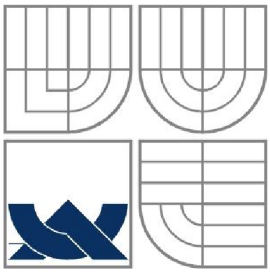
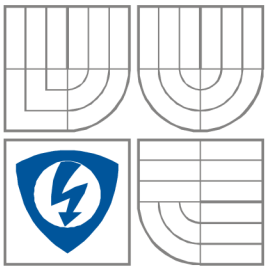


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

LABORATORNÍ PŘÍPRAVEK PRO VÝVOJ APLIKACÍ OBVODŮ CPLD FIRMY ALTERA

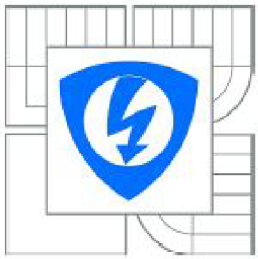
LABORATORY KIT FOR DEVELOPMENT APPLICATION OF ALTERA CPLD DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S PROJECT

AUTOR PRÁCE — Bc. Petr Gajdošík
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Jaromír Kolouch, CSc.
SUPERVISOR

BRNO, 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav radioelektroniky

Diplomová práce

magisterský navazující studijní obor
Elektronika a sdělovací technika

Student:	Bc. Petr Gajdošík	***ID:***	109650
Ročník:	2	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Laboratorní přípravek pro vývoj aplikací obvodů CPLD firmy Altera

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte vlastnosti a způsoby programování obvodů řady MAX-II firmy Altera. Vypracujte návrh laboratorního přípravku pro práci s těmito obvody a pro jejich demonstraci v laboratorních cvičeních, splňující tyto požadavky:
a) vstup údajů z přepínačů na desce, b) indikaci stavu výstupů signálkami LED a multiplexním displejem, c) možnost připojení dalších periferií pomocí konektorů.
Přípravek realizujte a ověřte jeho funkci včetně ověření použitelnosti všech dostupných programovacích prostředků (kabely Altera, Presto).
Zpracujte vzorové příklady k demonstraci funkce přípravku včetně simulace funkční a post-fit, a návod pro práci s přípravkem. Součástí příkladů má být šablona pro vkládání testovaných konstrukcí do přípravku s textovou i schematickou vrcholovou jednotkou. Prozkoumejte také možnosti stanovení časových parametrů konstrukcí statickou časovou analýzou.

DOPORUČENÁ LITERATURA:

[1] MAX II Device Handbook. Dostupné na www:
http://www.altera.com/literature/hb/max2/max2_mii5v1.pdf)
[2] KOLOUCH, J. Programovatelné logické obvody a návrh jejich aplikací v jazyku VHDL. Skriptum. Brno: FEKT VUT v Brně, 2006. ISBN 80-214-3271-3.

Termín zadání:	6.2.2012	***Termín odevzdání:***	18.5.2012

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

prof. Dr. Ing. Zbyněk Raida
Předseda oborové rady

ABSTRAKT

V diplomové práci se zaměřuji na návrh schematu laboratorního přípravku a prostudování způsobů programování obvodů CPLD firmy Altera. Přípravek slouží pro vývoj a demonstraci aplikací v obvodech CPLD firmy Altera. Přípravek je navržen pro programování kabely Altera a Presto (výrobce ASIX). Vstupní signály jsou realizovány soustavou přepínačů a tlačítek na desce. Stavy výstupů jsou zobrazovány na LED diodách, případně na připojeném multiplexním displeji. Uživatel má možnost připojit externí zařízení, přes externí vstupy.
Práce je dále zaměřena na návrh desky plošných spojů laboratorního přípravku, následné výrobě, oživení přípravku a ověření kompatibility programátorů ALTERA a PRESTO. Závěr práce je zaměřen na práci s návrhovým prostředím QUARTUS II. Zejména se jedná o návod na práci se šablonami a simulací VHDL konstrukcí.

KLÍČOVÁ SLOVA

Laboratorní přípravek, CPLD, Altera, JTAG, VHDL, QUARTUS II.

ABSTRACT

In this thesis I aim at a design of the laboratory kit and study ways how to programme CPLD devices made by Altera company. The product is used for development and demonstration of applications in CPLD devices made by Altera company. The kit is designed for Altera programming cables and Presto (made by ASIX). Input signals are implemented by a set of switches and buttons on the board. Output states are displayed by LED diods, possibly connected to multiplex the display. The user can connect to external devices via external inputs.

Thesis is also aimed at the design PCB of the laboratory kit, subsequent production, recovery and verification of compatibility ALTERA and PRESTO programmers. End of the thesis aims on working with the Quartus II design environment. In particular, it is a guide to working with templates and simulation of VHDL designs.

KEYWORDS

Laboratory kit, CPLD, Altera, JTAG, VHDL, QUARTUS II.

GAJDOŠÍK, P. *Laboratorní přípravek pro vývoj aplikací obvodů CPLD firmy Altera*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav radioelektroniky, 2012. 31 s., 11 s. příloh. Diplomová práce. Vedoucí práce: doc. Ing. Jaromír Kolouch, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Laboratorní přípravek pro vývoj aplikací obvodů CPLD firmy Altera jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce doc. Ing. Jaromír Kolouch, CSc. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne ....

(podpis autora)

OBSAH

Seznam obrázků viii

Úvod 1

1 Programovatelné logické obvody 2

1.1 SPLD 3

1.2 CPLD 4

1.3 FPGA 4

2 Použití obvodu ALTERA MAX II 6

2.1 Vlastnosti rodiny MAX II 6

2.2 Vlastnosti zvoleného obvodu EPM240 6

2.3 Konfigurace obvodů MAX II 7

2.3.1 Programátor PRESTO 7

3 Návrh schematu přípravku 9

3.1 Napájení 9

3.2 Programování obvodu CPLD 10

3.3 Oscilátory 11

3.4 Vstupní přepínače 12

3.5 Vstupní tlačítka 13

3.6 Výstupní LED diody 14

3.7 Multiplexně řízený 7segmentový displej 14

3.8 Konektory pro externí I/O zařízení 15

4 Návrh desky plošných spojů 16

5 Návod na práci s přípravkem 18

5.1 Postupy pro práci se šablonou v QUARTUS II 18

5.1.1 Přejmenování projektu 18

5.1.2 Seznámení s vrcholovou jednotkou 19

5.1.3 Vytvoření vlastního konstrukčního bloku 19

5.1.4 Přiřazení vývodů 22

5.1.5 Programování přípravku 23

5.2 Postupy pro simulace a analýzy VHDL konstrukcí 25

5.2.1 Funkční simulace 25
5.2.2 POST-FIT Simulace 27
5.2.3 Statická časová analýza 27

6 Vzorové VHDL konstrukce **29**
6.1 Hodiny 29
6.2 Bitové operace 30
6.3 Převod BIN to HEX 30
6.4 Čítač 30

7 Závěr **31**

Literatura **32**

Seznam symbolů, veličin a zkratek **33**

Seznam příloh **34**

SEZNAM OBRÁZKŮ

Obrázek 1.1 Struktura makrobuňky obvodů SPLD typu PAL (převzato z [1]) 2
Obrázek 1.2 Základní princip obvodů PAL (převzato z [2]) 3
Obrázek 1.3 Struktura obvodů CPLD (převzato z [2]) 4
Obrázek 1.4 Struktura FPGA (převzato z [2]) 5
Obrázek 2.1 Zapojení programátoru PRESTO (převzato z [4]) 8
Obrázek 3.1 Blokové schema přípravku 9
Obrázek 3.2 Zapojení stabilizátorů 10
Obrázek 3.3 Zapojení programování 10
Obrázek 3.4 Katalogové zapojení oscilátoru (převzato z [5]) 11
Obrázek 3.5 Zapojení oscilátorů 11
Obrázek 3.6 Zapojení přepínačů 12
Obrázek 3.7 Zapojení tlačítek 13
Obrázek 3.8 Zapojení LED diod 14
Obrázek 3.9 Zapojení displeje 15
Obrázek 4.1 TOP vrstva navržené desky 16
Obrázek 4.2 BOTTOM vrstva navržené desky 16
Obrázek 4.3 TOP vrstva osazené desky 17
Obrázek 4.4 BOTTOM vrstva osazené desky 17
Obrázek 5.1 Schematická vrcholová jednotka šablony 18
Obrázek 5.2 Připojení více zdrojů do sběrnice 21
Obrázek 5.3 Přiřazení vývodů přes Pin Planner 22
Obrázek 5.4 Přiřazení vývodů v Assignment Editoru 23
Obrázek 5.5 Generování SVF souboru 24
Obrázek 5.6 JTAG SVF Player 24
Obrázek 5.7 Průběhy simulace v ModelSim 27
Obrázek 5.8 TimeQuest - Zobrazení výsledků po spuštění Report Data 28
Obrázek 5.9 TimeQuest - nastavení cest pro analýzu 28
Obrázek 5.10 Quartus II - zobrazení výsledků analýzy 29

ÚVOD

Úkolem diplomové práce je prostudovat možnosti programování obvodů CPLD firmy Altera a navrhnout schema laboratorního přípravku s obvodem firmy Altera. Obvody CPLD neboli Complex Programmable Logic Devices, jsou složité logické obvody, které se využívají hlavně pro jejich rychlost a flexibilitu. Můžeme je nalézt v různých aplikacích, kde není možné využít mikroprocesoru a kde se klade důraz na rychlost jednoduchých operací jako je multiplexování, jednoduché aritmetické operace a další operace v kterých mikrokontrolery za těmito obvody zaostávají.

Laboratorní přípravek se tedy bude skládat z obvodu CPLD firmy Altera, který bude programován přes rozhraní JTAG. Hlavní zaměření bude na možnost programování přes externí programátor Presto firmy Asix, který je momentálně velmi rozšířeným prostředkem. Přípravek bude napájen zdrojem o napětí 5 V. Jako vstupní periferie zde budou použity sestava přepínačů a také několik tlačítek, které budou na desce ošetřeny proti zákmitům při stisku. Toto ošetření bude možné i odpojit, kvůli možnosti vytvoření programového ošetření těchto zákmitů. LED diody a 7segmentový multiplexní displej budou použity pro zobrazení výstupů. Nebude zde chybět ani možnost připojení externích modulů přes vyvedené konektory na desce.

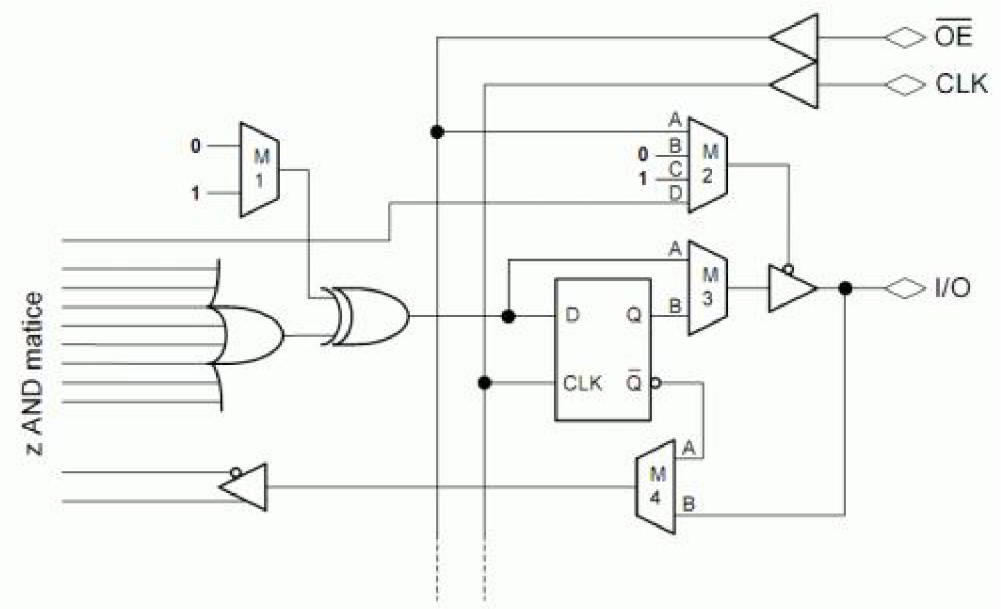
Dalším úkolem je navrhnout desku plošných spojů s následným osazením součástkami a oživením. Celá práce bude vrcholit několika příklady VHDL konstrukcí pro ověření funkčnosti desky a manuálem pro práci s přípravkem a s vytvořenou šablonou v návrhovém prostředí QUARTUS II.

1 PROGRAMOVATELNÉ LOGICKÉ OBVODY

V této kapitole jsou použity informace z pramenu [2]. Číslicové programovatelné obvody se všeobecně označují zkratkou PLD. Je to zkratka slovního spojení Programmable Logic Devices. Jsou to obvody, které jsou složeny z mnoha logických obvodů. Nejmenším stavebním prvkem je dvouvstupé hradlo NAND. Tyto obvody samozřejmě neobsahují jen je. Programovatelné jsou proto, že uživatel je schopný si tyto obvody nakonfigurovat podle svých potřeb.

Programovatelné logické obvody se dělí na tři základní typy. Jsou to SPLD (Simple Programmable Logic Devices), CPLD (Complex Programmable Logic Devices) a obvody FPGA (Field Programmable Gate Arrays).

Velikost PLD obvodů se uvádí většinou ve dvou jednotkách. Je to počet hradel, avšak jde o zavádějící pojem, protože řada hradel je použita na pomocné a systémové funkce. Z toho důvodu je vhodnější jednotkou velikosti PLD obvodů užívaný počet makrobuněk. Makrobuňka je vždy jeden paměťový člen (klopný obvod typu D), který je doplněný o řadu pomocných hradel. Podoba makrobuňky se u každého typu PLD obvodů liší. Typická struktura makrobuňky z nejjednodušších obvodů PAL je zobrazena na Obrázku 1.1.



Obrázek 1.1 Struktura makrobuňky obvodů SPLD typu PAL (převzato z [1])

Obvody CPLD mají makrobuňky složitější a obvody FPGA mají spíše logické bloky sdružující i řadu dalších funkcí.

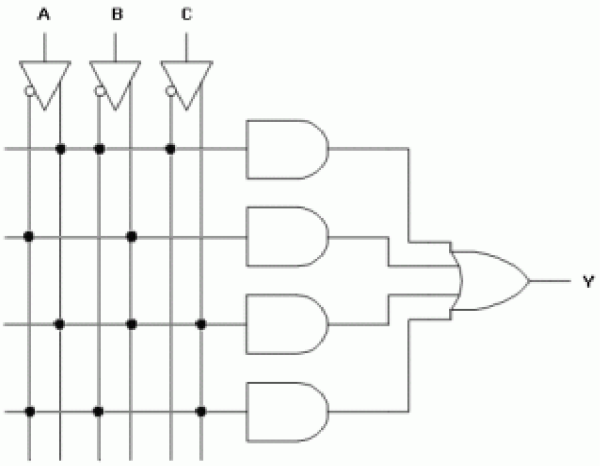
Výhody těchto obvodů jsou v tom, že jsou velmi rychlé. Jsou schopné pracovat na frekvencích stovek MHz. Zvláště kvůli jejich rychlosti jsou vhodné na místech, kde je potřeba rychlá reakce, například síťové přepínače (switch), zpracování rychlých signálů.

Další výhodou je tzv. pinová rovnocennost. Až na pár speciálních pinů můžeme kterýkoliv pin ustanovit na námi zvolenou funkci. To usnadňuje návrh desky plošného spoje.

Nevýhodou je častá nutnost napájení těchto obvodů více standardy napájecího napětí. Ale tato vlastnost není pravidlem, jelikož už mnoho obvodů má v sobě implementovaná tzv. multivoltage cores, což vede k možnosti napájení jádra a periferií stejnou velikostí napětí. Další nevýhodou těchto obvodů je jejich častá absence analogových obvodů (například DAC), ale to poslední dobou není problém, jelikož dosti často se využívá kombinace PLD obvodů s mikroprocesory.

1.1 SPLD

V této kapitole jsou použity informace z pramenu [2]. Jsou to jednoduché programovatelné obvody PAL, GAL, PLA a podobné. Jedná se o zastaralé obvody, které v sobě seskupovaly jen malé množství makrobuněk, což umožňovalo naprogramovat jen jednoduché logické funkce. Obvody mají předem definované vstupní a výstupní piny a uživatel jen konfiguruje vlastní logickou funkci pro každý výstup zvlášť, jelikož na každém výstupu je umístěna jedna makrobuňka. Základní princip obvodů řady PAL je na Obrázku 1.2.



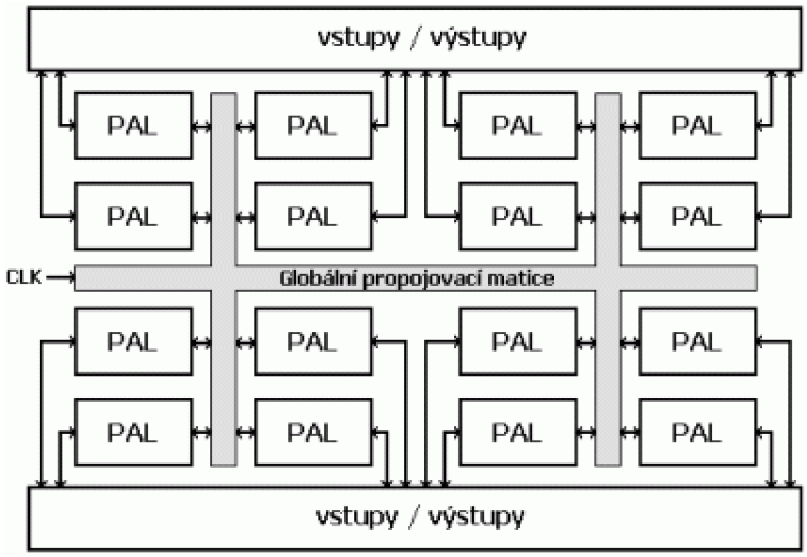
Obrázek 1.2 Základní princip obvodů PAL (převzato z [2])

Jde o zobrazení základního principu obvodů PAL. Vlevo nahoře jsou vstupy obvodu. Jejich invertovaná a neinvertovaná hodnota je přivedena na součinovou matici AND. Každý vodorovný spoj protínající tuto matici reprezentují vždy jedno součinové hradlo. Tento spoj v sobě sdružuje více vstupů hradla ke kterému vedou a tím je možné na každé takové hradlo připojit více vstupních signálů. Výstupy těchto hradel se spojují v součtovém hradle OR a přes případnou další logiku procházejí až na výstupní pin. Zároveň se tyto signály často vracejí i zpět do matice, takže je možné jejich stav využít i u ostatních částí obvodu.

Na každém výstupním pinu je navíc umístěna jedna (jednobitová) makrobuňka. Jejich největší zástupce je obvod PAL22V10, který obsahuje 10 výstupů a tedy 10 makrobuněk. Což mimo jiné znamená, že z takového obvodu bylo možné vytvořit maximálně desetibitový binární čítač, který již však nemohl vykonávat nic dalšího.

1.2 CPLD

V této kapitole jsou použity informace z pramenu [2]. Obvody SPLD jsou v podstatě základem pro obvody CPLD (neboli složité programovatelné logické obvody). Jejich typická struktura je zobrazena na Obrázku 1.3.



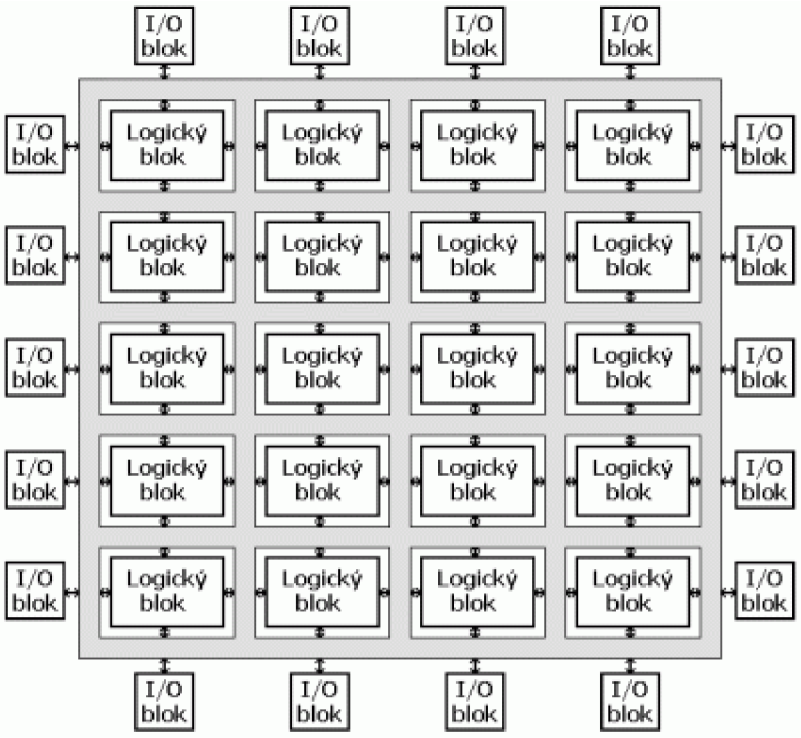
Obrázek 1.3 Struktura obvodů CPLD (převzato z [2])

Jak lze vidět ze struktury obvodu CPLD, jde o matici mnoha obvodů SPLD, které jsou spojeny globální propojovací maticí. To umožňuje vytvořit mnohem složitější konstrukci. Momentálně lze od společnosti Altera získat obvody o velikosti od 192 – 1700 makrobuněk, což znamená větší volnost, nežli u obvodů SPLD. Většina obvodů CPLD obsahuje i interní paměť EEPROM pro trvalé uložení konfigurace obvodu,což je výhodou ve chvíli, přerušení napájení obvodu, jednak nemusí být obvod nově naprogramován a nemusí se využívat externí paměti Flash pro obnovu konfigurace.

1.3 FPGA

V této kapitole jsou použity informace z pramenu [2]. Obvody FPGA jsou nejsložitějšími, zároveň však také nejobecnějšími PLD obvody. Místo klasických makrobuněk jsou složeny z tzv. logických bloků a obsahují až miliony ekvivalentních hradel (typické dvouvstupé hradlo NAND).

Na Obrázku 1.4 je typická struktura FPGA.



Obrázek 1.4 Struktura FPGA (převzato z [2])

Tyto obvody ve velké míře obsahují pro svou konfiguraci pouze integrovanou paměť RAM. Z toho důvodu je obvod nutné po každém zapnutí znovu nakonfigurovat. To lze zařídit externí pamětí eeprom, z které si dokážou obvody FPGA po zapnutí samy načíst svou konfiguraci.

2 POUŽITÍ OBVODU ALTERA MAX II

V této kapitole jsou použity informace z pramenu [3]. Pro návrh laboratorního přípravku byl zvolen obvod CPLD řady MAX II od společnosti ALTERA. Jedná se o EPM240T100I5N. Je to nejmenší velikost obvodu této řady a obsahuje 192 makrobuněk, což stačí pro základní aplikace v rozsahu zadaných funkcí laboratorního přípravku.

2.1 Vlastnosti rodiny MAX II

- nízká cena, nízká spotřeba CPLD
- obsahuje nevolatilní architekturu (není nutno po každém spuštění konfigurovat)
- odběr v pohotovostním režimu menší než 25 uA
- Možnost využití 4 globálních hodinových vstupů
- UFM (User Flash Memory) blok o velikosti až 8 Kbits nevolatilní paměti
- Jádro s podporou více napěťových standardů 3.3 V/2.5 V nebo 1.8 V
- Vstupně výstupní rozhraní podporující napěťové standarty 3.3 V, 2.5 V, 1.8 V, 1.5 V
- Architektura obvodu umožňuje programovatelnou rychlost přeběhu, a programovatelné pull-up a pull-down rezistory
- Možnost nastavení na každý pin průchod Schmittovým obvodem pro odrušení šumu
- Vstupy a výstupy obvodu jsou plně kompatibilní s PCI SIG (Peripheral Component Interconnect Special Interest Group) PCI Local Bus Specification, Revize 2.2 pro napěťový standart 3.3-V pracující na frekvenci 66 MHz
- Podpora připojování za běhu
- Zabudovaný JTAG (Joint Test Action Group), BST(boundary-scan test) systém vyhovující standartu IEEE Std. 1149.1-1990
- Systém ISP vyhovující standartu IEEE Std. 1532

2.2 Vlastnosti zvoleného obvodu EPM240

- Obsahuje 240 logických hradel
- Obsahuje 192makrobuměk
- Až 80 vstupně výstupních uživatelských pinů
- Použité pouzdro 100-pinové TQFP
- rozměry obvodu 16 mm x 16 mm, rozestup pinů 0,5 mm
- Napájení jádra napětím 3,3 V nebo 2,5 V
- Napájení vstupně-výstupních bloků standardy 1,5 V, 1,8 V, 2,5 V , 3,3 V

2.3 Konfigurace obvodů MAX II

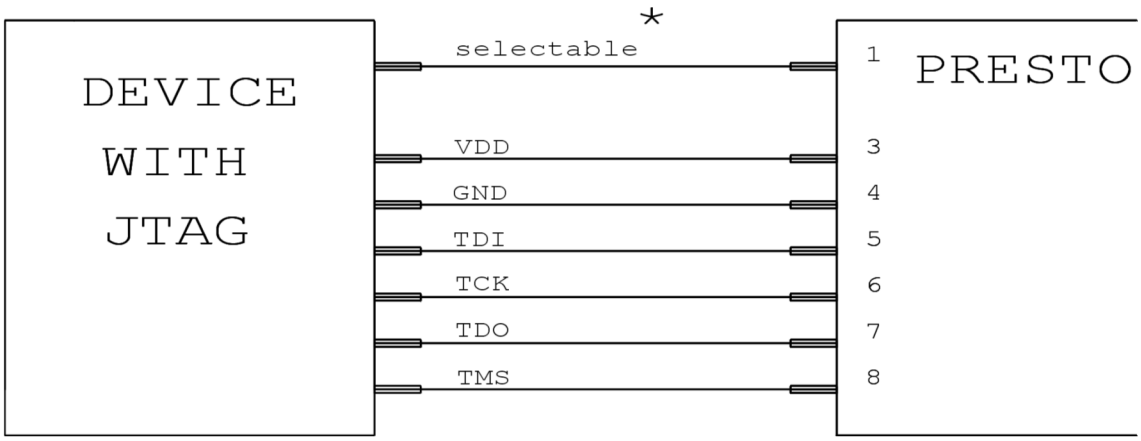
V této kapitole jsou použity informace z pramenu [3]. Konfigurace obvodů řady MAX II se provádí přes rozhraní JTAG (IEEE Std. 1149.1). Toto rozhraní se skládá ze čtyř signálů. Jsou to TDI (Test Data Input) sloužící pro vstupní data, TMS (Test Mode Select) sloužící pro volbu stavů programování, TCK (Test Clock), který slouží jako hodinový signál a TDO (Test Data Output) zařizuje odesílání dat zpět popřípadě do dalšího obvodu v řetězci. Přes rozhraní JTAG je možné programovat více obvodů zároveň. Využívá se k tomu zapojení obvodů za sebe do řetězce. Signály TMS, TCK jsou zapojeny do každého obvodu paralelně a signály TDI a TDO jsou zapojeny do série.

V této práci se budu zaměřovat na možnost programovaní za užití externích programátorů. Přípravek by mělo být možno naprogramovat přes programátor od společnosti Altera. Mezi ně patří Altera ByteblasterMV, MasterBlaster, ByteBlaster II a USB-Blaster cables. Tyto programátory by měli být navrženy pro tyto obvody CPLD, takže neočekávám problémy jejich použití a funkce. A jako druhá možnost konfigurace bude přes programátor PRESTO od společnosti ASIX. Tento programátor patří momentálně k dosti používanému zařízením, a to hlavně z důvodu široké možnosti programování nejrůznějších programovatelných obvodů.

2.3.1 Programátor PRESTO

V této kapitole jsou použity informace z pramenu [4]. Pro programování součástek přes rozhraní JTAG tímto programátorem slouží program s názvem JTAG SVF PLAYER. Je s ním možné programovat a testovat součástky, pro které existuje software poskytující data ve formátu SVF nebo XSVF. To jsou například obvody CPLD společnosti Altera, Xilinx, a další. Formát XSVF se výhradně využívá pro obvody společnosti Xilinx. Nás bude zajímat formát SVF.

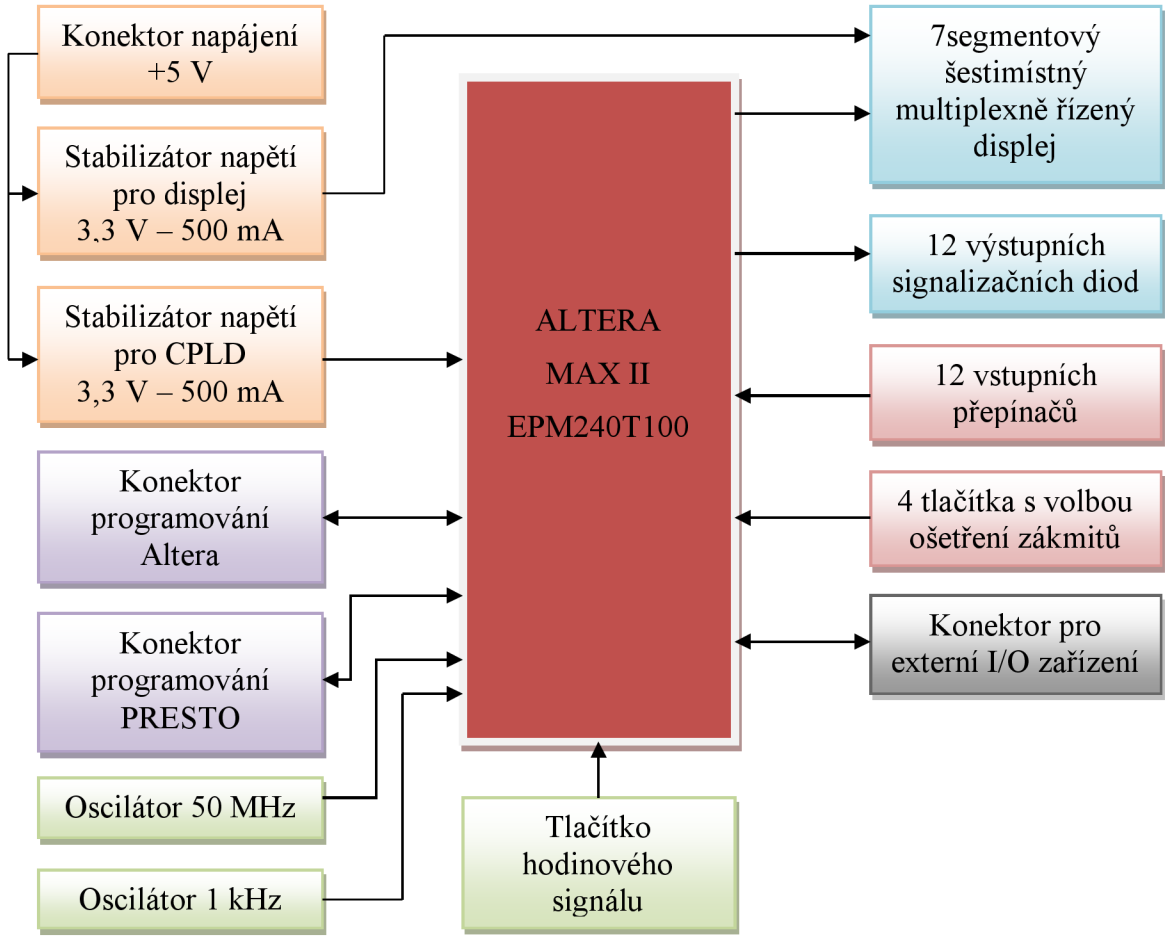
U programátoru PRESTO na nás čeká jediné omezení. Pro programování přes tento programátor je potřeba vygenerovat SVF soubor a následně ho nahrát do obvodu přes další software, který dodává ASIX. Přesněji přes JTAG Player. Dřívější problémy s programováním pomocí PRESTA už byly odstraněny.



Obrázek 2.1 Zapojení programátoru PRESTO (převzato z [4])

* Signál selectable se může chovat jako SCK nebo TRST popsaný v SVF souboru nebo může byt definovaný uživatelem (log.1, log.0, třetí stav, různý během programování a po něm)

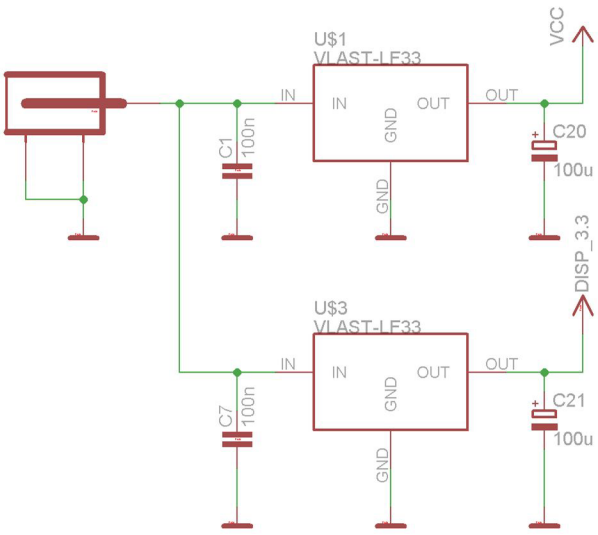
3 NÁVRH SCHEMATU PŘÍPRAVKU



Obrázek 3.1 Blokové schema přípravku

3.1 Napájení

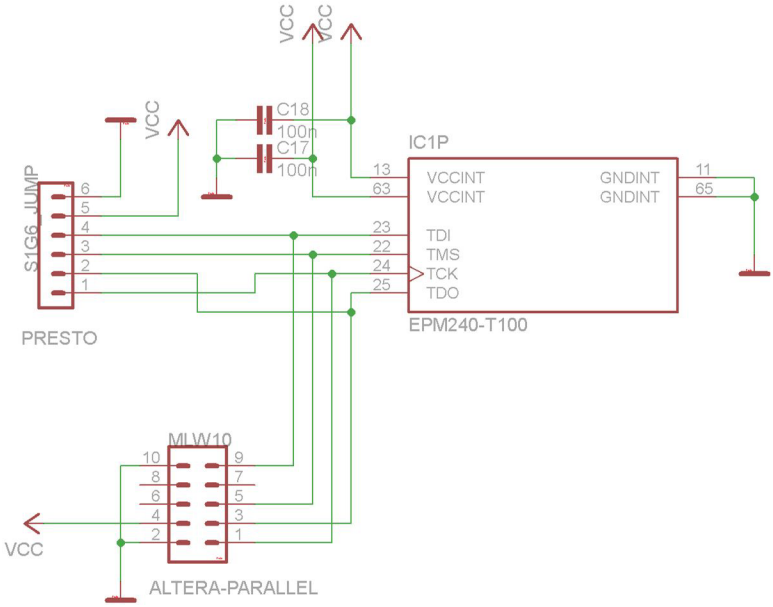
Napájení celého přípravku je navrženo pro adaptér +5 V. Toto napětí je dále stabilizováno dvěma stabilizátory LF33, které vstupní napětí 5 V stabilizují na 3,3 V. Tento stabilizátor dovoluje maximální proudový odběr 500 mA. Jeden stabilizátor je užit pro napájení 7segmentového displeje. Druhý stabilizátor je určený pro napájení obvodu CPLD a všech periferií. Zapojení stabilizátoru je navrženo podle doporučeného zapojení udaného v dokumentaci stabilizátoru. Každý napájecí vstup CPLD je osazen kondenzátorem o kapacitě 100 nF, aby se získalo stabilnější napětí, jelikož tyto obvody jsou na to citlivé. Zapojení stabilizátorů je zobrazeno na obrázku 3.2.



Obrázek 3.2 Zapojení stabilizátorů

3.2 Programování obvodu CPLD

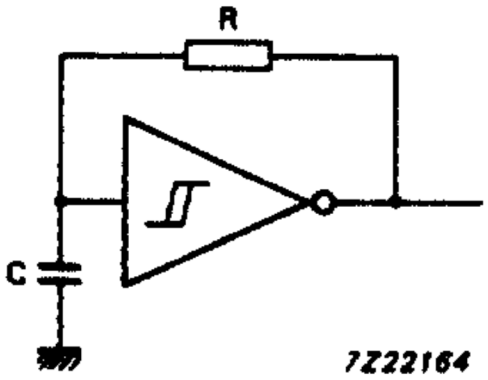
Pro programování jsou z CPLD vyvedeny dva konektory, které slouží k připojení externích programátorů. Konektor s názvem PRESTO je pro připojení programátoru PRESTO společnosti ASIX. Konektor nazvaný ALTERA-PARALLEL je pro připojení programátoru společnosti Altera. Piny jsou rozmístěny podle dokumentace k programátorům. Zapojení lze vidět na obrázku 3.3.



Obrázek 3.3 Zapojení programování

3.3 Oscilátory

Pro vstup hodinového signálu je navržen oscilátor o frekvenci 50 MHz, který je zamýšlen jako hlavní zdroj kmitočtu. Zapojení je převzato z doporučeného zapojení od výrobce. Poté je zde navržen oscilátor o frekvenci 1 kHz, který je vytvořen z obvodu 74HC14D, což je invertovaný Schmittův obvod. Tento oscilátor je zde použit pro řízení multiplexního displeje. To lze vyřešit i pomocí oscilátoru 50 MHz ale zvolil jsem tuto možnost kvůli plnému využití obvodu 74HC14D. A jako poslední možnost vstupu hodinového signálu je zde použito tlačítko s ošetřením proti zákmitům, které může sloužit ke krokování programu v obvodu CPLD.

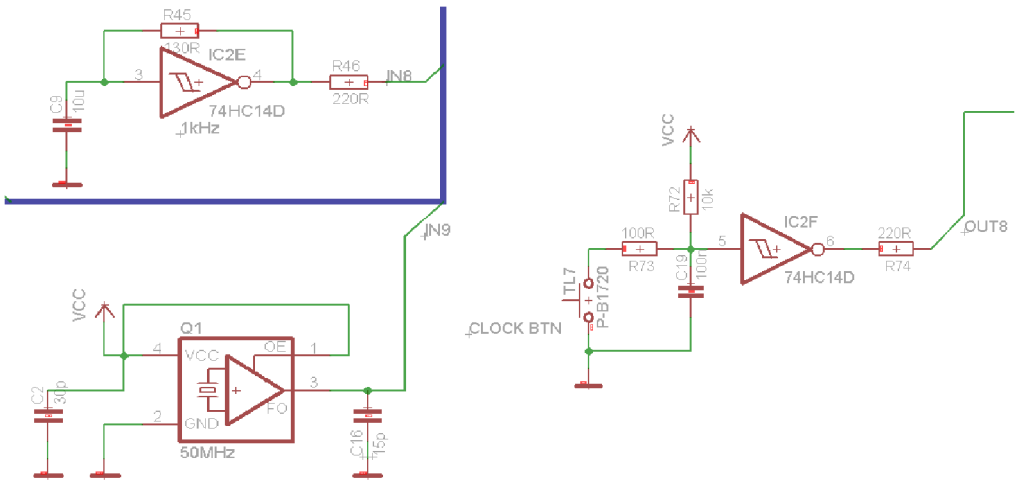


Obrázek 3.4 Katalogové zapojení oscilátoru (převzato z [5])

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

Rezistor R je zvolen na 130 Ω, takže dosazením a úpravou dopočítáme kondenzátor C, abychom dosáhli frekvence 1 kHz.

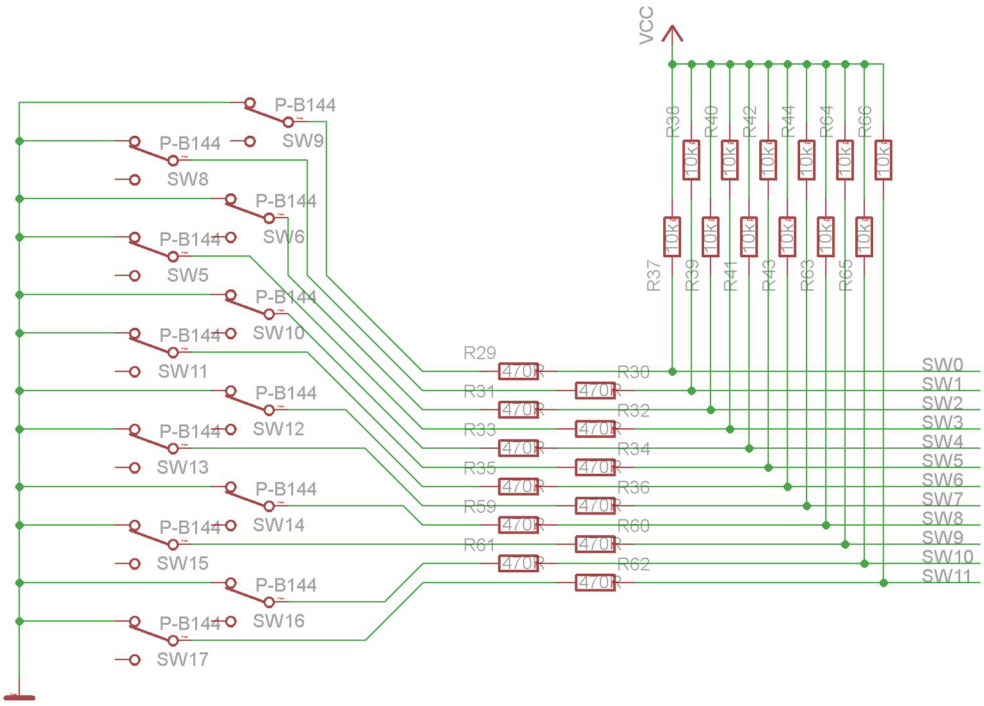
$$C = \frac{1}{0{,}8fR} = \frac{1}{0{,}8 \cdot 1000 \cdot 130} = 9{,}6\ \text{uF}$$



Obrázek 3.5 Zapojení oscilátorů

3.4 Vstupní přepínače

Pro demonstraci vstupních signálů jsem použil sestavu 12 přepínačů. Ty svým počtem můžou reprezentovat binární hodnotu až 4096. Zapojení je zobrazeno na obrázku 3.5



Obrázek 3.6 Zapojení přepínačů

Zapojení je navrženo jako dělič napětí. Při rozepnutém stavu spínačů je na vstup přivedeno napětí 3,3 V přes rezistor 10 kΩ, což omezuje proud vstupující do obvodu. V případě nastavení nízké úrovně vstupu bez omezení proudu, by mohlo dojít k přetížení obvodu CPLD. Tím pádem je proud omezen na hodnotu

$$I = \frac{Vcc}{R37} = \frac{3{,}3}{10000} = 330\ uA$$

V tomto případě jde o reprezentaci vysoké úrovně. Nízká úroveň je definovaná sepnutým přepínačem, kdy dojde k uzemnění obvodu přes rezistor o hodnotě 470 Ω. Tím získáme dříve zmíněný dělič napětí. V tomto případě na vstup CPLD je přiváděna nízká úroveň o velikosti napětí

$$U_2 = \frac{R29}{R29 + R37} \cdot Vcc = \frac{470}{470 + 10000} \cdot 3{,}3 = 0{,}148\ V$$

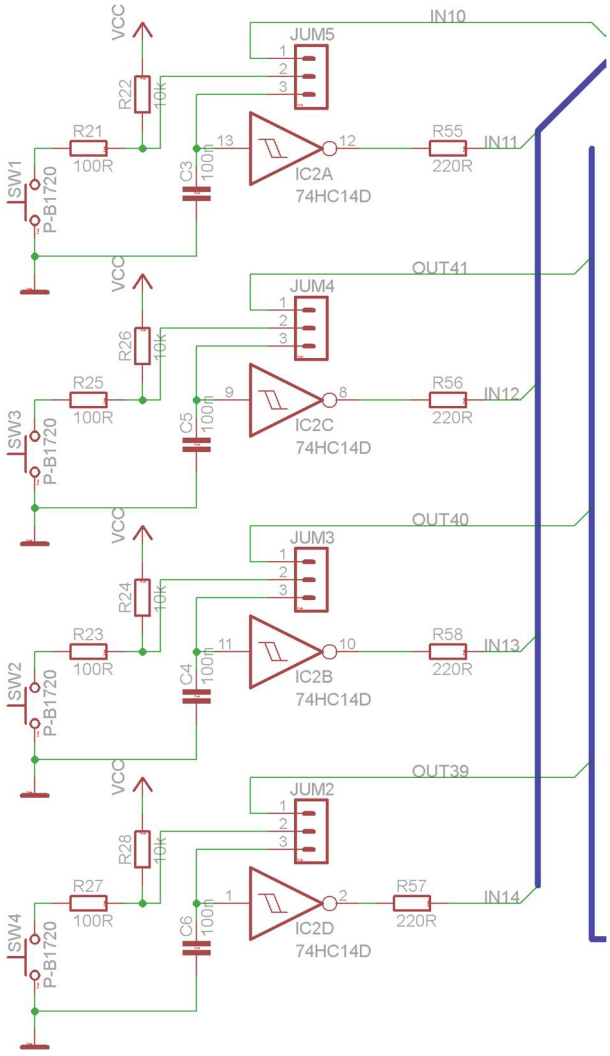
3.5 Vstupní tlačítka

Další vstup signálů je vytvořen čtveřicí tlačítek, u kterých je možná volba ošetření mechanických zákmitů. V případě nepoužití ošetřující volby, lze zákmity odstranit na úrovni softwaru. Volba zapojení tlačítek je rozhodována pomocí propojek.

Doba nestability tlačítek se většinou pohybuje v rozmezí 100 ns až 10 ms. V některých případech i větší. Pro ošetření zákmitů jsem zvolil hodnotu 1 ms. Při volbě rezistorů na 10 kΩ a 100 Ω, jsem dopočítal hodnotu použitého kondenzátoru pomocí vztahu

$$C = \frac{1}{(R21 + R22) \cdot f} = \frac{1}{(10000 + 100) \cdot 1000} = 99\ \text{nF}$$

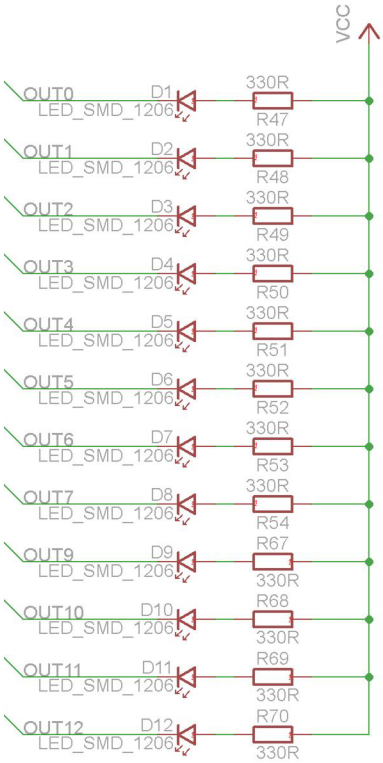
Z toho důvodu jsem zvolil hodnotu kondenzátoru na 100 nF. Zapojení je znázorněno na obrázku 3.7.



Obrázek 3.7 Zapojení tlačítek

3.6 Výstupní LED diody

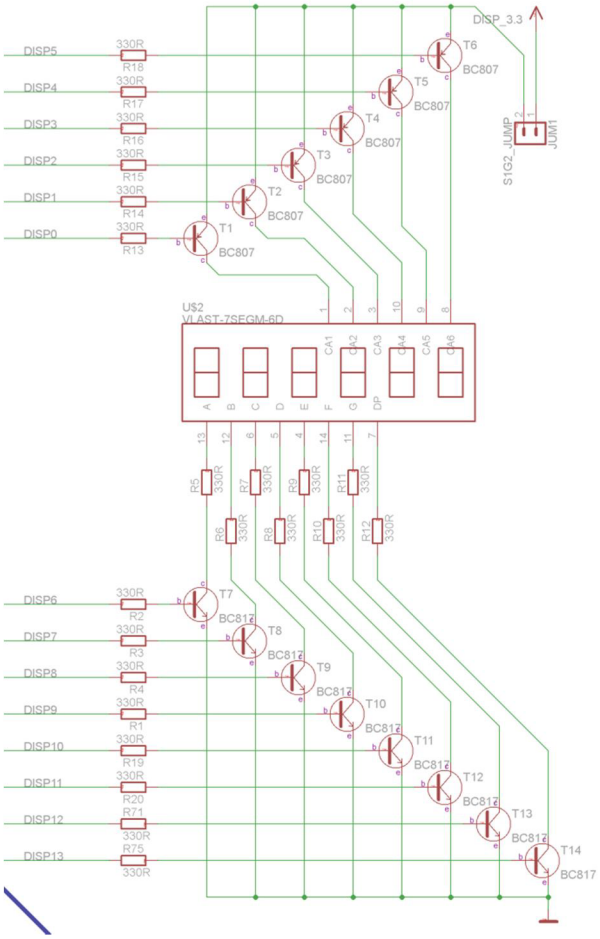
Jako demonstraci výstupních dat jsem navrhl 12 LED diod. Počet je totožný s počtem přepínačů. Diody jsou připojeny na napájecí napětí 3,3 V přes rezistor 330 Ω pro omezení proudu diodou na 10 mA. LED diody svítí ve chvíli nastavení na obvodu CPLD nízké úrovně na vstupu. Zapojení je na obrázku 3.8.



Obrázek 3.8 Zapojení LED diod

3.7 Multiplexně řízený 7segmentový displej

V návrhu jsem použil šestimístný 7segmentový multiplexně řízený displej se společnou anodou. Předem jsem ověřil kapacitní náročnost řadiče displeje na obvod CPLD. A řadič zabíral přibližně 10 procent kapacity obvodu. Z toho důvodu jsem se rozhodl pro použití multiplexně řízeného displeje. Pro omezení proudu segmentem jsou použity rezistory o velikosti 330 Ω. Tím dosáhneme proudu 10 mA na segment. Je zde přidána i možnost manuálního odpojení napájení displeje pomocí propojky. Spínání napájení bylo řešeno přes tranzistory BC 817 a BC 807 aby nedocházelo k velkému zatížení CPLD. Napájení je ze samostatného stabilizátoru. Schema zapojení je na obrázku 3.9.



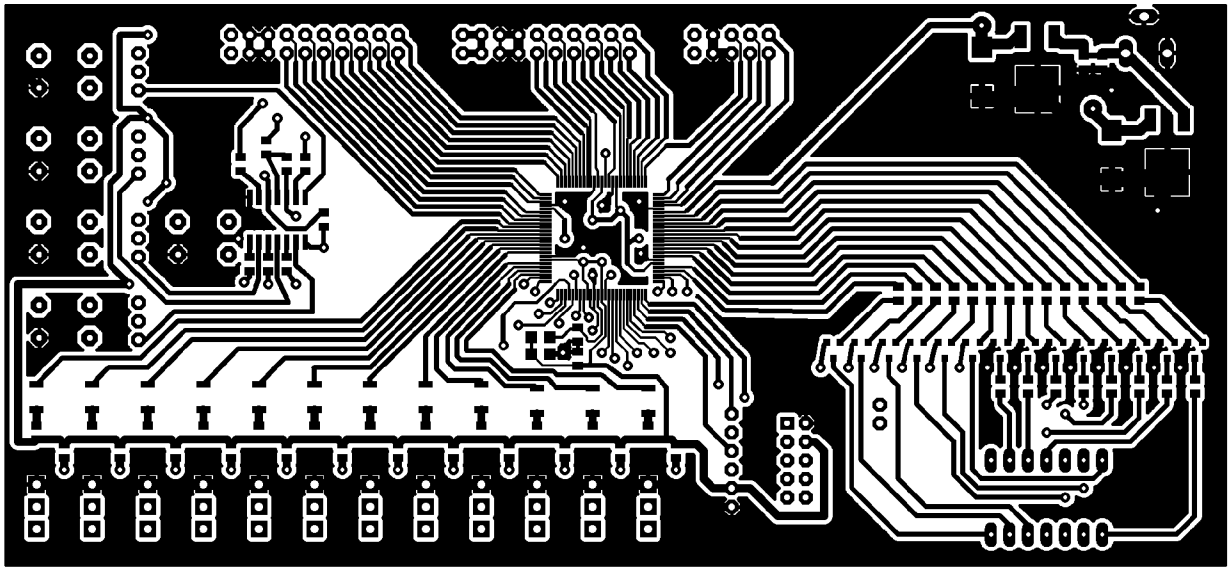
Obrázek 3.9 Zapojení displeje

3.8 Konektory pro externí I/O zařízení

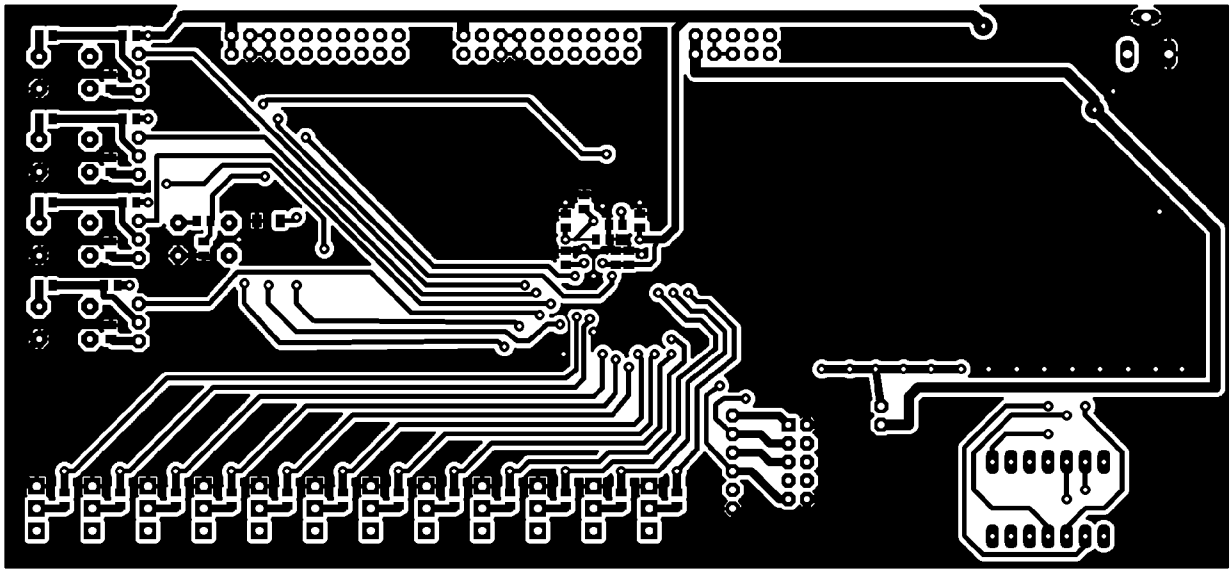
Jsou použity 3 konektory. Každý obsahuje i napájení 3,3 V a připojení GND. Dvojice konektorů EXT1 a EXT2 mají vyvedeno 14 a 15 uživatelských pinů obvodu CPLD. A konektor EXT3 má vyvedeno 5 uživatelských pinů. To znamená, že celkem je možné využít až 34 uživatelských pinů obvodu CPLD pro připojení externích zařízení.

4 NÁVRH DESKY PLOŠNÝCH SPOJŮ

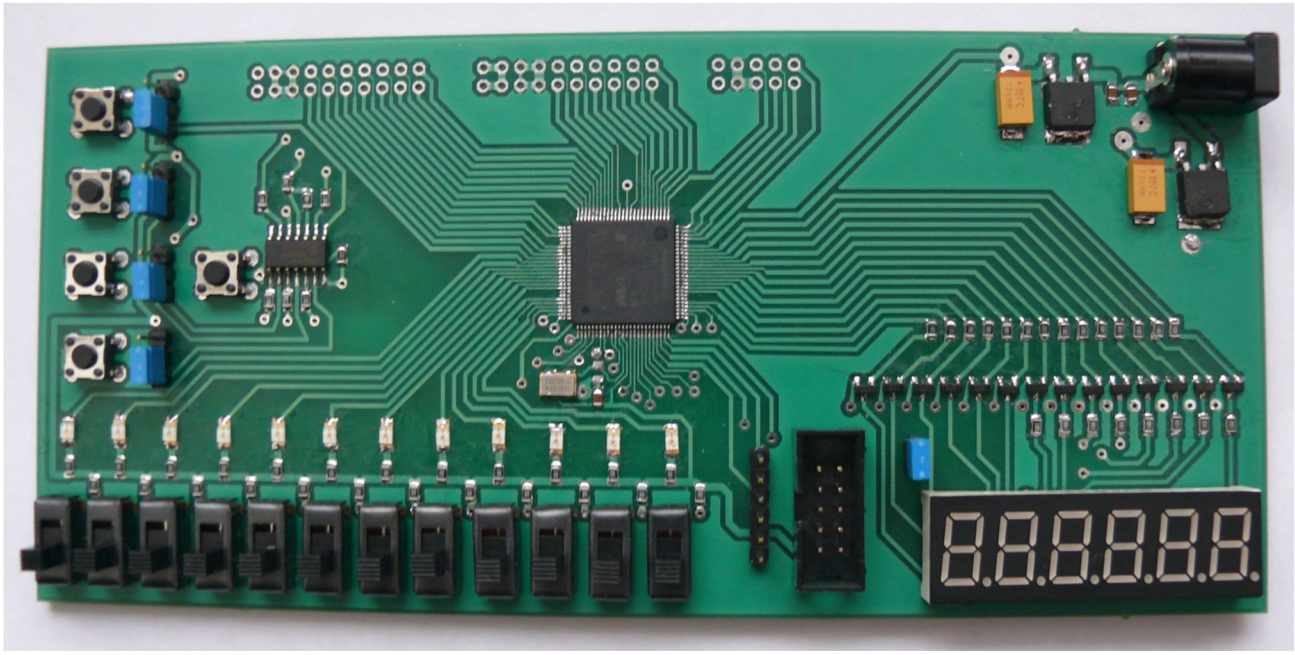
Návrh desky plošných spojů proběhl podle doporučení uveřejněných na webu UREL VUT Brno a podle doporučení vedoucího práce.



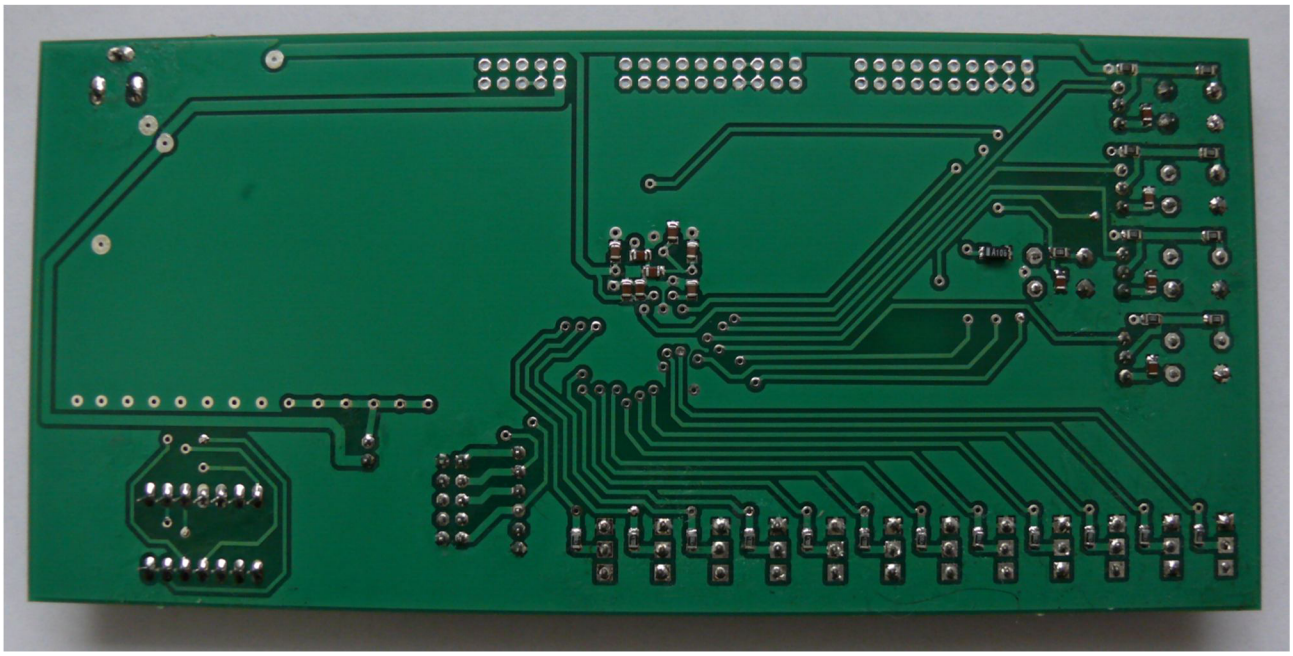
Obrázek 4.1 TOP vrstva navržené desky



Obrázek 4.2 BOTTOM vrstva navržené desky



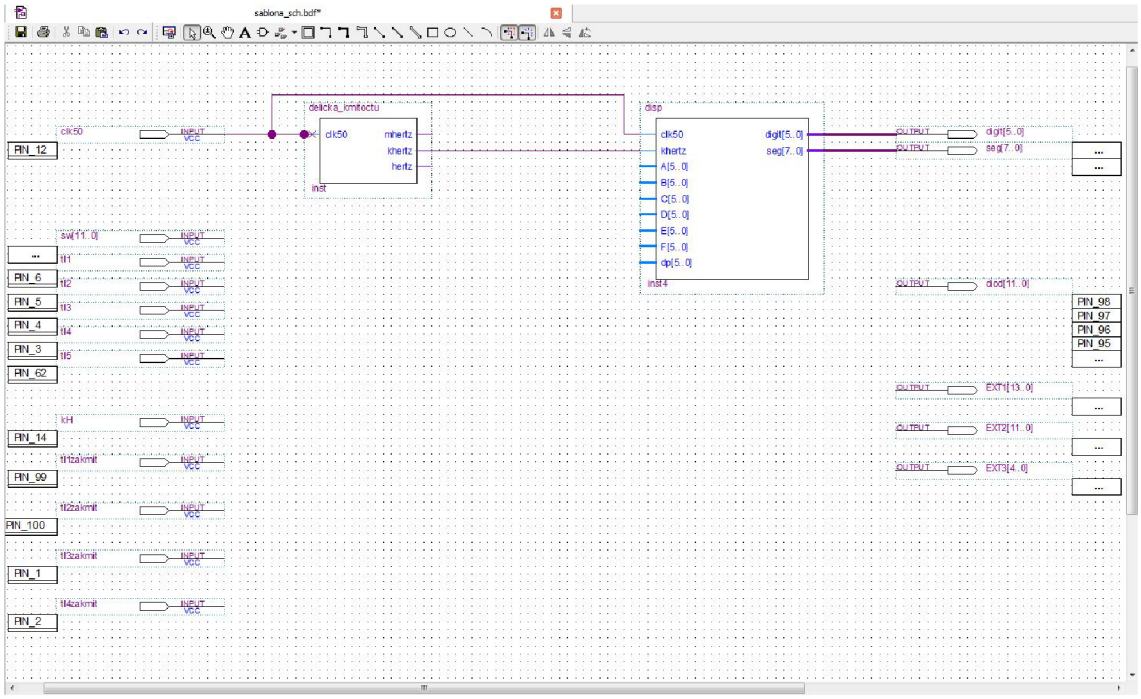
Obrázek 4.3 TOP vrstva osazené desky



Obrázek 4.4 BOTTOM vrstva osazené desky

5 NÁVOD NA PRÁCI S PŘÍPRAVKEM

Pro práci s přípravkem jsem vytvořil dvě šablony. Jedna šablona je se schematickou vrcholovou jednotkou. Druhá pak s textovou. Obě šablony jsou uloženy na doprovodném CD. Budeme se zabývat prací se šablonou se schematickou vrcholovou jednotkou.



Obrázek 5.1 Schematická vrcholová jednotka šablony

5.1 Postupy pro práci se šablonou v QUARTUS II

Po spuštění softwaru QUARTUS II se nám zobrazí nabídka na vytvoření nového projektu nebo otevření stávajícího projektu. Zvolíme otevření stávajícího projektu. A vybereme naši šablonu s názvem “sablona_sch.qpf“.

5.1.1 Přejmenování projektu

V tuto chvíli máme otevřenou šablonu. Ale abychom neměli všechny projekty se stejným jménem, provedeme následujících pár kroků pro přejmenování.

Z důvodu, že Quartus nedisponuje přímo funkcí *Save Project As,* rozklikneme si nabídku *Project* a zvolíme položku *Copy Project*. Otevře se nám nabídka pro zvolení nového umístění projektu a název projektu. V našem případě zvolíme jméno “vzor_sch“ a potvrdíme. V případě zvolení místa projektu, které neexistuje, budeme dotázáni, jestli se má složka vytvořit. Zvolíme *Ano*. Tímto krokem se nám vytvořil na novém umístění projekt s novým jménem. Ale pořád se nám zobrazuje vrcholová jednotka s názvem starým. Proto v okně *Project Navigator* přepneme na záložku *Files* a dvojklikem

otevřeme *šablona_sch.bdf*. Následně provedeme uložení pod jiným názvem. V našem případě pod jménem *vzor_sch.bdf*. Automaticky se nám tento nově vytvořený soubor přidal do projektu. Jelikož nám nyní zůstal v projektu i původní soubor, tak na něj klikneme pravým tlačítkem myši a zvolíme odstranění z projektu. Nyní klikneme na námi vytvořený soubor *vzor_sch.bdf* pravým tlačítkem a zvolíme ho za vrcholovou jednotku. Nyní máme projekt vytvořen. Pod hlavní nabídkou se nám zobrazuje poslední vzpomínka na šablonu. To značí jaké nastavení projektu je aktuální. V našem případě kam jsou přiřazené piny na pouzdře obvodu, atd. Jestli nám tato vzpomínka vadí, můžeme ji změnit. V menu rozklikneme *Project* a zvolíme položku *Revision*. Otevře se nám okno, v něm poklepáme na položku *New Revision*. Vyskočí nám okno ve kterém nastavíme nové jméno, v našem případě *vzor_sch* a v položce *Based On Revision* ponecháme *sablona_sch*. Tím zajistíme totožné nastavení se šablonou. Zkontrolujeme, jestli máme zaškrtnuté možnosti pro *Copy Database* a *Set As Current Revision*. Poté potvrdíme *OK*. Vidíme, že se nám přidal nový profil a že ten starý nám tam zůstal. Teď je jen na nás jestli ten starý smažeme nebo ne. Tímto nám už nestojí nic v cestě pokračovat v návrhu.

5.1.2 Seznámení s vrcholovou jednotkou

Nyní si představíme schematickou vrcholovou jednotku. Na *obrázku 5.1* vidíme vytvořené dva bloky. Zdrojové kódy jsou uvedeny v příloze *B.1* a *B.2*.

První blok nazvaný *dělička kmitočtu* slouží jako dělič kmitočtu. Na vstupu máme oscilátor o frekvenci 50 MHz. Na výstupu pak 1 MHz, 1 kHz a 1 Hz.

Druhý blok je nazván jako *disp* a funguje jako řadič multiplexně řízeného 7segmentového displeje. Vstupní signály jsou pojmenovány písmeny *A, B, C, D, E, F, dp, clk50 a khertz*. Vstupy *A* až *F* jsou šestibitové a každý vstup slouží právě jednomu digitu na displeji. Každý bit reprezentuje právě jen jeden segment z celé číslice. Vstup *dp* je taktéž šestibitový a každý bit reprezentuje právě jednu tečku ke každému digitu. Pomocí vstupu *clk50* je frekvencí 50 MHz obnovovány vstupní údaje ze vstupů *A – F* a *dp*. Vstupem *khertz* je s frekvencí 1 kHz postupně přepínané zobrazované digity na displeji. Na šabloně je kromě výše popsaných bloků i mnoho vstupů a výstupů. Ty slouží minimálně k ilustraci všech možných periferií, které se dají využít pro návrh. V jiném případě je možné vstupy a výstupy přímo využít a propojit je s návrhem.

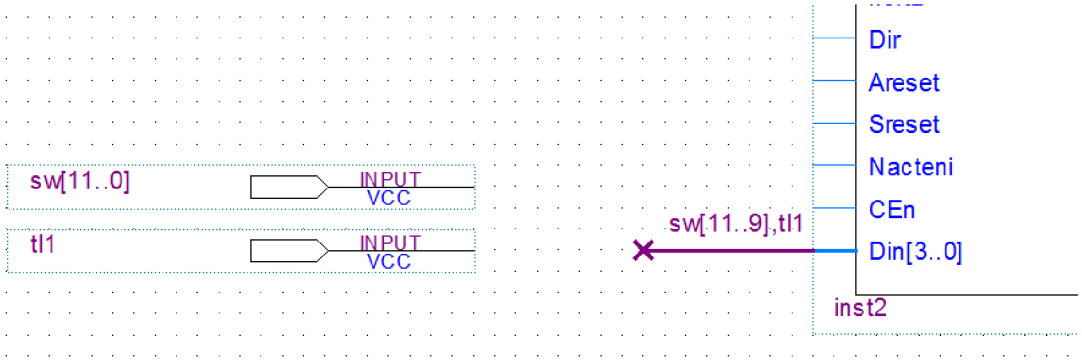
5.1.3 Vytvoření vlastního konstrukčního bloku

Pro náš návrh budeme potřebovat vytvořit další blok, který bude plnit funkci, kterou mu nadefinujeme. Pro vytvoření, v nabídce *File* klikneme na možnost *New*. V okně, které se nám zobrazí, vybereme *VHDL File* a potvrdíme *OK*. Otevře se nám soubor, který si následně uložíme pod námi žádaným názvem. Automaticky se nám přidá do projektu. Nyní nás čeká sestrojení konstrukce. Zápis je členěn následujícím způsobem.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

Entity Counter IS
  PORT(
        hertz, Smer, Areset:    IN std_logic;
        Sreset, Nacteni, Cen:   IN std_logic;
        Din:        IN std_logic_vector(3 downto 0);
        Count:      OUT std_logic_vector(3 downto 0)
        );
END Counter;

ARCHITECTURE CountArch OF Counter IS

  SIGNAL CntInt:    std_logic_vector(3 downto 0):= "1010";

BEGIN
PROCESS (hertz, Areset) BEGIN
  IF Areset = '1' THEN
        CntInt <= "0000";
  ELSIF (hertz'event AND hertz = '1') THEN
        IF Sreset = '1' THEN
                CntInt <= "0000";
        ELSIF Nacteni = '1' THEN
                              CntInt <= not Din;
        ELSIF CEn = '1' THEN
                IF Smer = '1' THEN
                      CntInt <= CntInt + 1;
                ELSE
                      CntInt <= CntInt - 1;
                END IF;
        END IF;
  END IF;
END PROCESS;
  Count <= not CntInt;
END CountArch;
```

Když máme vytvořenou konstrukci, tak soubor uložíme a provedeme Analýzu a syntézu, kterou spustíme dvojklikem v okně *Tasks* na *Analysis & Synthesis*. Pokud je vše v pořádku můžeme pokračovat ve tvoření schematického bloku. V okně projektu pravým tlačítkem klikneme na námi vytvořený soubor a vybereme možnost *Create symbol Files For Current File*. Jelikož celá šablona potřebuje obnovit schematické symboly všech bloků, provedeme tento krok i pro ostatní bloky. V dalším kroku přejdeme do vrcholové jednotky. Klikneme do schematu pravým tlačítkem a zvolíme *Update Symbol Or Block*. Dvojklikem na levé tlačítko nyní otevřeme okno s názvem *Symbol*. V okně rozklikneme položku *Project*, kde vybereme symbol, který jsme před chvílí vytvořili. Zkontrolujeme, jestli nemáme zaškrtnuté políčko *Insert Symbol As Block* a pokud není zaškrtnuté tak potvrdíme *OK*. Symbol následně vložíme do schematu. Vrátím se ještě k oknu *Symbol*. V případě, že potřebujeme do projektu vložit nějaký další symbol, který patří k běžným funkcím jako na příklad *AND*, *OR* a mnoho dalších můžeme si vybrat z knihovny Quartus. V tomto případě nebudeme hledat v položce *Project* ale v položce *Libraries*. Po nalezení toho co potřebujeme, se

postupuje dále stejně jako při vložení námi vytvořeného bloku. Když se vrátím zpátky do vrcholové jednotky, kde už máme vložen náš blok, je potřeba ho připojit. Připojuje se kliknutím a držením levého tlačítka myši na vývod bloku a tažením k vývodu, kterému ho chceme připojit. Tady musíme dávat pozor. V případě, že chceme připojit sběrnici například k symbolu napájení, musíme připojovat ve směru od sběrnice k napájení. V případě opačném se tyto dva objekty taktéž spojí, ale ne pomocí sběrnicového spoje ale jen jediného spoje, který nám následně při syntéze bude hlásit chybu z důvodu, že nejsou všechny vstupy připojeny. V tomto ohledu podle mého názoru je návrhový systém trochu nešikovný. V případě nutnosti připojení více zdrojů do sběrnice, vyvedeme kousek sběrnice, tu následně pojmenujeme jmény signálů oddělené čárkami, které chceme připojit. V případě nevyužití některých LED diod na desce je potřeba je v návrhu připojit na napájení, protože LED diody, když na ně přivedeme logickou nulu tak svítí. Připojením na logickou jedničku dosáhneme nesvítících LED diod, které nejsou použity.

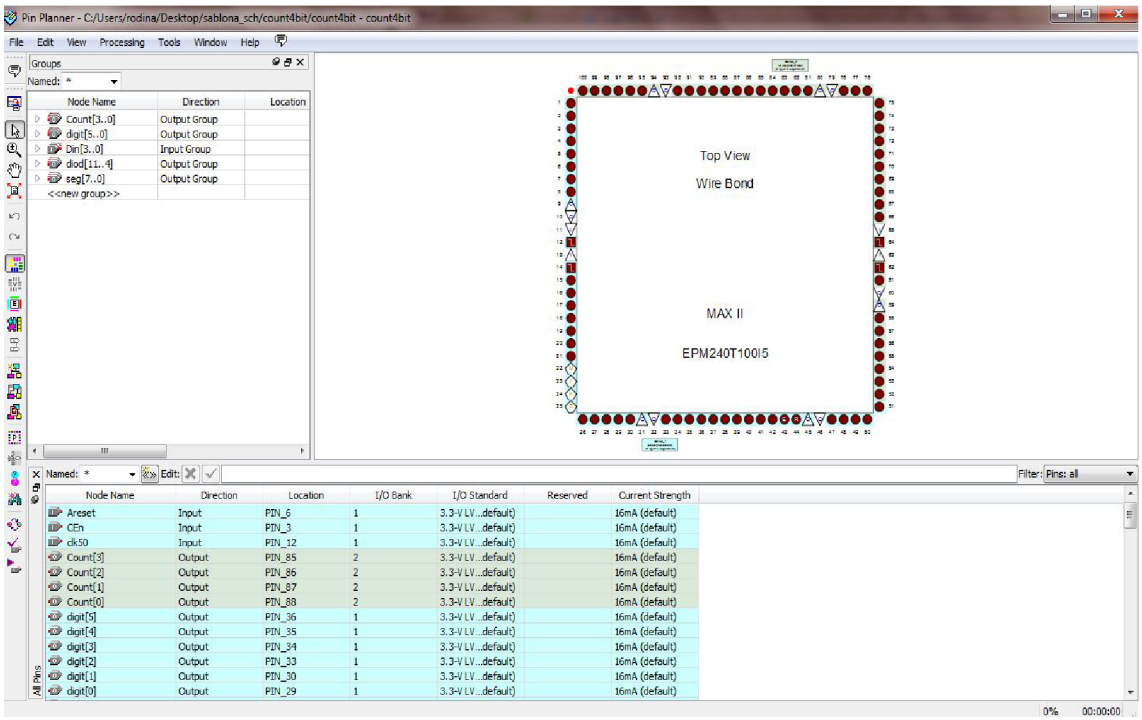


Obrázek 5.2 Připojení více zdrojů do sběrnice

Nyní můžeme provést syntézu, která nám odhalí, jestli se nám někde nachází syntaktická chyba.

5.1.4 Přiřazení vývodů

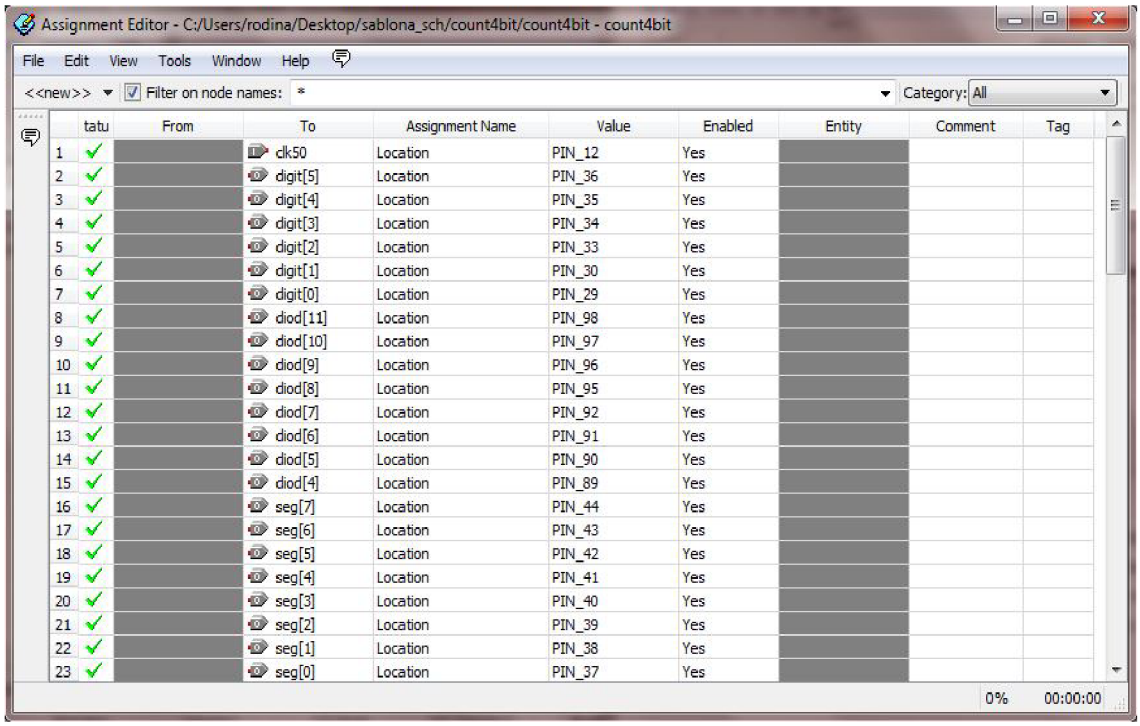
Přiřazení pinů lze v programu Quartus II provést třemi způsoby. První způsob je přes *Pin Planner*, v kterém je graficky zobrazené pouzdro obvodu. Pro názornost je to dobrý způsob, ale v případě zadávání více pinů je to způsob trochu zdlouhavý.



Obrázek 5.3 Přiřazení vývodů přes Pin Planner

Dalším způsobem je pomocí *Assignment Editoru*, který je v podstatě obdobný způsob přiřazování pinů jako od Xillinx ve formě UCF souboru. A posledním způsobem je pomocí *TCL scriptu*, který lze vygenerovat přes Quartus a následně jej jen lehce upravit podle našich potřeb. Pro moji osobu přišel jako nejpřívětivější způsob přiřazování právě *Assignment Editor*.

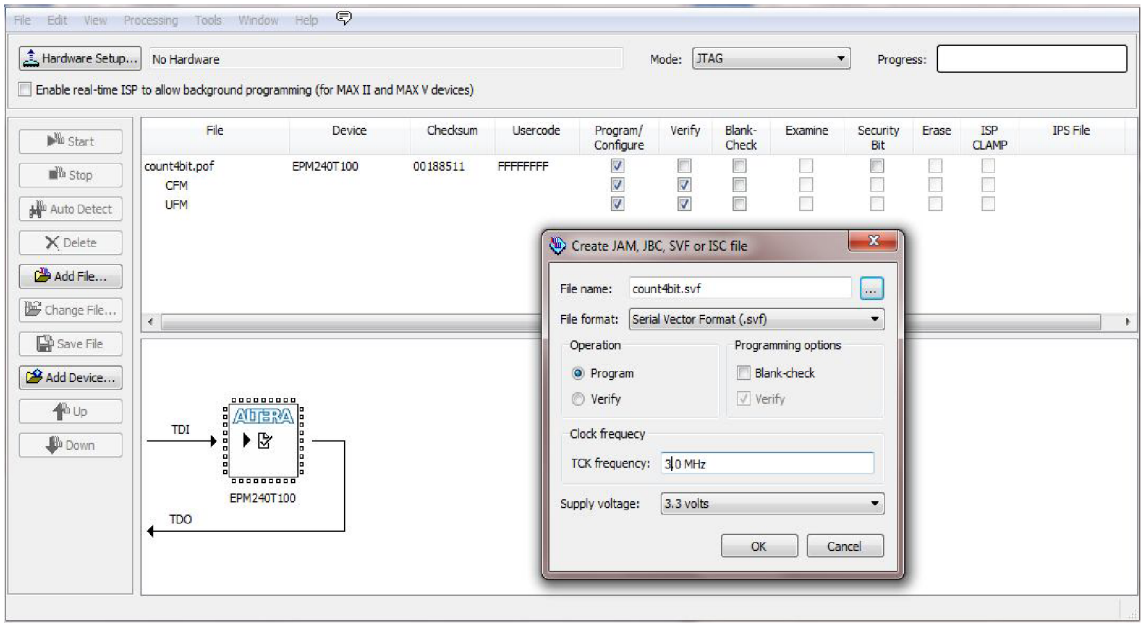
V naší šabloně jsou už všechny piny přiřazeny a tak už nám zbývá jen upravovat pomocí *Assignment Editoru* názvy pinů pro konstrukci. Způsoby přiřazování pinů jsou intuitivně řešeny, takže není potřeba nikterak do hloubky popisovat způsob.



Obrázek 5.4 Přiřazení vývodů v Assignment Editoru

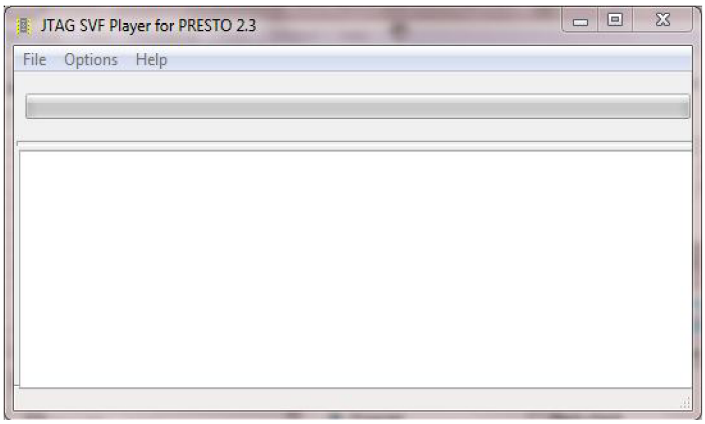
5.1.5 Programování přípravku

Ve chvíli jak máme správně přiděleny piny, tak nám nezbývá nic jiného než celý projekt zkompilovat dvojklikem na *Compile design* v okně *Tasks*. V případě když dojde k úspěšnému zkompilování naší konstrukce, tak se automaticky vygeneruje soubor pro zápis do obvodu, který má koncovku *pof (programmming output file)*. Pro nahrání konstrukce do obvodu slouží program s názvem *Programmer*, který spustíme dvojklikem na položku *Program Device* v okně *Tasks*. Tím, že ho spouštíme přímo přes Quartusu se nám automaticky načte i soubor s programem. V případě že máme programátor, který je podporován softwarem *Programmer*, tak postupujeme následujícím způsobem. Nyní nás čeká detekce a nastavení našeho programátoru, kterým chceme naprogramovat obvod. Klikneme na tlačítko *Hardware Setup* a vybereme programátor, který máme připojený k počítači a potvrdíme *OK*. Nyní stačí už jen kliknout na tlačítko *Start* v nabídce na levé straně okna. Provede se nahrání konstrukce do obvodu. A jestli je vše v pořádku, tak se nám zobrazí okno s potvrzením úspěšného nahrání programu.



Obrázek 5.5 Generování SVF souboru

V případě programování přípravku programátorem Presto od firmy ASIX, nebudeme klikat na tlačítko *Hardware Setup*, ale v menu *File* klikneme na možnost *Create JAM, JBV, SVF or ISC File*. Zobrazí se nám okno, kde nastavíme formát souboru na *Seriál Vector Format (.svf)* a v kolonce *TCK frequency* nastavíme jen 3 MHz z původních 18 MHz. To z toho důvodu, že programátor Presto podporuje maximálně 3 MHz hodiny. V případě pokud zanecháme 18 MHz, taktéž se nám podaří obvod naprogramovat, ale bude průběh programování zdlouhavější. Když máme vše nastaveno, tak stiskneme *OK*. Tím se nám vygeneruje *SVF* soubor s instrukcemi. K dokončení programování potřebujeme si nainstalovat software od výrobce ASIX s názvem *JTAG Player*. *JTAG Player* je uživatelsky velmi jednoduchý program. V první řadě rozklikneme položku *Options* a vybereme položku *Select Presto*. Zaškrtneme programem identifikované zařízení a potvrdíme. Poté už jen v nabídce *File* klikneme na *Open and Process* a vybereme soubor *.svf*, který jsme si nechali vygenerovat a potvrdíme. Okamžitě se začne programovat obvod.



Obrázek 5.6 JTAG SVF Player

5.2 Postupy pro simulace a analýzy VHDL konstrukcí

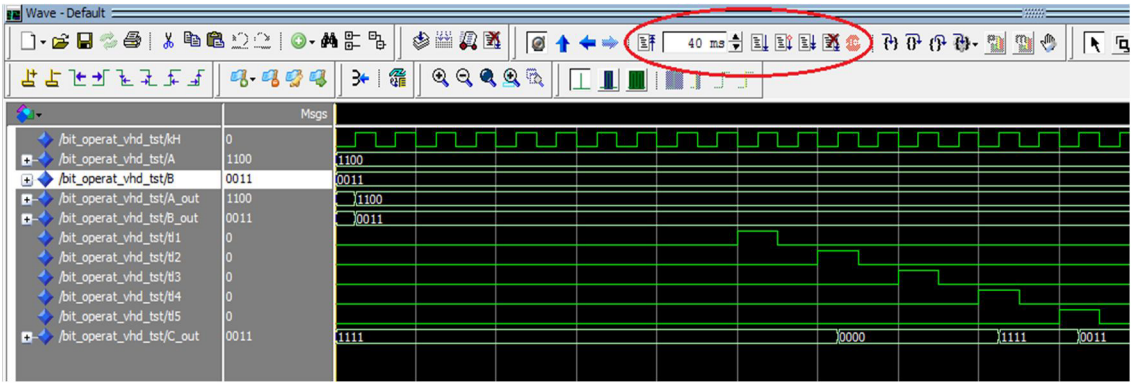
5.2.1 Funkční simulace

V prvé řadě budeme muset povolit generování souboru pro funkční simulaci. To provedeme v nabídce *Assignments -> Settings*. Vybereme kategorii *Simulation*, Jako prostředek nastavíme *ModelSim-Altera*. Rozklikneme možnost *More EDA Netlist Writer Settings*. Tady nastavíme *Generate third-party EDA tool command script for RTL function simulation* a také možnost pro *gate-level simulation*, tím získáme vygenerovaný soubor i pro post-fit simulaci. Vše potvrdíme. Vygenerujeme si šablonu na Test Bench. Dále budu používat výraz benč, který je už vcelku zdomácnělý pojem. V menu *Processing -> Start -> Start Test Bench Template Writer*, tím se nám vygeneruje benč. Nyní si spustíme *Modelsim Altera*. Přejdeme do nabídky *File -> Change Directory*, kde si nastavíme cestu k našemu projektu. Poté v menu *File -> New -> Library* v okně, které se nám zobrazí vybereme poslední možnost *A new library and a logical mapping to it* a zadáme název knihovny, ve které budeme mít soubory pro simulaci. Po potvrzení pokračujeme na menu *Compile -> Compile*. Zde si vybereme námi stvořenou knihovnu a označíme soubory **.vho a *.vht*. Potvrdíme a pravým tlačítkem klikneme na náš benč v knihovně a vybereme *Edit*. Nyní nás čeká úprava vygenerovaného benče a napsání vstupních stimulů pro simulaci. Pro příklad dokládám zdrojový text k benči.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY operator_vhd_tst IS
END operator_vhd_tst;

ARCHITECTURE operator_arch OF operator_vhd_tst IS
SIGNAL A :         STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL A_out :     STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL B :         STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL B_out :     STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL C_out :     STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL tl1 :       STD_LOGIC:='0';
SIGNAL tl2 :       STD_LOGIC:='0';
SIGNAL tl3 :       STD_LOGIC:='0';
SIGNAL tl4 :       STD_LOGIC:='0';
SIGNAL tl5 :       STD_LOGIC:='0';
SIGNAL kH :        STD_LOGIC:='0';
COMPONENT operator
  PORT (
             A :   in std_logic_vector(3 downto 0);
             B :   in std_logic_vector(3 downto 0);
             tl1, tl2, tl3, tl4, tl5:   in std_logic;
             C_out: out std_logic_vector(3 downto 0);
             A_out: out std_logic_vector(3 downto 0);
             B_out: out std_logic_vector(3 downto 0);
             kH:   in std_logic
  );
END COMPONENT;
```

```
BEGIN
  i1 : operator
  PORT MAP (
        A => A,
        A_out => A_out,
        B => B,
        B_out => B_out,
        C_out => C_out,
        tl1 => tl1,
        tl2 => tl2,
        tl3 => tl3,
        tl4 => tl4,
        tl5 => tl5,
        kH => kH
  );

Clock_GEN : PROCESS BEGIN
       loop
          kH <= '0'; wait for 500 us;
          kH <= '1'; wait for 500 us;
       end loop;
END PROCESS Clock_GEN;

always : PRCESS BEGIN
  A <= "1100" ;
  B <= "0011" ;
Wait for 10ms;
tl1 <= '1'; wait for 1ms; tl1 <= '0'; wait for 1ms;
tl2 <= '1'; wait for 1ms; tl2 <= '0'; wait for 1ms;
tl3 <= '1'; wait for 1ms; tl3 <= '0'; wait for 1ms;
tl4 <= '1'; wait for 1ms; tl4 <= '0'; wait for 1ms;
tl5 <= '1'; wait for 1ms; tl5 <= '0'; wait for 1ms;
END PROCESS always;
END operator_arch;
```

Ve chvíli jak máme benč připravený, tak ho uložíme a zkompilujeme ho. Poté už stačí jen dvakrát kliknout na test bech v knihovně a spustí se. Nyní si jen táhnutím myši přidáme kontrolované signály z okna *Objects* do okna *Wave*. Pokud není zobrazeno okno *waves* automaticky, tak si ho spustíme v menu *View -> Wave*. Teď už nezbývá nic jiného než nastavení délky simulace a spuštění.



Obrázek 5.7 Průběhy simulace v ModelSim

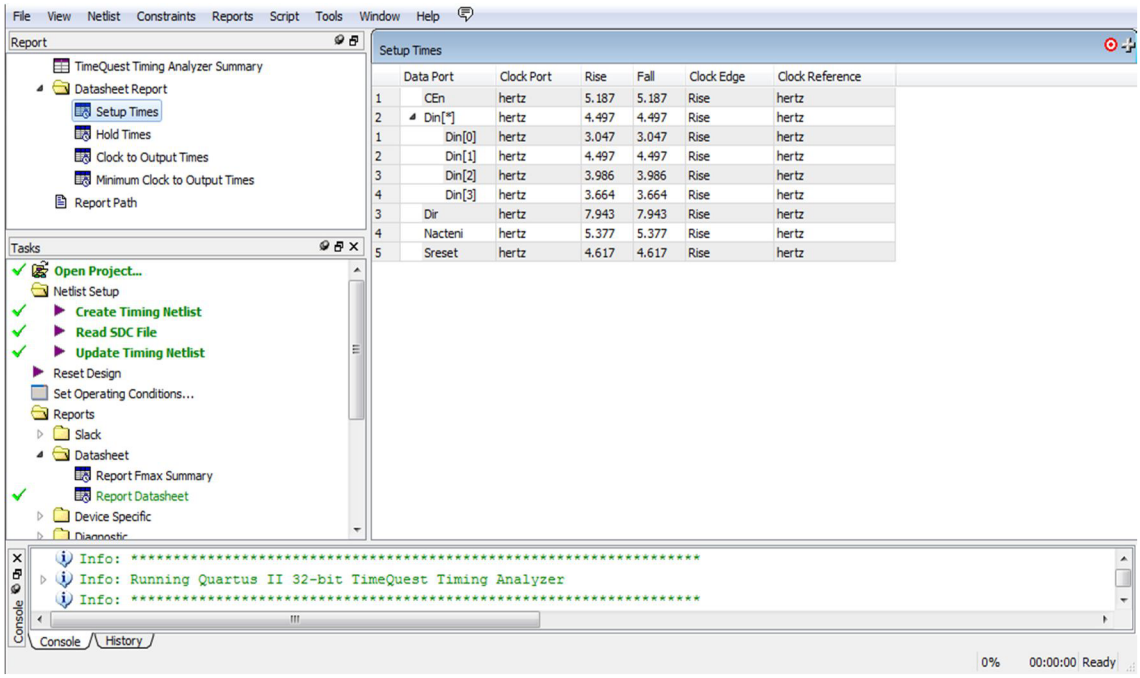
Nyní v okně *Waves* si můžeme zkontrolovat, jestli nám naše konstrukce opravdu dělá to co má. V případě, že nedělá tak tímto způsobem lze rychleji dohledat zdroj problému, který by se ve VHDL kódu samostatně obtížně hledal, zejména u složitějších konstrukcí.

5.2.2 POST-FIT Simulace

Simulace Post Fit neboli simulace po rozmístění je v podstatě simulace benče do kterého jsou započítány i časové prodlevy signálových drah. Proto taky postup je obdobný s benčem s tím rozdílem, že u Post Fit simulace je použit ještě vygenerovaný soubor **.sdo*, který nese informace o časových zpožděních. Postup je následující. Jelikož jsme si už u benče povolili generování i souboru **.sdo*, tak můžeme pracovat přímo v programu *ModelSim*. Po vytvoření naší knihovny a zkompilování souboru **.vho* a **.vht* jako u benče. Klikneme v menu na *Simulate -> Start Simulation*. Otevře se nám okno, kde si vybereme naši knihovnu a označíme soubor s benčem. Přejdeme na záložku *SDF*, kde pomocí tlačítka *Add* vybereme vygenerovaný soubor s časovým zpožděním a potvrdíme. Poté zaškrtneme možnosti *Disable SDF warnings* a *reduce SDF errors to warnings*. Tím potlačíme chyby, které nám vyskočí v případě, že nemáme připojené nějaké piny v návrhu. Potvrdíme *OK*. Nyní se nám spustila post fit simulace a následující práce je totožná s benčem. V mém případě při provedení post fit simulace pro konstrukci s bitovými operacemi se časové zpoždění nikterak neprojevilo.

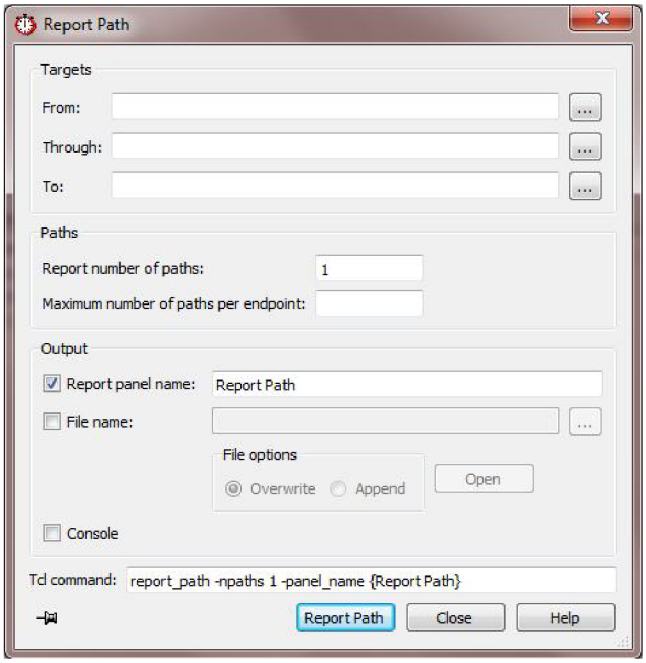
5.2.3 Statická časová analýza

Pro vytvoření statické časové analýzy je potřeba provést *Place&Route*. Poté v okně *Tasks* rozklikneme položku *TimeQuest Timing Analysis* a dvakrát poklepeme na položku *TimeQuest Timing Analyzer*. Tím se nám spustí *TimeQuest*. Budeme dotázáni, jestli chceme vygenerovat *SDC* soubor ze souboru *QSF*. To potvrdíme. Nyní v programu *TimeQuest* v okně *Tasks* spustíme *Report Datasheet*. Tím se nám spustí analýza a následně zobrazí výpis časových zpoždění naší konstrukce.



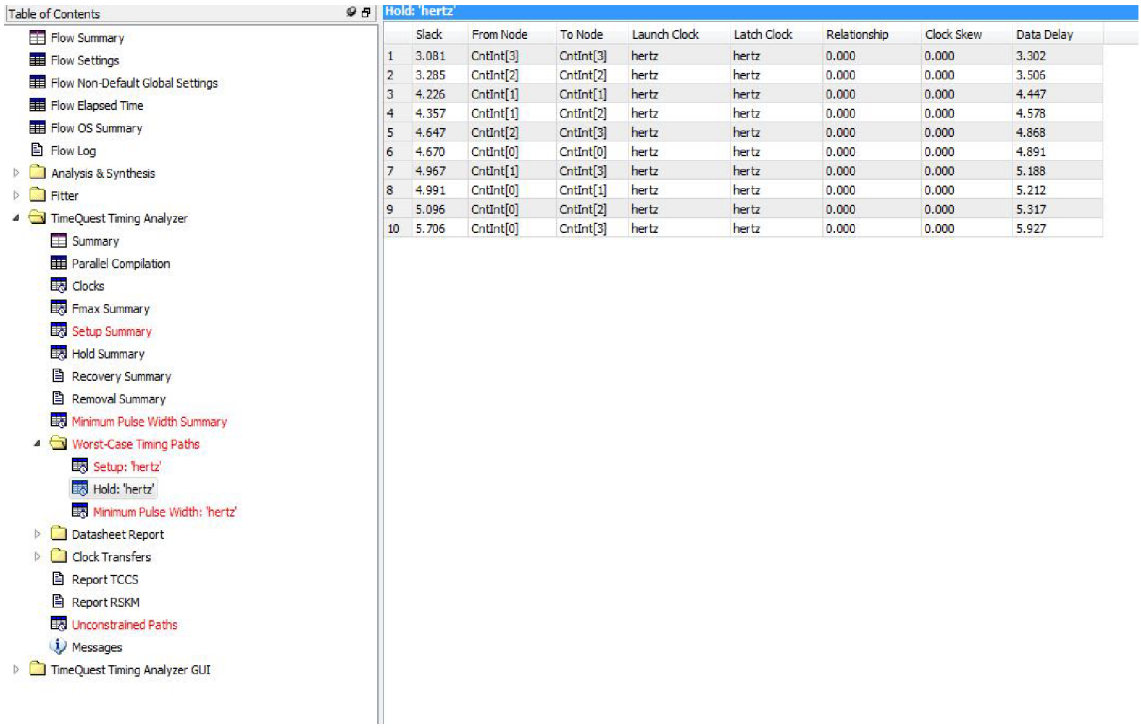
Obrázek 5.8 TimeQuest - Zobrazení výsledků po spuštění Report Data

V případě, že nás zajímají jen určité cesty, tak spustíme v okně *Tasks* položku *Report Path*, která se nachází v *Custom Reports*. Zobrazí se nám okno, kde si přímo nastavíme cestu. Poté jen potvrdíme tlačítkem *Report Path*. Tím se nám provede analýza zadaných cest.



Obrázek 5.9 TimeQuest - nastavení cest pro analýzu

Parametry statické časové analýzy lze získat i přímo v návrhovém systému Quartus. Stačí V okně *Tasks -> TimeQuest Timing Analysis* poklepat na položku *View Report*. Spustí se nám záložka *Compilation Report*, kde v položce *TimeQuest Timing Analyzer* se nám zobrazí získané časové parametry.



	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	3.081	CntInt[3]	CntInt[3]	hertz	hertz	0.000	0.000	3.302
2	3.285	CntInt[2]	CntInt[2]	hertz	hertz	0.000	0.000	3.506
3	4.226	CntInt[1]	CntInt[1]	hertz	hertz	0.000	0.000	4.447
4	4.357	CntInt[1]	CntInt[2]	hertz	hertz	0.000	0.000	4.578
5	4.647	CntInt[2]	CntInt[3]	hertz	hertz	0.000	0.000	4.868
6	4.670	CntInt[0]	CntInt[0]	hertz	hertz	0.000	0.000	4.891
7	4.967	CntInt[1]	CntInt[3]	hertz	hertz	0.000	0.000	5.188
8	4.991	CntInt[0]	CntInt[1]	hertz	hertz	0.000	0.000	5.212
9	5.096	CntInt[0]	CntInt[2]	hertz	hertz	0.000	0.000	5.317
10	5.706	CntInt[0]	CntInt[3]	hertz	hertz	0.000	0.000	5.927

Obrázek 5.10 Quartus II - zobrazení výsledků analýzy

6 VZOROVÉ VHDL KONSTRUKCE

Pro demonstraci funkce přípravku jsem vytvořil následující čtyři VHDL konstrukce. Jejich zdrojové kódy jsou uveřejněny v příloze. Zde uvádím jen jejich popis funkce. Všechny konstrukce jsou plně funkční a odzkoušené a jsou přiloženy na doprovodném CD.

6.1 Hodiny

Tato VHDL konstrukce funguje jako klasické hodiny. Údaj času je zobrazován na 7segmentovém displeji ve tvaru HH.MM.SS. Sekundy jsou také zobrazovány pomocí skupiny LED diod, které blikají v taktu 1 sekundy. Nastavování hodin je provedeno pomocí skupiny přepínačů. První šestice přepínačů slouží k nastavení hodin a druhá šestice k nastavení minut. Nastavení se uloží po stisku tlačítka tl1.

Zdrojový kód je v příloze B.3

6.2 Bitové operace

Bitové operace je program, který slouží k demonstraci logických funkcí AND, OR, EX OR, NOR a negace. Jako vstupy jsem použil dva čtyřbitové signály, které jsou reprezentovány skupinou přepínačů. Stav vstupů je zobrazován pomocí LED diod přímo nad přepínači. Výstup bitové operace je zobrazován pomocí zbylé čtveřice LED diod. Výběr prováděné funkce je volen pomocí pětice tlačítek. První tlačítko tl1 reprezentuje funkci AND, tlačítko tl2 funkci OR, tlačítko tl3 funkci EX OR, tlačítko tl4 funkci NOR a tlačítko tl5 reprezentuje funkci negace vstupu A.

Zdrojový kód je v příloze B.4

6.3 Převod BIN to HEX

Tato konstrukce převádí binární kód na hexadecimální. Jako binární vstup jsou použity přepínače. Ty jsou rozděleny na tři čtveřice bitů. Každá čtveřice bitů je převáděna na hexadecimální hodnotu, která je následně zobrazena na 7segmentovém displeji. Na displeji první tři digity pořád zobrazují nuly a na zbylých třech digitech jsou zobrazeny převody z přepínačů. V tomto programu je ještě vložena další funkce, která zobrazuje na LED diodách předdefinované posloupnosti podle toho, které tlačítko je zrovna stlačeno.

Zdrojový kód je v příloze B.5

6.4 Čítač

Jedná se o čtyřbitový čítač s volbou směru čítání, který je daný prvním v řadě přepínačů. Čítač má funkci synchronního i asynchronního resetu, Asynchronní reset je na tlačítku tl1, synchronní na tlačítku tl2. Na tlačítku tl3 je možnost načtení počáteční hodnoty čítání z poslední čtveřice přepínačů na desce. Pro povolení čítání je použito tlačítko tl4. Stav čítače je zobrazen na čtveřici LED diod. Čítání je prováděno signálem o frekvenci 1 Hz.

Zdrojový kód je v příloze B.6

7 ZÁVĚR

V diplomové práci jsem měl za úkol navrhnout a zrealizovat vývojovou desku pro práci s obvody CPLD od společnosti ALTERA. Tento bod jsem splnil. Přípravek jsem oživil a provedl ověření funkčnosti všech periferií. Přípravek je plně funkční. Zkoušky, které jsem při práci na projektu provedl, potvrdily, že programátor USB Blaster funguje správně. Podařilo se rovněž vyřešit problém s programátorem Presto, které uvádí výrobce programátoru. Zejména omezení frekvence programování a úprava SVF souboru. Zpracoval jsem několik VHDL konstrukcí pro ověření funkce přípravku. Tyto konstrukce byly vytvořeny na základě šablon, které jsem vytvořil. Oba typy šablon i všechny příklady jsou uvedeny na doprovodném CD. Dalším úkolem bylo vytvořit manuál pro práci s přípravkem a postupy prací v návrhovém systému QUARTUS II včetně doprovodných aplikací pro simulace a analýzu, jelikož Altera na svém webu nemá příliš mnoho informací o práci s návrhovým systémem QUARTUS II. V tomto případě má společnost Xilinx mnohem lépe řešenou uživatelskou podporu. Tento návod by měl posloužit pro praktickou práci s přípravkem pro nové uživatele se systémem QUARTUS II.

LITERATURA

[1] POUPA, M. *Programovatelné Logické Obvody* [online]. Dostupné na www: www.fel.zcu.cz.

[2] CPLD a FPGA 1.díl – popis obvodů [online]. Dostupné na www: http://pandatron.cz/?481&cpld_a_fpga_1.dil_-_predstaveni_obvodu.

[3] MAX II Device Family Data Sheet [online]. Dostupné na www: www.altera.com.

[4] Programátory ASIX – referenční příručka [online]. Dostupné na www: www.asix.cz.

[5] 74HC14 Data Sheet [online]. Dostupné na www: www.gme.cz.

[6] Quartus II Handbook v11.0 Vol.2[online]. Dostupné na www: www.altera.com

[7] Quartus II Handbook v11.0 Vol.3[online]. Dostupné na www: www.altera.com

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

PLD	Programmable Logic Device
SPLD	Simple Programmable Logic Device
CPLD	Complex Programmable Logic Device
FPGA	Field Programmable Gate Array
JTAG	Join Test Action Group
TDO	Test Data Output
TDI	Test Data Input
TMS	Test Mode Select
TCK	Test Clock
VHDL	Hardware Description Language
EEPROM	Electrically Erasable Programmable Read Only Memory
PLA	Programmable Logic Array
PAL	Programmable Array Logic
GAL	Generic Array Logic
LED	Light Emited Diode
SVF	Seriál Vector Format
NAND	Not And
EX OR	Exclusive Or
DAC	Digital to Analog Converter
RAM	Random Access Memory
UFM	User Flash Memory
ISP	In Systém Programming
QPF	Quartus Project File
BDF	Block Diagram File
UCF	User Constrainsts File
SDO	Standard Delay format Output file
VHO	VHDL Output file
VHT	VHDL Test file
SDC	Synopsys Design Constraint file
QSF	Quartus Settings File

SEZNAM PŘÍLOH

A Schema přípravku **35**

B Zdrojové kódy **36**

B.1 Dělič kmitočtu – zdrojový kód 36

B.2 Řadič displeje – zdrojový kód 38

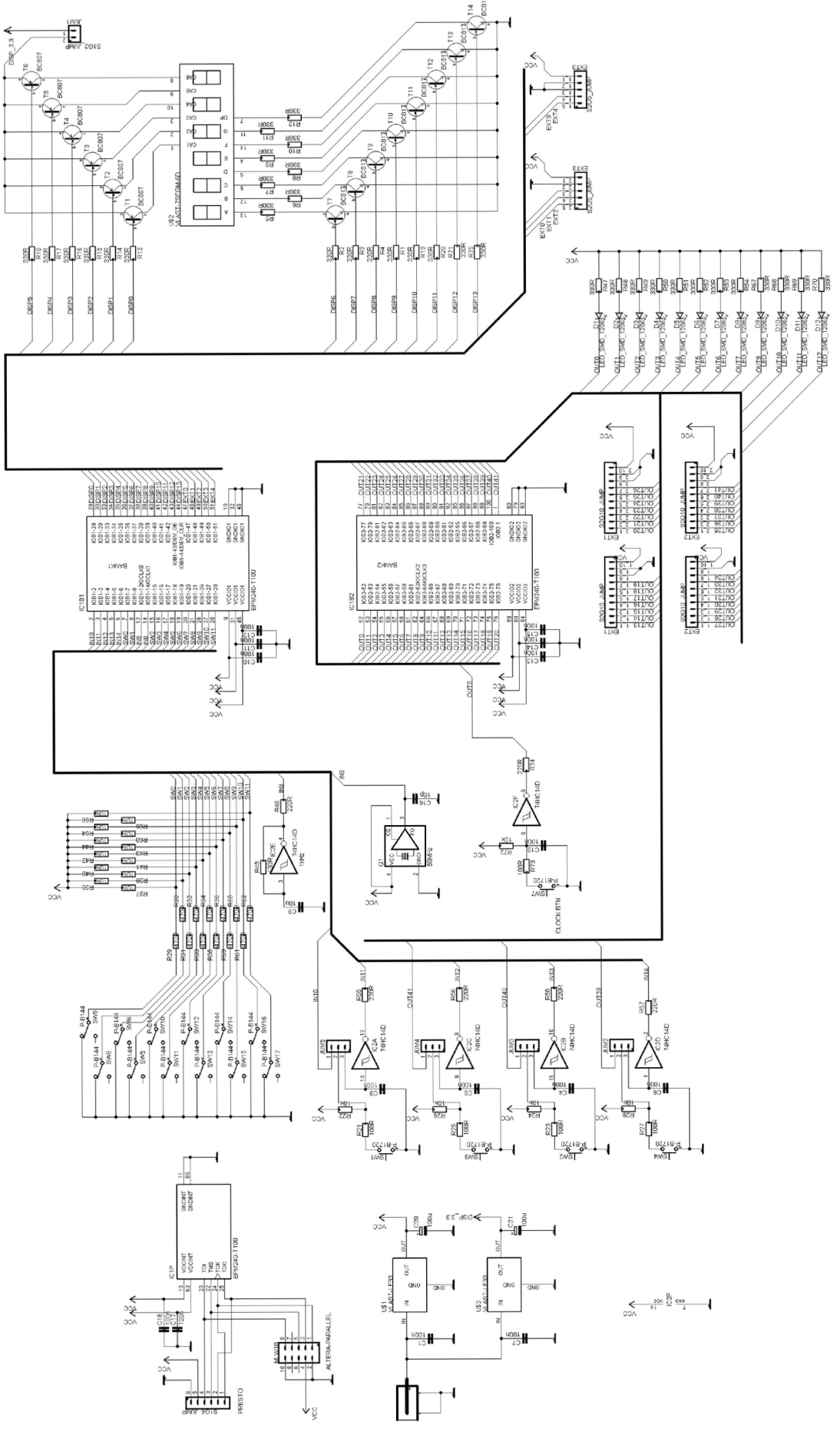
B.3 Hodiny – zdrojový kód 40

B.4 Bitové operace – zdrojový kód 42

B.5 Převod BIN to HEX – zdrojový kód 43

B.6 Čítač – zdrojový kód 45

A SCHEMA PŘÍPRAVKU



B ZDROJOVÉ KÓDY

B.1 Dělič kmitočtu – zdrojový kód

```
LIBRARY ieee;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE ieee.std_logic_1164.all;

ENTITY delicka_kmitoctu IS
      PORT
      (
      clk50:      IN STD_LOGIC;
      mhertz:     OUT STD_LOGIC;
      khertz:     OUT STD_LOGIC;
      hertz:      OUT STD_LOGIC
      );
END delicka_kmitoctu;

ARCHITECTURE delicka_kmitoctu_architecture OF delicka_kmitoctu IS
SIGNAL mhertz_count:  STD_LOGIC_VECTOR(5 DOWNTO 0) := (OTHERS => '0');
SIGNAL khertz_count:  STD_LOGIC_VECTOR(9 DOWNTO 0) := (OTHERS => '0');
SIGNAL hertz_count:   STD_LOGIC_VECTOR(9 DOWNTO 0) := (OTHERS => '0');
SIGNAL mhertz_en:     STD_LOGIC := '0';
SIGNAL khertz_en:     STD_LOGIC := '0';
SIGNAL hertz_en:      STD_LOGIC := '0';

BEGIN
PROCESS (clk50) BEGIN
      IF clk50'event AND clk50 = '1' THEN
                  mhertz_count <= mhertz_count + 1;
                  IF mhertz_count = "110001" THEN       -- 49
                              mhertz_en <= '1';
                              mhertz_count <= (OTHERS => '0');
                  ELSE
                              mhertz_en <= '0';
                  END IF;
      END IF;
END PROCESS;

mhertz<=mhertz_en;
```

```
-- generace 1 kHz z mhertz_en
PROCESS (clk50) BEGIN
      IF clk50'event AND clk50 = '1' THEN
            IF mhertz_en = '1' THEN
                  khertz_count <= khertz_count + 1;
                  IF khertz_count = "1111100111" THEN     -- 999
                        khertz_en <= '1';
                        khertz_count <= (OTHERS => '0');
                  ELSE
                        khertz_en <= '0';
                  END IF;
            ELSE
                  khertz_en <= '0';
            END IF;
      END IF;
END PROCESS;

khertz<=khertz_en;

-- generace 1 Hz z 1 kHz
PROCESS (clk50) BEGIN
      IF clk50'event AND clk50 = '1' THEN
            IF khertz_en = '1' THEN
                  hertz_count <= hertz_count + 1;
                  IF hertz_count = "1111100111" THEN     -- 999
                        hertz_en <= '1';
                        hertz_count <= (OTHERS => '0');
                  ELSE
                        hertz_en <= '0';
                  END IF;
            ELSE
                  hertz_en <= '0';
            END IF;
      END IF;
END PROCESS;

hertz<=hertz_en;

END delicka_kmitoctu_architecture;
```

B.2 Řadič displeje – zdrojový kód

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
------------------------------------------------------------------------
------------
ENTITY disp IS
  PORT(clk50:     IN  STD_LOGIC;
       khertz:    IN  STD_LOGIC;
       digit:     OUT STD_LOGIC_VECTOR(5 DOWNTO 0);
       seg:       OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
       A:         IN     STD_LOGIC_VECTOR (5 downto 0);
       B:         IN     STD_LOGIC_VECTOR (5 downto 0);
       C:         IN     STD_LOGIC_VECTOR (5 downto 0);
       D:         IN     STD_LOGIC_VECTOR (5 downto 0);
       E:         IN     STD_LOGIC_VECTOR (5 downto 0);
       F:         IN     STD_LOGIC_VECTOR (5 downto 0);
       dp:        IN  STD_LOGIC_VECTOR (5 downto 0)
       );
      END disp;
------------------------------------------------------------------------
ARCHITECTURE Behavioral OF disp IS
  SIGNAL cd:      STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";
  SIGNAL curr:    STD_LOGIC_VECTOR(5 DOWNTO 0) := "000000";
  SIGNAL dot:     STD_LOGIC := '0';
------------------------------------------------------------------------
BEGIN
  PROCESS (clk50) BEGIN

      IF clk50'event AND clk50 = '1' THEN

--inkrementace "ukazatele" - vyber vzdy jen jedne ze sesti segmentovek
      IF khertz = '1' THEN
            cd <= cd + 1;
            IF cd = "101" THEN
                  cd <= "000";
            END IF;
      END IF;

-- podle "ukazatele" cd je aktivovana prislusna pozice (aktivni v 0)
   CASE cd IS  -- curr je zobrazena cifra, digit je pozice na displeji
   WHEN "000"   =>  curr <= F;   digit <= "111110";    dot <= dp(0);
   WHEN "001"   =>  curr <= E;   digit <= "111101";    dot <= dp(1);
   WHEN "010"   =>  curr <= D;   digit <= "111011";    dot <= dp(2);
   WHEN "011"   =>  curr <= C;   digit <= "110111";    dot <= dp(3);
   WHEN "100"   =>  curr <= B;   digit <= "101111";    dot <= dp(4);
   WHEN OTHERS  =>  curr <= A;   digit <= "011111";    dot <= dp(5);
   END CASE ;

-- nastaveni prislusneho znaku na segmentovce podle BCD kodu na vstupu
   CASE curr IS
   WHEN "000000" => seg <= dot & "0111111";       -- 0
   WHEN "000001" => seg <= dot & "0000110";       -- 1
   WHEN "000010" => seg <= dot & "1011011";       -- 2
   WHEN "000011" => seg <= dot & "1001111";       -- 3
   WHEN "000100" => seg <= dot & "1100110";       -- 4
```

```
    WHEN "000101" => seg <= dot & "1101101";       -- 5
    WHEN "000110" => seg <= dot & "1111101";       -- 6
    WHEN "000111" => seg <= dot & "0000111";       -- 7
    WHEN "001000" => seg <= dot & "1111111";       -- 8
    WHEN "001001" => seg <= dot & "1101111";       -- 9
    WHEN "001010" => seg <= dot & "1110111";       -- A
    WHEN "001011" => seg <= dot & "1111100";       -- B
    WHEN "001100" => seg <= dot & "0111001";       -- C
    WHEN "001101" => seg <= dot & "1011110";       -- D
    WHEN "001110" => seg <= dot & "1111001";       -- E
    WHEN OTHERS => seg <= dot & "1110001";         -- F
    END CASE;

    END IF;
   END PROCESS;

END Behavioral;
```

B.3 Hodiny – zdrojový kód

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

ENTITY cas IS
  PORT(clk50:     IN  STD_LOGIC;
       hertz:     IN  STD_LOGIC;
       sw:        IN STD_LOGIC_VECTOR (0 to 11);
       tl1:       IN STD_LOGIC;
       A:         OUT STD_LOGIC_VECTOR (5 downto 0);
       B:         OUT STD_LOGIC_VECTOR (5 downto 0);
       C:         OUT STD_LOGIC_VECTOR (5 downto 0);
       D:         OUT STD_LOGIC_VECTOR (5 downto 0);
       E:         OUT STD_LOGIC_VECTOR (5 downto 0);
       F:         OUT STD_LOGIC_VECTOR (5 downto 0)
       );
      END cas;

ARCHITECTURE Behavioral OF cas IS

  SIGNAL diods:         STD_LOGIC_VECTOR (11 downto 0):="000000000000";
  SIGNAL hodiny:        STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL minuty:        STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL sekundy:       STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL hod:           STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL min:           STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL sec:           STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL A1:            STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL B1:            STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL C1:            STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL D1:            STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL E1:            STD_LOGIC_VECTOR (5 downto 0):="000000";
  SIGNAL F1:            STD_LOGIC_VECTOR (5 downto 0):="000000";

BEGIN

  PROCESS (clk50) BEGIN

      IF clk50'event AND clk50 = '1' THEN
            IF tl1='1' then
                  hodiny <= NOT sw(0 to 5);
                  hod <= NOT sw(0 to 5);
                  C1 <= "000000";
                  minuty <= NOT sw(6 to 11);
                  E1 <= "000000";
                  min <= NOT sw(6 to 11);
                  sekundy <= (OTHERS => '0');
            END IF;

            IF hertz='1' THEN
                  sekundy <= sekundy + 1;
                  sec <= sekundy;
                  A1 <= "000000";
                  min <= minuty;
                  hod <= hodiny;
```

```
                C1 <= "000000";
                E1 <= "000000";
                IF sekundy = "111011" THEN
                      sekundy <= "000000";
                      minuty <= minuty + 1;
                      IF minuty = "111011" THEN
                            minuty <= "000000";
                            hodiny <= hodiny + 1;
                            IF hodiny = "010111" THEN
                                  hodiny <= "000000";
                            END IF;
                      END IF;
                END IF;
          END IF;

          IF sec < "001010" then
                B1 <= sec;
          ELSIF sec >= "001010" then
                A1 <= A1 + "000001";
                sec <= sec - "001010";
          END IF;

          IF min < "001010" then
                D1 <= min;
          ELSIF min >= "001010" then
                C1 <= C1 + "000001";
                min <= min - "001010";
          END IF;

          IF hod < "001010" then
                F1 <= hod;
          ELSIF hod >= "001010" then
                E1 <= E1 + "000001";
                hod <= hod - "001010";
          END IF;

    END IF;
 END PROCESS;

     A<=A1; B<=B1; C<=C1; D<=D1; E<=E1; F<=F1;

END Behavioral;
```

B.4 Bitové operace – zdrojový kód

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

ENTITY operator IS
  PORT(
       A :        in std_logic_vector(3 downto 0);
       B :        in std_logic_vector(3 downto 0);
       tl1:       in std_logic;
       tl2:       in std_logic;
       tl3:       in std_logic;
       tl4:       in std_logic;
       tl5:       in std_logic;
       C_out:     out std_logic_vector(3 downto 0);
       A_out:     out std_logic_vector(3 downto 0);
       B_out:     out std_logic_vector(3 downto 0);
       khertz:    in std_logic
       );
      END operator;

ARCHITECTURE Behavioral OF operator IS
  SIGNAL C:       std_logic_vector(3 downto 0):="0000";
  SIGNAL A_in:    std_logic_vector(3 downto 0):="0000";
  SIGNAL B_in:    std_logic_vector(3 downto 0):="0000";

BEGIN
PROCESS (khertz, tl1, tl2, tl3, tl4, tl5) begin

      IF khertz'event AND khertz = '1' THEN
            A_in <= not A;
            B_in <= not B;

            IF tl1='1' then
                  C <= A_in and B_in;
            ELSIF tl2='1' then
                  C <= A_in or B_in;
            ELSIF tl3='1' then
                  C <= A_in xor B_in;
            ELSIF tl4='1' then
                  C <= A_in nor B_in;
            ELSIF tl5='1' then
                  C <= not A_in;
            END IF;
      END IF;
END PROCESS;

      C_out <= not C;
      A_out <= not A_in;
      B_out <= not B_in;

END Behavioral
```

B.5 Převod BIN to HEX – zdrojový kód

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

ENTITY disp IS
  PORT(clk50, kH, khertz:     IN  STD_LOGIC;
       sw:                    IN STD_LOGIC_VECTOR (11 DOWNTO 0);
       digit:                 OUT STD_LOGIC_VECTOR(5 DOWNTO 0);
       seg:                   OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
       diod:                  OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
       tl1, tl2, tl3, tl4, tl5:     IN STD_LOGIC
       );
      END disp;

ARCHITECTURE Behavioral OF disp IS
  SIGNAL cd:            STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";
  SIGNAL curr:          STD_LOGIC_VECTOR(3 DOWNTO 0) := "0000";
  SIGNAL dp:            STD_LOGIC := '0';
  SIGNAL bcdint:        STD_LOGIC_VECTOR(23 downto 0);
  SIGNAL sw1:           STD_LOGIC_VECTOR(11 downto 0):= "111111111111";
  SIGNAL tlacitko:      STD_LOGIC_VECTOR (4 downto 0):= "00000";

BEGIN

  PROCESS (clk50) BEGIN
       IF clk50'event AND clk50 = '1' THEN
            bcdint <= sw & sw1;
            IF khertz_en = '1' THEN
                  cd <= cd + 1;
                  IF cd = "101" THEN
                        cd <= "000";
                  END IF;
            END IF;

-- podle "ukazatele" cd je aktivovana prislusna pozice (aktivni v 0)
CASE cd IS   -- curr je zobrazena cifra, digit je pozice na displeji
WHEN "000" => curr <= bcdint(3 DOWNTO 0); digit <="111110"; dp <='0';
WHEN "001" => curr <= bcdint(7 DOWNTO 4); digit <="111101"; dp <='0';
WHEN "010" => curr <= bcdint(11 DOWNTO 8); digit <="111011"; dp <='0';
WHEN "011" => curr <= bcdint(15 DOWNTO 12); digit<="110111"; dp <='0';
WHEN "100" => curr <= bcdint(19 DOWNTO 16); digit<="101111"; dp <='0';
WHEN OTHERS => curr <= bcdint(23 DOWNTO 20);digit<="011111"; dp <='0';
END CASE ;

-- nastaveni prislusneho znaku na segmentovce podle BCD kodu na vstupu
CASE curr IS
WHEN "1111" => seg <= dp & "0111111";       -- 0
WHEN "1110" => seg <= dp & "0000110";       -- 1
WHEN "1101" => seg <= dp & "1011011";       -- 2
WHEN "1100" => seg <= dp & "1001111";       -- 3
WHEN "1011" => seg <= dp & "1100110";       -- 4
WHEN "1010" => seg <= dp & "1101101";       -- 5
WHEN "1001" => seg <= dp & "1111101";       -- 6
WHEN "1000" => seg <= dp & "0000111";       -- 7
WHEN "0111" => seg <= dp & "1111111";       -- 8
```

```
WHEN "0110" => seg <= dp & "1101111";       -- 9
WHEN "0101" => seg <= dp & "1110111";       -- A
WHEN "0100" => seg <= dp & "1111100";       -- B
WHEN "0011" => seg <= dp & "0111001";       -- C
WHEN "0010" => seg <= dp & "1011110";       -- D
WHEN "0001" => seg <= dp & "1111001";       -- E
WHEN OTHERS => seg <= dp & "1110001";        -- F
END CASE;

END IF;
END PROCESS;

END Behavioral;
```

B.6 Čítač – zdrojový kód

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

Entity Counter IS
  PORT(hertz, Smer, Areset, Sreset, Nacteni, CEn:IN std_logic;
       Din:   IN std_logic_vector(3 downto 0);
       Count: OUT std_logic_vector(3 downto 0)
       );
END Counter;

ARCHITECTURE CountArch OF Counter IS

  SIGNAL CntInt:    std_logic_vector(3 downto 0):= "1010";

BEGIN

  PROCESS (hertz, Areset) BEGIN
        IF Areset = '1' THEN
              CntInt <= "0000";
        ELSIF (hertz'event AND hertz = '1') THEN
                    IF Sreset = '1' THEN
                          CntInt <= "0000";
                    ELSIF Nacteni = '1' THEN
                          CntInt <= not Din;
                    ELSIF CEn = '1' THEN
                          IF Smer = '1' THEN
                                CntInt <= CntInt + 1;
                          ELSE
                                CntInt <= CntInt - 1;
                          END IF;
                    END IF;
        END IF;
  END PROCESS;

  Count <= not CntInt;

END CountArch;
```