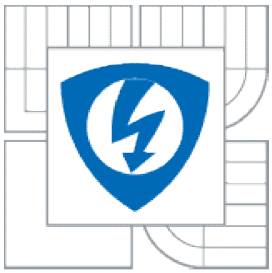
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

LABORATORNÍ PŘÍPRAVEK PRO EXPERIMENTÁLNÍ PRÁCI S OBVODY COOLRUNNER

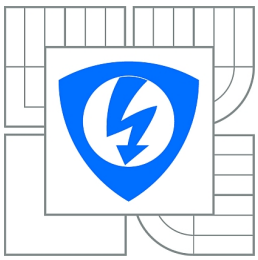
LABORATORY KIT FOR EXPERIMENTAL WORK WITH COOLRUNNER DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR GAJDOŠÍK
AUTHOR

VEDOUCÍ PRÁCE doc. Ing. JAROMÍR KOLOUCH, CSc.
SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Bakalářská práce

bakalářský studijní obor
Elektronika a sdělovací technika

Student:	Petr Gajdošík	***ID:***	109650
Ročník:	3	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Laboratorní přípravek pro experimentální práci s obvody CoolRunner

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte možnosti programování obvodů CPLD řady CoolRunner-II firmy Xilinx pomocí sběrnice USB. Navrhněte zapojení laboratorního přípravku s tímto obvodem, programovaným sběrnicí USB. Přípravek má umožňovat vstup údajů z přepínačů na desce, indikaci stavu výstupů signálkami LED a případně multiplexním displejem. Napájení přípravku volitelně z USB nebo ze zdroje 5 V. Přípravek má být vybaven konektory pro možnost připojení dalších periferií. Přípravek má být odolný proti případným chybám při používání. Vypracujte seznam součástek potřebných pro osazení přípravku.

Navrhněte plošný spoj pro tento přípravek, osaďte jej a oživte. Zpracujte vzorové příklady použití přípravku v laboratorním cvičení.

DOPORUČENÁ LITERATURA:

[1] CoolRunner-II CPLD Family. Dostupné na www: http://www.xilinx.com/support/documentation/data_sheets/ds090.pdf.

[2] KOLOUCH, J. Programovatelné logické obvody a návrh jejich aplikací v jazyku VHDL. Skriptum. Brno: FEKT VUT v Brně, 2006.

Termín zadání:	8.2.2010	***Termín odevzdání:***	28.5.2010

Vedoucí práce: doc. Ing. Jaromír Kolouch, CSc.

prof. Dr. Ing. Zbyněk Raida
Předseda oborové rady

ABSTRAKT

Bakalářská práce je zaměřena na programování obvodu CPLD řady CoolRunner – II společnosti Xilinx pomocí sběrnice USB. V dnešní době je tato sběrnice a její užití nedílnou součástí mnoha zařízení. Tyto programovatelné obvody jsou dosti využívané, hlavně pro jejich rychlost a flexibilitu. Dají se využít v mnoha aplikacích. Úkolem je navrhnout laboratorní přípravek, který se bude skládat ze vstupů realizovaných přepínači, výstupů realizovaných signalizačními diodami LED a 7segmentového displeje. A dále s možností připojit další externí vstupy a výstupy. Práce bude doprovázena i příklady aplikací v jazyce VHDL.

KLÍČOVÁ SLOVA

CPLD, CoolRunner II, USB, laboratorní přípravek, JTAG, VHDL.

ABSTRACT

This bachelor´s thesis deals with the programming of the CPLD device from the CoolRunner.II family, made by Xilinx company, through the use of USB bus. This bus and its application is an integral part of many devices nowadays. These programmable devices are used very often, mainly because of their speed and flexibility. They can be used in many different applications. The main task is to design a laboratory kit containing input switches, outputs of signaling LED diods and a 7segments display. Next possibility is to connect it with a chance plug in many external inputs and outputs. The thesis contains some examples of programs in the VHDL language.

KEYWORDS

CPLD, CoolRunner II, USB, laboratory kit, JTAG, VHDL.

GAJDOŠÍK, P. *Laboratorní přípravek pro experimentální práci s obvody CoolRunner*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 43 s. Vedoucí bakalářské práce: doc. Ing. Jaromír Kolouch, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma Laboratorní přípravek pro experimentální práci s obvody CoolRunner jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce doc. Ing. Jaromíru Kolouchovi, CSc. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

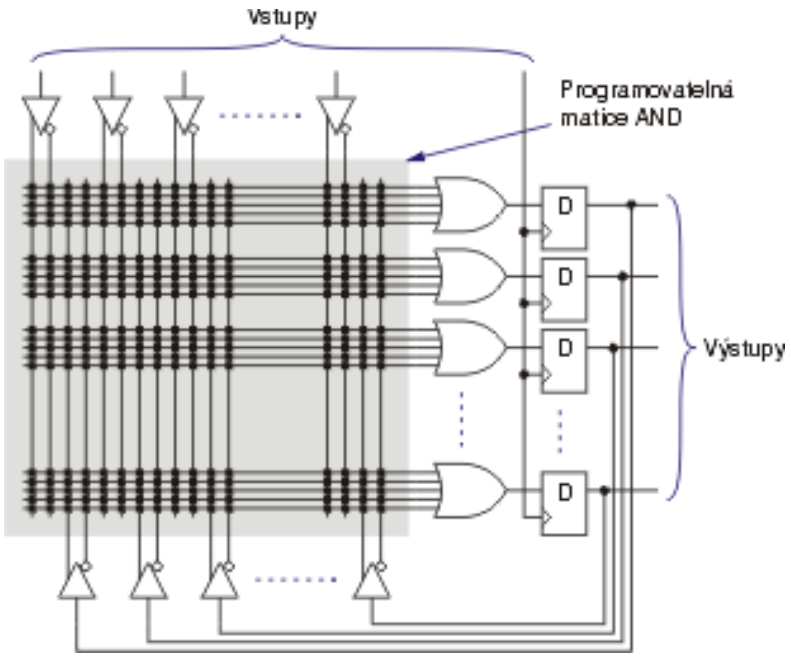
Úkolem v této bakalářské práci je navrhnout laboratorní přípravek pro experimentální práci s obvodem CoolRunner, vyráběným firmou Xilinx. Tyto obvody patří do skupiny CPLD, což je Complex Programmable Logic Device, neboli složitý programovatelný logický obvod. Jsou to rychlé a flexibilní obvody, které se dají naprogramovat aplikací, kterou si uživatel vytvoří sám. Tento obvod by v tomto přípravku měl být programovaný přes USB sběrnici. Zároveň v přípravku je zahrnuta i možnost programování přes nějaký externí programátor. Co se týče napájení, tak je použito 5 V ze sběrnice USB a taktéž možnost připojení externího zdroje napětí 5 V. Celý tento přípravek je v co možná největší míře sestaven z SMD součástek z důvodů dosažení menších rozměrů.
Přípravek je navržen tak, aby mohl být používán v laboratorních cvičeních k demonstraci funkce studentských konstrukcí implementovaných do obvodů PLD. Z pohledu periferií, které jsou připojeny k obvodu CPLD, je pro generování vstupních signálů použito 12 přepínačů. Výstupní signály jsou zobrazeny dvanácti diodami LED a 7segmentovým displejem. Na zbylé volné kontakty je připojen konektor pro připojení externích periferií. 7segmentový displej je použit staticky řízený, z důvodu použití obvodu CPLD s malou velikostí. Bakalářská práce se zabývá i programováním obvodu CPLD přes sběrnici USB za použití mikrokontroléru. Dále jsou realizovány příklady programů ve VHDL kódu pro obvod CPLD, zejména pro demonstraci funkčnosti přípravku.

1 PROGRAMOVATELNÉ LOGICKÉ OBVODY

V této kapitole jsou použity informace z pramenu [4]. Programovatelné logické obvody jsou v dnešní době důležitou součástí elektroniky. Představují jednoduchý způsob, jak si pořídit a vyrobit vlastní integrovaný obvod, pracující podle svých představ. Tato možnost je i cenově dostupnou. Všechny číslicové obvody se souhrnně nazývají PLD, což znamená Programmable Logic Device. Tyto logické obvody se dají rozdělit do tří skupin. První skupina jsou SPLD obvody, druhá skupina jsou složité obvody nazývané zkratkou CPLD a do třetí skupiny patří obvody typu FPGA.

1.1 SPLD

Tyto obvody jsou charakterizovány vnitřní strukturou podle následujícího obrázku.



Obrázek 1.1 Vnitřní struktura PLD (převzato z [4])

Každá vodorovná čára v programovatelné matici AND představuje vždy jedno součinové hradlo. Což znamená, že na vstupy každého hradla je možné zapojit libovolnou kombinaci vstupních signálů, zpětných vazeb a jejich negací. Počet vstupů součinového hradla je omezen.

Do kategorie jednoduchých obvodů PLD je možné zařadit obvody následujících typů:

- PAL

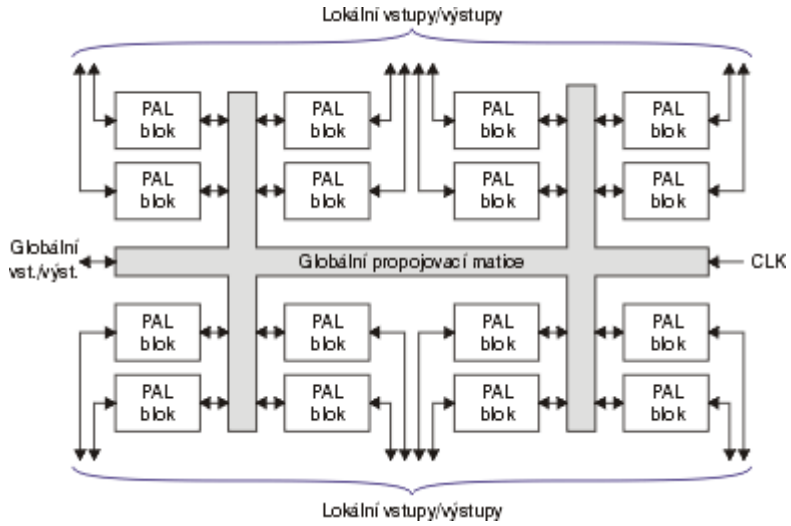
Obvody typu PAL (Programmable Array Logic) mají strukturu podle výše uvedeného obrázku. Některé starší typy neměly například výstupní registry, takže byly vhodné pro kombinační logiku. Zástupci této kategorie jsou obvody PAL, GAL.

- PLA

Obvody typu PLA (Programmable Logic Array) mají obecnější strukturu, než PAL. Mají totiž programovatelnou nejen matici součinů ale i následující matici logických součtů. [4]

1.2 Složité PLD obvody (CPLD)

Klasické obvody PLD mají velmi omezené prostředky, takže umožňují realizovat pouze jednodušší funkce. Proto výrobci začali sdružovat více takových obvodů na jednom čipu spolu s nutnými prostředky pro propojení. Takové obvody se většinou označují jako CPLD (Complex Programmable Logic Device). Typická struktura obvodu CPLD je znázorněna na následujícím obrázku.



Obrázek 1.2 Typická struktura CPLD (převzato z [4])

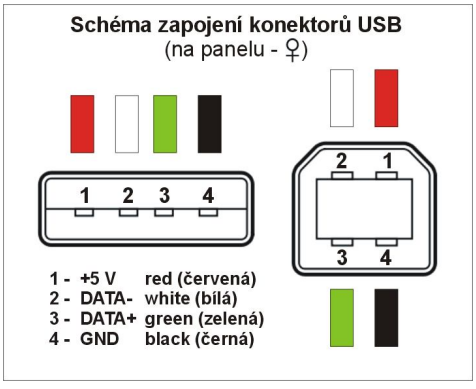
Každý výrobce CPLD používá trochu jinou interní strukturu obvodů, ale ve většině případů využívají strukturu zobrazenou na předešlém obrázku. Obvody CPLD od různých výrobců se obvykle liší v provedení bloků vlastní programovatelné logiky, ale většinou se vychází z klasické struktury PAL. [4]

2 UNIVERSAL SERIAL BUS

Informace o USB sběrnici jsou čerpané z pramenu [1]. USB je zkratka názvu Universal Serial Bus, nahradila do té doby hojně používanou sériovou sběrnici RS-232. Takže u obou sběrnic lze tvrdit, že přenos dat je realizován sériově bit po bitu. Přechod na USB s sebou přinesl výhody ale i nevýhody. Pro uživatele je tato sběrnice ulehčením práce. Má větší šířku pásma než RS-232. Ve verzi USB 1.1 jsou dvě rychlosti, pomalá (low speed), která má přenosovou rychlost 1,5 MB/s a také rychlá verze (full speed) s rychlostí 12 MB/s. Další výhodou USB sběrnice je i fakt, že k jednomu USB je možno připojit až 127 zařízení, což u RS-232 nebylo možné.
Sběrnice USB má tu výhodu, že ji lze využít i jako zdroj napětí 5 V s malým odběrem proudu. Není problém odebírat z něho proud 100 mA. Po speciálním přihlášení může systém odebírat až 500mA. Tím je vyřešen problém při, kterém muselo být připojené zařízení ještě dalším kabelem napájeno. USB zařízení je taktéž možno připojovat a odpojovat za běhu bez jakýchkoli problémů, operační systém okamžitě zavede ovladač pro toto zařízení, takový systém se nazývá Plug and Play (Připoj a pracuj).
Nevýhodou je velká složitost, USB zařízení obsahuje přinejmenším jeden řadič USB, obsahující rozsáhlý program. Na straně počítače je nutnost ovladače, který taktéž není jednoduché zrealizovat. Každé zařízení má interní číslo dodavatele (vendor ID), které je oficiálně udělováno Organizací USB. Zařízení je možno dodávat na trh jen s platným VID.
Vyskytuje se i verze USB 2.0, u které se rychlost přenosu vyšplhala až na 480 MBit/s.

2.1 Připojení na kabel

Sběrnice USB využívá dva základní konektory, typ A a typ B. Nelze je zaměnit. Užívají se také konektory microUSB a miniUSB, ty jsou určené zejména do menších zařízení, jako jsou telefony, fotoaparáty, atd. Zapojení konektorů je názorně vyjádřeno na obrázku 2.1.

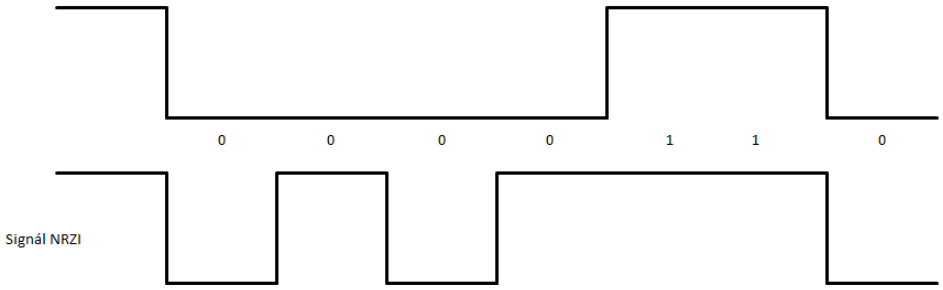


Obrázek 2.1 Zapojení konektorů USB (převzato z [1])

Délka, průřez a stínění kabelů je přesně dáno normou pro danou rychlost přenosu. Každopádně by délka prodlužovacího kabelu neměla přesáhnout 5 m.
Signály na linkách D+ a D- jsou rozdílové signály s napěťovými úrovněmi 0 V/3,3 V. Celý systém je navržen tak, aby i při maximálním zatížení napájecí napětí nekleslo pod 4,2 V. [1]

2.2 Sériový přenos

Sběrnice USB má jen jedno zařízení typu master a tím je PC, tzn. veškeré aktivity vycházejí z PC. Zařízení USB nemůže vysílat data jen tak, aniž by si je master nevyžádal. Data se odesílají v paketech, které jsou dlouhé od 8 bytů až po 256 bytů. Veškerý datový přenos se provádí v tzv. rámech (frame), které mohou trvat nejdéle 1 ms. Pakety v těchto rámech se mohou přenášet jak rychlostí low speed, tak i high speed. Rychlost pro low speed je 1,5 MB/s, což jeden bit je dlouhý 666,7 ns a pro high speed rychlost je 12 MB/s a tady je jeden bit dlouhý 83,3 ns. Uvnitř jednoho rámu můžou být zpravovány pakety i pro více zařízení a to i v rychlé i pomalé verzi. To, aby nedocházelo např. k přenosu rychlého paketu na pomalé zařízení, řeší rozdělovač sběrnice (hub).
Rychlost komunikace udává master. Slave zařízení se musí zasynchronizovat na datový tok, jelikož není žádný odděleně vedený hodinový signál. Proto se využívá systému NRZI (Non-Return-To Zero). Nuly v datovém signálu mění úroveň a jedničky ponechávají úroveň beze změny.



Obrázek 2.2 Příklad systému NRZI

Kódování a dekódování signálu se provádí hardwarově. Přijímač musí získat hodinový signál, přijmout a dekódovat data. Speciální prostředky zajišťují, aby nedocházelo ke ztrátě synchronizace. Obsahuje-li původní datový tok šest jedniček po sobě, vsune vysílač automaticky jednu nulu, aby tím došlo ke změně, a přijímač ji automaticky odebírá. Každý datový paket má za účelem synchronizace synchronizační byte, tzv. sync-byte (00000001b), který reprezentuje osm střídajících se bitových stavů a na tomto bytu se může přijímač zasynchronizovat. [1]

2.3 Rozdělovače sběrnice (HUBS)

USB má hvězdicovou strukturu s jedním masterem (PC), pro připojení více zařízení je nutný USB hub, který má za úkol rozšířit množství připojitelných zařízení, rozpoznání připojeného zařízení, rozpoznání rychlosti zařízení. USB hubů je možno připojit za sebe maximálně 7. Každé rozšíření znamená časovou prodlevu.
Nepoužitý port USB je neaktivní a na signálových spojích má úroveň low a mají vnitřní odpor 15 kΩ. Každé přidané zařízení má odpor 1,5 kΩ, který jednu ze signálových linek připojuje

na napětí 3,3 V. U rychlého zařízení je takto připojena linka D+ a u pomalého D-. Díky tomu hub rozpozná rychlost zařízení. Jako první se provede reset zařízení a to tak, že na 10 ms obě datové linky stáhnou na aktivní nízkou úroveň. Poté se zařízení rozpozná a začne se zavádět firmware. Hub zásobuje hned po připojení zařízení proudem až 100 mA. Pokud takový proud nestačí, musí požádat o navýšení a tím dojde k povolení až 500 mA. [1]

2.4 Napájení přes sběrnici USB

Díky tomu, že sběrnice USB umožňuje dodání napájecího napětí, odpadá u většiny zařízení nutnost pořizovat další napájecí zdroje. Sběrnice dodává napětí 5 V, z které je možné hned po připojení čerpat 100 mA a po enumeraci až 500 mA. To dostačuje k napájení většiny běžně používaných zařízení. V reálných přídavných zařízeních USB se většinou pracuje s napětím 3,3 V. Hlavně kvůli tomu, že na datových linkách D+ a D- je rozdíl úrovní 3,3 V a taky kvůli tomu, že při poklesu napětí v USB na 4 V, je stabilizátor na 3,3 V schopen bez problémů do obvodu dodat dané napětí, aniž by se na něm pokles projevil. [1]

3 POUŽITÉ INTEGROVANÉ OBVODY

3.1 ATMEL AT90USB162

Informace o AT90USB jsou čerpány z pramenu [3]. Tento integrovaný obvod je 8bitový mikrokontrolér s 16 kB ISP flash pamětí a USB řadičem. Bude využit pro zprostředkování a řízení programování CPLD obvodu přes USB.

Základní vlastnosti:

Pokročilá RISC Architektura
- -Instrukční sada obsahuje 135 intrukcí
- -Obsahuje 32 8bitových pracovních registrů
- -Plně statické operace
- -Až 16 miliónů instrukcí za vteřinu při 16 MHz

Stálost programu a datové paměti
- -16 kB ISP flash paměť – 10 000 cyklů
- -512 Bytů EEPROM paměti – 100 000 cyklů
- -512 Bytů interní SRAM paměti
- -Programovací zámek pro softwarovou ochranu

USB 2.0 Full-speed modul s přerušením na dokončení přenosu
- -Plně vyhovuje standardu USB specifikaci REV 2.0
- -4 programovatelné koncové body:
  - Vstupní nebo výstupní směry
  - Přerušení a asynchronní přenos
  - Programovatelné pakety o velikosti od 8 do 64 bytů
  - Programovatelný jednoduchý nebo dvojitý zásobník
- -Zapnutím vyvolá reset a reset USB sběrnice

Periferní vlastnosti
- -PS/2 platforma
- -8bitový čítač/časovač s odděleným voláním a porovnávacím režimem (dva 8bitové PWM kanály)
- -16bitový čítač/časovač s odděleným voláním, porovnáním a zachycením (tři 8bitové PWM kanály)
- -Master/slave SPI sériové rozhraní
- -Programovatelný Watchdog Timer s odděleným vnitřním oscilátorem
- -Vnitřní analogový komparátor
- -Přerušení a přerušení na změnu Pinu

Speciální vlastnosti mikrokontroléru
- -Interně kalibrovatelný oscilátor
- -Externí a interní zdroje přerušení

I/O a pouzdro
- -22 programovatelných vstupů/výstupů
- -pouzdro TQFP32

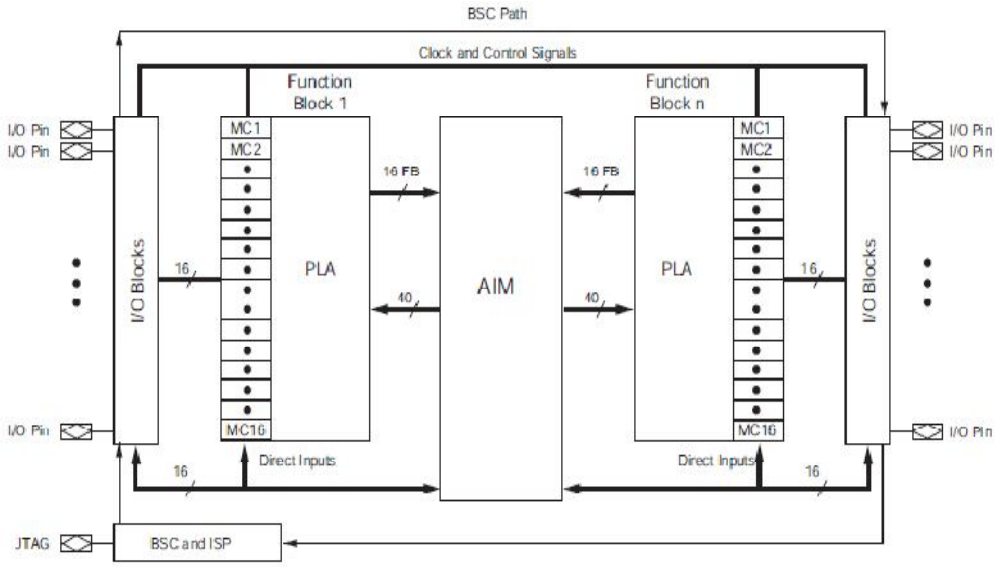
Pracovní napětí
- 2,7 – 5,5 V [3]

3.2 XILINX CoolRunner II, XC2C128

Informace o obvodu CPLD jsou čerpány z pramenu [2]. Je to obvod CPLD s napájením jádra napětím 1,8 V. Jeho základní vlastnosti jsou.

Základní vlastnosti:

Jen 13 uA klidový proud
- Hodinový dělič (dělení 2, 4, 6, 8, 10, 12, 14, 16)
- CoolClock
- DualEDGE
- PLA architektura
- 128 makrobuněk
- Nízká spotřeba, možné napájení z baterie
- Statický odběr menší než 100 uA
- Nevolatilní 0.18mikronové CMOS provedení
- Garantováno 1000 cyklů programování/mazání
- Garantováno 20 let udržení programu
- V použité verzi obvodu 80 I/O pinů (100 pinů celkem)
- I/O napěťově kompatibilní s 1,5 V, 1,8 V, 2,5 V, 3,3 V logikou
- Vstupy mohou být nakonfigurovány do funkce Schmidtova KO
- Možnost řízení registrů oběma hranami hodinového signálu (DualEDGE triggered registers)
- Volitelná možnost konfigurace nevyužitých I/O vývodů jako přídavných zemnicích vývodů
- Možnost zasouvání při připojeném napájení – hot plugging
- Programování ISP JTAG
- Podpora Boundary Scan [2]



Obrázek 3.1 Vnitřní struktura obvodu (převzato z [2])

Dělení kmitočtu:

Přiváděný signál lze děličkou kmitočtu dělit číslem (2, 4, 6, 8, 10, 12, 14, 16). Kmitočet je dělen se střídou 50 %. Je zabezpečeno, aby nedocházelo k rušení popřípadě k nějakým nechtěným impulzům

DualEDGE:

Touto volbou lze hodinový signál každé makrobuňky zdvojnásobit. Používá se například v aplikacích synchronní paměti.

CoolCLOCK:

Rozvod hodinového signálu znamená velikou spotřebu. Spotřeba se pohybuje v závislosti na frekvenci. Takže když se frekvence sníží na polovinu a následně se v makrobuňce povolí DualEDGE pro zdvojnásobení frekvence, tak se dostaneme na původní frekvenci, ale s nižší spotřebou.

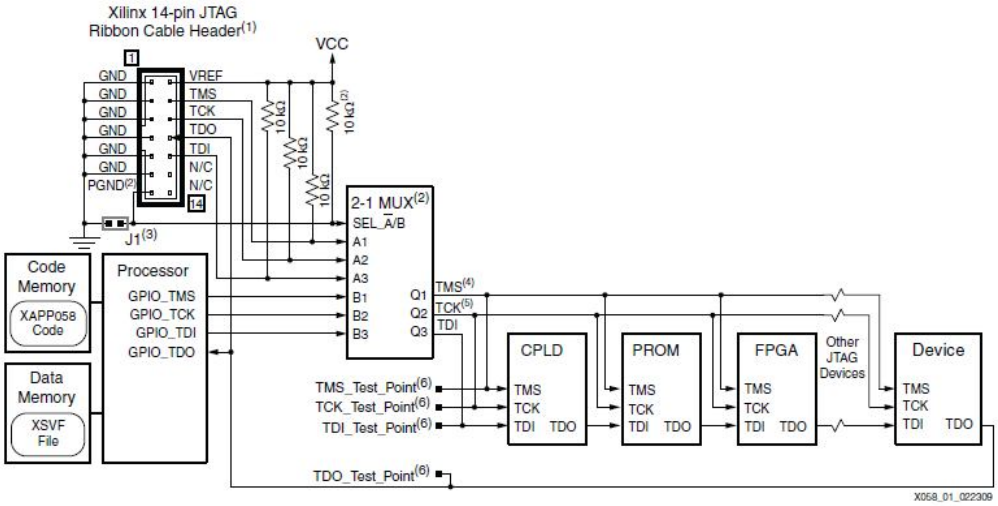
DataGATE:

CoolRunnery využívají CMOS architekturu i pro hradlové pole, což dovoluje snížení statického odběru. DataGate je funkce, která dokáže blokovat vstupy, na kterých volně běží signály, které v tu chvíli nejsou potřeba. Tím se u obvodů CoolRunner lze snížit spotřeba energie. Funkce DataGate je implementována u obvodů od velikosti 128 makrobuněk, což je i v našem případě. Každý vstup obsahuje spínač řízený touto funkcí a latch. Tato funkce využívá jeden vývod na obvodě. V případě nevyužití této funkce, lze vývod použít k běžnému použití.

3.2.1 Programování a vývoj

Programování tohoto obvodu je zprostředkováno přes standart IEEE 1149.1 ISP JTAG. Je to zkratka ze slov Joint Test Action Group. Jedná se o architekturu Boundary Scan, která slouží k testování plošných spojů, programování Flash pamětí apod.

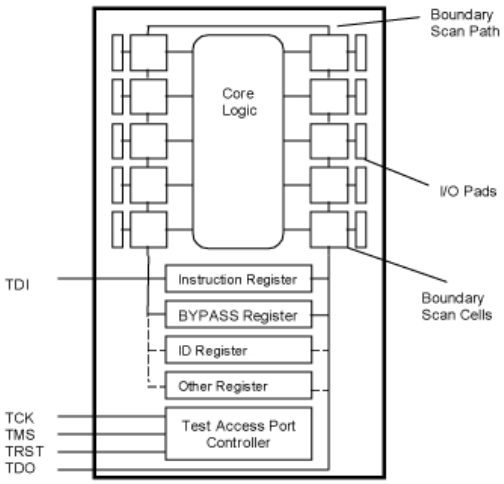
Pomocí použití rozhraní JTAG, lze Xilinx logické obvody snadno naprogramovat a testovat bez použití drahého hardware. Pro programování více logických obvodů zaráz se tyto logické obvody zapojí do řetězce. Toto zapojení lze vidět na následujícím obrázku. [5]



Obrázek 3.2 Blokové schéma řetězcového programování (převzato z [5])

Na obrázku 3.2 lze také vidět, že pomocí 2-1 MUX se volí cesta programování. Buď přes procesor a nebo přes externí programátor.

Programování probíhá přes čtyři povinné signály. Jsou to TCK, TMS, TDI, TDO a nepovinný TRMS. TCK je hodinový signál, TMS se používá pro posílání příkazů do zařízení, TDI a TDO se používají pro příjem/odesílání dat. Nepovinný signál TRMS, který vykonává reset, lze nahradit signálem TMS, nastaveným na logickou 1, po dobu minimálně pěti cyklů. To zaručeně provede reset. [5]



Obrázek 3.3 Náčrtek obvodu s řetězcem Boundary Scan (převzato z [6])

Podle stavu automatu je do řetězce mezi TDI a TDO připojen určitý registr. Základní povinné registry jsou instrukční registr, boundary scan registr, bypass registr a nepovinný ID registr, který slouží k detekci typu a výrobce součástky. Jednotlivé testované obvody se propojují do tzv. boundary řetězce seriově pomocí TDI a TDO. Ostatní signály jsou paralelně připojeny na každou součástku. Jelikož existuje bypass registr, který pouze jakoby propojuje TDI a TDO, můžeme kdykoliv "odpojit" veškeré prvky řetězce a ponechat aktivní pouze jeden obvod, se kterým budeme pracovat. Test Access Port (TAP) slouží pro kontrolu počtu zařízení Xilinx nebo ostatních zařízení kompatibilních s JTAG. [5]

Vývoj aplikace, která se následně bude programovat do obvodu CPLD, lze vytvořit v softwaru WebPACK, dodávanému společností XILINX zdarma, která je dostupná z jejich webových stránek.

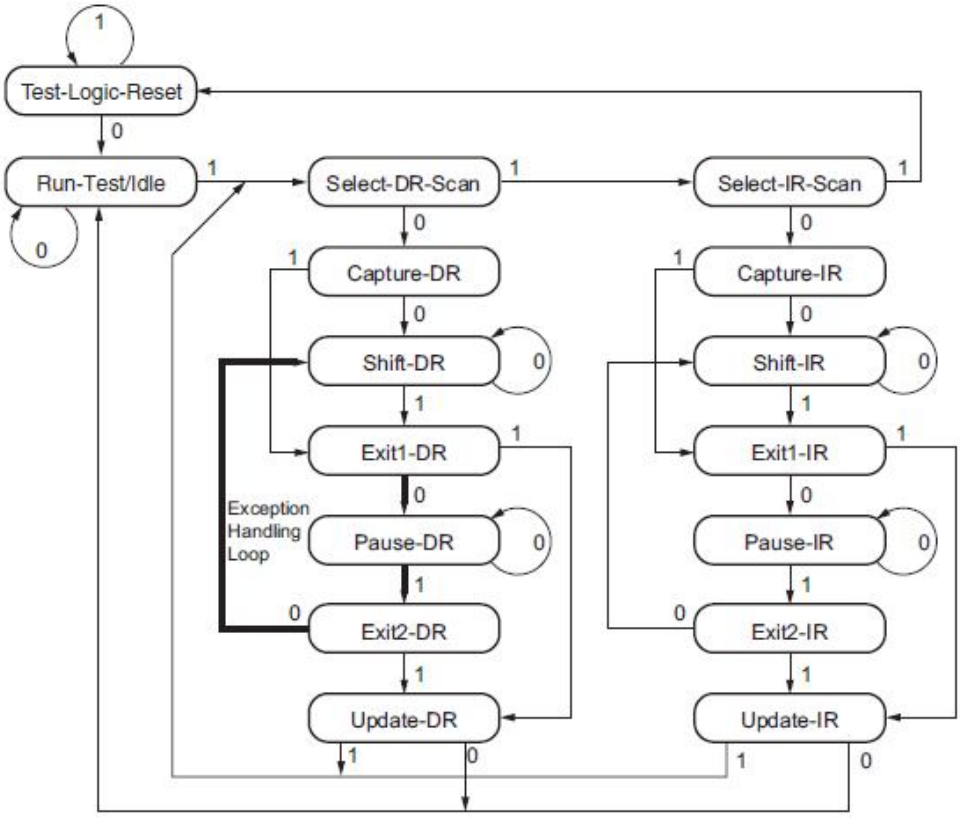
3.2.2 Realizace programátoru

Nejjednodušším způsobem, jak naprogramovat obvody firmy Xilinx, je přes paralelní port, pro který Xilinx dalo veřejnosti k dispozici schématické zapojení JTAG programátoru. Tímto programátorem lze bez problémů naprogramovat přípravek v této práci realizovaný. Schéma a potřebné materiály jsou k dispozici na adrese: *http://www.xilinx.com/itp/3_1i/pdf/docs/jtg/jtg.pdf*

Realizace JTAG programátoru přes mikrokontrolér je řešen v aplikační poznámce XAPP058 od Xilinx. Programování probíhá pomocí souboru XSVF, který vychází z typu SVF (Serial Vector Format). SVF je specifická syntaxe popisující operace JTAG (IEEE Std 1149.1) sběrnice. Je to textový soubor, který obsahuje příkazy a parametry oddělené mezerami a ukončené středníkem. Pomocí příkazů obsažených v souboru SVF se řídí TAP (Test access port). Typ souboru XSVF je binární verze SVF souboru. Tento formát je velikostně mnohem úspornější než SVF a užívá se častěji než formát SVF. Příklad celého souboru SVF, který vykonává jen mazání obvodu XC2C128, je dodán v příloze C. [5]

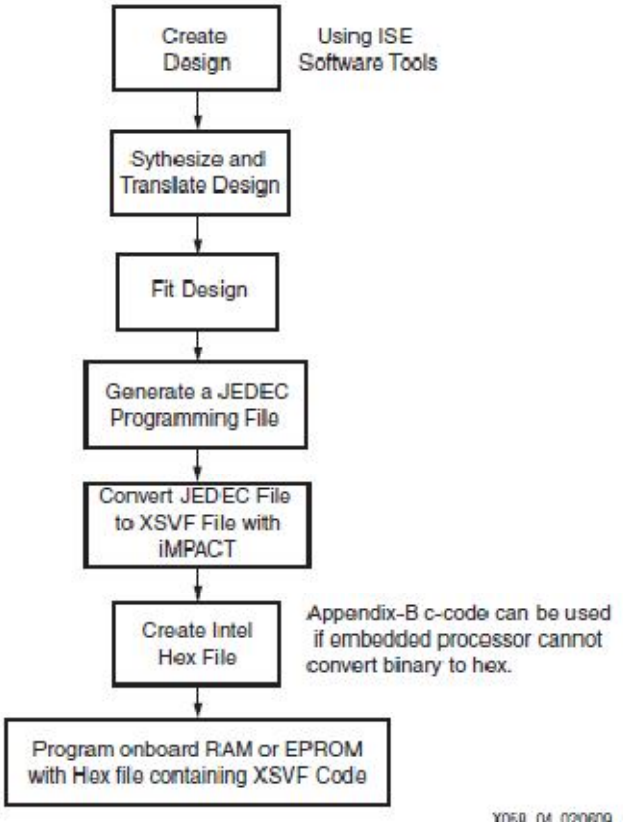
Popis některých příkazů:

STATE	uvede JTAG rozhraní do definovaného stavu
ENDIR	definuje stav, kde se má skončit po instrukci SIR
ENDDR	definuje stav, kde se má skončit po instrukci SDR
HIR	definuje hlavičku, která se bude odesílat před každou SIR instrukcí
HDR	definuje hlavičku, která se bude odesílat před každou SDR instrukcí
SIR	odeslání dat do IR registru
SDR	odeslání dat do DR registru
RUNTEST	přepne JTAG do stavu běh (RUNSTATE) na definovanou dobu
! nebo //	komentář



Obrázek 3.4 Stavový diagram kontroléru (převzato z [5])

Tvorba programovacího souboru je zřejmá z obrázku 3.5. Navrhne se program v jazyce VHDL, provede se syntéza a překlad kódu. Provede se vhodný návrh obvodu a poté se vygeneruje programovací soubor typu JEDEC. V případě programování pomocí paralelního rozhraní počítače programátorem uveřejněným společností Xilinx to stačí, v tom případě se soubor JEDEC pošle pomocí softwaru iMPACT do zařízení. V případě programování pomocí procesoru se v iMPACT provede převod JEDEC souboru na XSVF. V podstatě jde o zadání jednotlivých úkonů, které bychom zadávali při práci s připojeným programátorem Xilinx s tím rozdílem, že tady programátor připojený nemáme a instrukce se nevykonávají. Tedy úkony, které zadáváme, se v přesném pořadí zapíší do XSVF souboru v podobě instrukcí pro TAP. To znamená, soubor XSVF je jakoby manuál, jež má vykonat programátor, jelikož už neobsahuje jen převedený JEDEC kód ale i instrukce, které provádějí mazání CPLD obvodu před samotným programováním, atd. Poté se vytvoří Intel Hex soubor a v případě, že procesor neumí převést binární kód do hexadecimálního, slouží k tomu zdrojový kód v jazyce C (Appendix-B), uveřejněný v XAPP058. Poté se Hex soubor, obsahující XSVF kód, nahraje do paměti na desce (např. paměť procesoru). [5]



Obrázek 3.5 Tvorba a uložení CPLD programovacího souboru (převzato z [5])

Hardwarové požadavky na programování:

- je potřeba mikroprocesor
- dostatečná paměť pro pojmutí zkompilovaného C kódu a XSVF souboru
- 3 procesorem řízené výstupy pro JTAG TCK, TMS a TDI signály
- 1 procesorem čtený vstup pro JTAG TDO signál

Přibližná velikost XSVF souboru pro obvod XC2C128 se pohybuje kolem 50 bytů. Mikrokontrolér AT90USB162 disponuje pamětí Flash 16 kB, RAM 512 B a EEPROM 512 B. Z toho důvodu usuzuji, že paměťové dispozice procesoru jsou dostačující. Pro programování CPLD jsou využity piny 0, 1, 2, 3 Portu B. Procesor je programovatelný přes USB sběrnici pomocí programu Flip 3.4, který je dodáván od výrobce Atmel.

PB0	TMS
PB1	TCK
PB2	TDI
PB3	TDO

Tabulka 3.1 Zapojení rozhraní JTAG k MCU

Aplikační zpráva XAPP058 uvádí základní kroky pro implementaci firmwaru pro užitý mikrokontrolér.

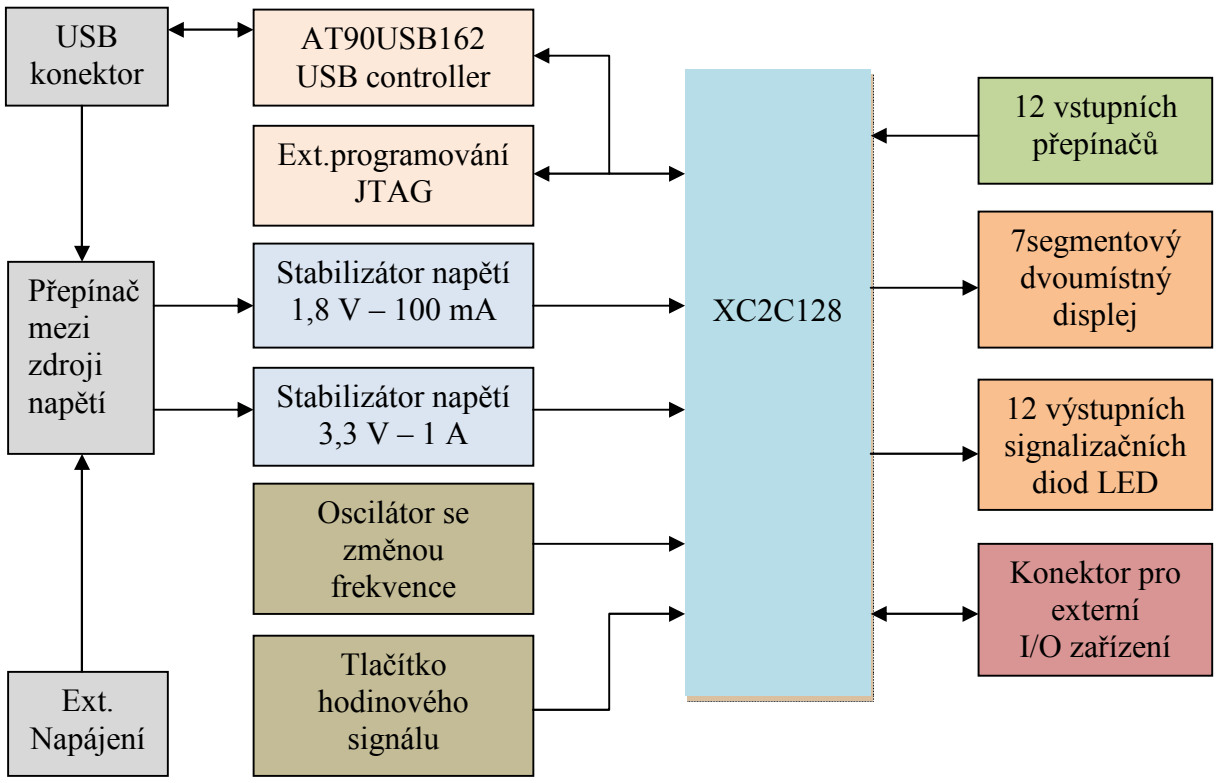
1) Definice JTAG řetězce a programovacích příkazů
   Úkolem je identifikovat zařízení, která jsou zapojena do JTAG řetězce a poté identifikovat cílové zařizení, které bude programováno mikrokontrolérem.
2) Určení velikosti XSVF souboru
   Po identifikaci cílového zařízení může být zjištěna velikost XSVF souboru. Poté následuje generování XSVF souboru aplikací iMPACT software s přímo instrukcemi pro cílové zařízení.
3) Určení místa pro XSVF soubor
   Po zjištění velikosti XSVF souboru dojde k určení místa, kde bude uložen. Konfigurační soubor může být uložen staticky v nevolatilní paměti mikrokontroléru nebo může být dynamicky přenášený do RAM paměti procesoru.
4) Rozdělení procesorového zdroje
   Velikost zdrojového kódu závisí především na instrukční sadě a kompilátoru. Provozní požadavky jsou v první řadě určeny velikostí datové struktury SXsvfInfo, definované ve zdrojovém souboru micro.c. Velikost SXsvfInfo je ovládaná datovou strukturou lenVal. LenVal definuje vyrovnávací paměť pro bitové hodnoty, užívané v JTAG posuvných operacích. Velikost lenVal vyrovnávací paměti je definovaná příkazem MAX_LEN ve zdrojovém souboru lenval.c. Přednastavená velikost MAX_LEN je na 7,000 bytes. Tato hodnota není potřeba měnit.
5) Návrh I/O procesoru
   Jde o užití jednotlivých pinů procesoru k řízení a čtení JTAG signálů.
6) Úprava zdrojového kódu ports.c
   Klíčové funkce, které implementují porty procesoru. Funkce setPort nastavuje výstupní signály (TCK, TMS, TDI) v procesoru. Funkce readTDOBit má za úkol číst vstupní signál TDO. Funkce readByte vrací další byte z XSVF souboru. Její funkcí je i to, aby si pamatovala místo posledně načteného bytu v paměti. A funkce waitTime udává hodnotu v mikrosekundách, je to čekání na dokončení probíhající operace.

V XAPP058 se odvolávají na zdrojové kódy, které slouží k čtení XSVF souborů a k řízení TAP kontrolních bitů.

lenval.c – tento zdrojový kód obsahuje pokyny k používání lenVal datové struktury
micro.c – obsahuje volání základních funkcí pro čtení XSVF z paměti, interpretaci XSVF příkazů a řízení JTAG signálů.
ports.c – obsahuje pravidla pro výstupní JTAG porty, čtení TDO bitu a čtení bytů XSVF z paměti.

Poté je odkaz na hlavičkové soubory, které nesou stejná jména jako zdrojové kódy. Jsou to lenval.h, micro.h a ports.h.
Je potřeba před kompilací těchto kódů provést 4 úpravy funkcí ve zdrojovém kódu ports.c a to funkce setPort, readTDOBit, readByte a waitTime. [5]

4 NÁVRH SCHÉMATU PŘÍPRAVKU



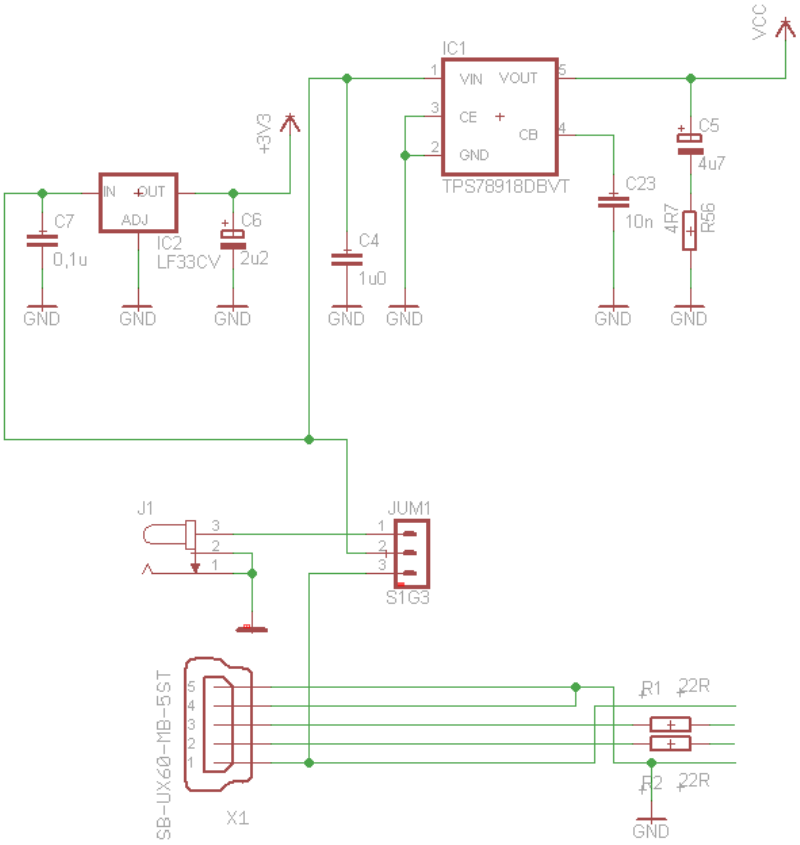
Obrázek 4.1 Blokové schéma přípravku

4.1 Napájení

Napájení celého přípravku je navrženo tak, že je možnost přípravek napájet jednak přímo z USB sběrnice nebo je možnost napájet obvod z externího napájecího zdroje stejnosměrného napětí 5 V. K volbě napájení slouží trojice pinů propojených pomocí propojek. Následnou stabilizaci napětí zabezpečují dva stabilizační integrované obvody. Integrovaný obvod LF33CV zajišťuje stabilizaci napětí na úroveň 3,3 V, u jehož zapojení se vychází z doporučeného dodávaného v jeho datasheetu. Tento stabilizátor je určený až do proudu 1 A. Zajišťuje napájení veškerých periferií v tomto přípravku včetně napěťových úrovní na vývodech hradlového pole. A pomocí obvodu TPS78918DBVT se napětí stabilizuje na úroveň 1,8 V s maximální proudovou zatížitelností 100 mA. Toto napětí zajišťuje napájení jádra celého hradlového pole XC2C128. Z důvodů nízké spotřeby tohoto pole tato proudová zatížitelnost postačuje. U stabilizátoru na 3,3 V je použit stabilizátor na 1 A z důvodu proudové náročnosti 7segmentového displeje a signalizačních diod. Pro získání stabilnějšího napětí jsou přidány na vstupy napájení CPLD kapacitou o hodnotě 100 nF.

Číslo pinů	Volba
1 – 2	Externí napájení
2 – 3	USB napájení

Tabulka 4.1 Výběr napájení



Obrázek 4.2 Schéma napájení

4.2 AT90USB162 USB kontrolér a programování

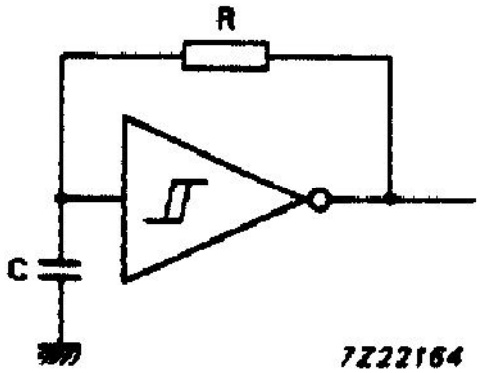
Pro programování CPLD je využit obvod AT90USB162 od firmy Atmel, který má v sobě implementován USB kontrolér. U tohoto obvodu je vycházeno z doporučeného zapojení podle datasheetu. Pro tento obvod je použit 16 MHz krystal. Tento obvod dovoluje užívání dvou napěťových standardů, jak 5 V tak i 3,3 V a to díky vnitřnímu napěťovému regulátoru. Změna standardu lze změnit jen způsobem zapojení. V tomto přípravku je využit 3,3 V. Programování bude možné jednak přes USB přímo z počítače přes obvod AT90USB162 a nebo přes externí programátor standardu JTAG, definovaný normou IEEE 1149.1. Pro externí způsob programování je vyvedena šestice pinů na konektoru JUM2, jejíž rozpoložení je uvedeno v tabulce. Odpojení obvodu AT90USB162 se provádí přes skupinu propojek JP4.

Číslo pinu	Obsazení
1	TMS
2	TCK
3	TDI
4	TDO
5	GND
6	+3V3

Tabulka 4.2 Zapojení programovacího konektoru

4.3 Oscilátor se změnou frekvence

Oscilátor je tvořen pomocí integrovaného obvodu 74HC14, což je Schmidtův obvod. Je připojen na hodinový vstup GC3. Tento obvod má 6 Schmidtových invertorů. Vstupy nevyužitých invertorů jsou ošetřeny připojením na GND. Oscilátor je navržen, aby osciloval ve čtyřech různých frekvencích. Jsou to 5 Hz, 50 Hz, 500 Hz a 5000 Hz. Návrh byl proveden podle výpočetního vztahu (4.1):



Obrázek 4.3 Katalogové zapojení oscilátoru (převzato z [7])

$$f = \frac{1}{T} \approx \frac{1}{0{,}8RC} \tag{4.1}$$

Rezistor R je zvolen na 47 kΩ, takže úpravou vztahu budeme dopočítávat kondenzátory, tak abychom přibližně dosáhli požadované frekvence.

Pro f = 5 Hz:

$$C = \frac{1}{0{,}8fR} = \frac{1}{0{,}8 \cdot 5 \cdot 47000} = 5{,}32 \text{ uF} \tag{4.2}$$

Volíme kondenzátor o kapacitě 4,7 uF
Následným dopočtem zjistíme přesnou hodnotu frekvence:

$$f = \frac{1}{T} \approx \frac{1}{0{,}8RC} = \frac{1}{0{,}8 \cdot 47000 \cdot 4{,}7 \cdot 10^{-6}} = 5{,}65 \text{ Hz} \quad (4.3)$$

Pro f = 50 Hz:

$$C = \frac{1}{0{,}8fR} = \frac{1}{0{,}8 \cdot 50 \cdot 130000} = 531{,}9 \text{ nF} \quad (4.4)$$

Volíme kondenzátor o kapacitě 470 nF

$$f = \frac{1}{T} \approx \frac{1}{0{,}8RC} = \frac{1}{0{,}8 \cdot 47000 \cdot 470 \cdot 10^{-9}} = 56{,}58 \text{ Hz} \quad (4.5)$$

Pro f = 500 Hz:

$$C = \frac{1}{0{,}8fR} = \frac{1}{0{,}8 \cdot 500 \cdot 47000} = 53{,}2 \text{ nF} \quad (4.6)$$

Volíme kondenzátor o kapacitě 47 nF

$$f = \frac{1}{T} \approx \frac{1}{0{,}8RC} = \frac{1}{0{,}8 \cdot 47000 \cdot 47 \cdot 10^{-9}} = 565{,}86 \text{ Hz} \quad (4.7)$$

Pro f = 5000 Hz:

$$C = \frac{1}{0{,}8fR} = \frac{1}{0{,}8 \cdot 5000 \cdot 47000} = 5{,}3 \text{ nF} \quad (4.8)$$

Volíme kondenzátor o kapacitě 4,7 nF

$$f = \frac{1}{T} \approx \frac{1}{0{,}8RC} = \frac{1}{0{,}8 \cdot 47000 \cdot 4{,}7 \cdot 10^{-9}} = 5658{,}66 \text{ Hz} \quad (4.9)$$

Tímto jsou určené hodnoty součástek realizující oscilátor. Přeměna frekvencí se děje pomocí propojek na desce. Volba frekvence je zobrazena v následující tabulce.

Označení pinů	Frekvence
1 – 2	56 Hz
3 – 4	566 Hz
5 – 6	5 Hz
7 – 8	5660 Hz

Tabulka 4.3 Volba frekvence propojením

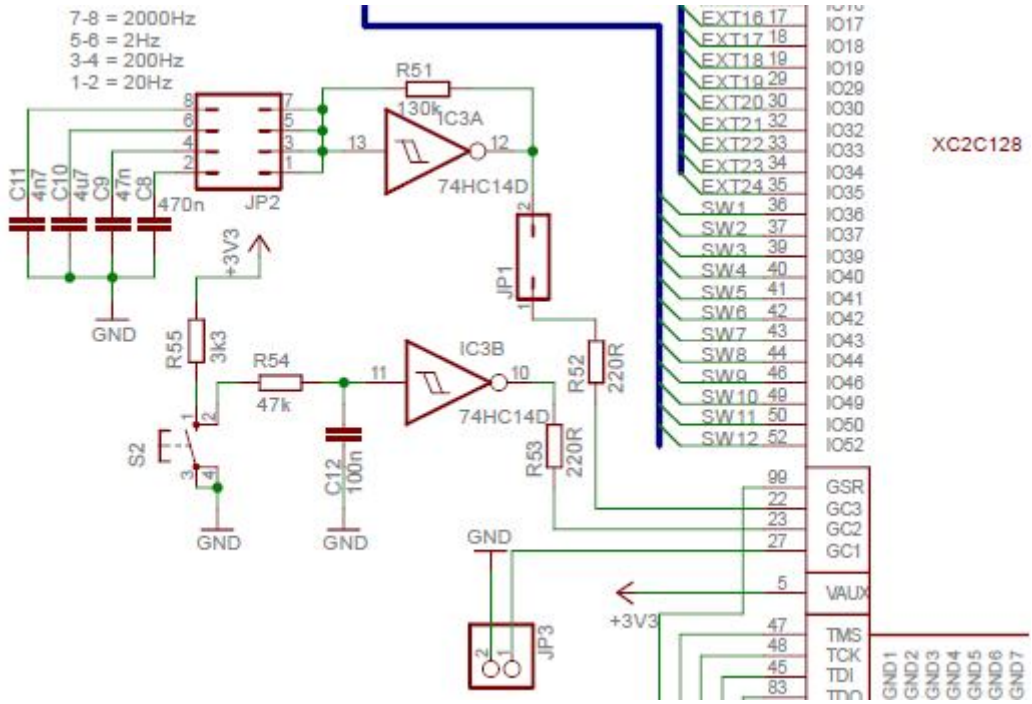
Pomocí propojky JP1 lze oscilátor odpojit od CPLD. A využívat jako hodinový vstup je možné tlačítko S2, připojené na druhý hodinový vstup GC2. Toto tlačítko je taktéž

realizováno přes Schmidtův invertor a je ošetřen mechanický zákmit tlačítka. Pro výpočet dolní propusti pro ošetření jsem zvolil 30 Hz a R jsem zvolil 47 kΩ.

$$f = \frac{1}{2\pi \cdot R \cdot C} => C = \frac{1}{f \cdot 2\pi \cdot R} = \frac{1}{30 \cdot 2\pi \cdot 47000} = 112{,}9\ \text{nF} \qquad (4.10)$$

Zvolil jsem kondenzátor o velikosti 100 nF.

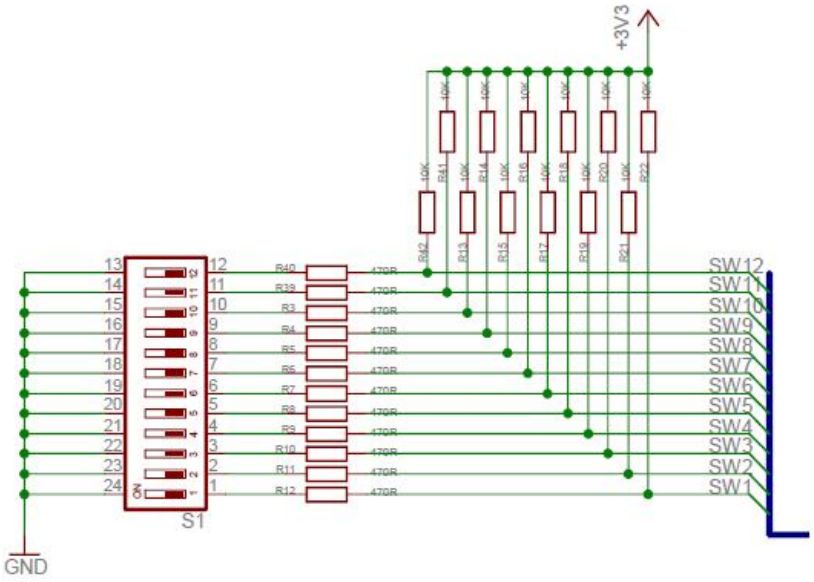
Na třetí hodinový vstup GC1 je připojen konektor JP3, pomocí kterého je možno připojit externí hodinový signál. Na pinu 2 je přivedena zem.



Obrázek 4.4 Schéma zapojení oscilátoru a tlačítka hodinového vstupu

4.4 Vytvoření vstupních signálů

Jako vstup je zvoleno dvanáct přepínačů, pomocí kterých je možno přivádět do hradlového pole binárně číslo až 4096. Zapojení je provedeno následujícím způsobem.



Obrázek 4.5 Schéma zapojení přepínače vstupů

Při rozepnutém stavu spínače dochází k tomu, že na vstup logického obvodu je přivedeno napětí o hodnotě 3,3 V. Rezistor o hodnotě 10 kΩ, který zaručuje, že do obvodu nepoteče větší proud než

$$I = \frac{U}{R} = \frac{3{,}3}{10000} = 330\ \text{uA} \tag{4.11}$$

Tímto je reprezentována vysoká úroveň na vstupu logického obvodu.
Při sepnutém stavu spínače dojde k uzemnění a vznikne dělič napětí, který je sestaven z rezistorů 10 kΩ a 470 Ω. Tím dosáhneme toho, že na vstup logického obvodu bude přivedeno napětí o velikosti

$$U_2 = \frac{R_1}{R_1+R_2} \cdot U_1 = \frac{470}{470+10000} \cdot 3{,}3 = 0{,}148\ \text{V} \tag{4.12}$$

Tímto je určena nízká úroveň na vstupu obvodu.

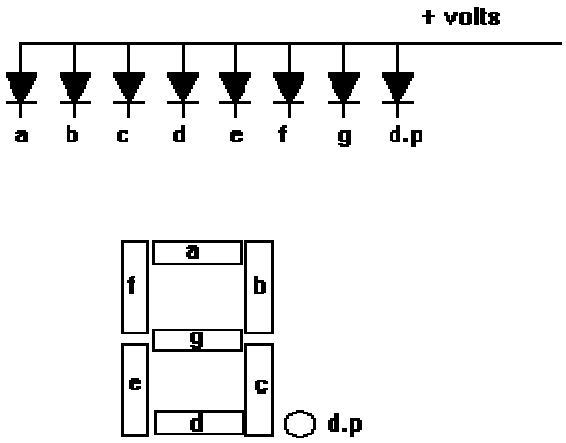
4.5 7segmentový výstupní LED displej

Jenou možností zobrazení výstupu funkce naprogramované do CPLD je 7segmentový displej, který je řízený staticky. Tento displej má dva digity a jeho zapojení je se společnou anodou. To znamená, že na společnou anodu každého digitu bude připojeno napájecí napětí 3,3 V.

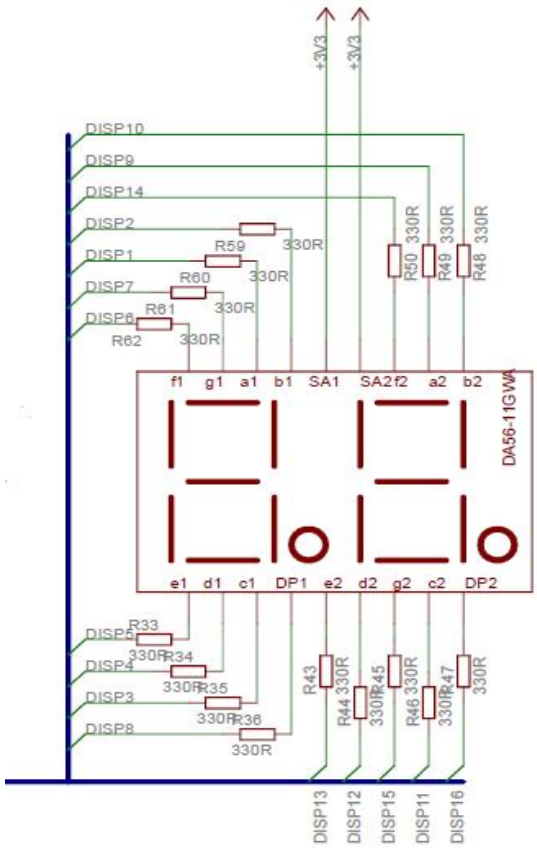
Výstupy každého segmentu je přes rezistor R o velikosti

$$R = \frac{U}{I} = \frac{3{,}3}{10 \cdot 10^{-3}} = 330\ \Omega \tag{4.13}$$

A bude jím protékat maximální proud I = 10 mA. Rozsvícení jednotlivých segmentů je provedeno přivedením nízké úrovně na výstup z obvodu CPLD.



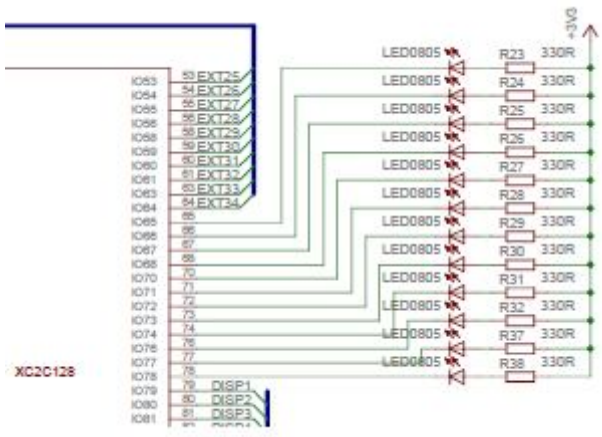
Obrázek 4.6 Vnitřní zapojení 7segmentového displeje



Obrázek 4.7 Schéma zapojení 7segmentového displeje

4.6 Výstupní signalizační diody LED

K výstupní signalizaci je možno také využít 12 LED diod. Ty jsou zapojeny tak, že na anodu diody je přivedeno napětí 3,3 V přes rezistor snižující maximální průchod proudu diodou. Katoda diody je připojena k obvodu CPLD. Rozsvícení diody je provedeno nízkou úrovní na výstupu obvodu CPLD. Zapojení je vidět na obrázku níže. Velikost odporu je vypočtena na hodnotu 330 Ω, který zaručuje, že nebude diodou LED protékat větší proud než 10 mA.



Obrázek 4.8 Schéma zapojení signalizačních diod LED

4.7 Konektor pro externí I/O zařízení

Pro připojení a rozšíření vstupních nebo výstupních zařízení je připraven konektor typu BLW240G. Je to dvouřadý konektor lomený o 90º od desky plošného spoje. Je možné pomocí tohoto konektoru využít 34 vstupů/výstupů obvodu XC2C128. Na konektoru je vyvedeno napájecí napětí 3,3 V a také zem GND. Zapojení konektoru je uvedeno v následující tabulce.

PIN		PIN		PIN		PIN	
1	GOE1	11	I/O	21	I/O	31	I/O
2	GOE2	12	I/O	22	I/O	32	I/O
3	GOE3	13	I/O	23	I/O	33	I/O
4	GOE4	14	I/O	24	I/O	34	I/O
5	I/O	15	I/O	25	I/O	35	+3V3
6	I/O	16	I/O	26	I/O	36	+3V3
7	I/O	17	I/O	27	I/O	37	GND
8	I/O	18	I/O	28	I/O	38	GND
9	I/O	19	I/O	29	I/O	39	GND
10	I/O	20	I/O	30	I/O	40	GND

Tabulka 4.4 Zapojení konektoru pro externí zařízení

GOE = Global output enable (může se využít i jako vstup)

5 REALIZACE

Realizace laboratorního přípravku probíhala navržením plošného spoje, výrobou plošného spoje, osazením a oživením přípravku a sestavením několika příkladů ve VHDL kódu.

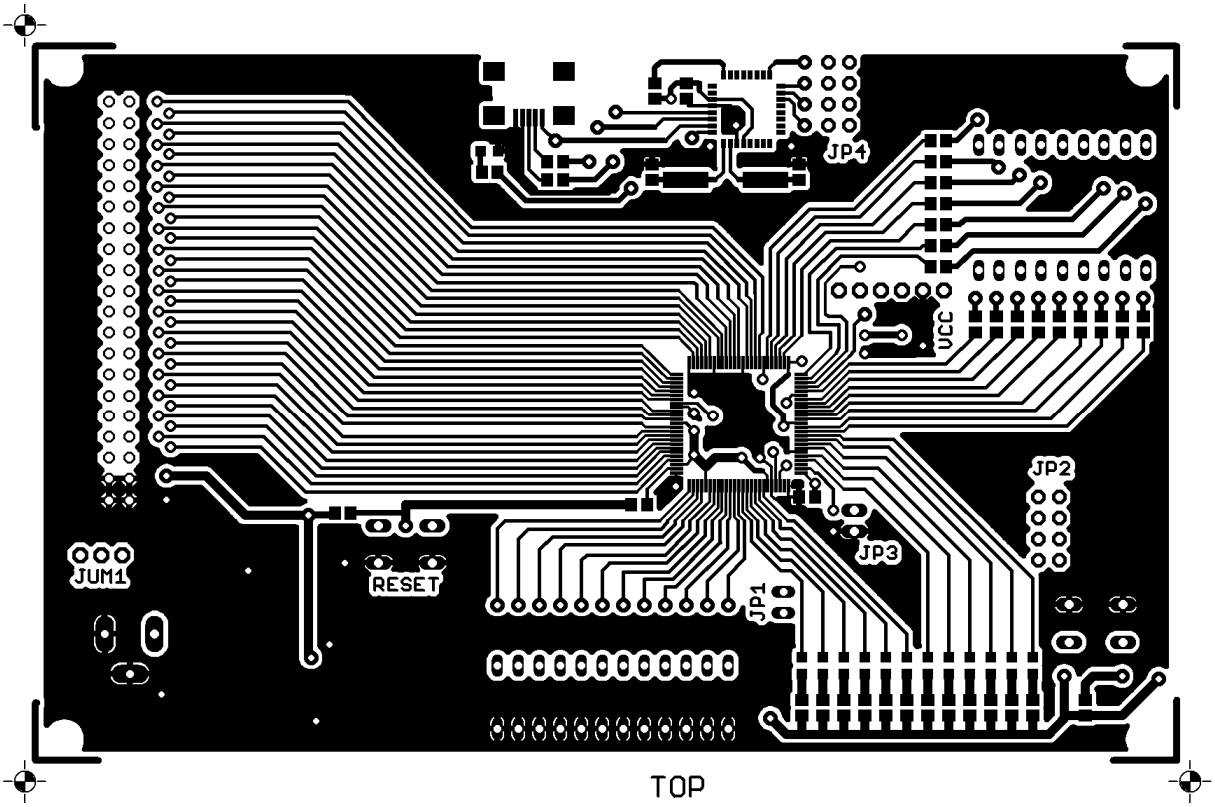
5.1 Návrh plošného spoje

Návrh byl proveden v návrhovém softwaru EAGLE a soubory návrhu jsou přiloženy na doprovodném CD. Přípravek je realizován jako dvouvrstvá deska plošného spoje s prokovenými spoji.

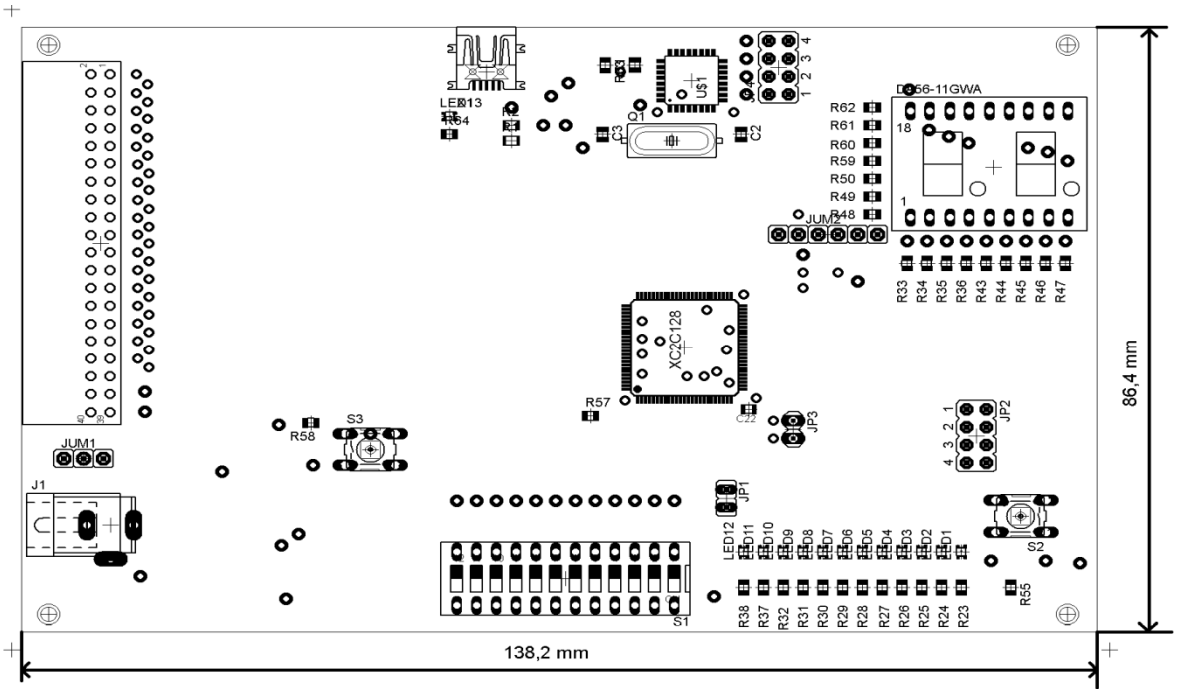
Rozměry desky plošného spoje jsou: šířka 138,2 mm, výška 86,4 mm.

Nejmenší průměr vrtaných děr je 0,6 mm.

5.1.1 Předloha pro výrobu DPS – TOP

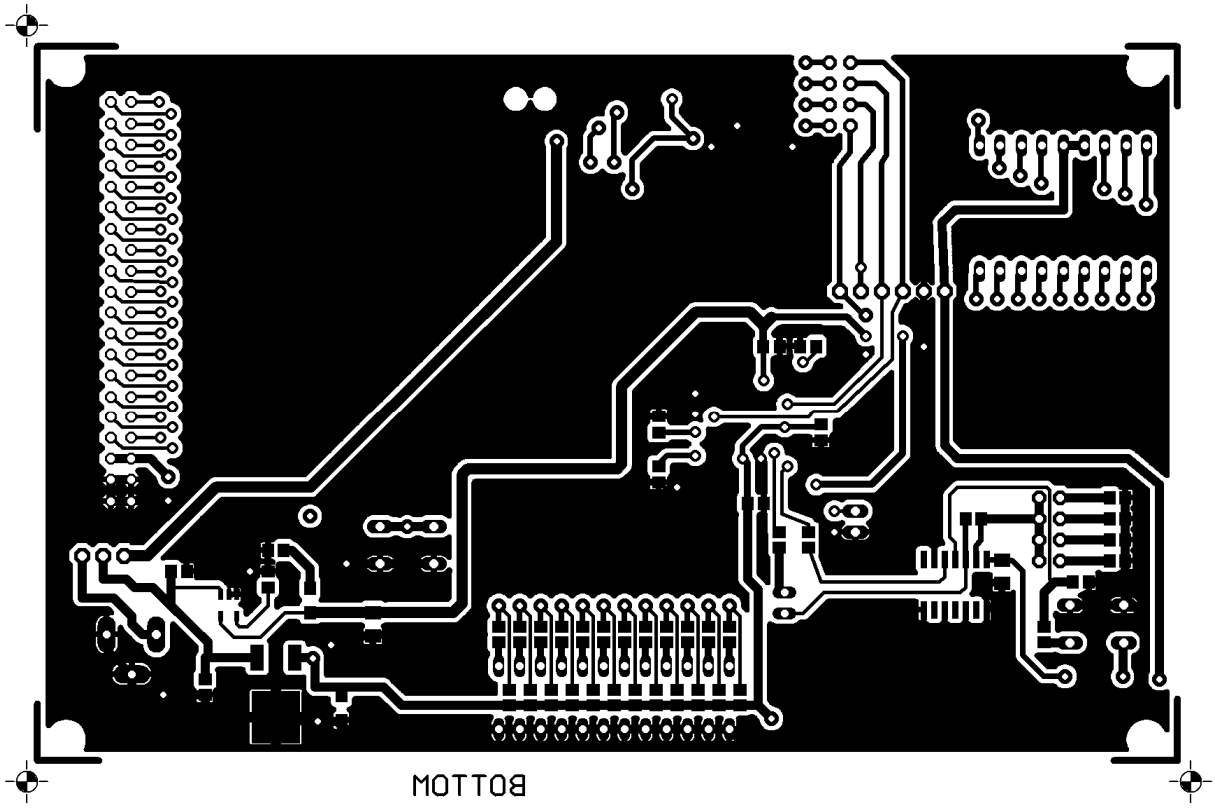


Obrázek 5.1 Předloha pro výrobu DPS - vrstva TOP

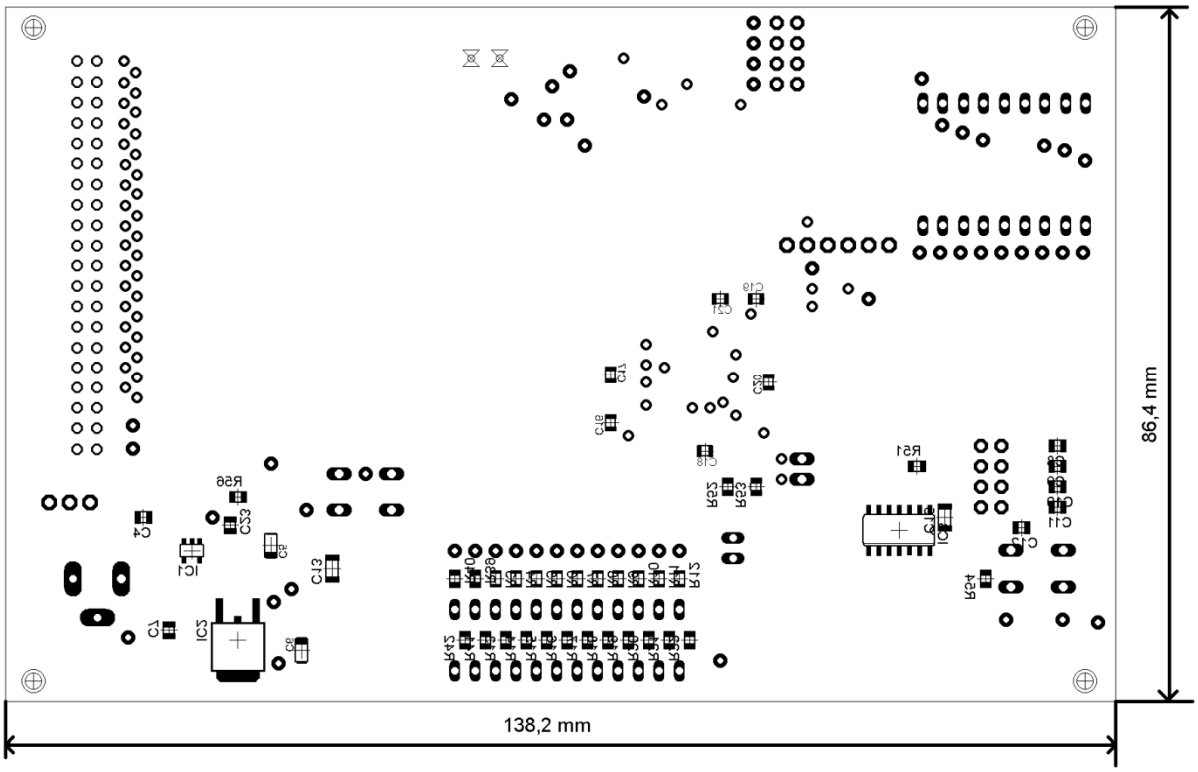


Obrázek 5.2 Osazovací plánek DPS - vrstva TOP

5.1.2 Předloha pro výrobu DPS – BOTTOM

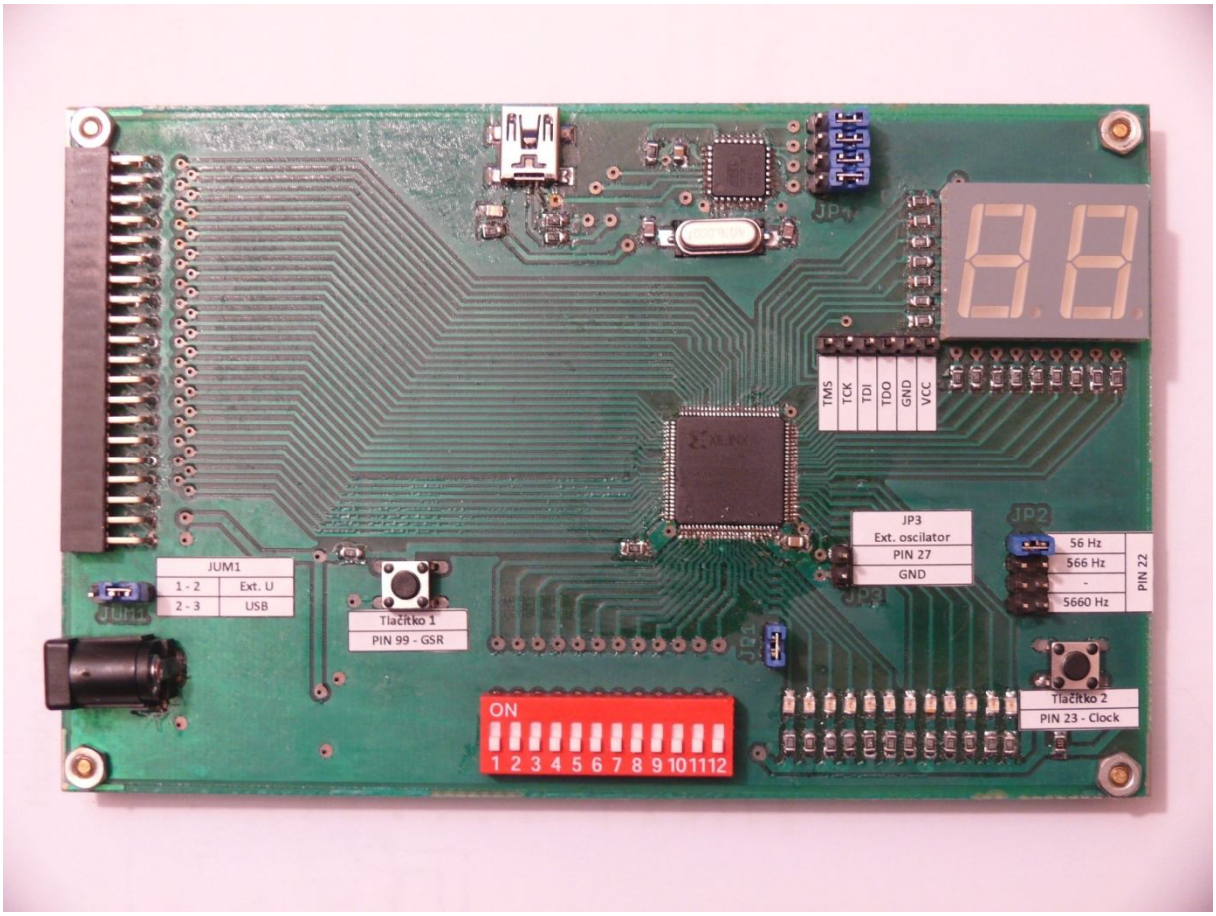


Obrázek 5.3 Předloha pro výrobu DPS - vrstva BOTTOM



Obrázek 5.4 Osazovací plánek DPS - vrstva BOTTOM

5.2 Osazená deska



Obrázek 5.5 Osazená deska

5.3 Příklady programů ve VHDL jazyce

Pro tento laboratorní přípravek jsem v softwaru Xilinx ISE navrhl příklady programů, které simulují funkčnost přípravku. Všechny tady uvedené příklady jsou na přípravku odzkoušené a funkční.

1. Příklad STOPKY

Jedná se o stopky, které jsou navrženy tak, že vteřiny se zobrazují na dvoumístném 7segmentovém displeji, minuty se zobrazují pomocí 12 diod LED. Takže tyto stopky mají omezenou dobu stopování. Dosahují 12 minut a 59 vteřin. Je to omezeno periferiemi. Start a Stop je realizovaný pomocí Tlačítka 2. Pomocí tlačítka 1 je realizováno nulování stopek.

Zdrojový kód ke stopkám je uveden v příloze D.1.

2. Příklad KÓDOVÝ ZÁMEK

Je to příklad jednoduchého zabezpečovacího systému, ve kterém je zadávací vstupní periferie soustava přepínačů. K ovládání zařízení slouží tlačítka 1, 2. Tlačítkem 1 je možné kódovou kombinaci nastavenou na přepínačích uložit do paměti. Pomocí tlačítka 2 probíhá vždy potvrzení nastavené kombinace, při kterém se ověřuje shodnost s bezpečnostním kódem. V případě, že je zadaná kombinace shodná, rozsvítí se všech 12 diod LED a na 7segmentovém displeji se zobrazí OP (Open). V případě neshody s bezpečnostním kódem diody LED zůstanou zhaslé a na displeji se zobrazí Er (Error).

Zdrojový kód je uveden v příloze D.2.

3. Příklad BĚŽÍCÍ SVĚTLO

Je to jednoduchý program, při kterém se postupně rozsvěcují všechny diody LED. Až jsou rozsvícené všechny, dojde k zhasnutí a vše se znovu opakuje.

Zdrojový kód je uveden v příloze D.3.

4. Příklad SYNCHRONNÍ KLOPNÝ OBVOD TYPU J-K

Je to program, který realizuje klopný obvod typu J-K. Jako vstupní hodnota J, slouží přepínač číslo 1 a jako hodnota K přepínač číslo 12. Nastavené hodnoty J a K jsou hned zobrazeny pomocí diod LED na pozicích odpovídající přepínačům. Svítící dioda LED značí logickou jedničku. Výstup obvodu J-K je realizovaný pomocí 2. segmentu na 7segmentovém displeji. Řídící hodinový signál se ovládá tlačítkem 2.

Zdrojový kód je uveden v příloze D.4.

5. Příklad 12BITOVÝ ČÍTAČ

Tento program je 12bitový inkrementující čítač s asynchronní nulování a s povolením čítat. Pro hodinový signál, který realizuje čítání je užit oscilátor na pinu 22. V programu je tento kmitočet dělen 30. Povolování je navrženo pro tlačítko 2. A asynchronní nulování je pomocí tlačítka 1. Výstup čítače je zobrazován na diodách LED.

Zdrojový kód je uveden v příloze D.5.

5.4 Oživení programátoru pomocí AT90USB162

Oživit JTAG programátor pomocí procesoru se mi nepodařilo, jelikož zdrojové kódy potřebné k realizaci programátoru, o kterých se zmiňuje aplikační poznámka XAPP058 chybí. Společnost Xilinx, která uvádí místo možného stažení z jejich webu, už soubory odstranila a při hledání na internetu se mi je bohužel nepodařilo najít. Za další, jejich aplikační zpráva se mi zdá poněkud velmi obecně napsána a neposkytuje dostatečné množství informací k realizaci JTAG programátoru, bez poskytnutých zdrojových kódů společností Xilinx. Mikrokontrolér, který je osazen na laboratorním přípravku, je ověřen, že je funkční. Z důvodů nefungujícího programování přes rozhraní USB jsem CPLD programoval přes externí programátor skrze paralelní port, který je uveřejněný společností Xilinx, jejíž odkaz na schéma je uveden výše.

6 ZÁVĚR

Tato bakalářská práce v sobě shrnuje jednak teoretické podklady pro realizaci laboratorního přípravku, tak také realizaci samotnou. Teoretická část obsahuje kompletní návrh funkčního schémata přípravku a teoretické podklady pro realizaci programování obvodu CPLD rozhraním JTAG. Realizační část obsahuje kompletní návrh plošného spoje, který byl po výrobě osazen. Vyhotovená deska je obvodově funkční. Co se týče oživení, se mi nepodařilo zrealizovat programování pomocí mikrokontroléru AT90USB, z důvodu nedostatku materiálů pro realizaci. Zejména zdrojových kódů, na které se odkazuje aplikační poznámka výrobce obvodu CPLD. Zbylá část desky s obvodem CPLD je plně funkční a oživená. Příklady v jazyce VHDL, které jsou vyhotoveny, jsou funkční a na přípravku jsou řádně přezkoušeny.

LITERATURA

[1] BURKHARD, K. *USB – měření, řízení a regulace pomocí sběrnice USB*, Praha: BEN – technická literatura, 2002.

[2] CoolRunner-II CPLD Family. Dostupné na www: http://www.xilinx.com/support/documentation/data_sheets/ds090.pdf

[3] ATMEL AT90USB82/162. Dostupné na www: http://www.atmel.com/dyn/resources/prod_documents/doc7707.pdf

[4] KOLOUCH, J. Programovatelná logika proniká do všech oblastí číslicové technik[online], 2006. Dostupné na www: http://www.odbornecasopisy.cz/index.php?id_document=30927

[5] Xilinx In-System Programming Using an Embedded Microcontroller. Dostupné na www: http://www.xilinx.com/support/documentation/application_notes/xapp058.pdf.

[6] JTAG – A technical overview. Dostupné na www: http://www.xjtag.com/support-jtag/jtag-technical-guide.php

[7] Datasheet 74HC14. Dostupné na www: http://pdf1.alldatasheet.com/datasheet-pdf/view/15537/PHILIPS/74HC14.html

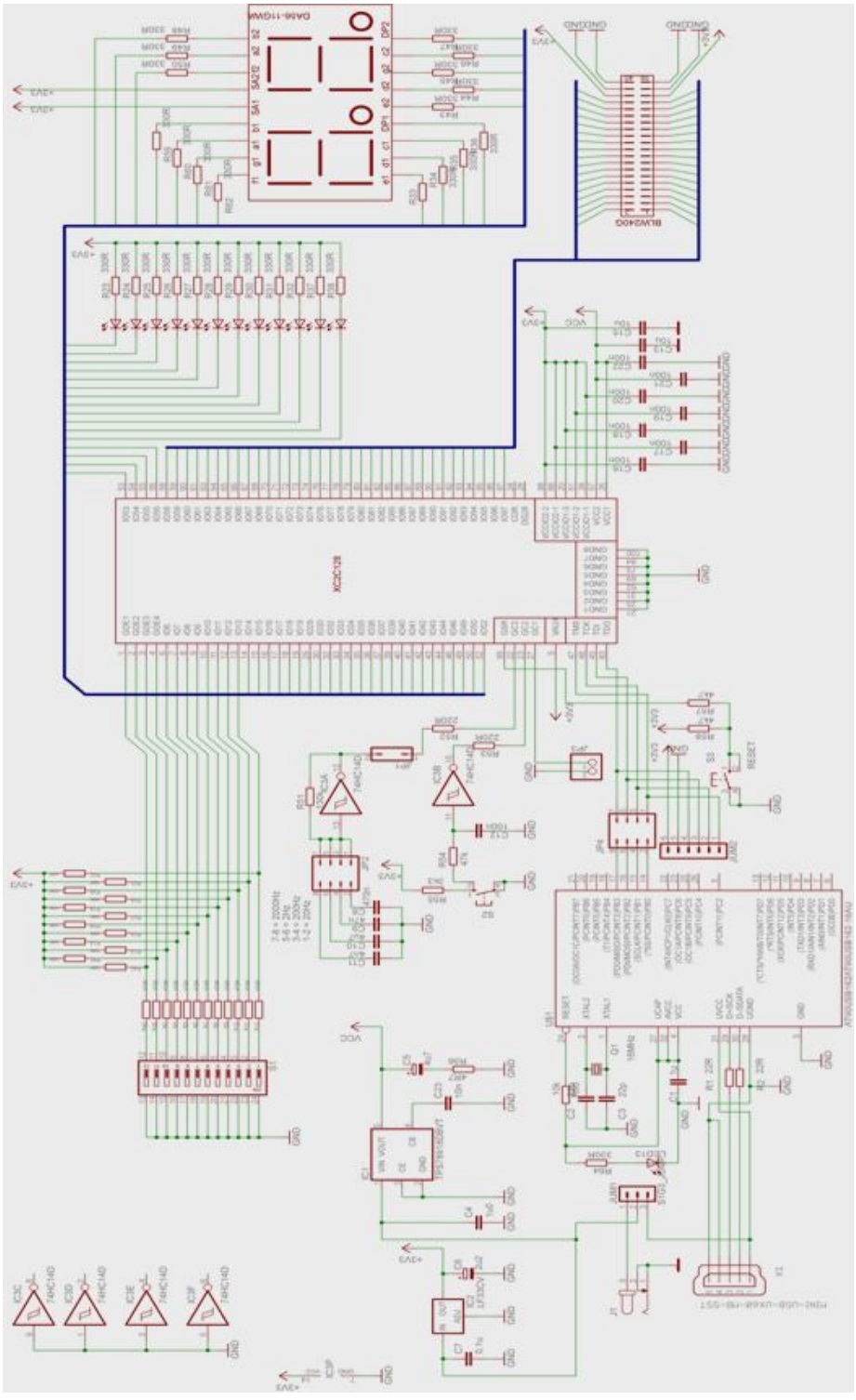
SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

U	veličina napětí
I	veličina proudu
R	veličina odporu
C	veličina kapacity
T	veličina periody
f	veličina frekvence
V	jednotka napětí
A	jednotka proudu
F	jednotka kapacity
Ω	jednotka odporu
s	jednotka času
π	konstanta 3,14
CPLD	Complex Programmable Logic Device
SPLD	Simple Programmable Logic Device
FPGA	Field Programmable Gate Array
USB	Universal Serial Bus
VID	Vendor ID‘s
RS-232	Sériový COM port
JTAG	Join Test Action Group
ISP	In System Programming
JEDEC	Joint Electron Devices Engeneering Council
RAM	Random Access Memory
EEPROM	Electrically Erasable Programmable Read Only Memory
PLA	Programmable Logic Array
PAL	Programmable Array Logic
GAL	Generic Array Logic
NRZI	Non Return to Zero Inverted
PC	Personal Computer
SMD	Surface Mount Device
LED	Light Emited Diode
TAP	Test Access Port
VHDL	Hardware Description Language
TDO	Test Data Out
TDI	Test Data In
TMS	Test Mode Select
TCK	Test ClocK
SVF	Serial Vector Format

SEZNAM PŘÍLOH

A NÁVRH PŘÍPRAVKU

A.1 Obvodové zapojení



B SEZNAM SOUČÁSTEK

Označení	Hodnota	Popis	Pouzdro
C1	1u	SMD kondenzátor	C0805
C2	22p	SMD kondenzátor	C0805
C3	22p	SMD kondenzátor	C0805
C4	2u2	SMD kondenzátor	C0805
C5	4u7	SMD kondenzátor	SMC_A
C6	2u2	SMD kondenzátor	SMC_A
C7	0,1u	SMD kondenzátor	C0805
C8	470n	SMD kondenzátor	C0805
C9	47n	SMD kondenzátor	C0805
C10	4u7	SMD kondenzátor	C0805
C11	4n7	SMD kondenzátor	C0805
C12	100n	SMD kondenzátor	C0805
C13	10u	SMD kondenzátor	C3216
C14	10u	SMD kondenzátor	C3216
C15	100n	SMD kondenzátor	C0805K
C16	100n	SMD kondenzátor	C0805K
C17	100n	SMD kondenzátor	C0805K
C18	100n	SMD kondenzátor	C0805K
C19	100n	SMD kondenzátor	C0805K
C20	100n	SMD kondenzátor	C0805K
C21	100n	SMD kondenzátor	C0805K
C22	100n	SMD kondenzátor	C0805K
IC3	74HC14D	Schmidtův obvod	SO14
IC2	LF33CV	Stabilizátor napětí 3,3V	DPACK
IC1	TPS78918DBVT	Stabilizátor napětí 1,8V	SOT25
J1		Konektor externího napájení	SPC4077
JP1		Konektor s propojkami	JP1
JP2		Konektor s propojkami	JP4Q
JP3		Konektor s propojkami	1X02
JP4		Konektor s propojkami	JP4Q
JUM1	S1G3_JUMP	Konektor pro volbu napájení	S1G3_JUM
JUM2	S1G6_JUMP	Konektor pro programování	S1G6_JUM
LED1	LED0805	LED dioda	CHIPLED_0805
LED2	LED0805	LED dioda	CHIPLED_0805
LED3	LED0805	LED dioda	CHIPLED_0805
LED4	LED0805	LED dioda	CHIPLED_0805
LED5	LED0805	LED dioda	CHIPLED_0805
LED6	LED0805	LED dioda	CHIPLED_0805
LED7	LED0805	LED dioda	CHIPLED_0805

Označení	Hodnota	Popis	Pouzdro
LED8	LED0805	LED dioda	CHIPLED_0805
LED9	LED0805	LED dioda	CHIPLED_0805
LED10	LED0805	LED dioda	CHIPLED_0805
LED11	LED0805	LED dioda	CHIPLED_0805
LED12	LED0805	LED dioda	CHIPLED_0805
LED13	LED0805	LED dioda	CHIPLED_0805
Q1	16MHz	Krystal	HC49UP
R1	22R	SMD rezistor	M0805
R2	22R	SMD rezistor	M0805
R3	470R	SMD rezistor	M0805
R4	470R	SMD rezistor	M0805
R5	470R	SMD rezistor	M0805
R6	470R	SMD rezistor	M0805
R7	470R	SMD rezistor	M0805
R8	470R	SMD rezistor	M0805
R9	470R	SMD rezistor	M0805
R10	470R	SMD rezistor	M0805
R11	470R	SMD rezistor	M0805
R12	470R	SMD rezistor	M0805
R13	10K	SMD rezistor	R0805
R14	10K	SMD rezistor	R0805
R15	10K	SMD rezistor	R0805
R16	10K	SMD rezistor	R0805
R17	10K	SMD rezistor	R0805
R18	10K	SMD rezistor	R0805
R19	10K	SMD rezistor	R0805
R20	10K	SMD rezistor	R0805
R21	10K	SMD rezistor	R0805
R22	10K	SMD rezistor	R0805
R23	330R	SMD rezistor	R0805
R24	330R	SMD rezistor	R0805
R25	330R	SMD rezistor	R0805
R26	330R	SMD rezistor	R0805
R27	330R	SMD rezistor	R0805
R28	330R	SMD rezistor	R0805
R29	330R	SMD rezistor	R0805
R30	330R	SMD rezistor	R0805
R31	330R	SMD rezistor	R0805
R32	330R	SMD rezistor	R0805
R33	330R	SMD rezistor	R0805
R34	330R	SMD rezistor	R0805
R35	330R	SMD rezistor	R0805
R36	330R	SMD rezistor	R0805

Označení	Hodnota	Popis	Pouzdro
R37	330R	SMD rezistor	R0805
R38	330R	SMD rezistor	R0805
R39	470R	SMD rezistor	R0805
R40	470R	SMD rezistor	R0805
R41	10K	SMD rezistor	R0805
R42	10K	SMD rezistor	R0805
R43	330R	SMD rezistor	R0805
R44	330R	SMD rezistor	R0805
R45	330R	SMD rezistor	R0805
R46	330R	SMD rezistor	R0805
R47	330R	SMD rezistor	R0805
R48	330R	SMD rezistor	R0805
R49	330R	SMD rezistor	R0805
R50	330R	SMD rezistor	R0805
R51	130k	SMD rezistor	R0805
R52	220R	SMD rezistor	R0805
R53	220R	SMD rezistor	R0805
R54	47k	SMD rezistor	R0805
R55	3k3	SMD rezistor	R0805
R56	100k	SMD rezistor	R0805
R57	4k7	SMD rezistor	R0805
R58	4k7	SMD rezistor	R0805
R59	330R	SMD rezistor	R0805
R60	330R	SMD rezistor	R0805
R61	330R	SMD rezistor	R0805
R62	330R	SMD rezistor	R0805
R63	10k	SMD rezistor	R0805
R64	330R	SMD rezistor	R0805
R65	330R	SMD rezistor	R0805
R66	330R	SMD rezistor	R0805
R67	330R	SMD rezistor	R0805
R68	330R	SMD rezistor	R0805
S1		Vstupní přepínače	DS-12
S2		Tlačítko hodinového signálu	B3F-10XX
S3	RESET	Tlačítko RESET	B3F-10XX
SV1		Rozšiřující konektor	BLW240G
U$1	AT90USB162AT90USB162	AT90USB162	VQFP32
U$2	DA56-11GWA	7 segmentový LED display	DA56-11GWA
X1	MINI-USB-UX60-MB-5ST	Mini-USB konektor	UX60-MB-5ST
XC2C128VQ1	XC2C128	Xilinx XC2C128	VQ100

C PŘÍKLAD SVF SOUBORU – MAZÁNÍ

```
// Created using Xilinx iMPACT Software [ISE - 11.1]
// Date: Wed Apr 21 17:58:35 2010

TRST OFF;
ENDIR IDLE;
ENDDR IDLE;
STATE RESET;
STATE IDLE;
FREQUENCY 1E6 HZ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
TIR 0 ;
HIR 0 ;
HDR 0 ;
TDR 0 ;
//Loading device with 'idcode' instruction.
SIR 8 TDI (01) SMASK (ff) ;
SDR 32 TDI (00000000) SMASK (ffffffff) TDO (f6d8f093) MASK (0fff8fff) ;
//Check for Read/Write Protect.
SIR 8 TDI (ff) TDO (01) MASK (03) ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
TIR 0 ;
HIR 0 ;
HDR 0 ;
TDR 0 ;
//Loading device with 'idcode' instruction.
SIR 8 TDI (01) ;
SDR 32 TDI (00000000) TDO (f6d8f093) ;
//Check for Read/Write Protect.
SIR 8 TDI (ff) TDO (01) MASK (03) ;
TIR 0 ;
HIR 0 ;
HDR 0 ;
TDR 0 ;
// Loading devices with 'enable' or 'bypass' instruction.
SIR 8 TDI (e8) ;
// Loading devices with 'erase' or 'bypass' instruction.
ENDIR IRPAUSE;
SIR 8 TDI (ed) SMASK (ff) ;
ENDIR IDLE;
STATE  IREXIT2 IRUPDATE DRSELECT DRCAPTURE DREXIT1 DRPAUSE;
RUNTEST DRPAUSE 20 TCK;
STATE  IDLE;
RUNTEST IDLE 100000 TCK;
STATE  DRPAUSE;
RUNTEST DRPAUSE 5000 TCK;
```

```
RUNTEST IDLE 1 TCK;
// Loading devices with 'init' or 'bypass' instruction.
ENDIR IRPAUSE;
SIR 8 TDI (f0) SMASK (ff) ;
STATE  IDLE;
RUNTEST IDLE 20 TCK;
// Loading devices with 'init' or 'bypass' instruction.
ENDIR IRPAUSE;
SIR 8 TDI (f0) SMASK (ff) ;
STATE  IREXIT2 IRUPDATE DRSELECT DRCAPTURE DREXIT1 DRUPDATE IDLE;
RUNTEST 800 TCK;
ENDIR IDLE;
// Loading devices with 'conld' or 'bypass' instruction.
SIR 8 TDI (c0) ;
RUNTEST 100 TCK;
// Loading devices with 'conld' or 'bypass' instruction.
SIR 8 TDI (c0) ;
RUNTEST 100 TCK;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 8 TDI (ff) ;
SDR 1 TDI (00) SMASK (01) ;
```

D PŘÍKLADY ZDROJOVÉHO TEXTU V JAZYKU VHDL

D.1 Stopky

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity stopky is
      Port  (Sa: out std_logic_vector (7 downto 0);
                  Sb: out std_logic_vector (7 downto 0);
                  clk: in std_logic;
                  leds: out std_logic_vector (11 downto 0);
                  reset: in std_logic;
                  start: in std_logic
                 );
end stopky;
architecture Behavioral of stopky is
      subtype cint is integer range 0 to 11;
      subtype cntint is integer range 0 to 25;
      signal poc : cntint :=0;
      signal iclk : std_logic := '1';
      signal pocb : cint := 1;
      signal poca : cint := 0;
      signal Saa : std_logic_vector(7 downto 0) :="00000011";
      signal Sbb : std_logic_vector(7 downto 0) :="00000011";
      signal ileds : std_logic_vector(11 downto 0) :="111111111111";
      signal svetylko : std_logic_vector(1 downto 0) :="10";
      signal stav : std_logic :='0';
      signal res : std_logic :='0';
      signal minn : std_logic :='0';
begin
      Sa <= Saa;
      Sb <= Sbb;
      leds <= ileds;

      process (clk)     --dělič kmitočtu
      begin
            if clk'event and clk='1' then
                  if poc=cntint'high then
                        poc <= 0;
                        iclk <= not iclk;
                  else
                        poc <= poc +1;
                  end if;
            end if;
      end process;
```

```
	process (start)   --ovládání tlačítka start/stop
		begin
		if start'event and start='1' then
			stav <= not stav;
		end if;
	end process;
	process (reset)   --ovládání nulovacího tlačítka
	begin
	if reset'event then
		res <= not res;
	end if;
	end process;

	process (iclk)    --vlastní program
	begin
		if iclk='1' and iclk'event and stav='1' then
				pocb <= pocb + 1;
				if minn='1' then
					minn <= '0';
					ileds <= svetylko(0) & ileds(11 downto 1);
				end if;
			case poca is      --řízení 1.segmentu displeje
				when 0 => Saa <= "00000011";
				when 1 => Saa <= "10011111";
				when 2 => Saa <= "00100101";
				when 3 => Saa <= "00001101";
				when 4 => Saa <= "10011001";
				when others => Saa <= "01001001";
			end case;
				if pocb=9 then
					pocb <= 0;
					poca <= poca + 1;
				end if;
				if pocb=9 and poca=5 then
					poca <= 0;
					minn <= not minn;
				end if;
			case pocb is      --řízení 2.segmentu displeje
				when 0 => Sbb <= "00000011";
				when 1 => Sbb <= "10011111";
				when 2 => Sbb <= "00100101";
				when 3 => Sbb <= "00001101";
				when 4 => Sbb <= "10011001";
				when 5 => Sbb <= "01001001";
				when 6 => Sbb <= "01000001";
				when 7 => Sbb <= "00011111";
				when 8 => Sbb <= "00000001";
				when others => Sbb <= "00001001";
			end case;
		end if;
		if res='1' then         --akce po přijetí nulovacího signálu
				poca <= 0;
				pocb <= 1;
				Saa <= "00000011";
				Sbb <= "00000011";
				ileds <= "111111111111";
		end if;
	end process;
end Behavioral;
```

D.2 Kódový zámek

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity kodovy_zamek is
  Port  (clk : in std_logic;
              set : in std_logic;
              pre : in std_logic_vector(11 downto 0);
              leds : out std_logic_vector(11 downto 0);
              SegA : out std_logic_vector(7 downto 0);
              SegB : out std_logic_vector(7 downto 0)
             );
end kodovy_zamek;

architecture Behavioral of kodovy_zamek is
  signal kod : std_logic_vector (11 downto 0);
  signal SA : std_logic_vector (7 downto 0) := "11111111";
  signal SB : std_logic_vector (7 downto 0) := "11111111";
  signal iled: std_logic_vector (11 downto 0) := "111111111111";

begin
  SegA <= SA;
  SegB <= SB;
  leds <= iled;

  process(clk)      --akce pro tlačítko2
        begin
              if clk='0' and clk'event then
                    if kod=pre then
                          iled <= "000000000000";
                          SA <= "00000011";
                          SB <= "00110001";
                    else
                          iled <= "111111111111";
                          SA <= "01100001";
                          SB <= "11110101";
                    end if;
              end if;
  end process;

  process(set)      --nastavení kodu
        begin
              if set='0' and set'event then
                    kod <= pre;
              end if;
        end process;
end Behavioral;
```

D.3 Běžící světlo

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity blikac is
    Port ( clk : in  STD_LOGIC;
           leds : out std_logic_vector(11 downto 0);
                dis : out std_logic_vector(15 downto 0)
  );
end blikac;

architecture beh2 of blikac is

 --definice subtypu integer, max. hodnota urcuje maximum citace
 --a zaroven potrebny pocet bitu
 subtype cntint is integer range 0 to 60;
 --signal udavajici stav citace
 signal  poc   : cntint := 0;
 signal iclk  : std_logic := '1';
 signal ileds : std_logic_vector(11 downto 0) := "111111111110";

begin
  leds <= ileds;
  dis <= "0000000000000000";

  -- delicka vstupniho kmitoctu pomoci citace
  process (clk)
  begin
    if clk='1' and clk'event then
       if poc = cntint'high then
          poc <= 0;
          iclk <= not iclk;
       else
          poc <= poc + 1;
       end if;
    end if;
  end process;

  -- vlastni blikac, pri kazde nastupne hrane iclk se
  -- provede posun obsahu registru ileds doprava
  process (iclk)
  begin
    if iclk = '1' and iclk'event then
          ileds <= ileds(0) & ileds(11 downto 1);
    end if;
  end process;


end beh2;
```

D.4 Klopný obvod typu J-K

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity KlopnyObvodJK is
port          (J: in std_logic;
              K, CLK, C : in std_logic;
              Disp_out : out std_logic_vector(7 downto 0);
              LEDs : out std_logic_vector(1 downto 0)
              );
end KlopnyObvodJK;

architecture Behavioral of KlopnyObvodJK is
  signal Q : std_logic;
  signal Disp : std_logic_vector(7 downto 0) := "11111111";
  signal JK : std_logic_vector (1 downto 0);
begin
  process (C)
  begin
        if C'event and C = '1' then
              JK <= (not J & not K);
              case Q is
                    when '0' => Disp <= "00000011";
                    when '1' => Disp <= "10011111";
                    when others => Disp <= "11111111";
              end case;
        end if;
  end process;

  process (CLK)
  begin
        if CLK'event and CLK = '0' then
              case JK is
                    when "01" => Q <= '0';
                    when "10" => Q <= '1';
                    when "11" => Q <= not (Q);
                    when "00" => Q <= Q;
                    when others=> Q <= 'X';
              end case;
        end if;
  end process;

  Disp_out <= Disp;
  LEDs <= not (JK);

end Behavioral;
```

D.5 Binární čítač s asynchronním nulováním a s povolením čítat

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity citac is
port          (CLK, RES, POV : in std_logic;
              LEDs : out std_logic_vector(11 downto 0));
end citac;

architecture Behavioral of citac is

signal tmp: std_logic_vector(LEDs'range);
subtype cntint is integer range 0 to 30;
signal iclk: std_logic;
signal poc: cntint := 0;

begin
  process (CLK)     --dělič kmitočtu
  begin
        if CLK'event and CLK='1' then
              if poc=cntint'high then
                    poc <= 0;
                    iclk <= not iclk;
              else
                    poc <= poc +1;
              end if;
        end if;
  end process;

  process (iclk, RES)
  begin
        if (RES='0') then
              tmp <= "000000000000";
        elsif (iclk'event and iclk='1') then
              if (POV='1') then
              tmp <= tmp + 1;
              end if;
        end if;
  end process;

  LEDs <= not (tmp);
end Behavioral;
```