to zesilovače třídy D s diferenciálními vstupy a výstupy.
Napájení pro oba dva se vybírá propojkou R24/R25. R24 vybírá 5V, R23 vybírá 3,3V (nikdy nezapojovat současně !!!).
Propojkou JP3/JP4 nebo JP7/JP8 se vybírá modulační frekvence zesilovače.

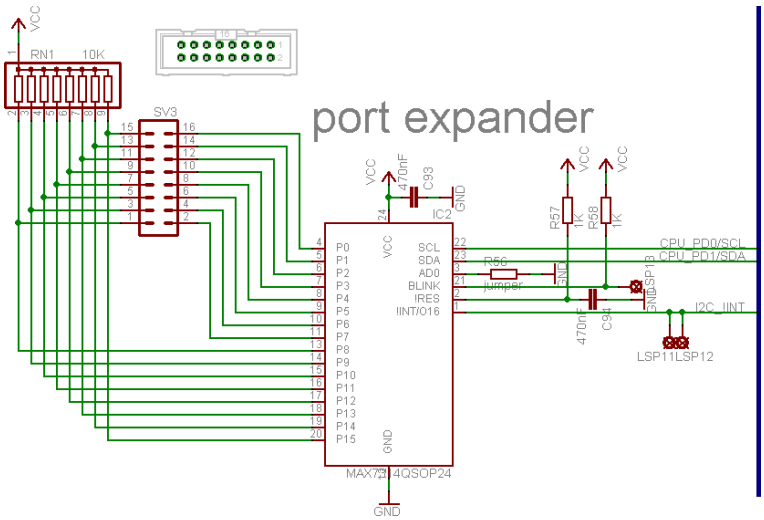
Konektor je typ MLW10G (přímý) nebo typ MLW10A (zahnutý o 90°).



Obrázek 3.14: zapojení zvukové části

3.12 Expander pro klávesnici (IC2 + SV3)

Použitý obvod je typ MAX7314 v pouzdru QSOP24, výrobce Maxim.
Je připojen na konektor SV3, FPGA a na rozhraní I^2C. Tento obvod je na desce použit pro připojení ovládacích tlačítek nebo klávesnice. Lze s ním ovládat LED, měnit intenzitu jejich svitu lze, když je na jeho vstup BLINK připojen zdroj hodinového signálu. Vývody připojené na konektor lze nakonfigurovat jako vstupní, výstupní nebo s otevřeným kolektorem. U vstupních vývodů signalizuje změnu stavu výstupem !INT, pokud je tak nakonfigurován. Protože nezbyl vývod na AVR pro vyvolání přerušení, je vývod !INT připojen na FPGA.
Konektor je typ MLW16G (přímý) nebo typ MLW16A (zahnutý o 90°).



Obrázek 3.15: zapojení expanderu portu

3.13 Diferenciální zesilovače (IC7 + IC10), AD převodník (IC6), logický vstup a výstup (IC17), rozhraní RS485 (IC16) a SV6

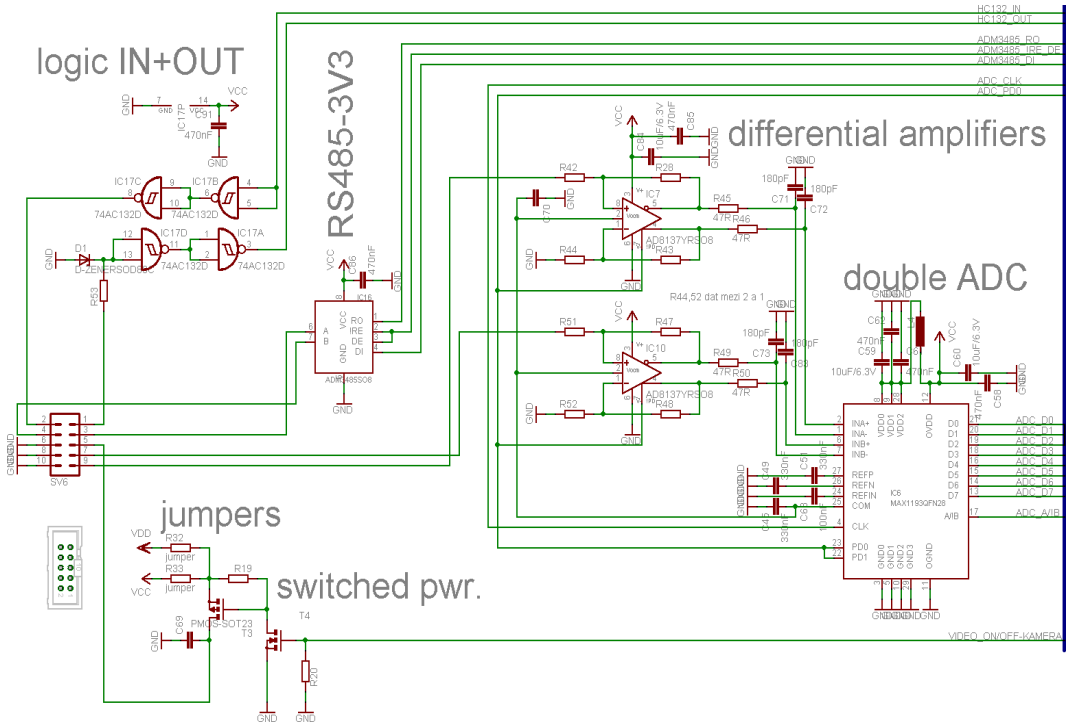
AD8137 jsou diferenciální zesilovače v pouzdru SO8, výrobce Analog Devices.
Na této desce převádějí single-ended signál na diferenciální signál, který je následovně připojen na AD převodník MAX1193.

MAX1193 je dvojitý AD převodník v pouzdru QFN28, výrobce Maxim.
Tento převodník je připojen na FPGA.
Jeho (vybrané) parametry jsou :
dvojitý osmibitový AD převodník
45 MSPS
diferenciální vstupy s napěťovým rozsahem ±0,512 V
multiplexovaný datový výstup
nízká spotřeba 27 mA (standby 10 µA)

74AC132 slouží k bezpečnému oddělení FPGA od logických signálů na konektoru SV6.
Zenerova dioda D1 omezuje napětí na vstupu 74AC132, rezistor R53 omezuje proud diodou D1.

ADM3485 je převodník LVTTL signálu na diferenciální signál RS485 v pouzdru SO8, výrobce Analog Devices. Požít lze i rychlejší typ ADM3485E se zabudovanou ochranou proti elektrostatickému výboji.

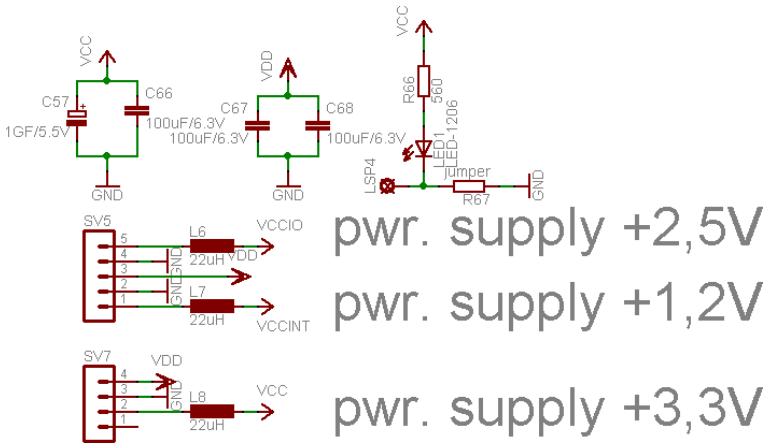
T4 je libovolný tranzistor NMOS, který spíná při 3,3V. T5 je tranzistor PMOS s malým odporem (R_{DSON}), který spíná při 3,3V (5V - podle propojky). Propojka R32/R33 vybírá napětí, které bude připínáno na konektor SV6.
Konektor je typ MLW10G (přímý) nebo typ MLW10A (zahnutý o 90°).



Obrázek 3.16: zapojení analogového vstupu, logického vstupu a výstupu, RS485

3.14 Ostatní

Dioda LED je použita při oživování desky a lze ji připojit na FPGA.
Do konektorů SV5 a SV7 se připojují napájecí moduly se stabilizátory napětí.
Napájení desky je filtrováno tlumivkami L6 - L8 a kondenzátory v příslušné napájecí větvi.



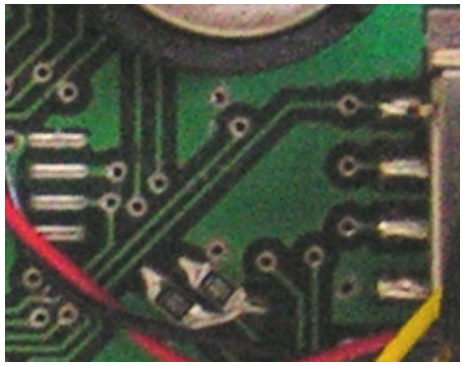
Obrázek 3.17: zapojení LED a napájení desky

Kapitola 4

Chyby na desce

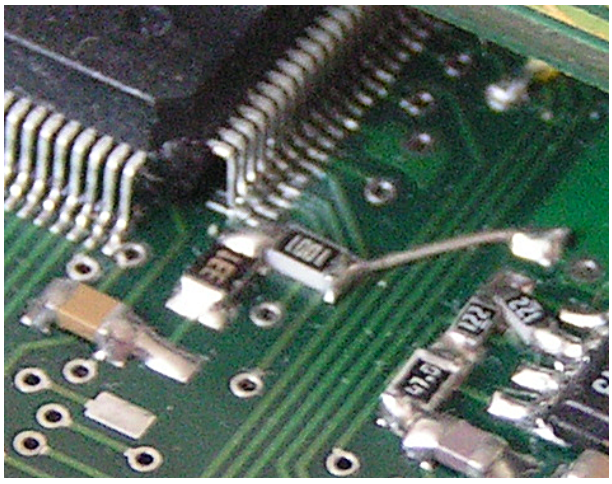
Jako na každém zařízení vytvořeném člověkem, jsou i na této desce chyby, jsou to:

a) Rozhraní I^2C nemá pull-up rezistory



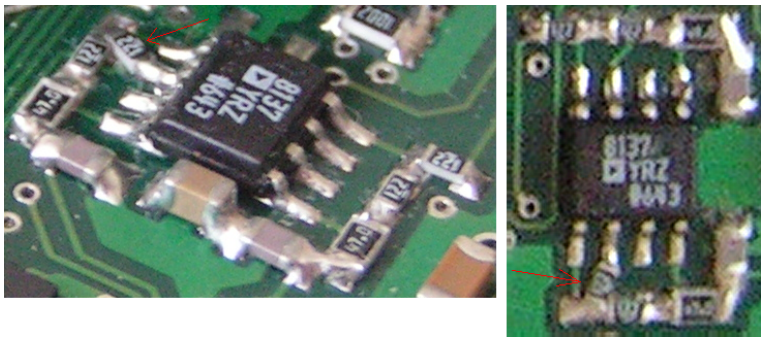
Obrázek 4.1: doplnění rezistorů pro I^2C

b) Při oživování se ukázalo, že pro konfiguraci s FPGA resetovaným AVRkem je vhodnější FPGA držet resetované, protože jinak se po zapnutí snaží načíst konfigurační data a pokud je vymazáno AVR, tak to ruší jakoukoliv komunikaci programátoru s AVR nebo s SPI pamětí a oboje tak nelze nahrát.
Pro konfiguraci, kdy je FPGA resetované samotným MAX6714, tato úprava neplatí.



Obrázek 4.2: doplnění rezistoru pro reset FPGA

c) Pro některé vstupní napěťové rozsahy AD8137 je třeba rezistory R44 a (nebo) R52 připojit mezi vývody 1 a 2 u AD8137.

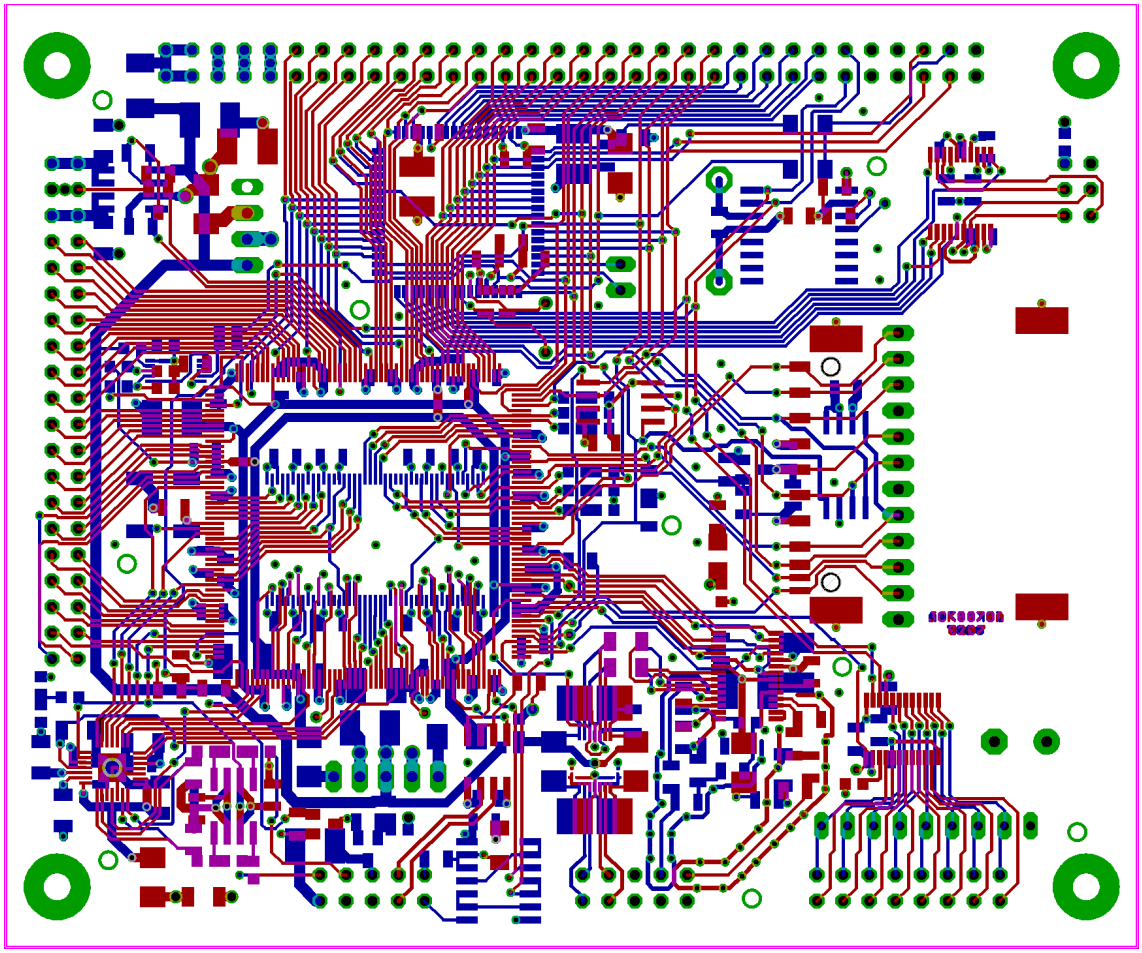


Obrázek 4.3: úprava umístění rezistorů pro AD8137

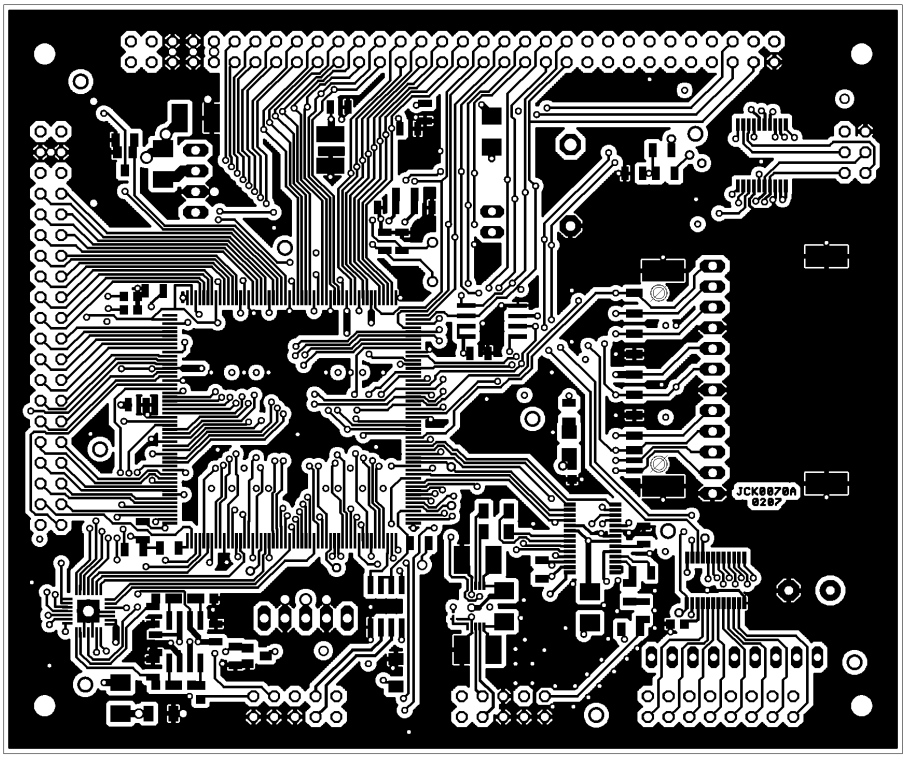
Kapitola 5

Spoje desky a rozmístění součástek na desce

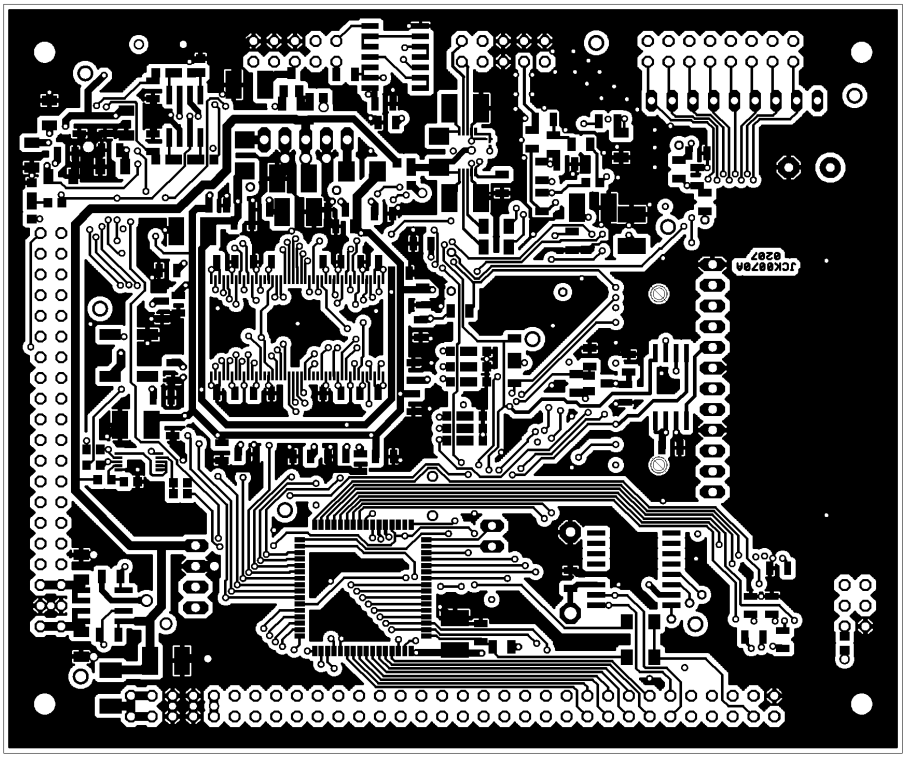
V této kapitole nejsou přiloženy obrázky vnitřních vrstev, protože nesdělují nějakou zásadní informaci, lze si je prohlédnout v originálním tvaru na přiloženém DVD.



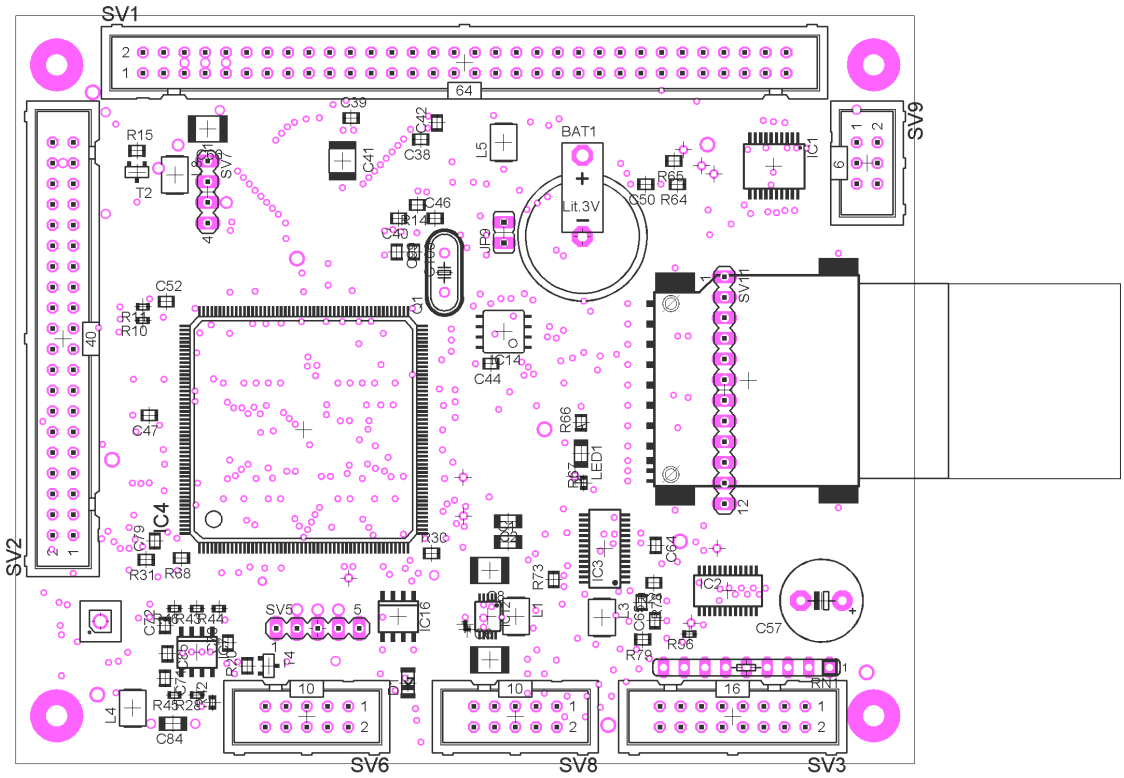
Obrázek 5.1: spoje desky z obou stran



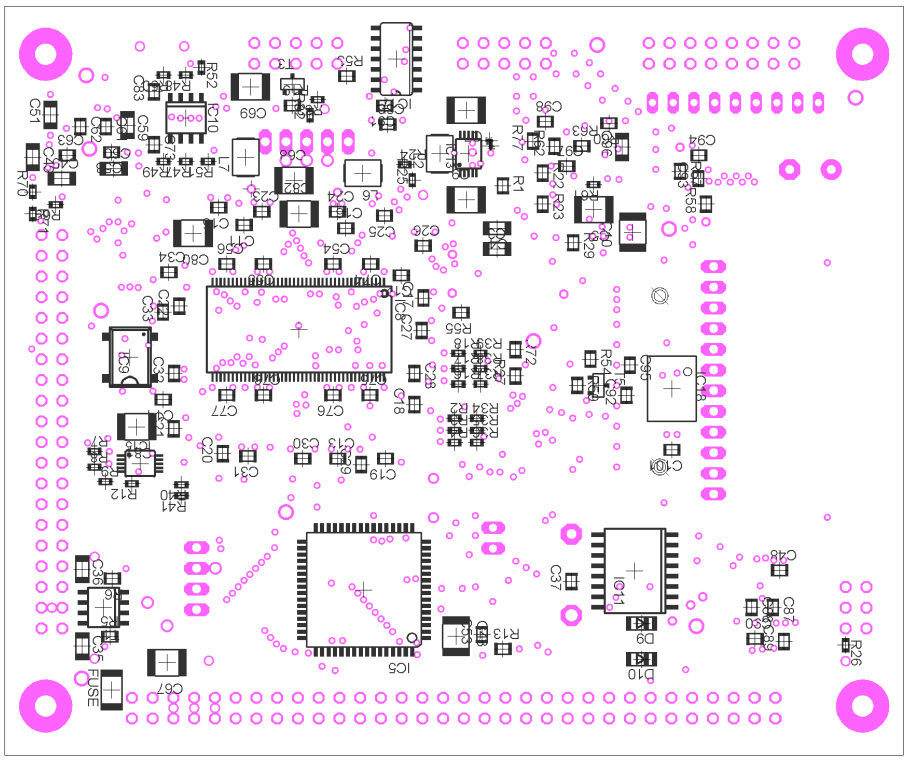
Obrázek 5.2: vrchní strana spojů desky



Obrázek 5.3: spodní strana spojů desky



Obrázek 5.4: rozmístění součástek na desce - vrchní strana



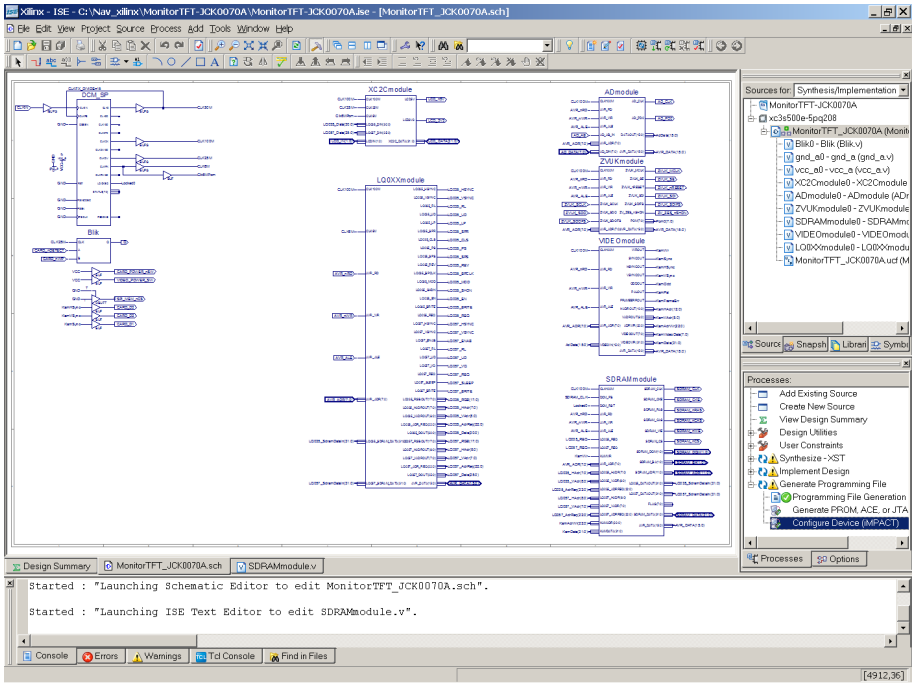
Obrázek 5.5: rozmístění součástek na desce - spodní strana

Kapitola 6

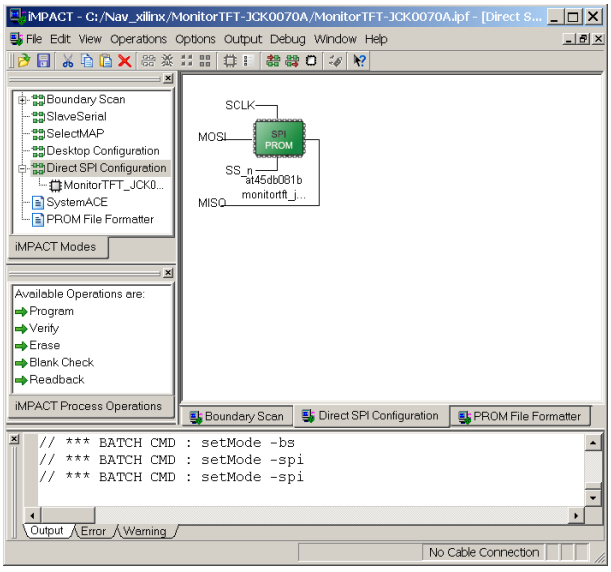
Popis programů pro desku

6.1 Popis programu pro FPGA

Program pro FPGA je vytvořen a přeložen v návrhovém systému ISE WebPACK, verze 8.2.03i, firmy Xilinx.

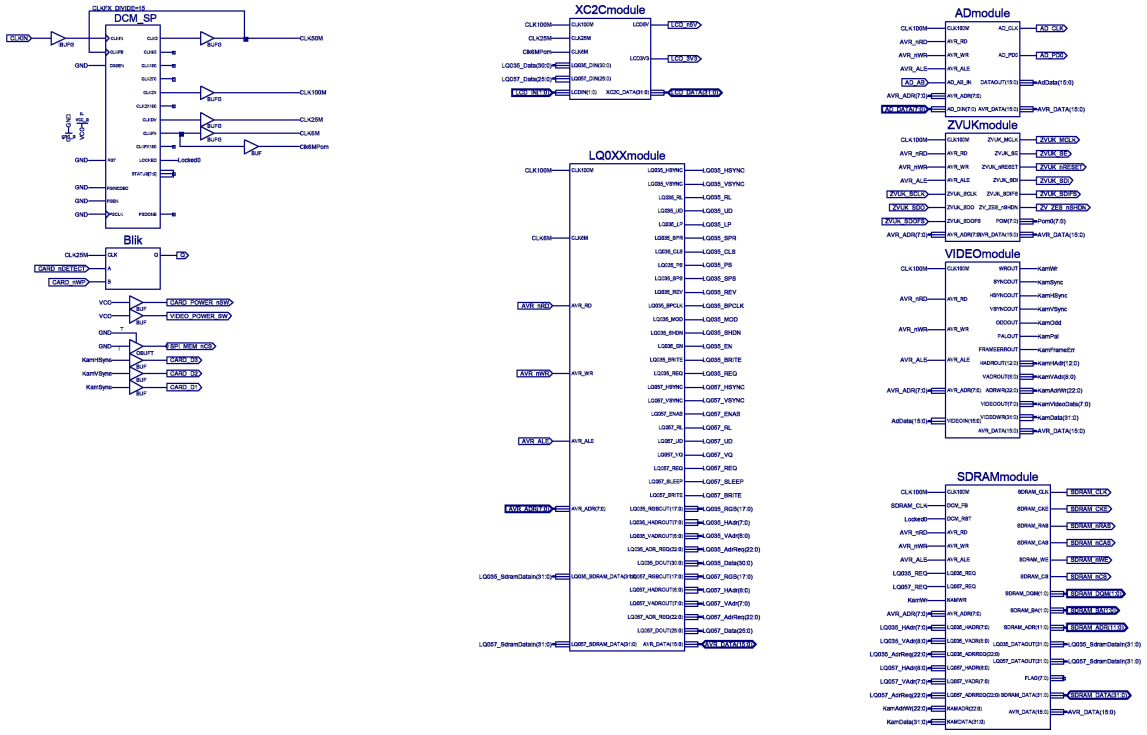


Obrázek 6.1: návrhový systém ISE WebPACK



Obrázek 6.2: nahrávací program iMPACT

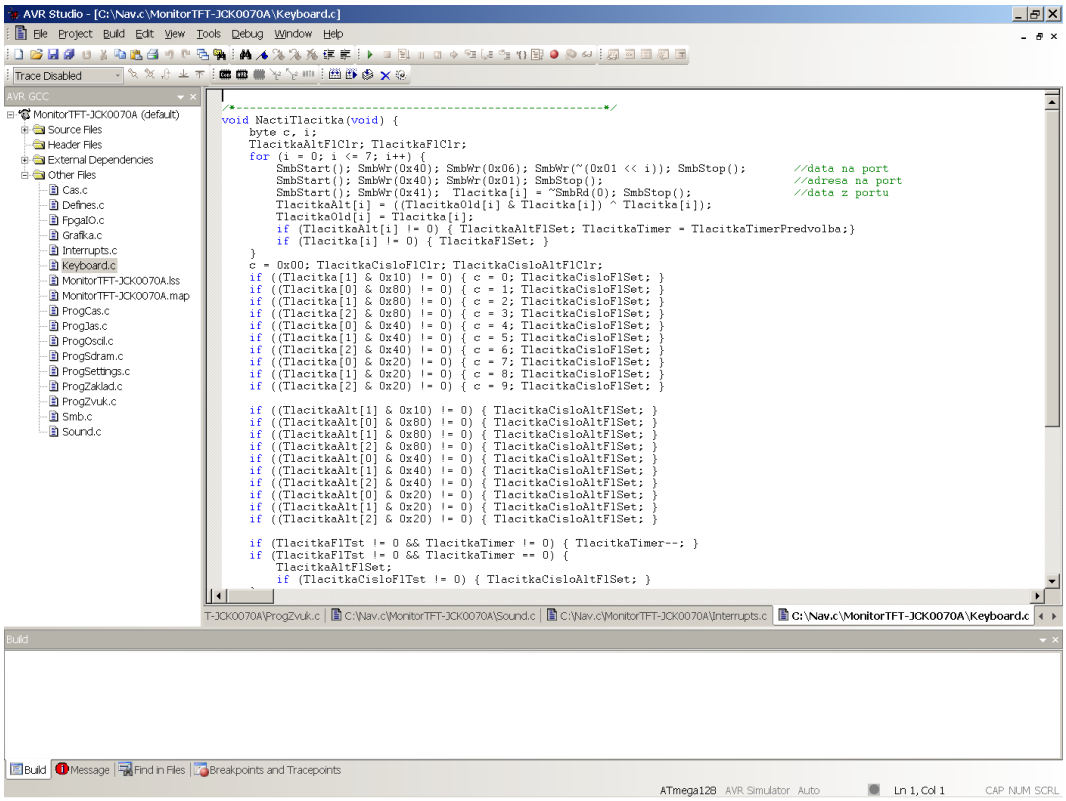
Program pro FPGA je tvořen moduly, které jsou mezi sebou propojeny.
Hlavní částí programu pro FPGA je řadič displeje, jeho součástí je znakový generátor s výběrem barvy pro písmo a pozadí, a generátor svislých čar (pro funkci osciloskopu). Jeho součástí je také funkce blikání znaku a „průhledná“ barva, která odkrývá data z SDRAM a generátoru čar.
Další modul je řadič paměti, který zapisuje data do SDRAM ze své FIFO a na požádání načítá obrazový řádek pro řadič displeje, v budoucnu i pro AVR.
Zvukový modul posílá data ze své FIFO do zvukového kodeku a také je z něj přijímá do své druhé FIFO, do obou pamětí FIFO má přístup AVR.
AD modul slouží k oddělení jednotlivých kanálů z AD převodníku.
Video modul dekóduje obrazový signál z AD modulu a žádá řadič paměti o jeho zápis do SDRAM.
Jednotlivé moduly jsou o rozsahu 50 - 500 řádek zdrojového kódu, pro ukázku složitosti je v příloze výpis zdrojového kódu řadiče paměti.



Obrázek 6.3: celkové schéma programu pro FPGA

6.2 Popis programu pro AVR

Program pro AVR je vytvořen a přeložen v návrhovém systému AVR studio, verze 4.13.528, firmy Atmel, s nainstalovaným překladačem WinAVR pro jazyk C.



Obrázek 6.4: návrhový systém AVR studio

Program pro AVR je napsán v programovacím jazyce C.
Hlavní částí programu je menu, ze kterého se spouštějí různé podprogramy.
Program se ovládá klávesnicí připojenou na konektor SV3.
Podprogramy vykonávají různé funkce, např. nastavují čas obvodu RTC, přehrávají ukázkový zvuk, zobrazují menu na displeji a nebo kreslí některé grafické symboly na displeji.
Jednotlivé části programu AVR jsou v rozsahu 35 - 580 řádek zdrojového kódu, pro ukázku je v příloze výpis zdrojového kódu pro generování zvuku piezoměničem.

Kapitola 7

Závěr

Nejvíce času jsem strávil samotným vytvářením spojů desky v editoru plošných spojů (Eagle). Velkou část času jsem také strávil při tvorbě programu pro AVR a hlavně pro FPGA, asi nejsložitější bylo naprogramovat řadič paměti SDRAM.

Možností pro vylepšení desky je několik, obvod ADM3485 má zbytečně připojeny tři vývody na FPGA, přestože stačí připojit pouze dva. Také paměť SDRAM má některé vývody zbytečně připojeny na FPGA a zbytečně mu tak vývody zabírá. Dalšími chybami jsou již zmíněné chyby v kapitole 4.

Samostatnou kapitolou pro vylepšování jsou programy, např. rychlost zápisu do SDRAM řadičem lze zvýšit. I v programu pro AVR lze udělat změny k lepšímu, ale tvorbu programového vybavení jsem si nekladl za hlavní cíl této práce, protože programové vybavení je vždy napsáno pro konkrétní zařízení, kde je umístěno.

Části jako jsou RS485, RS232, SD-karta a logický vstup/výstup jsem nemohl z časových důvodů vyzkoušet, ale domnívám se, že jsou plně funkční. Funkci ostatních částí desky jsem s úspěchem otestoval.

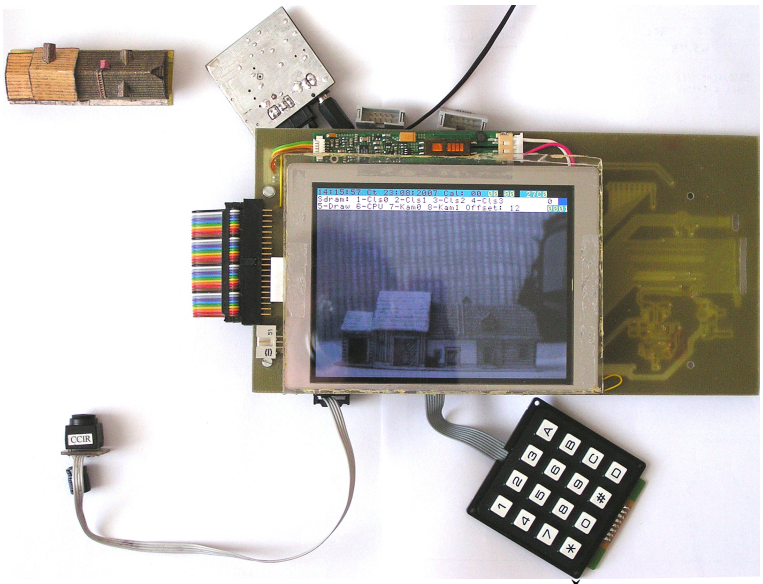
Na vzorové desky jsem připájel moduly se stabilizátory napětí, které jinde v textu neuvádím, protože je nepovažuji za přímou součást návrhu desky a předpokládám, že jejich popis později umístím na internet. Lze je nahradit lineárními stabilizátory, pokud odběr desky není příliš velký.
Na fotografiích s displejem je vidět jednostranná deska, tu jsem vyrobil pro řízení dvou displejů, ale z časových důvodů jsem na ni umístil pouze jeden. Nepředpokládám, že bych ji s tímto umístěním řídicího konektoru ještě někdy vyráběl, raději bych ji někdy v budoucnu přepracoval.

Přestože deska má některé drobné chyby, domnívám se, že nejsou natolik zásadní pro její použití v nějakém zařízení. Proto se domnívám, že zadaný cíl práce jsem splnil.

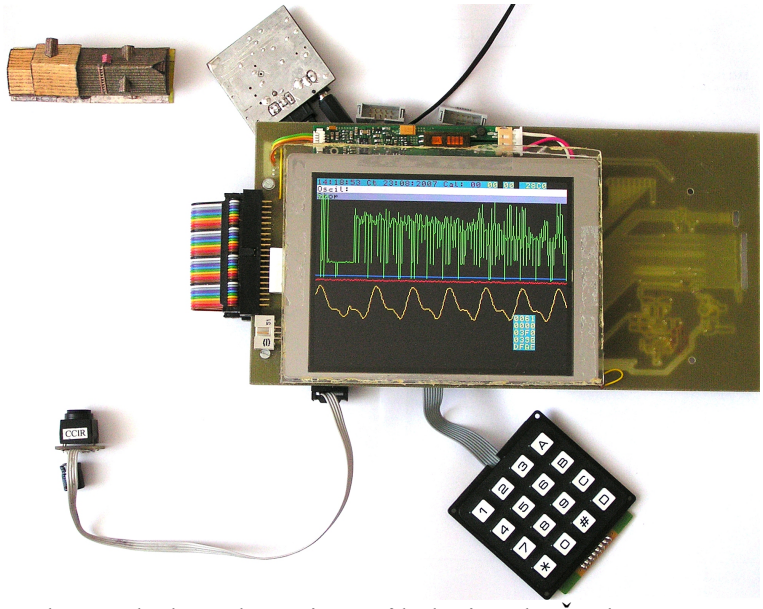
Kapitola 8

Přílohy

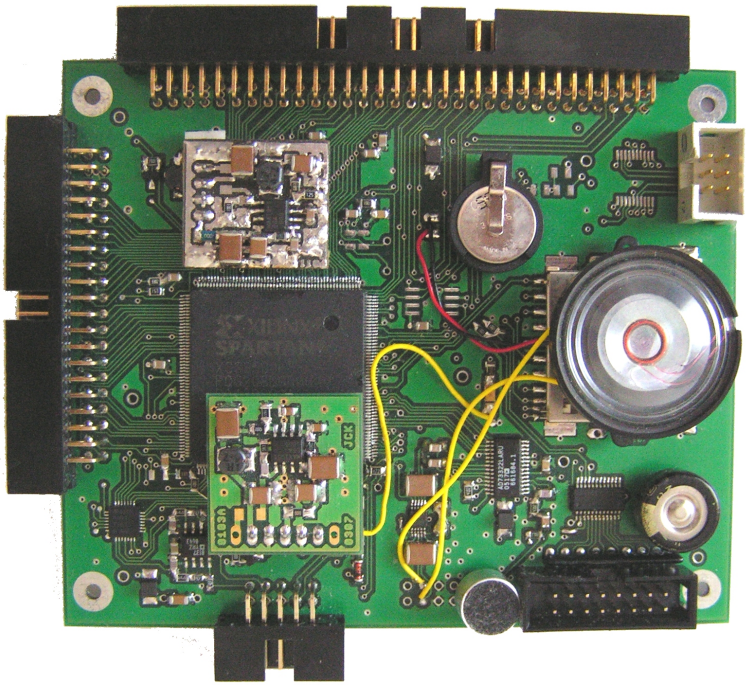
8.1 Fotografie desky



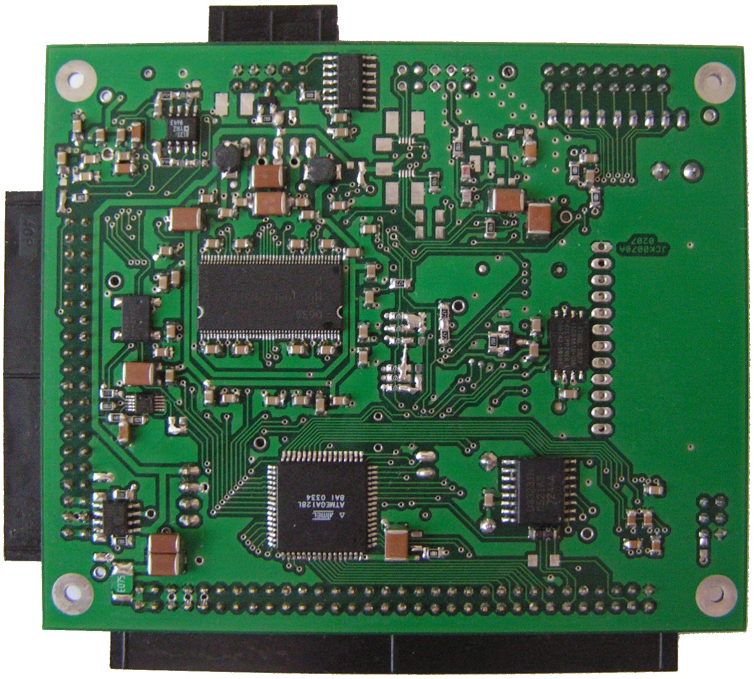
Obrázek 8.1: deska zobrazující signál z ČB kamery



Obrázek 8.2: deska zobrazující průběh signálu ČB kamery a zvuku



Obrázek 8.3: deska při pohledu zhora



Obrázek 8.4: deska při pohledu zdola

8.2 Ukázka zdrojového kódu pro AVR - generátor zvuku

```
/*-------------------------------------------------------*/
void SIG_INPUT_CAPTURE1 (void) __attribute__ ((signal));
void SIG_INPUT_CAPTURE1 (void) {
       if (ZvukLength == 0x0000) { TCCR1B &= 0xF8; TIMSK &= ~(1 << TICIE1); }
       else { ZvukLength--; }
}



/*-------------------------------------------------------*/
void Beep(dword i, dword j, dword k, byte Wait) {
//frekvence, doba trvani, plneni, cekat na dohrani
       word ZvukFreq, ZvukPwm;
       ZvukFreq = (word) ((CpuFreq / 2) / (dword) i);       //pro 8MHz
       ZvukLength = (word) ((i * j) / 1000);
       ZvukPwm =(word) ((ZvukFreq * k) / 200);

       ICR1 = ZvukFreq; OCR1A = ZvukPwm; OCR1B = ZvukPwm;
       TCCR1B = ((1 << WGM13) | 1 << CS10);
       TIMSK |= (1 << TICIE1);

       if (Wait != 1) { return; }
       do { _SLEEP(); } while (ZvukLength != 0x0000);
}


/*-------------------------------------------------------*/
void Click(void) {
       Beep(1000, 1, 100, 1); Beep(5000, 1, 100, 1); Beep(1000, 1, 100, 0);
}
```

8.3 Ukázka zdrojového kódu pro FPGA - řadič paměti SDRAM

```
module SDRAMmodule(
       CLK100M, DCM_FB, DCM_RST, AVR_RD, AVR_WR, AVR_ALE, LQ035_REQ, LQ057_REQ,
KAMWR,
       AVR_ADR, LQ035_HADR, LQ035_VADR, LQ035_ADRREQ, LQ057_HADR, LQ057_VADR,
LQ057_ADRREQ, KAMADR, KAMDATA,
       SDRAM_CLK, SDRAM_CKE, SDRAM_RAS, SDRAM_CAS, SDRAM_WE, SDRAM_CS, SDRAM_DQM,
SDRAM_BA, SDRAM_ADR, LQ035_DATAOUT, LQ057_DATAOUT, FLAG,
       SDRAM_DATA, AVR_DATA);
input CLK100M, DCM_FB, DCM_RST, AVR_RD, AVR_WR, AVR_ALE, LQ035_REQ, LQ057_REQ,
KAMWR;
input [7:0] AVR_ADR;
                                        //ve skutecnosti jsou na AVR jako A8-15
input [7:0] LQ035_HADR;
input [8:0] LQ035_VADR;
input [22:0] LQ035_ADRREQ;
input [8:0] LQ057_HADR;
input [7:0] LQ057_VADR;
input [22:0] LQ057_ADRREQ;
input [22:0] KAMADR;
input [31:0] KAMDATA;

output SDRAM_CLK, SDRAM_CKE, SDRAM_RAS, SDRAM_CAS, SDRAM_WE, SDRAM_CS;
output [1:0] SDRAM_DQM, SDRAM_BA;
output [11:0] SDRAM_ADR;
output [31:0] LQ035_DATAOUT, LQ057_DATAOUT;
output [7:0] FLAG;

inout [31:0] SDRAM_DATA;
inout [15:0] AVR_DATA;


parameter [7:0] ModuleAdr = 8'hFC;

parameter [2:0] CasLatencyParam  = 3'h3;
parameter [1:0] RasToCasParam          = 2'h2;
parameter [3:0] KeepIdleParam          = 4'h8;
parameter [10:0] BurstRdParam          = 11'h0FF;
parameter [10:0] BurstWrParam          = 11'h0FF;
parameter [16:0] RefreshParam          = 17'h005DC; //decimal 1500 for a 15us
refresh command interval
parameter [7:0] ST_IDLE0               = 8'h00;            //
parameter [7:0] ST_IDLE1               = 8'h01;            //
parameter [7:0] ST_PRECH               = 8'h02;            //Precharge
parameter [7:0] ST_LOAD_MR             = 8'h03;            //Load Mode Register
parameter [7:0] ST_PRE_AR              = 8'h04;            //Pre-Auto_refresh
parameter [7:0] ST_AR                        = 8'h05;            //Auto_refresh
parameter [7:0] ST_ACT0                      = 8'h06;            //pauza pro
zapis adresy
parameter [7:0] ST_ACT                       = 8'h07;            //Activate row
parameter [7:0] ST_READ_W              = 8'h08;            //Read Wait for Ras-
to-Cas delay
parameter [7:0] ST_READ_CS             = 8'h09;            //Read command
parameter [7:0] ST_READ_C              = 8'h0A;            //Read command, wait
for Cas latency
parameter [7:0] ST_READ                      = 8'h0B;            //Read data
parameter [7:0] ST_WRITE_W             = 8'h0C;            //Write Wait for Ras-
to-Cas delay
parameter [7:0] ST_WRITE_C             = 8'h0D;            //Write command
parameter [7:0] ST_WRITE               = 8'h0E;            //Write data
parameter [7:0] ST_BST_W               = 8'h0F;            //Burst Stop Wait for
KeepIdleEndFl
parameter [7:0] ST_BST_C               = 8'h10;            //Burst Stop Command
```

```
parameter [7:0] ST_BST                          = 8'h11;           //Burst Stop
parameter [13:0] ModeRegParam          = {7'h00, CasLatencyParam[2:0], 4'h7};
/*
parameter clk = 10;
parameter tAC  = 5.4;
parameter tOH  = 2.7;
parameter tHZ  = 5.4;
parameter tMRD =   2;     // 2 Clk Cycles
parameter tRC  =  66;
parameter tRAS =  44;
parameter tRCD =  20;
parameter tRP  =  20;
parameter tRRD =  15;
parameter tWR  =  1*clk + 7.5ns;
*/


//vystupni registry a draty
wire SDRAM_CLK;
reg SDRAM_CKE, SDRAM_RAS, SDRAM_CAS, SDRAM_WE, SDRAM_CS;
reg [1:0] SDRAM_BA, SDRAM_DQM;
reg [11:0] SDRAM_ADR;
wire [31:0] LQ035_DATAOUT, LQ057_DATAOUT;
wire [7:0] FLAG;

//vstupni registry
reg [15:0] DataInReg0, DataInReg, AdrInReg0, AdrInReg;
reg [1:0] AleReg = 2'h0, RdReg = 2'h3, WrReg = 2'h3;
reg RdFl = 1'b0, WrFlRise = 1'b0, WrFlFall = 1'b0;

//vnitrni pameti a registry
reg [16:0] FifoOutDelayCnt = 17'h1FFFF;
reg [8:0] FifoOutCntWr = 9'h000, FifoOutCntRd = 9'h000;
wire [8:0] FifoOutFullSize;

reg [10:0] BurstRdCnt, BurstWrCnt;     //Burst Counter
reg [2:0] CasLatencyCnt;                       //Cas Latency Counter
reg [1:0] RasToCasCnt;                                 //RAS to CAS Delay Counter
reg [3:0] KeepIdleCnt;                                 //Keep Idle Counter
reg [16:0] RefreshCnt;                                 //Refresh Counter
reg AutoRefreshReg;

reg [22:0] AdrReg;
reg [15:0] ModeReg = 16'h0000;

reg [22:0] AdrCnt;
reg [16:0] StartUpCnt = 16'h0000;
reg [15:0] DataReg;
reg [7:0] St = 8'h00, StPom = 8'h00;
reg [6:0] AdrHiReg = 7'h00;
reg [3:0] TftReqReg, RdSdramReg;
reg [1:0] WrSdramReg, KamWrReg = 2'h0;

wire [22:0] FifoOutAdr;
reg [22:0] FifoOutAdr1;
reg StWriteWReg, StIdleReg;

wire LoadKeepIdleCntFl, LoadCasLatencyCntFl, LoadBurstRdCntFl, LoadBurstWrCntFl,
LoadRasToCasCntFl, ClrRef;
```

```
wire KeepIdleCntEndFl, CasLatencyCntEndFl, BurstRdCntEndFl, BurstWrCntEndFl,
RasToCasCntEndFl;
wire RdSdramFl, WrSdramFl, FifoRdFl, sd_dqm0, sd_dqm1;
wire FifoOutEmptyFl, FifoOutHalfFl, FifoOutFullFl, StartRdFifoOutFl,
StopRdFifoOutFl, FifoOutDelayCntEndFl;
reg BufEn;

wire ModuleEn = (AdrInReg[15:4] == {ModuleAdr[7:0], 4'h0}) ? 1'b1 : 1'b0;
      //jako 8'h00 - 8'h0F

wire st_idle0 = (St[7:0] == ST_IDLE0[7:0]) ? 1'b1 : 1'b0;
wire st_idle1 = (St[7:0] == ST_IDLE1[7:0]) ? 1'b1 : 1'b0;
wire st_prech = (St[7:0] == ST_PRECH[7:0]) ? 1'b1 : 1'b0;
wire st_load_mr = (St[7:0] == ST_LOAD_MR[7:0]) ? 1'b1 : 1'b0;
wire st_pre_ar = (St[7:0] == ST_PRE_AR[7:0]) ? 1'b1 : 1'b0;
wire st_ar = (St[7:0] == ST_AR[7:0]) ? 1'b1 : 1'b0;
wire st_act0 = (St[7:0] == ST_ACT0[7:0]) ? 1'b1 : 1'b0;
wire st_act = (St[7:0] == ST_ACT[7:0]) ? 1'b1 : 1'b0;
wire st_read_w = (St[7:0] == ST_READ_W[7:0]) ? 1'b1 : 1'b0;
wire st_read_cs = (St[7:0] == ST_READ_CS[7:0]) ? 1'b1 : 1'b0;
wire st_read_c = (St[7:0] == ST_READ_C[7:0]) ? 1'b1 : 1'b0;
wire st_read = (St[7:0] == ST_READ[7:0]) ? 1'b1 : 1'b0;
wire st_write_w = (St[7:0] == ST_WRITE_W[7:0]) ? 1'b1 : 1'b0;
wire st_write_c = (St[7:0] == ST_WRITE_C[7:0]) ? 1'b1 : 1'b0;
wire st_write = (St[7:0] == ST_WRITE[7:0]) ? 1'b1 : 1'b0;
wire st_bst_w = (St[7:0] == ST_BST_W[7:0]) ? 1'b1 : 1'b0;
wire st_bst_c = (St[7:0] == ST_BST_C[7:0]) ? 1'b1 : 1'b0;
wire st_bst = (St[7:0] == ST_BST[7:0]) ? 1'b1 : 1'b0;

AutoRefreshReg));
wire st_read_fast = st_read_c & CasLatencyCntEndFl;
wire st_write_c_fast = st_write_w & RasToCasCntEndFl;
wire st_write_fast = st_write_c;


assign BurstWrCntEndFl = BurstWrCnt[10];
assign BurstRdCntEndFl = BurstRdCnt[10];
assign CasLatencyCntEndFl = (CasLatencyCnt[2:0] == 3'h0) ? 1'b1 : 1'b0;
assign RasToCasCntEndFl = (RasToCasCnt[1:0] == 2'h0) ? 1'b1 : 1'b0;
assign KeepIdleCntEndFl = (KeepIdleCnt[3:0] == 4'h0) ? 1'b1 : 1'b0;
assign FifoOutDelayCntEndFl = FifoOutDelayCnt[16];

assign LoadKeepIdleCntFl = ~(st_idle1 | st_bst_w | st_bst);
assign LoadCasLatencyCntFl = ~(st_read_cs | st_read_c | st_read);
assign LoadBurstRdCntFl = ~(st_read_cs | st_read_c | st_read);
assign LoadBurstWrCntFl = ~(st_write | st_write_c);
assign LoadRasToCasCntFl = ~(st_act | st_read_cs | st_read_w | st_write_c |
st_write_w);

assign ClrRef = (st_pre_ar | st_ar);
```

```
/*-------------------------------------------------------------------------------
-----------------------------------------------*/
//vstupni pamet do/pro LQ035
reg [31:0] LQ035_MemTft [511:0];

wire [8:0] LQ035_AdrMemTftRd0 = {LQ035_VADR[1:0], LQ035_HADR[7:1]};
reg [8:0] LQ035_AdrMemTftRd;

wire [8:0] LQ035_AdrMemTftWr = AdrCnt[8:0];

wire [31:0] LQ035_DataMemTftWr = SDRAM_DATA[31:0];

assign LQ035_DATAOUT[31:0] = LQ035_MemTft[LQ035_AdrMemTftRd[8:0]];

always @(posedge CLK100M) begin
LQ035_AdrMemTftRd[8:0] <= LQ035_AdrMemTftRd0[8:0];
if (st_read && RdSdramReg[0]) LQ035_MemTft[LQ035_AdrMemTftWr[8:0]] <=
LQ035_DataMemTftWr[31:0];
end

/*-------------------------------------------------------------------------------
-----------------------------------------------*/
//vstupni pamet do/pro LQ057
reg [31:0] LQ057_MemTft [511:0];

wire [8:0] LQ057_AdrMemTftRd0 = {LQ057_VADR[0], LQ057_HADR[8:1]};
reg [8:0] LQ057_AdrMemTftRd;

wire [8:0] LQ057_AdrMemTftWr = AdrCnt[8:0];

wire [31:0] LQ057_DataMemTftWr = SDRAM_DATA[31:0];

assign LQ057_DATAOUT[31:0] = LQ057_MemTft[LQ057_AdrMemTftRd[8:0]];

always @(posedge CLK100M) begin
LQ057_AdrMemTftRd[8:0] <= LQ057_AdrMemTftRd0[8:0];
if (st_read && RdSdramReg[1]) LQ057_MemTft[LQ057_AdrMemTftWr[8:0]] <=
LQ057_DataMemTftWr[31:0];
end

/*-------------------------------------------------------------------------------
-----------------------------------------------*/
//vystupni pamet do SDRAM
reg [56:0] FifoOut [511:0];

wire [56:0] DataFromFifoOut;
reg [8:0] AdrFifoOutRd;

wire FifoWrFl;

assign DataFromFifoOut[56:0] = FifoOut[AdrFifoOutRd[8:0]];

always @(posedge CLK100M) begin
AdrFifoOutRd[8:0] <= FifoOutCntRd[8:0];
if (FifoWrFl) FifoOut[FifoOutCntWr[8:0]] <=
       (ModeReg[0]) ? {2'h0, KAMADR[22:0], KAMDATA[31:0]} : {AdrInReg[3:2],
AdrReg[22:0], DataReg[15:0], DataInReg[15:0]};
end

assign FifoWrFl = ((~ModeReg[0] & WrFlRise & ModuleEn) | (ModeReg[0] &
~KamWrReg[1] & KamWrReg[0])) & ~FifoOutFullFl;
assign FifoRdFl = ~StopRdFifoOutFl & st_write_c;
```

```
/*-------------------------------------------------------------------------------
-----------------------------------------------*/
wire Clk_Fb0, SdramClkPom;
IBUFG IBufg_0 (.I(DCM_FB), .O(Clk_Fb0));
//DCM_SP DCM_0 (.CLKIN(CLK100M), .CLKFB(DCM_FB), .DSSEN(1'b0), .RST(~DCM_RST),
.CLK0(SDRAM_CLK));
DCM_SP DCM_0 (.CLKIN(CLK100M), .CLKFB(Clk_Fb0), .DSSEN(1'b0), .RST(~DCM_RST),
.CLK0(SdramClkPom));
OBUF_F_16 obuf0 (.I(SdramClkPom), .O(SDRAM_CLK));


assign sd_dqm0 =
      (((st_read | st_read_c | st_read_cs) & ~BurstRdCntEndFl) | (st_write_c &
~DataFromFifoOut[55] & ~StopRdFifoOutFl));
assign sd_dqm1 =
      (((st_read | st_read_c | st_read_cs) & ~BurstRdCntEndFl) | (st_write_c &
~DataFromFifoOut[56] & ~StopRdFifoOutFl));
assign SDRAM_DATA[31:0] = (BufEn) ? DataFromFifoOut[31:0] : 32'hz;
//assign SDRAM_DATA[31:0] = (BufEn && WrSdramReg[2]) ? {DataReg1[7:0],
DataReg0[7:0]} : 16'hz;


assign RdSdramFl = (RdSdramReg[3:0] != 4'h0) ? 1'b1 : 1'b0;
assign WrSdramFl = (WrSdramReg[1:0] != 2'h0) ? 1'b1 : 1'b0;

assign FifoOutAdr[22:0] = DataFromFifoOut[54:32];

assign FifoOutFullSize[8:0] = FifoOutCntWr[8:0] - FifoOutCntRd[8:0];
assign FifoOutEmptyFl = (FifoOutFullSize[8:0] == 9'h000) ? 1'b1 : 1'b0;
assign FifoOutHalfFl = (FifoOutFullSize[8:0] >= 9'h100) ? 1'b1 : 1'b0;
assign FifoOutFullFl = (FifoOutFullSize[8:0] >= 9'h1F8) ? 1'b1 : 1'b0;

assign StartRdFifoOutFl = FifoOutHalfFl | (FifoOutDelayCntEndFl &
~FifoOutEmptyFl);
assign StopRdFifoOutFl =
      ((FifoOutAdr[22:8] != FifoOutAdr1[22:8] && ~StWriteWReg) || FifoOutEmptyFl
|| BurstWrCntEndFl) ? 1'b1 : 1'b0;

assign AVR_DATA[15:0] =
      (~RdReg[0] && AdrInReg[15:0] == {ModuleAdr[7:0], 8'h78}) ? {13'h0000,
FifoOutFullFl, FifoOutHalfFl, FifoOutEmptyFl} :
      16'hz;
//assign FLAG[7:0] = {4'h0, st_read, FifoOutFullFl, FifoOutHalfFl,
FifoOutEmptyFl};
assign FLAG[7:0] = {4'h0, st_read, KAMADR[20], KAMADR[9], KAMWR};



/*-------------------------------------------------------------------------------
-----------------------------------------------*/
always @(posedge CLK100M) begin
AleReg[1:0] <= {AleReg[0], AVR_ALE}; RdReg[1:0] <= {RdReg[0], AVR_RD}; WrReg[1:0]
<= {WrReg[0], AVR_WR};
RdFl <= (RdReg[1:0] == 2'h2) ? 1'b1 : 1'b0;
WrFlRise <= (WrReg[1:0] == 2'h1) ? 1'b1 : 1'b0; WrFlFall <= (WrReg[1:0] ==
2'h2) ? 1'b1 : 1'b0;

DataInReg0[15:0] <= AVR_DATA[15:0]; DataInReg[15:0] <= DataInReg0[15:0];
AdrInReg0[15:0] <= {AVR_ADR[7:0], AVR_DATA[7:0]};
if (AleReg[1:0] == 2'h2) AdrInReg[15:0] <= AdrInReg0[15:0];

if (WrFlRise && AdrInReg[15:2] == {ModuleAdr[7:0], 6'h04}) ModeReg[15:0] <=
DataInReg[15:0];                                                    //jako
8'h10

if (WrFlRise && AdrInReg[15:0] == {ModuleAdr[7:0], 8'h80}) DataReg[15:0] <=
DataInReg[15:0];
```

```
AdrReg[22:0] <=
       (WrFlRise && AdrInReg[15:0] == {ModuleAdr[7:0], 8'h84}) ? {AdrHiReg[6:0],
DataInReg[15:0]} :
       (WrFlRise && AdrInReg[1:0] == 2'h1) ? AdrReg[22:0] + 23'h000001 :
       (WrFlRise && AdrInReg[1:0] == 2'h2) ? AdrReg[22:0] - 23'h000001 :
       AdrReg[22:0];
if (WrFlRise && AdrInReg[15:0] == {ModuleAdr[7:0], 8'h88}) AdrHiReg[6:0] <=
DataInReg[6:0];

KamWrReg[1:0] <= {KamWrReg[0], KAMWR};

BurstRdCnt[10:0] <=                          //Burst Counter
       (LoadBurstRdCntFl) ? BurstRdParam[10:0] :
       (BurstRdCntEndFl) ? BurstRdCnt[10:0] : BurstRdCnt[10:0] - 11'h001;
BurstWrCnt[10:0] <=                          //Burst Counter
       (LoadBurstWrCntFl) ? BurstWrParam[10:0] :
       (BurstWrCntEndFl) ? BurstWrCnt[10:0] : BurstWrCnt[10:0] - 11'h001;
CasLatencyCnt[2:0] <=                        //Cas Latency Counter
       (LoadCasLatencyCntFl) ? CasLatencyParam[2:0] :
       (CasLatencyCntEndFl) ? 3'h0 : CasLatencyCnt[2:0] - 3'h1;
RasToCasCnt[1:0] <=                          //RAS to CAS Delay Counter
       (LoadRasToCasCntFl) ? RasToCasParam[1:0] :
       (RasToCasCntEndFl) ? 2'h0 : RasToCasCnt[1:0] - 2'h1;
KeepIdleCnt[3:0] <=                          //Keep Idle Counter
       (LoadKeepIdleCntFl) ? KeepIdleParam[3:0] :
       (KeepIdleCntEndFl) ? 4'h0 : KeepIdleCnt[3:0] - 4'h1;
RefreshCnt[16:0] <=                          //Refresh Counter
       (ClrRef) ? RefreshParam[16:0] :
       (RefreshCnt[16]) ? RefreshParam[16:0] : RefreshCnt[16:0] - 17'h00001;
AutoRefreshReg <=   (ClrRef) ? 1'b0 : (RefreshCnt[16]) ? 1'b1 : AutoRefreshReg;

St[7:0] <= StPom[7:0];
StartUpCnt[16:0] <=
       (StartUpCnt[16]) ? StartUpCnt[16:0] :
       (st_idle1 && KeepIdleCntEndFl) ? StartUpCnt[16:0] + 17'h00001 :
       StartUpCnt[16:0];

FifoOutDelayCnt[16:0] <=
       (FifoOutEmptyFl) ? 17'h00000 :
       (FifoOutDelayCntEndFl) ? FifoOutDelayCnt[16:0] : FifoOutDelayCnt[16:0] +
17'h00001;

FifoOutCntWr[8:0] <=
       (FifoWrFl) ? FifoOutCntWr[8:0] + 9'h001 :
       FifoOutCntWr[8:0];
FifoOutCntRd[8:0] <=
       (FifoRdFl) ? FifoOutCntRd[8:0] + 9'h001 :
       FifoOutCntRd[8:0];

TftReqReg[3:0] <=
       (BurstRdCntEndFl && st_read) ? {2'h0, LQ057_REQ, LQ035_REQ} |
(TftReqReg[3:0] & ~RdSdramReg[3:0]) :
       {2'h0, LQ057_REQ, LQ035_REQ} | TftReqReg[3:0];

RdSdramReg[3:0] <=
       (st_idle0 && TftReqReg[3:0] == 4'h0) ? 4'h0 :
       (st_idle0 && TftReqReg[0]) ? 4'h1 :
       (st_idle0 && TftReqReg[1]) ? 4'h2 :
       (st_idle0 && TftReqReg[2]) ? 4'h4 :
       (st_idle0 && TftReqReg[3]) ? 4'h8 :
       RdSdramReg[3:0];
```

```
WrSdramReg[1:0] <= (st_idle0) ? {1'h0, StartRdFifoOutFl} : WrSdramReg[1:0];

AdrCnt[22:0] <=
       (RdSdramReg[0] && st_idle1) ? LQ035_ADRREQ[22:0] :
       (RdSdramReg[1] && st_idle1) ? LQ057_ADRREQ[22:0] :
       (~BurstRdCntEndFl && st_read) ? AdrCnt[22:0] + 23'h000001 :
      AdrCnt[22:0];


FifoOutAdr1[22:0] <= FifoOutAdr[22:0];


StWriteWReg <= st_write_w;
StIdleReg <= st_idle1;


SDRAM_CKE <= 1'b1;
SDRAM_CS <= st_idle0 || st_idle1;             //1'b0;
SDRAM_RAS <= ~(st_act | st_prech | st_load_mr | st_ar);
SDRAM_CAS <= ~(st_read_cs | st_write_c | st_load_mr | st_ar);
//SDRAM_WE <= ~(st_write_c | st_prech | st_load_mr | st_bst_c);
SDRAM_WE <= ~((st_write_c & ~StopRdFifoOutFl) | st_prech | st_load_mr |
st_bst_c);
SDRAM_ADR[11:0] <=
       (st_ar || st_pre_ar || st_prech) ? 12'h400 :
              //precharge pro vsechny banky (nezavisle na BA0-1)
       (st_load_mr) ? ModeRegParam[11:0] :
       ((st_act0 || st_act) && RdSdramFl) ? {1'b0, AdrCnt[18:8]} :
       (~(st_act0 || st_act) && RdSdramFl) ? {4'h0, AdrCnt[7:0]} :
       ((st_act0 || st_act) && WrSdramFl) ? {1'b0, FifoOutAdr[18:8]} :
       (~(st_act0 || st_act) && WrSdramFl) ? {4'h0, FifoOutAdr[7:0]} :
       (st_act0 || st_act) ? {1'b0, AdrCnt[18:8]} :
       {4'h0, AdrCnt[7:0]};
SDRAM_BA[1:0] <=
       (st_load_mr) ? ModeRegParam[13:12] :
       (RdSdramFl) ? AdrCnt[20:19] :
       (WrSdramFl) ? FifoOutAdr[20:19] :
      AdrCnt[20:19];
SDRAM_DQM[1:0] <= {~sd_dqm1, ~sd_dqm0};
BufEn <= st_write_c & WrSdramFl;

end
```

```
/*-----------------------------------------------------------------------------
-----------------------------------------------*/
always @ (St or AutoRefreshReg or ClrRef or RdSdramFl or WrSdramFl or StartUpCnt
or KeepIdleCntEndFl
                        or CasLatencyCntEndFl or BurstRdCntEndFl or
BurstWrCntEndFl or RasToCasCntEndFl or StopRdFifoOutFl) begin
      StPom = 8'h00;      //has default values for outputs so synthesis tool
don't infer latches
      casex (St[7:0])

      ST_IDLE0 : StPom = ST_IDLE1;
      ST_IDLE1 :
            StPom =
            (StartUpCnt[16] && KeepIdleCntEndFl && AutoRefreshReg) ? ST_PRE_AR :
            (StartUpCnt[16] && KeepIdleCntEndFl && (RdSdramFl || WrSdramFl)) ?
ST_ACT0 :
            (StartUpCnt[15:0] == 16'h4000 && KeepIdleCntEndFl) ? ST_PRECH :
            (StartUpCnt[15:12] == 4'h5 && KeepIdleCntEndFl) ? ST_AR :
            (StartUpCnt[15:0] == 16'h6000 && KeepIdleCntEndFl) ? ST_PRECH :
            (StartUpCnt[15:0] == 16'h6008 && KeepIdleCntEndFl) ? ST_LOAD_MR :
            (StartUpCnt[15:0] == 16'h6010 && KeepIdleCntEndFl) ? ST_PRECH :
            (StartUpCnt[15:12] == 4'h8 && KeepIdleCntEndFl) ? ST_AR :
            ST_IDLE1;

      ST_PRECH :          StPom = ST_IDLE0;
      ST_LOAD_MR : StPom = ST_IDLE0;
      ST_PRE_AR :         StPom = ST_AR;
      ST_AR :                   StPom = ST_IDLE0;
      ST_BST :                  StPom = (KeepIdleCntEndFl) ? ST_BST_C : ST_BST;
      ST_BST_C :          StPom = ST_BST_W;
      ST_BST_W :          StPom = (KeepIdleCntEndFl) ? ST_PRECH : ST_BST_W;

      ST_ACT0 :           StPom = ST_ACT;
      ST_ACT :
            StPom = (RdSdramFl) ? ST_READ_W :  (WrSdramFl) ? ST_WRITE_W :
ST_PRECH;

      //Write
      ST_WRITE_W : StPom = (RasToCasCntEndFl) ? ST_WRITE_C : ST_WRITE_W;
      ST_WRITE :          StPom = ST_WRITE_C;
      ST_WRITE_C : StPom = (StopRdFifoOutFl) ? ST_BST : ST_WRITE;

      //Read
      ST_READ_W :         StPom = (RasToCasCntEndFl) ? ST_READ_CS : ST_READ_W;
      ST_READ_CS : StPom = ST_READ_C;
      ST_READ_C :         StPom = (CasLatencyCntEndFl) ? ST_READ : ST_READ_C;
      ST_READ :           StPom = (BurstRdCntEndFl) ? ST_BST : ST_READ;
      default:                  StPom = ST_IDLE0;
      endcase
end
endmodule
```

Literatura

http://www.analog.com/en/
http://www.atmel.com/
http://www.fairchildsemi.com/
http://www.gme.cz/
http://www.maxim-ic.com/
http://www.micron.com/
http://www.onsemi.com/
http://www.sandisk.com/
http://www.xilinx.com/

Internetové stránky s návrhovými systémy
http://winavr.sourceforge.net/
http://www.atmel.com/
http://www.xilinx.com/

čistý list