ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

KATEDRA ŘÍDICÍ TECHNIKY

BAKALÁŘSKÁ PRÁCE

Základní úlohy s ALTERA DE2

Praha 2011 Martin Štěpánek

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze, dne 26. 5. 2011 ..... Martin Štěpánek .....

podpis

Anotace

Tato práce byla vytvořena pro seznámení s perifériemi VGA a LCD a s programovatelným hradlovým polem (FPGA), kterým se tyto periférie ovládají, na vývojové desce Altera DE2 pro předmět A0B35SPS (Struktury počítačových systémů). Studenti se učí s těmito periferiemi pomocí navržených úloh, které jsou vyřešeny v této práci.

Pro seznámení s prostředím Quarus 9.1 sp2 a se základními perifériemi byly úlohy zpracovány ve schematickém návrhu pomocí čítačů. Dalším stupněm řešení úloh bylo jejich zpracování v jazyce VHDL, který je dále také popisován.

Annotation

This work was create for introduction with peripherals VGA and LCD and with Field-programmable gate array (FPGA), which control these peripherals on the development board Altera DE2, for the subject A0B35SPS (Struktury počítačových systémů). Students learn with these peripherals using prepared tasks, which included in this work.

For introduction with the programming environment Quartus 9.1 sp2 and with basic peripherals have been designed in Schematic file using counters. The other tasks have been designed in the language VHDL, which is also described below.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Martin Štěpánek**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Základní úlohy s Altera DE2**

Pokyny pro vypracování:

1. Seznamte se s vývojovou FPGA deskou Altera DE2 od firmy Terasic. Bližší informace lze získat na stránkách výrobce http://www.terasic.com.tw/. Seznamte se s vývojovým prostředím Quartus II, Version 9.1 Servise Pack 2.
2. Seznamte se s programováním v jazyce VHDL a připravte základní návod pro použití VHDL jazyka ve formě PowerPointové prezentace.
3. Po dohodě s vedoucím práce navrhněte a realizujte základní úlohy pro předmět A0B35SPS Struktury počítačových systémů – (např. digitální hodiny, digitální stopky a digitální budík se zobrazováním na sedmisegmentových zobrazovačích a na LCD displeji, různé typy čítačů, snímání znaků pomocí PS/2 klávesnice atd.)
4. Vypracujte návod k jednotlivým úlohám ve formě webové prezentace včetně nezbytných dokumentů potřebných pro realizaci základních úloh.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Martin Hlinovský, Ph.D.

Platnost zadání: do konce zimního semestru 2011/2012

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 29. 10. 2010

Obsah

Seznam tabulek

Seznam obrázků

1. Úvod

Od doby kdy byl sestaven první počítač uběhlo mnoho let. Za tu dobu má za sebou vývoj počítačů nejen průběžné, ale i skokové změny, a to jak v jejich velikosti, tak i v možnostech připojitelných periférií. Při vytváření nových připojitelných komponent bylo zapotřebí vymyslet nová standardní rozhraní.

Tato práce nás seznámí s perifériemi VGA a LCD a s programovatelným hradlovým polem (FPGA), kterým se tyto periférie ovládají, a to ve třech částech, kde se v úvodní části seznámíme s deskou, na které jsou úlohy simulovány. V druhé části jsou navržené a zpracované tři úlohy. Složitost úloh se zvyšuje a to následovně:

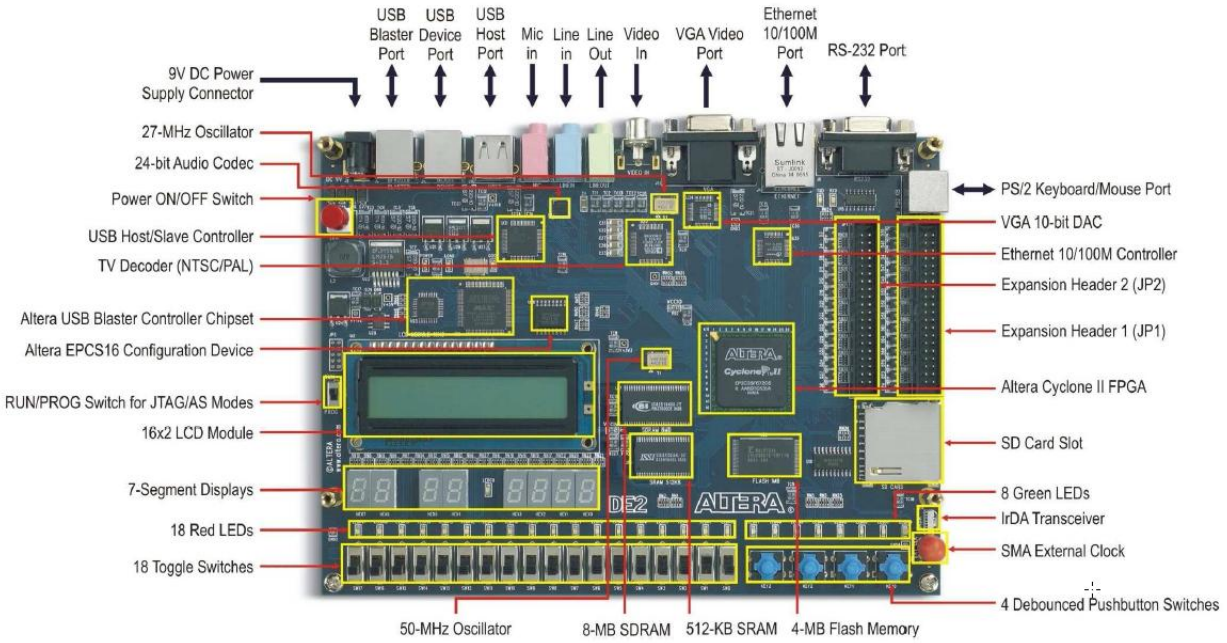
- v prvních dvou úlohách se naučíme obsluhovat jednoduché periferie (LED dioda, sedmisegment, tlačítko a „switch“) pomocí čítačů 7490 a 74193, přičemž v druhé úloze řešíme zadání i pomocí jazyka VHDL, kterým se bude programovat i zbylá úloha (VGA).
- ve třetí části vysvětlujeme syntaxi a pravidla zápisu v jazyce VHDL, které jsou dokumentovány na přiložených programech, resp. částech kódu.

Obr. 1. Vývojová deska Altera DE2

2. Altera DE2

Vývojová deska Altera DE2 od firmy Terasic Technologies Inc. je vhodná pro všechny, kteří se chtějí naučit s digitální logikou, FPGA a počítačovými periferiemi. Tato deska obsahuje mnoho periférií a zařízení (jak je vidět na obr. 2) a další zařízení se k ní dají připojit přes rozšiřující periferie. Mezi další výhody patří možnost simulace vytvořeného programu přímo v prostředí Quartus. Zde uživatel vidí jestli program pracuje tak, jak ho navrhl. Nesmíme také zapomenout na návody, které výrobce k desce a programovacímu prostředí nabízí a rozsáhlé internetové fórum [10].

Jak již bylo zmíněno, výhodou samotné desky je množství periférií, které se na ní nacházejí. To nám umožňuje vytvářet malé a jednoduché programy, ale i mnohem složitější programy, u kterých můžeme obsluhovat více periférií najednou, jak je vidět v programech, které jsou na CD (toto CD je dodávané spolu s deskou), nebo na “defaultním“ programu, který se spustí při zapnutí desky. Než začneme vytvářet svůj první program, můžeme se s deskou seznámit pomocí programu “Control Panel Setup“, tento program se také nachází na daném CD, pomocí kterého si můžeme vyzkoušet jak jednotlivá zařízení na desce pracují.



Obr. 2. Popis desky

Seznam zařízení na desce Altera DE2:

- Altera Cyclone® II 2C35 FPGA device
- Altera Serial Configuration device - EPCS16
- USB Blaster (na desce) pro programování a ovládání uživatelského rozhraní API (JTAG a Active Seriál)
- 512-Kbyte SRAM
- 8-Mbyte SDRAM
- 4-Mbyte Flash paměť
- SD Card socket
- 4 tlačítka
- 18 přepínačů
- 18 červených LED diod
- 9 zelených LED diod
- 50-MHz oscilátor a 27-MHz oscilátor pro hodinový zdroj
- 24-bit CD-quality audio CODEC s line-in, line-out a vstupem pro mikrofon
- VGA DAC (10-bit high-speed triple DACs) s výstupním VGA konektorem
- TV Decoder (NTSC/PAL) a TV-in konektor
- 10/100 Ethernet Controller with a connector
- USB Host/Slave zařízení s USB konektory typu A a typu B
- RS-232 s 9-pinovým konektorem
- PS/2 konektor pro myš/klávesnice
- IrDA vysílač
- dva 40-pinové rozšiřující konektory, ke kterým můžeme připojit další zařízení

3. Návrh světelného hada pomocí čítačů 7490

3.1. Zadání

Pomocí čítačů 7490, vytvořte 4-bitového světelného hada, který se bude pohybovat danou rychlostí. Pohyb bude zobrazen na **LED_0 – LED_15**. Had bude mít několik druhů pohybu:

a) Had se bude pohybovat vzestupně a pomocí *switchů* se nastaví jeho velikost. Velikost bude určena nejvyšším spuštěným *switchem*. Pokud nebude spuštěn žádný *switch*, pak bude jeho velikost maximální.

b) Obdobně jako v bodě a). S rozdílem ve směru pohybu. Had se bude pohybovat sestupným směrem.

c) Po dosažení konce se had začne pohybovat opačným směrem.

Každou úlohu vytvořte zvlášť.

3.2. Rozbor úlohy

Abychom mohli pomocí schematického popisu vytvořit světelného hada, musíme si na začátku definovat vstupy a výstupy, kterými se bude program obsluhovat. Jako vstupní hodnoty použijeme hodinový vstup na desce *clock_50* (pro pohyb hada) a *switche,* které určují velikost hada. Pro výstup využijeme LED diody podle zadání.

Jako další krok úlohy si určíme z jakých prvků obvod složíme: čítače 7490, dekodér, kodér a logické obvody. Čítače využijeme jako děličku vstupních hodin z desky na požadovanou hodnotu hodinového signálu pro pohyb hada, který určuje jeho rychlost. Druhé využití čítače je na posuv světelné značky na LED diodách. Pomocí dekodéru převedeme binární kód z čítače (pozice hada) na adresy 1 z N, na jejichž koncích budou připojeny LED diody. Kodér bude sloužit pro převedení omezení velikosti hada ze *switchů*. Logické obvody slouží pro vytváření funkcí tak, aby obvod pracoval podle zadání.

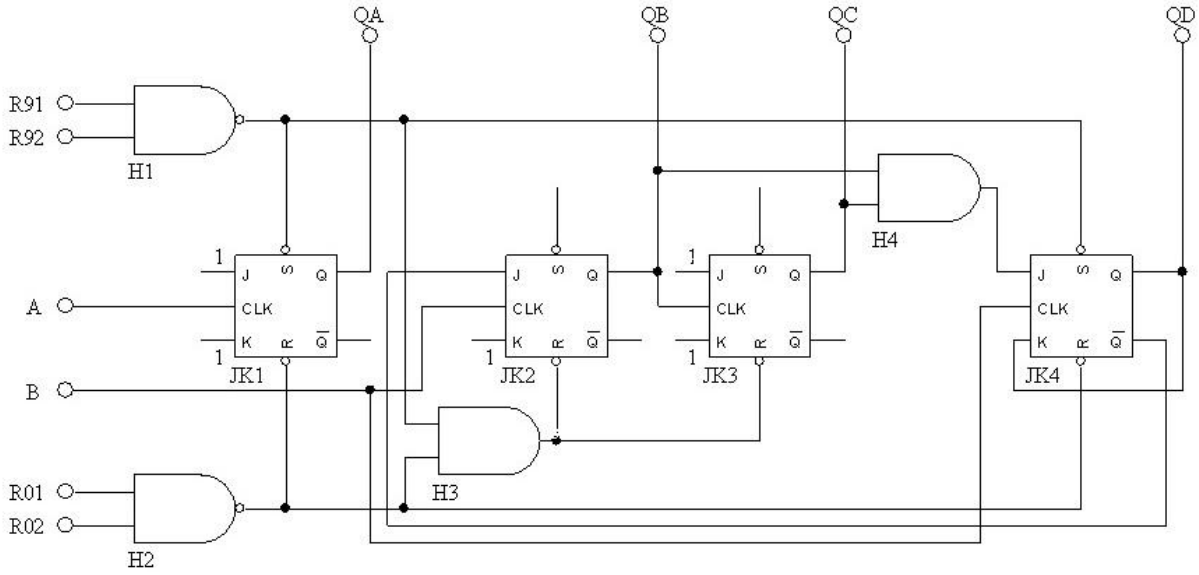
3.3. Použité obvody

V této části jsou popsány jednotlivé obvody, které jsou využity v návrhu.

3.3.1. Čítač 7490

Integrovaný obvod 7490 je navržen jako asynchronní dekadický čítač, který čítá na sestupnou hranu hodinového signálu. Obvod je složen ze čtyř bistabilních klopných obvodů JK, který je rozdělen na dvě části - viz Obr. 3**. Vnitřní schéma čítače 7490**. První klopný obvod (se vstupem „A“) je realizován jako dělička 2. Obvod se vstupem „B“ je zapojen jako dělička 5. Pokud se propojí obě části dohromady (QA=>B nebo QD=>A), tak potom čítač koná funkci děličky 10. Čítač je také opatřen hradly, které jeho výstupy nastaví buď do hodnoty b“0000“ (značí binární hodnotu 0000) nebo b“1001“/“1100“ (v závislosti na zapojení A=>B/B=>A), a to se děje pomocí vstupu R0 nebo R9, které jsou aktivovány nastavením do logické 1.

Tento obvod je definován ve vývojovém prostředí Quartus. Je však navržen podle fyzického zapojení špatně. Místo asynchronně zapojených obvodů JK, je obvod složen ze synchronních obvodů D, což znamená, jsou-li čítače 7490 zapojeny do série, pak první blok čítá 0 – 9, ale ostatní v prvním cyklu čítají jako první blok a pak už jen 1 – 9. Tato chyba dělá velké problémy, a to jak pro nastavení požadovaného kmitočtu, tak i pro posuv světla. Proto jsme použili doporučený a mnou vytvořený obvod 7490MS , který odpovídá reálnému zapojení 7490 podle Obr. 3**. Vnitřní schéma čítače 7490**.



Obr. 3. Vnitřní schéma čítače 7490

3.3.2. Dekodér

Dekodér je kombinační logický obvod, který ze vstupních dat v určitém kódu vytváří na výstupu jiná, často jednodušší data. Funkce dekodéru je inverzní k funkci kodéru, který je v úloze také použit. V tomto případě použijeme zapojení binárního dekodéru (kodéru), který převádí vstupní signál o *n* bitech na 2^n výstupů - viz obr. 4. V tomto případě jsou na výstupy

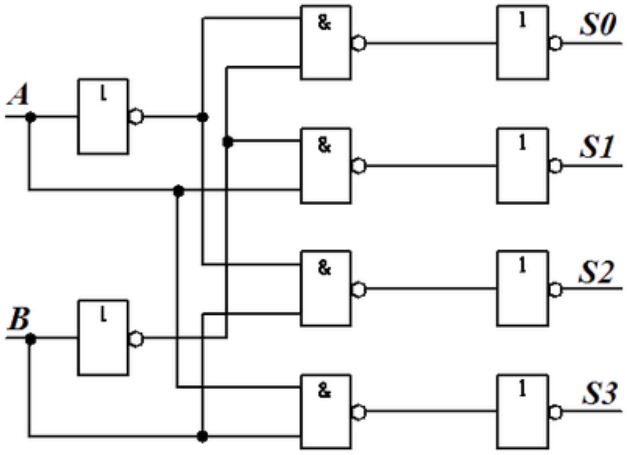
připojeny LED diody, které zobrazují pohyb hada. Pro kodér jsou na vstupy připojeny *switche*, které určují velikost hada.

V naší úloze jsou možné dva způsoby řešení, jak takový dekodér zhotovit. Buď pomocí funkcí složených z logických obvodů, jako na obr. 4 (tyto funkce nám vyjdou z Karnaughových map [1]), nebo pomocí dekodéru, který je v Quartusu označen jako *demultiplexor*, pro kodér je zde obvod označený jako 74147. Jelikož zadání přímo neurčuje, jaký dekodér a kodér má být použit, vybereme si *demultiplexor* a obvod 74147 z důvodu přehlednosti úlohy, proto se již dále nebudeme zabývat Karnaughovými mapami.

B	A	S0	S1	S2	S3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

$S0 = \overline{B} \cdot \overline{A}$ $S1 = \overline{B} \cdot A$

$S2 = B \cdot \overline{A}$ $S3 = B \cdot A$



Obr. 4. Binární dekodér s pravdivostní tabulkou

3.3.3. Logické obvody

Logický (digitální) obvod je elektronický obvod, který pracuje s diskrétními stavy. Jednotlivé obvody jsou složeny z analogových součástek, ale pracují ve spínacích módech. To umožňuje zacházet s obvody tak, jako by byly diskrétní.

Logické obvody se dělí na kombinační logické obvody a sekvenční logické obvody. Pro kombinační obvody je charakteristické to, že na jistou kombinaci hodnot vstupních signálů reagují vždy stejnou kombinací hodnot výstupních signálů, a to bez ohledu na minulé hodnoty signálů. Kombinace hodnot signálů se nazývá stav. Sekvenční obvody se liší od obvodů kombinačních tím, že na jednu a tutéž kombinaci hodnot vstupních signálů může obvod reagovat pokaždé jinak. Výstupní signály sekvenčních obvodu jsou tedy závislé nejen na vstupních signálech v daném okamžiku, ale též na vstupních signálech předchozích – jinými slovy – sekvenční obvod má vnitřní paměť. V předmětném zadání použijeme první typ obvodů a to ve variantách: logický součin, logický součet, logická neekvivalence (exkluzivní logický součet)a logická negace, a jeden sekvenční obvod: klopný obvod JK.

		AND	NAND	OR	NOR	XOR
A	B	A . B	$\overline{A . B}$	A + B	$\overline{A + B}$	A ⊕ B
0	0	0	1	0	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	0	1	0	0

Tab. 1. Pravdivostní tabulka kombinačních logických obvodů

3.4. Realizace úlohy

Pro všechny úlohy použijeme stejnou děličku, která bude převádět vstupní hodinový signál *clock_50* na požadovanou hodnotu (rychlost pohybu hada). Ta se bude skládat z čítačů 7490 zapojených do série - viz příloha A.

Úlohy ad a) a ad b) budou zpracovány obdobně. Musíme však zohlednit směr pohybu hada (vzestupně / sestupně) a podle toho program upravit. Dále musíme vyřešit problém dekadického kodéru, na kterém se bude zadávat velikost hada, jelikož potřebujeme ovládat více vstupů (vzestupně *switch_0* – *switch_14* nebo sestupně *switch_15* – *switch_1*). Tento problém vyřešíme pomocí dvou integrovaných obvodů 74147 a deseti logických obvodů tak, aby vznikl 4-bitový kodér. Následně sestavíme obvod, který bude porovnávat signál z tohoto kodéru a signál přiváděný do dekodérů, který udává pozici hada na LED diodách. Poté přivedeme tento výsledný signál na resetovací vstup čítačů 7490 (na pohyb hada).

U úlohy ad c) musíme vyřešit zadání tak, aby čítač čítal vzestupně a po dosažení *LED_15* čítal sestupně a to až na *LED_0* a zajistit aby se celý cyklus opakoval. Jako nejlepší řešení uvedeného vybereme způsob, kdy budeme čítat pouze vzestupně a po dosažení LED_15 budeme výstupy negovat. U tohoto způsobu zpracování úlohy musíme tedy dořešit aby se had v krajních bodech nezastavil (př.: po *LED_15* nastane znovu *LED_15* a to po negaci vstupní hodnoty, která je rovna 0). Tato nepravidelnost se odstraní tak, že čítač omezíme na čítání od 0 do 14 a pomocí klopného obvodu JK zajistíme vyhodnocování směru čítání (vzestupné / sestupné čítání). Signál z tohoto obvodu spojíme se vzestupným čítačem pomocí logických obvodů tak, aby vysílal požadované hodnoty do *demultiplexoru*, který má své výstupy připojeny na LED diody.

4. Sestavení časovacích obvodů z čítačů a návrh v jazyce VHDL

4.1. Zadání

Sestavte tyto časovací obvody pomocí čítačů 7490 nebo 74193 ve schematickém popisu nebo v jazyce VHDL.

a) Realizujte s využitím sedmi segmentových zobrazovačů digitální stopky s rozsahem měření 60 minut a s přesností setiny sekundy. Digitální stopky budou mít vstupy START/STOP a NULOVÁNÍ (jako vstupy využijte *pushbutton* tlačítka). Na pozicích **HEX7, HEX6** se budou zobrazovat minuty, na pozicích **HEX5, HEX4** se budou zobrazovat sekundy, na pozicích **HEX3, HEX2** se budou zobrazovat setiny sekundy a pozice **HEX1** a **HEX0** budou zhasnuté.

b) S využitím LCD displeje realizujte digitální hodiny se zobrazením aktuálního času tak, že po stisku definovaného tlačítka pushbutton se po dobu jeho stisku zobrazí na displeji čas budíku. Oba časy budou nastavovány pomocí *switchů*. Alarm zobrazte na LED diodách a jeho vypnutí obslužte tlačítkem, které jej nastavuje.

c) Vytvořte elektronickou minutku 0 – 999s. Čas se bude zadávat pomocí *switchů*. Čas bude zobrazen na **HEX3 – HEX1**. Signalizace uplynutí zadaného času, bude zobrazena na LED diodách.

Každou úlohu vytvořte zvlášť.

4.2. Rozbor úlohy

Dříve než začneme úlohu zpracovávat je zapotřebí si nadefinovat veškeré vstupy a výstupy, které budou použity. Jako první musíme nastavit vstup *clock*_50, který pro všechny úlohy bude zajišťovat hodinový vstup. Tyto hodiny nám budou určovat časová kvanta pro jednotlivé úlohy (vteřiny a setiny). Pro ovládání stopek nastavíme pushbutton tlačítka *key1* a *key2.* Tyto vstupy použijeme i v ostatních úlohách. V úlohách ad b) a ad c) nastavujeme čas. Ten budeme nastavovat pomocí *switchů*. V poslední řadě nesmíme zapomenout na výstupy úloh. Pro úlohy ad a) a ad c) nastavíme sedmi segmenty podle zadání. K zadání ad b) si najdeme všechny porty, které obsluhují LCD displej [7] a to jsou: *LCD_RW, LCD_EN, LCD_RS,*

LCD_DATA[0 - 7] a *LCD_ON*. Veškeré vstupy budou vysvětleny v kapitole 4.4.2.

Pokud budeme dělat úlohy ve schematickém popisu, musíme si určit obvody, z kterých jednotlivé úlohy složíme: čítače 7490 a 74193, převodník BCD kódu 7447 a logické obvody. Pomocí čítačů 7490 nastavíme vstupní hodiny z 50MHz na požadovanou hodnotu a dále budou konat funkci určování času v úloze stopky a na pozici vteřin u hodin. Zbylé pozice časovacích obvodů, v zadáních ad b) a ad c), bude zastávat čítač 74193.

4.3. Použité obvody

Tato část slouží pouze pro popis jednotlivých obvodů ve schematickém zpracování. Jazyk VHDL bude vysvětlen v kapitole 6.

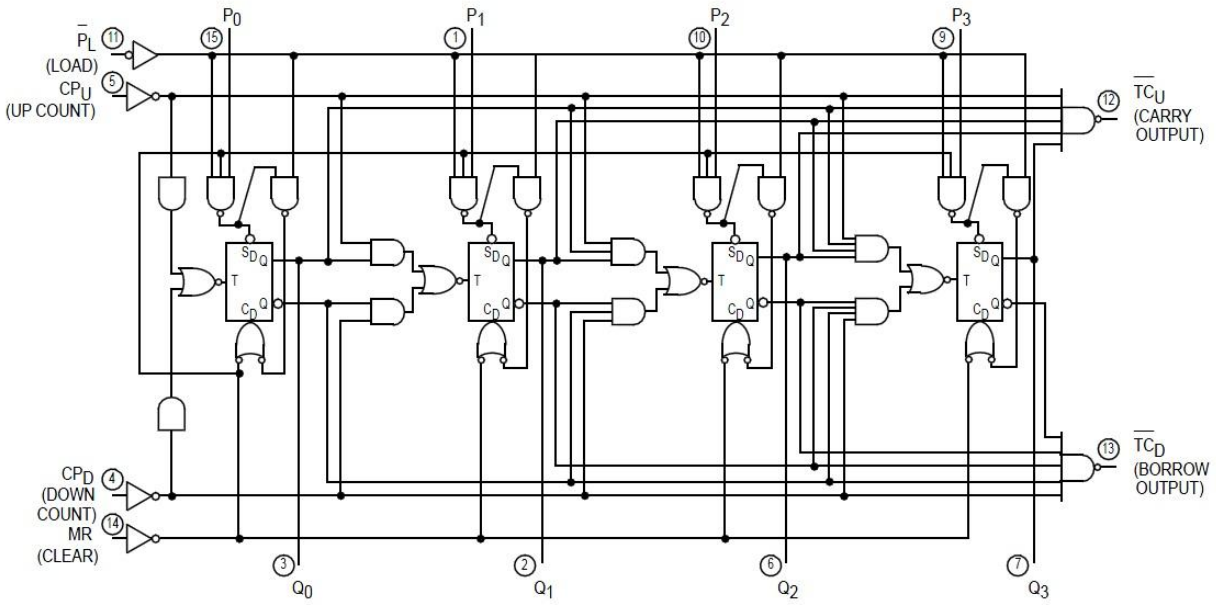
4.3.1. Čítač 7490

Viz kapitola 3.3.1.

4.3.2. Čítač 74193

Integrovaný čítač 74193 je binární synchronní obousměrný čítač. Který je sestaven ze čtyř dvojitých klopných obvodů a z kombinační sítě logických členů zajišťující chod čítače. Obvod je opatřen dvěma hodinovými vstupy (vpřed a vzad). Uplatňuje se vždy ten vstup, na který jsou přiváděny impulsy, zatímco na druhém je úroveň logická 1. Čítač má programovatelný obsah. Jeho výstupy mohou být nastaveny na potřebný výchozí stav prostřednictvím vstupů P0 – P3 viz obr. 5. Aby se zápis realizoval, je nutno na tyto vstupy přivést žádané úrovně a poté se uvede vstup *load*, na úroveň logická 0 (impuls). Obsah čítače je možno také vynulovat a to pomocí asynchronního vstupu *clear*. Čítače je možno pohodlně zařadit do kaskád. K tomu slouží výstupy pro přetečení *carry* a *borrow* (přetečení / podtečení).

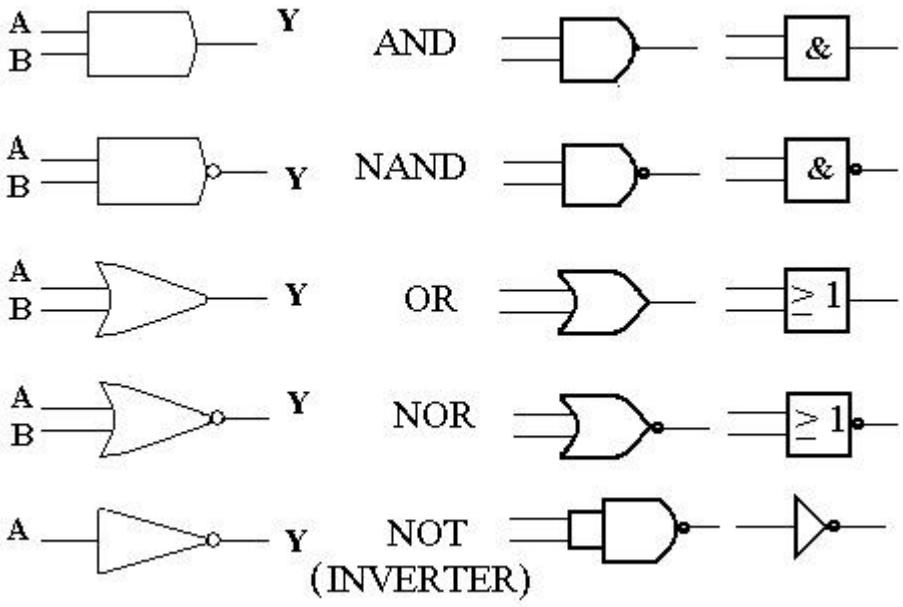
Ve vývojovém prostředí Quartus je tento obvod definován ve správném fyzickém zapojení – viz obr. 5. Proto tento čítač můžeme bez úprav rovnou použít.



Obr. 5. Vnitřní schéma čítače 74193

4.3.3. Logické obvody

Viz kapitola 3.3.3.



Obr. 6. Značky logických obvodů

4.3.4. Převodník 7447

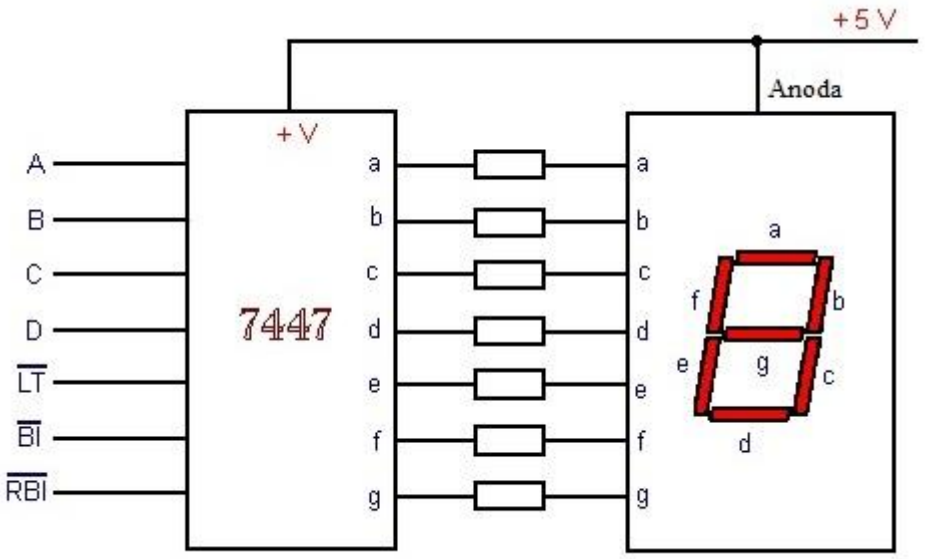
Tento obvod je další typ dekodéru, který používá jako vstup číslo v BCD kódu a výstupem je sedm signálů v sedmi segmentovém kódu. Dekodér je určen pro buzení

zobrazovacích jednotek tvořených sedmi segmentovými displeji. Jeho výstupní stavy rozsvěcují některé ze sedmi světelných segmentů tak, aby tvořily dekadické číslice od 0 do 9. Zobrazovací displeje jsou tvořeny svítivými diodami, uspořádanými v LED jednotce, které mají rozmístěny jednotlivé segmenty do tvaru znázorněném na obrázku 7. Při návrhu zobrazovacího systému vycházíme z pravdivostní tabulky, která je pro sedmi segmentový dekodér uvedena v tabulce 2.

Přivedením logické 0 na vstup LT se rozsvítí celý displej (svítí číslice 8). Pokud přivedeme logickou 0 na RBO, pak displej zůstane zhasnutý. Jestliže chceme snížit spotřebu proudu nebo zvýšit čitelnost displeje, tak k tomu použijeme poslední vstup RBI, který potlačí nevýznamnou 0 na dané segmentu (př.: místo 05,03200 zobrazí 5,032).

Dekadická hodnota	Vstup BCD				Výstup segmentů						
	D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	0	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	0	0

Tab. 2. Převodní tabulka z BCD na sedmi segment



Obr. 7. Propojení dekodéru 7447 se sedmi segmentem

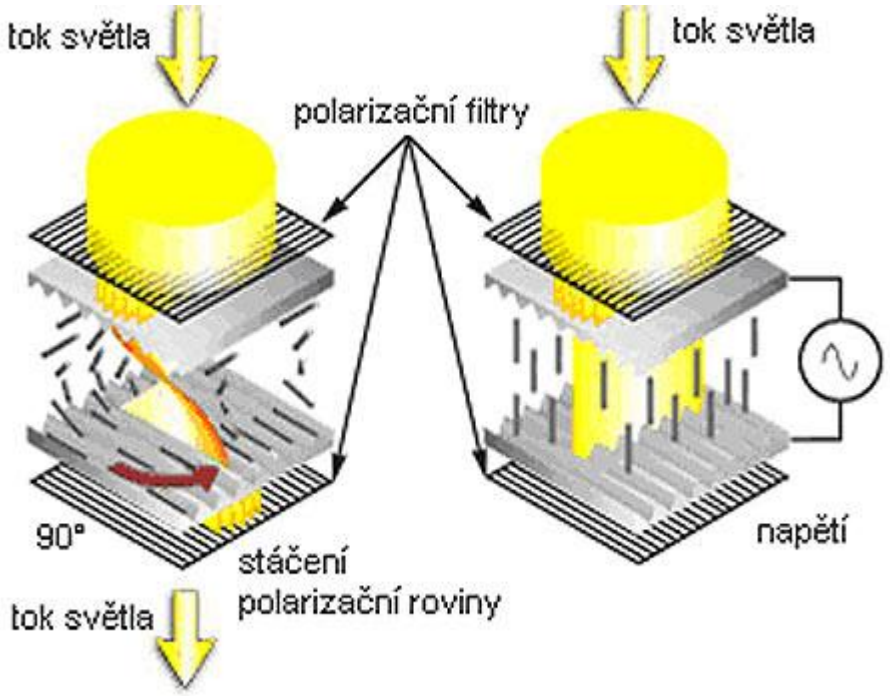
4.3.5. Sedmi segment

Jak už bylo zmíněno v kapitole 4.3.4 jedná se o zobrazovací displej. Displej je v zapojení se společnou anodou nebo katodou. V prvním případě jsou spojeny anody diodových segmentů, které se připojí na kladné napájecí napětí +Ucc. Jednotlivé segmenty rozsvěcujeme spojením katod diodových segmentů s logickou 0. Tento typ sedmi segmentu se nachází na vývojové desce Altera DE2. Displeje se společnou katodou fungují opačným způsobem. Místo -Ucc je většinou použit potenciál země.

4.3.6. LCD displej

Displej z tekutých krystalů (LCD) je tenké a ploché zobrazovací zařízení skládající se z omezeného (velikostí monitoru) počtu barevných nebo monochromatických pixelů seřazených před zdrojem světla nebo reflektorem. Každý pixel LCD se skládá z molekul tekutých krystalů uložených mezi dvěma polarizačními filtry, přičemž osy polarizace jsou na sebe kolmé. Bez krystalů mezi filtry by bylo světlo, procházející jedním filtrem blokováno druhým filtrem. Molekuly tekutých krystalů jsou bez vnějšího elektrického pole srovnány do spirálové struktury a stáčí polarizaci procházejícího světla o 90 stupňů, což světlu umožňuje projít i druhým filtrem. V okamžiku vzniku pole jsou molekuly tekutých krystalů taženy rovnoběžně s elektrickým polem, což snižuje rotaci vstupujícího světla. Pokud nejsou tekuté krystaly vůbec stočené, pak bude procházející světlo polarizováno kolmo k druhému filtru, bude tudíž úplně blokováno a daný pixel se bude jevit jako nerozsvícený. Pomocí ovlivnění stočení krystalů v pixelu lze kontrolovat množství procházejícího světla, a tudíž i celkovou svítivost pixelu.

V naší úloze bude využit monochromatický displej o velikosti 2x16 znaků.



Obr. 8. Průchod světla v LCD

4.4. Realizace úlohy

Jelikož zadání můžeme řešit dvěma odlišnými způsoby a to buď pomocí jednotlivých bloků ve schematickém popisu, nebo v programovacím jazyce VHDL, bude i popis zpracovaných úloh rozdělen dvě části.

4.4.1. Zpracování ve schematickém popisu

Pro všechny úlohy použijeme podobnou děličku, která bude převádět vstupní hodinový signál *clock_50* na požadovanou hodnotu (nejmenší časový úsek pro „jednotlivé hodiny“). Tuto děličku sestavíme z čítačů 7490, sestavených do série.

V úloze ad a) sestavíme časomíru jen pomocí čítačů 7490. Při stavění obvodů do série si musíme dát pozor na pozici desítek u vteřin a minut. To vyřešíme pomocí hradla AND, který bude čítač nulovat po zobrazení hodnoty 6 na jeho výstupu. Výstup z jednotlivých čítačů přivedeme na převodníky BCD kódu 7447. Výstup z převodníku přivedeme na jednotlivé segmenty tak, jak je znázorněno na obr. 7. K celému obvodu musíme ještě připojit dvě tlačítka, která jej budou ovládat podle zadání.

Úlohu ad b) nelze přesně splnit podle zadání, protože se při skládání složitějších funkcí (inicializace displeje) mohou jednotlivé bloky čítačů chovat překvapivě v důsledku nepřesného naprogramování funkcí čítačů v programu Quartus. Protože úloha musí být implementována ve schematickém popisu, musí se pozměnit zadání na: Čas bude zobrazen na sedmi segmentech a alarm se nebude implementovat. S doporučením na zpracování v jazyce VHDL.

Při skládání časových obvodů dáme na pozici vteřin obvody 7490. Na další pozice dáme již obvody 74193 kvůli nastavení hodin (pozice vteřin se nastaví na 00). Na nastavovací vstupy přivede jednotlivé *switche.* A na vstupy LDN přivedeme tlačítko pro načtení času (u 7490 je to R0). Jelikož je čítač 4-bitový musíme pomocí logických obvodů omezit jeho velikost, aby ukazoval správné hodnoty. Pak už jen propojíme výstupy z čítačů se vstupy převodníků a jeho výstupy přivedeme na sedmi segmenty.

U úlohy ad c) není omezení, kvůli kterému bychom museli žádat o změnu zadání. Musíme pouze využít jiný čítač (místo čítače 74193 použijeme čítač 74190, který pracuje obdobně jako čítač 74193 s výjimkou toho, že čítač 74190 je dekadický), jelikož nám ten původní způsoboval problémy, které by se v reálném zapojení nevyskytly.

Veškeré polohy času obsloužíme integrovaným obvodem 74190 na jehož vstupy přivedeme *switche*, kterými se nastavuje čas a na vstupy LDN připojíme tlačítko, kterým nastavíme zadaný čas ze *switchů*. Zobrazení na sedmi segmentech bude obdobné jako v úlohách ad a) a ad b). Aby se, po zobrazení času „000“ časomíra zastavila a začaly blikat LED diody musíme, pomocí logických obvodů, sestavit vyhodnocovací obvod, který tento požadavek zajistí. Nesmíme však přitom zapomenout na vytvoření spouštěcího tlačítka.

4.4.2. Zpracování v jazyce VHDL

Jako první si napíšeme program děličky, který budeme moci použít pro všechna zadání jako vstupní hodinový signál, jen budeme muset změnit dělící poměr.

Úlohy ad a) a ad c) můžeme vypracovat současně, jelikož jsou podobné. Pro první úlohu - ad a) - si vytvoříme čítací programy o velikosti 6 a velikosti 10. Pro druhou úlohu - ad c) - změníme pouze ten program z ad a), který čítal původně do 10. Tuto změnu jsme realizovali tak, že jeho výstup byl nastaven sestupně – ve směru od 10 do 0 s tím, že stejný systém odčítání je zajištěn od maximálního nastavení “999“ do stavu “000“ (po propojení čítačů), které jsme nastavili pomocí *switchů*. V této poloze se program zastaví a spustí alarm, kterým jsou blikající LED diody. Opětovné spuštění systému je možné až po dalším nastavení časového údaje na sedmi segmentu a to v intervalu “001 - 999“.

Výstup z těchto programů bude ve tvaru přímo pro 7-segment. Tyto výstupy budou přiřazeny k příslušným pozicím na časomíře. K oběma úlohám jsme naprogramovali dvě

tlačítka, kterými se programy obsluhují podle zadání.

V zadání ad b) si naprogramujeme ještě jednu děličku, která nám bude udávat takt pro zadávání parametrů do LCD. Čítací programy budou umět nastavit čas ze *switchů* po stisknutí tlačítka „SET“. Výstupní hodnoty těchto čítačů budou v ASCII-tvaru, jelikož LCD umí pracovat pouze s tímto formátem.

Displej zapneme vstupním pinem *LCD_ON* nastaveným do logické 1. Pomocí vstupu *LCD_RW* určujeme, jestli zapisujeme nebo čteme z tohoto zařízení. Pro naší úlohu bude tento vstup nastaven pouze na zápis. Vstupem *LCD_RS*, nastavujeme zda-li chceme pracovat s daty nebo nastavovat parametry displeje. *LCD_DATA[0 – 7]* jsou datové vstupy / výstupy podle nastavení „RW“. Přivedením pulsu na vstup *LCD_EN*, řekneme LCD, aby provedlo danou operaci.

Po zapnutí LCD musíme provézt tzv. inicializaci, která je přesně popsána v [7], kde nastavíme veškeré parametry displeje. Poté už můžeme zapisovat znaky na displej podle [7]. V zadané úloze se bude displej obnovovat po každé změně času.

Character located	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DDRAM address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
DDRAM address	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Obr. 9. Adresy znaků pro LCD 2x16 znaků

5. VGA rozhraní vytvořené v jazyce VHDL

5.1. Zadání

Pomocí VGA vykreslete vlajku.

a) Lehčí varianta: Arménie, Belgie, Čad, Estonsko, Irsko, Itálie, Maďarsko, Mauricius, Německo, Peru, Polsko, Rakousko, San Marino, Senegal, Sierra Leone nebo Vatikán
b) Těžší varianta: Bosna a Hercegovina, Česká republika, Eritrea, Island, Jamajka, Jordánsko, Kongo, Kuba, Kuvajt, Norsko, Skotsko, Spojené státy americké, Středoafrická republika, Švýcarsko nebo Tanzanie

U vlajek vykreslujte pouze barevná pole, ne znaky a ne hvězdy. Cvičící každému určí jednu vlajku z obou variant.

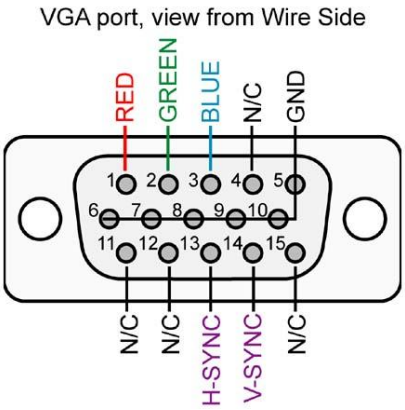
5.2. Rozbor úlohy

Jelikož v zadané úloze pracujeme pouze s jednou periférií a tou je VGA, které je obsluhováno integrovaným obvodem ADV7123 [8]. Jsou vstupy a výstupy zadání jasně definovány na: *VGA_R[0 - 9], VGA_G[0 - 9], VGA_B[0 - 9], VGA_CLK, VGA_BLANK, VGA_HS, VGA_VS* a *VGA_SYNC.* Jak se dozvíme v kapitole 5.4 musíme přivést do integrovaného obvodu vstupní hodinový signál *CLOCK_50*, který bude určovat rychlost vysílaných pixelů.

5.3. VGA

VGA, někdy též označováno jako D-SUB, je počítačový standard pro počítačovou zobrazovací techniku, vydaný roku 1987 společností IBM. Jedná se o analogovou periférii, která pomocí 3 pinů vysílá barvu daného pixelu (modrá, červená a zelená v bitových hloubkách – pro naše zadání se jedná o 10-bitové hloubky). Dále obsahuje dva řídící piny, které zajišťují vertikální a horizontální synchronizaci podle, kterých monitor určuje kde skončil řádek a kde

daný „frame“. Ostatní piny na desce Altera DE2 nejsou zapojeny.



Obr. 10. Rozložení pinů VGA konektoru na desce Altera DE2

5.4. Nastavení VGA a vykreslení dané vlajky

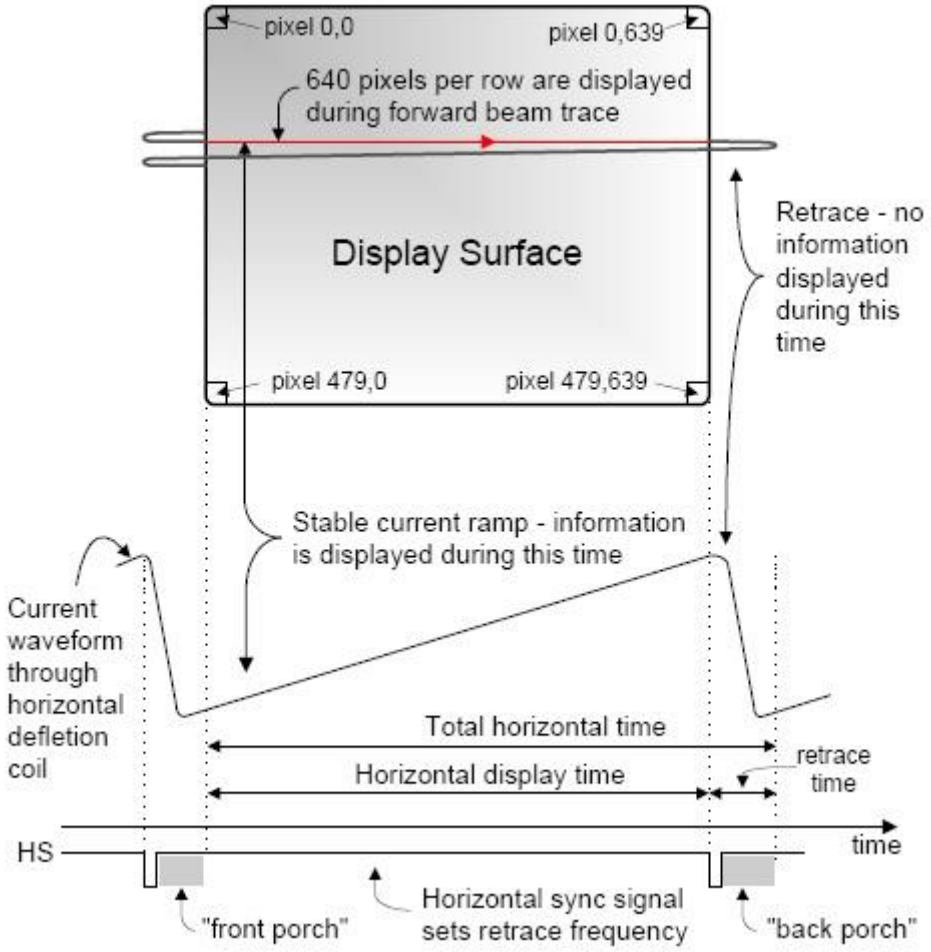
Než začneme úlohu řešit, musíme si určit jakou budeme mít frekvenci snímků a jak velké bude jejich rozlišení. Kvůli vertikálnímu a horizontálnímu časování - jak je vidět v tabulkách 3 a 4 a na obrázku 11 – má vertikální synchronizace stejný typ průběhu jako horizontální, ale jiný časový průběh, a to v závislosti na uvedených parametrech (a - d). Jelikož zadání neurčuje jaké rozlišení máme použít, vybereme si hned to první podle přiložených tabulek, a tím je VGA 60Hz. Frekvenci 25MHz vytvoříme nejsnadněji vydělením vstupních hodin (50MHz) dvěma. Pro ostatní frekvence bychom si museli vytvořit fázový závěs s využitím MegaWizard Plug-In Manageru. Tato frekvence určuje rychlost vysílaných pixelů.

VGA mode		Horizontal Timing Spec				
Configuration	Resolution HxV	a (us)	b (us)	c (us)	d (us)	Pixel clock (MHz)
VGA(60Hz)	640x480	3,8	1,9	25,4	0,6	25
VGA(85Hz)	640x480	1,6	2,2	17,8	1,6	36
SVGA(60Hz)	800x600	3,2	2,2	20,0	1,0	40
SVGA(75Hz)	800x600	1,6	3,2	16,2	0,3	49
SVGA(85Hz)	800x600	1,1	2,7	14,2	0,6	56
XGA(60Hz)	1024x768	2,1	2,5	15,8	0,4	65
XGA(70Hz)	1024x768	1,8	1,9	13,7	0,3	75
XGA(85Hz)	1024x768	1,0	2,2	10,8	0,5	95
1280x1240(60Hz)	1280x1240	1,0	2,3	11,9	0,4	108

Tab. 3. Časové konstanty pro horizontální synchronizaci

VGA mode		Vertical Timing Spec			
Configuration	Resolution HxV	a (lines)	b (lines)	c (lines)	d (lines)
VGA(60Hz)	640x480	2	33	480	10
VGA(85Hz)	640x480	3	25	480	1
SVGA(60Hz)	800x600	4	23	600	1
SVGA(75Hz)	800x600	3	21	600	1
SVGA(85Hz)	800x600	3	27	600	1
XGA(60Hz)	1024x768	6	29	768	3
XGA(70Hz)	1024x768	6	29	768	3
XGA(85Hz)	1024x768	3	36	768	1
1280x1240(60Hz)	1280x1240	3	36	1024	1

Tab. 4. Časové konstanty pro vertikální synchronizaci



Obr. 11. Průběh horizontální synchronizace

(synchronizace – a, back porch – b, displey time – c, front porch - d)

Při programování průběhů obou signálů musíme zajistit, aby při synchronizaci na monitoru (*LCD_HS a LCD_VS*) jsme spouštěli synchronizaci i do integrovaného obvodu na vstupu *VGA_SYNC*, který vysílá příslušný signál do monitoru. Po dobu synchronizace a

„zakrývacích signálů“ nesmíme vysílat žádnou informaci o barvě, to řešíme nastavením *VGA_BLANK* do logické 0.

V programu si musíme dát pozor na ideální rozložení čítacích cyklů, aby se obraz dlouho nesynchronizoval v monitoru, který by jej vykreslil až po delší době (př.: prodleva 10 vteřin místo 1,5 vteřiny). V závislosti na „chytrosti“ monitorů by obraz nemusel být vykreslen vůbec.

Obr. 12. Seznam použitých vlajek mimo Skotska

6. Jazyk VHDL

V této části si vysvětlíme jak se programuje v jazyce VHDL a jak mohou být použity jednotlivé příkazy. Nebudeme se ovšem zabývat vším, to popisuje specializovaná literatura [1,2], ale pouze tím, co využijeme nebo bychom mohli použít (v přiložené prezentaci na CD bude však mnohem víc). Ke každé kapitole je přiřazen obrázek, který ukazuje co bylo v kapitole popsáno.

6.1. Syntaxe v jazyce VHDL

Dříve než začneme s jakýmkoliv popisem jazyku VHDL, si musíme vysvětlit co je v daném jazyce přípustné a co nikoliv.

Jazyk VHDL není Case sensitive, a to jak pro příkazy, tak i pro proměnné (*mojePromena* nebo *mojePROMENA* je jedno označení pro proměnnou *mojepromena*). Při psaní programu nesmíme používat háčky a čárky v názvu proměnných a komentářích.

Komentáře se uvozují znakem dvojité pomlčky (‘--‘), komentář může začít kdekoliv na řádku a končí s koncem řádku. Bohužel neexistuje žádný znak pro hromadný komentář, a proto musíme souvislejší část programu, kterou chceme „zakomentovat“ na každém řádku, uvodit tímto znakem.

Názvy proměnné smí obsahovat písmeno (A – Z nebo a - z), číslo (0 - 9) a podtržítko (’_’). Platí zde však pravidla, jak tyto znaky smíme používat:

- První znak smí být pouze písmeno
- Poslední znak nesmí být podtržítko
- V názvu proměnné se nesmí nacházet dvě podtržítka za sebou

U názvu proměnných si musíme dát pozor abychom nepoužili název, který je již vyhrazen pro jazyk VHDL .

```
6   last@rank       --obsahuje neplatny znak
7   5bit_counter    --zacina cislem
8   _A0             --zacina podtrzitkem
9   A0_             --konci podtrzitkem
10  clock__pulse    --obsahuje dve podtrzitka za sebou
```

Obr. 13. Nesprávné pojmenování proměnných

Při **zápisu čísel** můžeme použít podtžítko mezi číslicemi pro přehlednost zápisu. Jako další možnost je zápis dekadického exponentu (E), následující číslice značí mocninu exponentu. Exponent nesmí být záporný.

Bitové řetězce slouží k reprezentaci zápisu bitových polí ve dvojkové, osmičkové (O) a šestnáctkové soustavě (X).

```
6   "1111_0000_1111"    --zapis binarniho retezce "111100001111"
7   o"74_17"            --zapis retezce v osmickove soustave
8   x"fof"              --zapis hexadecimalniho retezce
9   23e3                --zapis celeho cisla 23000
10  54                  --zapis celeho cisla
```

Obr. 14. Zápis hodnot v různých formátech

6.2. Charakteristika zdrojového programu

Na začátku každého programu musíme uvést s jakými knihovnami budeme pracovat. Smysl knihoven je definování typů, objektů nebo operace nad nimi, které mohou být zpřístupněny v různých entitách či architekturách.

Jako první musíme definovat jaké knihovny budeme používat. V našem případě se jedná o knihovnu IEE a dále příkazem **use** vybereme z této knihovny dílčí části, které budeme potřebovat pro náš program. Pro naše úlohy bychom si měli vystačit s knihovnou *std_logic_1164*, ale můžeme přidat i matematické knihovny *std_logic_unsigned* a *std_logic_arith*. podle obr. 15.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity prvni is
7       port (clock_50:in std_logic;
8             sw:in std_logic_vector(3 downto 0);
9             ledr:out std_logic_vector(0 downto 0) );
10  end prvni;
11
12  architecture prvni_priklad of prvni is
13      --deklarační část
14  begin
15      --sekce paralelních příkazů
16  end prvni_priklad;
```

Obr. 15. Základní rozvržení programu

Konstrukce v jazyku VHDL má dvě základní části: deklaraci *entity* a popis *architecture.*

Deklarace *Entity* specifikuje vstupní a výstupní porty modelu. Tyto porty (brány) jsou jediným prostředkem pro komunikaci modelu s prostředím. Všechny porty se deklarují v závorce za klíčovým slovem **port**. Deklarace brány se skládá ze jména brány a dále z určení jejího přenosu (mód) a typu dat, která signál představují. Módy mohou být:

- in – vstup
- out – výstup
- inout – vstupně/výstupní – pro obousměrnou komunikaci
- buffer - výstupní port, u kterého můžeme číst jeho hodnotu.

Typ dat:

- std_logic – bitová velikost
- std_logic_vector – vektorová velikost, do závorky uvádíme jeho velikost a směr pomocí **to** (MSB) nebo **downto** (LSB).

Popis *Architecture* definuje chování nebo strukturu modelu a je vždy svázán s entitou, která definuje rozhraní s okolím. Každá komponenta může být popsána na úrovni strukturální, behaviorální nebo dataflow (bude vysvětleno dále). V „deklarační části“ definujeme všechny proměnné a konstanty – viz kapitola 6.3., které jsou použity v architektuře. Součástí „sekce paralelních příkazů“ mohou být instance komponent nebo procesy, které jsou vzájemně propojeny signály.

Typy popisů:

- Strukturální – architektura obsahuje pouze instance komponent
- Behaviorální – architektura je složena z jednoho nebo více procesů (příkazy, podmínky, cykly)
- Dataflow – modeluje datové závislosti

6.3. Datové typy

V zadáních můžeme využít tři třídy datových objektů: konstanty, signály a proměnné.

- **Konstanty** mají hodnotu, která je neměnná a ta je přiřazena při její deklaraci.
- **Signály** jsou datové objekty, které se v realizované konstrukci skutečně vyskytují. Patří k nim i signály příslušné branám – viz kapitola 6.2. (přiřazení hodnoty pomocí znaku ‘<=‘)
- **Proměnné** slouží pouze jako lokální paměťové buňky (přiřazení hodnoty pomocí ‘:=‘)

Při programování nesmíme zapomenout, že VHDL je paralelní jazyk, to pro signály a jednorozměrné pole jako *std_logic* znamená, že se do nich zapíše pouze poslední přiřazená hodnota v daném cyklu. Předchozí přiřazení nebudou vůbec zobrazena. To neplatí pro proměnné, které v jednom cyklu můžeme měnit neomezeněkrát.

```
12  architecture prvni_priklad of prvni is
13    constant pi:real:=3.14;
14    variable cislo:integer;
15    signal hotovo: std_logic:='3';
16    signal blik: std_logic_out(2 to 0);
17  begin
18        cislo:=5+3;
19        hotovo<='1';
20        blik<="010";
21    end prvni_priklad;
```

Obr. 16. Vytvoření proměnné a přiřazení hodnoty

Jazyk VHDL je definován jako přísně typový, což znamená, že před každým použitím nějakého objektu musí být deklarován jeho typ:

- Integer – celočíselný typ ($-2^{31} + 1$) až ($2^{31} - 1$)
- Real – pohyblivá desetinná čárka -1E308 až 1E308
- Boolean

Výše uvedené určuje, jakých hodnot může deklarovaný objekt nabývat, a jaké operace s ním jsou přípustné. Datové objekty různých typů nemohou být vzájemně přiřazovány bez konverze. Kromě předdefinovaných datových typů si můžeme vytvořit vlastní datové typy a to buď úplně nové nebo jako podtyp již existujícího typu.

```
12  architecture priklad of prvni is
13    type my_count is range 0 to 250;
14    type fruit is (apple,banana,orange,strawberry);
15    variable ovoce:fruit:=orange;
16  begin
17
18    end priklad;
```

Obr. 17. Vytvoření vlastní proměnné

6.4. Operátory

Operátory pro srovnání (=, /=, >, <, <=, >=) jsou v základní definici jazyka definovány pro objekty celočíselné a výčtových typů a pro objekty *std_logic_vector*, ty však musejí mít stejnou velikost.

Předdefinované datové typy a operace

- Integer + real: sčítání (+), odčítání (-), násobení (*), dělení (/), mocnina (**), absolutní hodnota (abs) a modulo (rem / mod)

  A rem B = A –(A/B)*B (kde je A/B integer)

  A mod B = A – B * N (kde N je integer)
- Boolean + *std_logic*: logické a relační operace nebo konkatenace (&)

```
6   5 rem (-3)       -- =2
7   5 mod (-3)       -- =-1
8   "11" and "01"    -- ="01"
9   '1' & '0'        -- ="10"
```

Obr. 18. Operátory rem, mod, AND a konkatenace

6.5. Sekvenční prostředí, strukturní popis a příkazy typu data-flow

6.5.1. Podmíněné příkazy a příkazy cyklů

Dříve než se začneme zabývat těmito typy příkazů, tak si uvedeme, že všechny tyto příkazy mohou být použity pouze v *process.* Proces jako celek představuje jeden z možných paralelních příkazů VHDL, jehož výsledný efekt je dán vnitřním sekvenčním algoritmem procesu. Při definici procesu zadáváme do závorky parametr (může být i více jak 1) při kterém se má proces spustit. Spuštění se provede vždy při změně parametru (jakéhokoliv). Programování procesu se provádí stejně jako v architektuře – viz kapitola 5.2.

```
44  stopky_bezi:process(key2)       --nazev_procesu:process(citlivostni seznam)
45  begin
46
47  end process;
```

Obr. 19. Syntaxe *process*

Pro podmíněný příkaz *if*, který smí být použit pouze v procesu, použijeme srovnávací operátory - viz kapitola 5.4. Výraz musí být standardního booleovského typu. Pokud výraz vede k hodnotě **True**, provede se příkaz, který následuje za *then*. Když je výsledek výrazu **False** a součástí příkazu je klauzule *else* (jediná) nebo *elsif* (neomezené množství) s přiřazenou podmínkou , provede se příkaz, který následuje za *else* nebo *elsif*. Když není klauzule *else* nebo *elsif* součástí příkazu, provádí se příkaz bezprostředně následující za strukturovaným příkazem *if*. Celý podmíněný příkaz se ukončuje pomocí *end if.* Pokud potřebujeme podmínku pro náběžnou hranu použijeme příkaz *rising_edge(promena)* pro sestupnou hranu je to příkaz *falling_edge(promena).*

```
12  if start='1' then
13      --sekvence prikazu
14  elsif rising_edge(clock) then
15      --sekvence prikazu
16  else
17      --sekvence prikazu
18  end if;
```

Obr. 20. Syntaxe příkazu *if else*

Podmíněný příkaz *case* použijeme pokud máme více možností řešení a nechceme použít *if* s *elsif.* Jako poslední musí být možnost *others* (ostatní neuvedené možnosti). Příkaz se ukončuje pomocí *end case.*

```
33      case promena is
34          when podminka1 => --seznam prikazu;
35          when podminka2 => --seznam prikazu;
36          ...
37          ...
38          when others => --seznam prikazu;
39      end case;
```

Obr. 21. Syntaxe příkazu *case*

6.5.2. Podmíněné přiřazovací příkazy

V této kapitole se budeme zabývat pouze s příkazem *with select*. Jedná se o výběrový přiřazovací příkaz, který je analogií sekvenčního příkazů *case* v paralelním prostředí. Výhoda oproti příkazu *case* je v tom, že nemusíme tento příkaz vkládat do procesu a nepíše se žádný speciální ukončovací znak. Příkaz končí řádkou *when others.*

```
32      with podminkova_promena select
33          promena <=  hodnota_a when podminka1,
34                      hodnota_b when podminka2,
35                          ...
36                      hodnota_x when others;
```

Obr. 22. Syntaxe příkazu *with select*

6.5.3. Deklarace komponent

Vyjděme z analogie s reálnými obvody. Lokální komponenta je prostředek jazyka VHDL, který reprezentuje patici na fyzické desce tištěných spojů a který umožňuje vložit do určitého místa architektury další model. Deklarace komponenty je umístěna v deklarační části architektury. Tuto deklaraci můžeme považovat za definici určitého typu komponenty, který umožní dále deklarovat libovolný počet konkrétních a jednoznačně pojmenovaných komponent tohoto typu, které budou mít stejné porty. Syntaxe portů je stejná jako v případě entit.

```
16  component clc50 is
17      Port (  clk : in std_logic;
18              start: in std_logic;
19              clkrun : buffer std_logic );
20      end component;
21   begin
22      hodiny:clc50 port map (clock_50,start,clkrun);  --volani komponenty
23      --pojmenovani_komponenty:komponenta port map (prirazeni vsech portu)
```

Obr. 23. Deklarace komponent

6.5.4. Ukázka kódů

V této kapitole budou ukázány funkční příklady.

```
1   entity delicka is
2       port ( clkin : in std_logic;    -- definice vnitřních signálů. Pro tento příklad je vstupem
3           clkout: buffer std_logic ); --  clkin a výstupem clkout
4   end delicka;
5   architecture mojedelicka of delicka is
6   begin
7       delickahodin:process(clkin)     -- process, který se spouští na každou změnu clkin
8       variable citej:integer:=0;      -- vytvoření vnitřní proměnné
9       begin
10          if rising_edge(clkin) then  -- test clkin na vzestupnou hranu
11              if citej<100 then
12                  citej:=citej+1;
13              else
14                  citej:=0;
15                  clkout<=not clkout;
16              end if;                 -- ukončovací syntaxe příkazu if
17          end if;
18      end process;                    -- ukončovací syntaxe příkazu process
19  end mojedelicka;
20
```

Obr. 24. Dělička

```
1  entity dekoder is
2      Port (  sw: in std_logic_vector(1 downto 0);
3              led: out std_logic_vector(3 downto 0) );
4  end dekoder;
5      -- vstup (sw) je 2-bitový a výstup (led) je 4-bitový
6  architecture mujdekoder of dekoder is
7  begin
8      preved:process(vstup)
9      begin
10     case sw is
11         when "00" => led<="1000";
12         when "01" => led<="0100";
13         when "10" => led<="0010";
14         when others => led<="0001";
15                        end case;          -- ukončovací syntaxe příkazu case
16     end process;
17 end mujdekoder;
```

Obr. 25. Dekodér

```
1  entity scitac is
2      Port (  a,b: in std_logic_vector(3 downto 0);
3              c: in std_logic_vector(7 downto 0);
4              x: buffer std_logic_vector(3 downto 0);
5              y: out std_logic_vector(7 downto 0) );
6  end scitac;
7  architecture mujscitac of scitac is
8  variable pom: std_logic_vector(8 downto 0)
9  begin
10     pom <= a + b;
11     x <= pom(3 downto 0);
12     y <= c and (0000 & b);  -- logické operace mužou být
13                             --pouze na vektorech stejné velikosti
14     y <= (pom(7 downto 0) or (0000 & x)) and (1111 & b);
15             --provede se pouze r.14 příkaz na r.12 se neprovede vubec
16 end mujscitac;
```

Obr. 26. Funkce operátorů
(znak “&“ značí konkatenaci)

7. Závěr

V závěrečné části této bakalářské práce považujeme za potřebné upozornit na několik zásadních skutečností jednotlivých úloh, jakož i na naše doporučení, týkající se zadané problematiky.

Při zpracovávání prvního úkolu se vyskytla jediná chyba a to při použití čítače 7490, která je popsána v kapitole 3.3.1., ta se vyřešila pomocí čítače 7490MS, který jsem vytvořil a bude doporučen studentům při vytváření programů pomocí čítačů. V úloze jsme se dále naučili, jak se do programu Quartus přidávají jednotlivé porty a jak se obsluhují a také jsme se naučili jak fungují základní periferie (LED dioda a „switch“) na desce.

V druhém zadání máme volbu jak úlohu vyřešit. Doporučuji zpracování v jazyce VHDL přesto, že už víme jak čítače pracují a jak se s nimi tvoří program. Při vytváření složitějších návrhů ve schematickém návrhu, se nám začal obvod chovat neočekávaně (př.: při zapojení čítačů 74193 do série se jeden z nich po hodnotě b“0011“ nastavil hodnotu b“1100“), a to z důvodu špatného naprogramování součástek v Quartusu. Navíc ovládání LCD se ve schematickém návrhu jevilo jako velmi složité. Z tohoto důvodu jsme museli použít VHDL pro dořešení tohoto bodu úlohy, nebo jsme museli změnit zadání, aby výstup nebyl na LCD, ale na sedmi segmentu jako v předchozích bodech úlohy a aby tento program pracoval pouze jako ukazatel času bez budíku.

Zadání třetího úkolu se po dohodě s vedoucím bakalářky změnilo z PS/2 na VGA z toho důvodu, že studenti tento typ úlohy úspěšně zprovoznili, a proto jsem nemusel dělat návrh této úlohy. V tomto zadání jsme museli navrhnout protokol pro komunikaci s VGA. Při vytváření tohoto programu nebylo nejsložitější nastavení časování pro vysílání jednotlivých pixelů, které si můžeme ověřit v simulaci, ale zvolení správných čítacích cyklů. Při nesprávné volbě se na „chytřejších“ monitorech obrázek zobrazí, ale až po delší době kvůli synchronizaci na monitoru. Tato doba může být i více než deseti násobek standardní doby. U „hloupějších“ monitorů se obrázek nemusí zobrazit vůbec, nebo může být nějak poškozen.
Všechny tyto zpracované úlohy jsou přiloženy jak v příloze, tak na CD.

Použitá literatura

[1] Jiří Pinker, Martin Poupa: Číslicové systémy a jazyk VHDL
Praha : BEN - technická literatura, 2006

[2] Peter J. Ashenden: The designer's guide to VHDL 3rd ed.
Burlington : Morgan Kaufmann, 2008

[3] Jiří Král: Řešené příklady ve VHDL : hradlová pole FPGA pro začátečníky
Praha : BEN - technická literatura, 2010

[4] Jiří Douša: Jazyk VHDL
ČVUT v Praze, Fakulta elektrotechnická

[5] Datasheet k čítači 7490
http://pdf1.alldatasheet.com/datasheet-pdf/view/82082/ETC/7490.html

[6] Datasheet k čítači 74193
http://pdf1.alldatasheet.com/datasheet-pdf/view/51050/FAIRCHILD/74193.html

[7] Datasheet k LCD HD44780, DE2 System CD-ROM (CD-ROM dodávaný s deskou) http://support.dce.felk.cvut.cz/mediawiki/index.php/DE2_Board

[8] Datasheet k ADV7123, DE2 System CD-ROM (CD-ROM dodávaný s deskou)
http://support.dce.felk.cvut.cz/mediawiki/index.php/DE2_Board

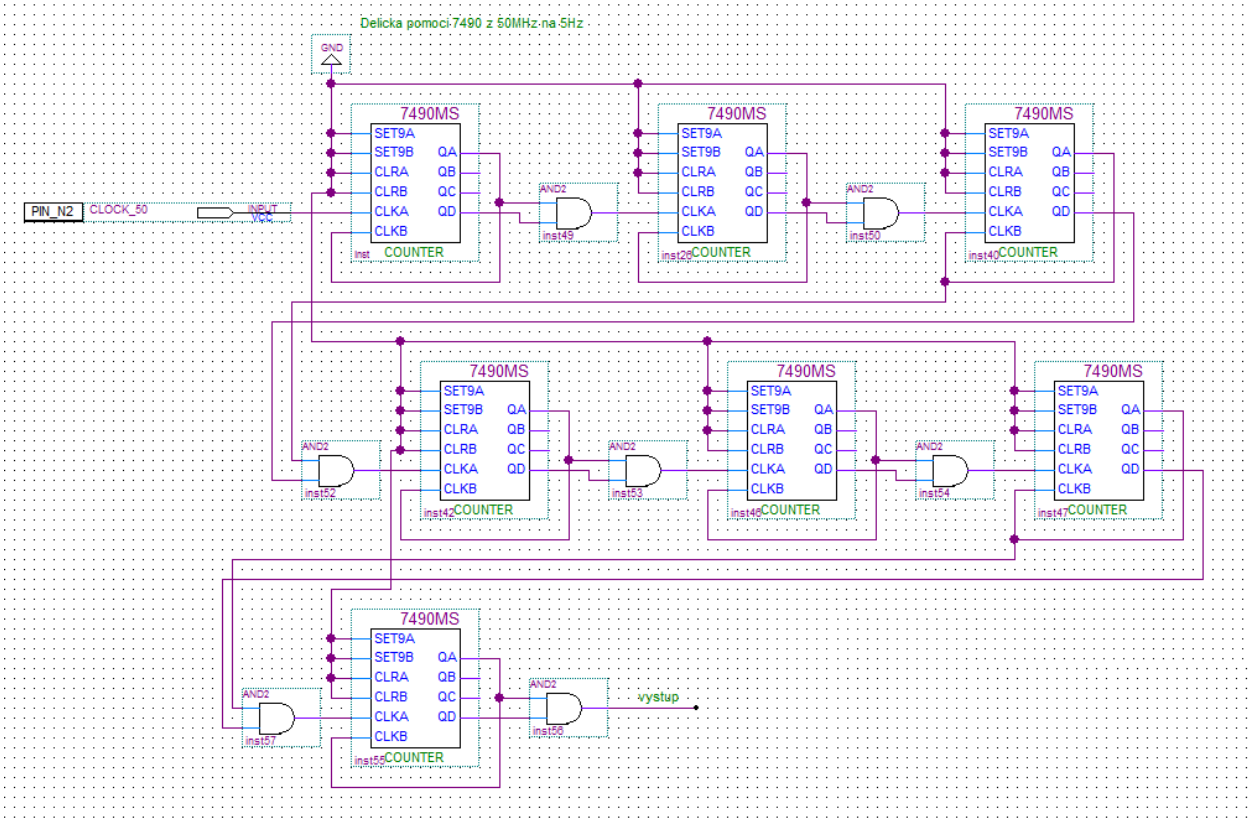
[9] VGA generator for the XSA Boards
http://www.xess.com/appnotes/an-101204-vgagen.pdf

[10] Internetové fórum k desce Altera DE2
http://www.alteraforum.com/forum/

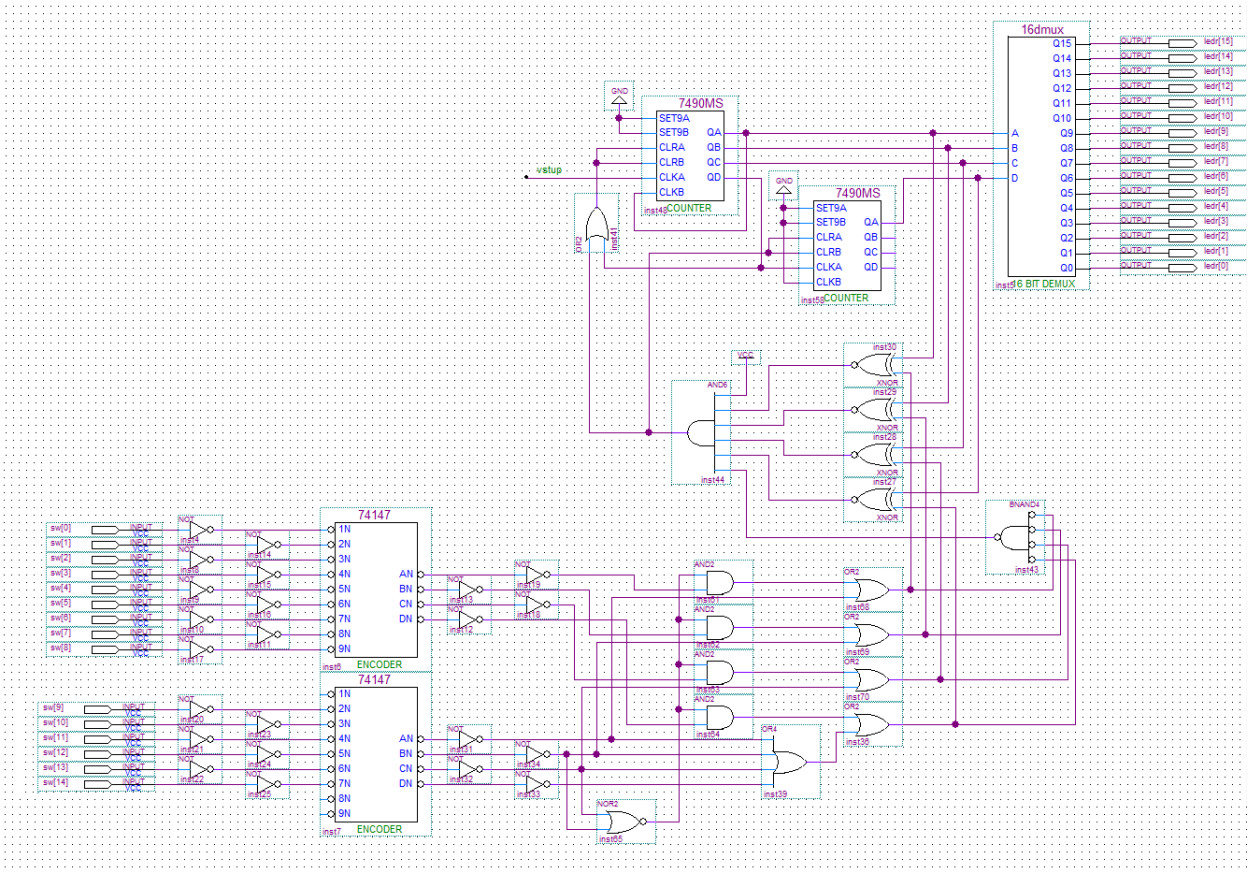
Obsah přiloženého CD

- bakalářská práce ve formátu pdf
- zpracované úlohy
- obsah CD k Altera DE2
- návod na jazyk VHDL v PowerPointové prezentaci

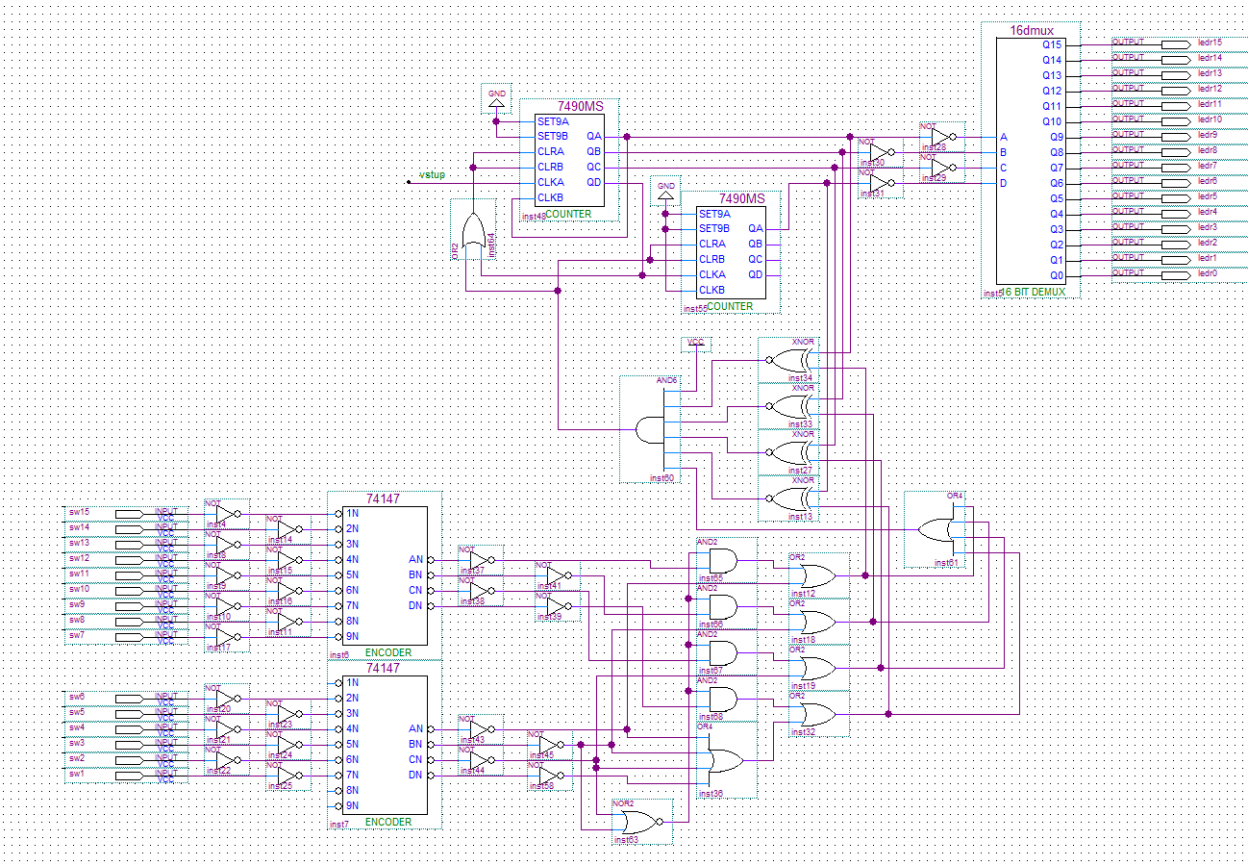
Příloha A – světelný had



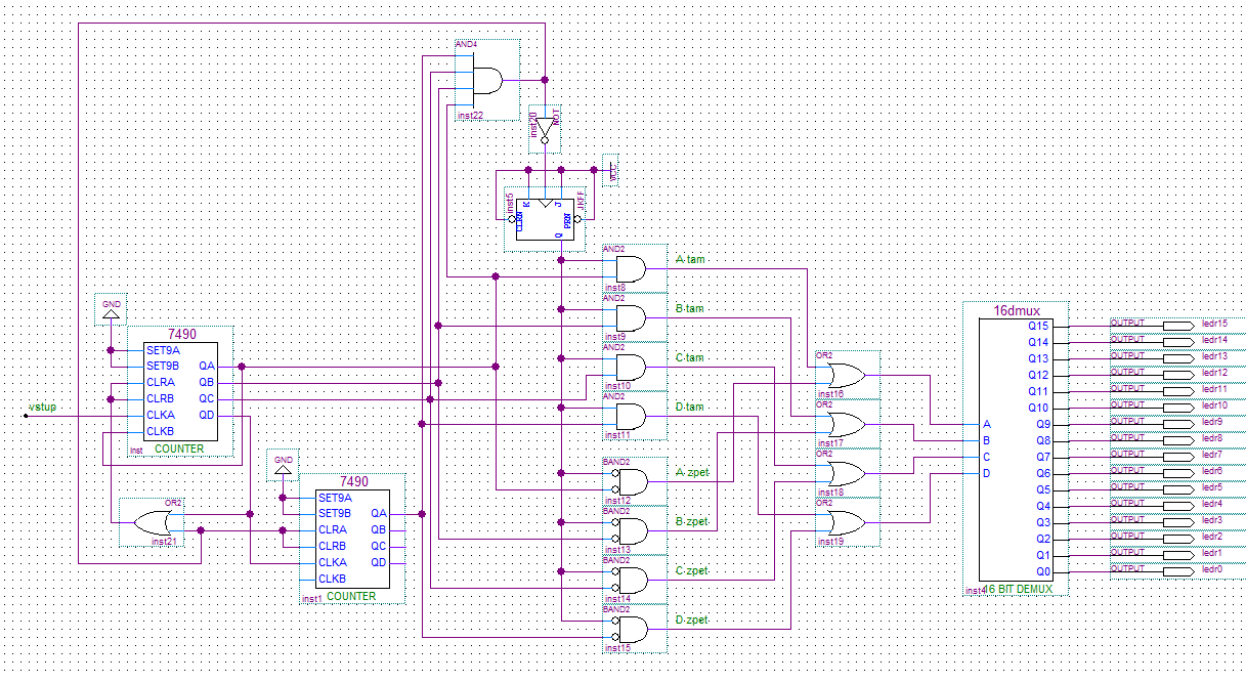
Dělička vstupních hodin



Obvod pro vzestupný pohyb hada



Obvod pro vzestupný pohyb hada



Obvod pro pohyb hada, který se odráží od krajních bodů

Příloha B – časovací obvody

Úlohy ve schematickém popisu jsou pouze na CD z důvodu jejich velikosti, která jim neumožňuje vložení do přílohy. Proto jsou zde programy pouze ve VHDL

Stopky

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

entity stopky is
port ( clock_50 : in std_logic;
                        key1,key2 : in std_logic;
                        hex0,hex1,hex2,hex3,hex4,hex5,hex6,hex7 : out std_logic_vector(6 downto 0) );
end stopky;

architecture mojestopky of stopky is
component clc50 is
Port (      clk : in std_logic;
                        start: in std_logic;
                        clkrun : buffer std_logic );
end component;

component citac10 is
Port (      clk,reset: in std_logic;
                        owf: out std_logic;
                        hex: out std_logic_vector(6 downto 0) );
end component;

component citac6 is
Port (      clk,reset: in std_logic;
                        owf: out std_logic;
                        hex: out std_logic_vector(6 downto 0) );
end component;

signal owf0,owf1,owf2,owf3,owf4,owf5 : std_logic;
signal start : std_logic := '0';
signal clkrun : std_logic:= '0';

begin
hodiny:clc50 port map (clock_50,start,clkrun);
setiny_jednotky:citac10 port map(clkrun,key1,owf0,hex2);
setiny_desitky:citac10 port map(owf0,key1,owf1,hex3);
sekundy_jednotky:citac10 port map(owf1,key1,owf2,hex4);
sekundy_desitky:citac6 port map(owf2,key1,owf3,hex5);
minuty_jednotky:citac10 port map(owf3,key1,owf4,hex6);
minuty_desitky:citac6 port map(owf4,key1,owf5,hex7);

stopky_bezi:process(key2)
begin
        if key2='0' then
                        start<=not start;
        end if;
end process;

hex0<="1111111";
hex1<="1111111";
end mojestopky;



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

```
entity clc50 is
port ( clk : in std_logic;
                start : in std_logic;
                clkrun: buffer std_logic );
end clc50;

architecture mojeclc50 of clc50 is
begin
delicka:process(clk)
variable citej:integer:=0;
begin
        if start='1' then
                if clk='1' and clk'event then
                        if citej<250000 then
                                citej:=citej+1;
                        else
                                citej:=0;
                                clkrun<=not clkrun;

                        end if;
                end if;
        else
                citej:=0;
                clkrun<='0';
        end if;
end process;
end mojeclc50;



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

entity citac10 is
Port (  clk,reset: in std_logic;
                owf: out std_logic;
                hex: out std_logic_vector(6 downto 0) );
end citac10;

architecture mujcitac10 of citac10 is
begin
citej10:process(reset,clk)
variable citani:integer:=0;
begin
        if reset='1' then
                if clk='1' and clk'event then
                        if citani=9 then
                                owf<='1';
                                citani:=0;
                        else
                                citani:=citani+1;
                                owf<='0';
                        end if;
                end if;
        else
                citani:=0;
                owf<='0';
        end if;
case citani is
        when 0 => hex<="1000000";
        when 1 => hex<="1111001";
        when 2 => hex<="0100100";
        when 3 => hex<="0110000";
        when 4 => hex<="0011001";
        when 5 => hex<="0010010";
        when 6 => hex<="0000010";
        when 7 => hex<="1111000";
        when 8 => hex<="0000000";
        when others => hex<="0010000";
end case;
end process;
end mujcitac10;
```

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

entity citac6 is
Port (	clk,reset: in std_logic;
		owf: out std_logic;
		hex: out std_logic_vector(6 downto 0) );
end citac6;

architecture mujcitac6 of citac6 is
begin
citej6:process(reset,clk)
variable citani:integer:=0;
begin
	if reset='1' then
		if clk='1' and clk'event then
			if citani=5 then
				owf<='1';
				citani:=0;
			else
				citani:=citani+1;
				owf<='0';
			end if;
		end if;
	else
		citani:=0;
		owf<='0';
	end if;
case citani is
	when 0 => hex<="1000000";
	when 1 => hex<="1111001";
	when 2 => hex<="0100100";
	when 3 => hex<="0110000";
	when 4 => hex<="0011001";
	when others =>hex<="0010010";
end case;
end process;
end mujcitac6;
```

Minutka

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

entity minutka is
port ( clock_50 : in std_logic;
		key1,key2 : in std_logic;
		sw: in std_logic_vector(11 downto 0);
		ledr: buffer std_logic_vector(9 downto 6);
		hex0,hex1,hex2,hex3,hex4,hex5,hex6,hex7 : buffer std_logic_vector(6 downto 0) );
end minutka;

architecture mojeminutka of minutka is
component clc50 is
Port (	clk : in std_logic;
		start: in std_logic;
		clkrun : buffer std_logic );
end component;

component citac10 is
Port (	clk,set,stop: in std_logic;
		nacti: in std_logic_vector(3 downto 0);
		owf: out std_logic;
		hex: out std_logic_vector(6 downto 0) );
end component;
signal owf0,owf1,owf2,start,clkrun,stop0,stop1,stop2 : std_logic := '0';

begin
hodiny:clc50 port map (clock_50,start,clkrun);
jednotky:citac10 port map(clkrun,key1,stop0,sw(3 downto 0),owf0,hex0);
desitky:citac10 port map(owf0,key1,stop1,sw(7 downto 4),owf1,hex1);
```

```
stovky:citac10 port map(owf1,key1,stop2,sw(11 downto 8),owf2,hex2);
konec:process(hex0,hex1,hex2)
begin
	if hex0="1000000" and hex1="1000000" and hex2="1000000" then
		stop0<='1';
		stop1<='1';
		stop2<='1';
	else
		stop0<='0';
		stop1<='0';
		stop2<='0';
	end if;
end process;
blikej:process(clock_50)
variable citej:integer:=0;
begin
	if stop0='1' and stop1='1' and stop2='1' then
		if rising_edge(clock_50) then
			if citej/=5000000 then
				citej:=citej+1;
			else
				citej:=0;
				ledr<=not ledr;
			end if;
		end if;
	else
		ledr<="0000";
	end if;
end process;
minutka_bezi:process(key2)
begin
	if key2='0' then
		start<=not start;
	end if;
end process;
hex3<="1111111";
hex4<="1111111";
hex5<="1111111";
hex6<="1111111";
hex7<="1111111";
end mojeminutka;



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

entity clc50 is
port ( clk : in std_logic;
		start : in std_logic;
		clkrun: buffer std_logic );
end clc50;

architecture mojeclc50 of clc50 is
begin
delicka:process(start,clk)
variable citej:integer:=0;
begin
	if start='1' then
		if clk='1' and clk'event then
			if citej<25000000 then
				citej:=citej+1;
			else
				citej:=0;
				clkrun<=not clkrun;

			end if;
		end if;
	else
		citej:=0;
		clkrun<='0';
	end if;
end process;
end mojeclc50;
```

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

entity citac10 is
Port (      clk,set,stop: in std_logic;
                    nacti: in std_logic_vector(3 downto 0);
                    owf: out std_logic;
                    hex: out std_logic_vector(6 downto 0) );
end citac10;

architecture odcitani10 of citac10 is
signal citani: std_logic_vector(3 downto 0):="0000";

begin
citej10:process(set,clk,stop)
begin
        if set='0' then
                if nacti<"1010" then
                        citani<=nacti;
                else
                        citani<="1001";
                end if;
        elsif stop='0' then
                if clk='1' and clk'event then
                        if citani="0000" then
                                owf<='1';
                                citani<="1001";
                        else
                                citani<=citani-'1';
                                owf<='0';
                        end if;
                end if;
        end if;
case citani is
        when "0000" => hex<="1000000";
        when "0001" => hex<="1111001";
        when "0010" => hex<="0100100";
        when "0011" => hex<="0110000";
        when "0100" => hex<="0011001";
        when "0101" => hex<="0010010";
        when "0110" => hex<="0000010";
        when "0111" => hex<="1111000";
        when "1000" => hex<="0000000";
        when others => hex<="0010000";
end case;
end process;
end odcitani10;
```

Hodiny s alarmem

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

entity hodiny is
        port ( sw : in std_logic_vector(13 downto 0);
                        key : in std_logic_vector(2 downto 1);
                        clock_50: in std_logic;
                        LCD_RW,LCD_EN,LCD_RS,LCD_ON: out std_logic;
                        LCD_DATA:out std_logic_vector(7 downto 0);
                        ledr:buffer std_logic_vector(17 downto 0) );
end hodiny;

architecture mojehodiny of hodiny is
component clccitac is
                port( clkin: in std_logic;
                                settime: in std_logic;
                                clkout: buffer std_logic );
                end component;
```

```
component citac is
		port( clk: in std_logic;
				settime: in std_logic;
				priznak: in std_logic;
				nastav_hodnotu: in std_logic_vector(7 downto 0);
				owf: out std_logic;
				hexj,hexd:out std_logic_vector(7 downto 0) );
		end component;

component clclcd is
	port( clkin: in std_logic;
			clkout: buffer std_logic );
end component;

component inicializace is
	port( clc,clktime,settime,setalarm: in std_logic;
			RW,EN,RS: out std_logic;
			DATA: out std_logic_vector(7 downto 0);
			time1,time2,time3,time4,time5,time6,alarm1,alarm2,alarm3,alarm4: in std_logic_vector(7 downto 0) );
end component;

signal clktime,clklcd,mujclc,owf0,owf1,owf2,owf3,owf4,alarmon: std_logic:='0';
signal hex2,hex3,hex4,hex5,hex6,hex7,alarm1,alarm2,alarm3,alarm4:  std_logic_vector(7 downto 0);
begin
	alarm: process (key(1),owf0,clklcd)
	variable citej:integer:=0;
	begin
		if key(1)='0' then
			alarmon<='0';
		elsif falling_edge(owf0) then
			if hex4=alarm1 and hex5=alarm2 and hex6=alarm3 and hex7=alarm4 then
				alarmon<='1';
			end if;
		end if;
		if rising_edge(clklcd) then
			if citej=0 then
				if alarmon='1' then
					ledr<=not ledr;
				else
					ledr<="00000000000000000";
				end if;
				citej:=50000;
			else
				citej:=citej-1;
			end if;
		end if;
	end process;

	mujcas: process(clklcd)
	variable citejcas:integer:=0;
	begin
		if clklcd='1' then
			if citejcas=50000 then
				mujclc<=not mujclc;
				citejcas:=0;
			else
				citejcas:=citejcas+1;
			end if;
		end if;
	end process;

		lcd_on<='1';
		hodinylcd: clclcd port map(clock_50,clklcd);
		prvnikrok: inicializace portmap(clklcd,mujclc,key(2),key(1),lcd_rw,lcd_en,lcd_rs,lcd_data,hex2,hex3,hex4,
		hex5,hex6,hex7,alarm1,alarm2,alarm3,alarm4);

		delicka_na_1hz:clccitac port map (clock_50,key(2),clktime);
		vteriny:citac port map (clktime,key(2),'1',"00000000",owf0,hex2,hex3);
		minuty:citac port map (owf0,key(2),'1',sw(7 downto 0),owf1,hex4,hex5);
		hodiny:citac port map (owf1,key(2),'0',"00" & sw(13 downto 8),owf2,hex6,hex7);
		alarmminuty:citac port map (owf4,key(1),'1',sw(7 downto 0),owf3,alarm1,alarm2);
		alarmhodiny:citac port map (owf3,key(1),'0',"00"&sw(13 downto 8),owf4,alarm3,alarm4);
end mojehodiny;
```

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

entity citac is
	port( clk: in std_logic;
			settime: in std_logic;
			priznak: in std_logic;
			nastav_hodnotu: in std_logic_vector(7 downto 0);
			owf: out std_logic;
			hexj,hexd:out std_logic_vector(7 downto 0) );
	end citac;

architecture mujcitac of citac is
signal citejj,citejd: std_logic_vector(3 downto 0):="0000";
begin
	citejcas:process (clk,settime,nastav_hodnotu)
	begin
		if settime='0' then
			owf<='0';
			if priznak='1' then
				if nastav_hodnotu(7 downto 4)>"0100" then
					citejd<="0101";
				else
					citejd<=nastav_hodnotu(7 downto 4);
				end if;
				if nastav_hodnotu(3 downto 0)>"1000" then
					citejj<="1001";
				else
					citejj<=nastav_hodnotu(3 downto 0);
				end if;
			else
				if nastav_hodnotu(7 downto 4)>"0001" then
					citejd<="0010";
					if nastav_hodnotu(3 downto 0)>"0010" then
						citejj<="0011";
					else
						citejj<=nastav_hodnotu(3 downto 0);
					end if;
				else
					citejd<=nastav_hodnotu(7 downto 4);
					if nastav_hodnotu(3 downto 0)>"1000" then
						citejj<="1001";
					else
						citejj<=nastav_hodnotu(3 downto 0);
					end if;
				end if;
			end if;
		elsif rising_edge(clk) then
			if priznak='0' and citejd="0010" and citejj="0011" then
				citejj<="0000";
				citejd<="0000";
			elsif citejj="1001" then
				citejj<="0000";
				if citejd="0101" then
					citejd<="0000";
					owf<='1';
				else
					citejd<=citejd+"0001";
				end if;
			else
				citejj<=citejj+"0001";
				owf<='0';
			end if;
		end if;
		case citejj is
			when "0000" => hexj<="00110000";
			when "0001" => hexj<="00110001";
			when "0010" => hexj<="00110010";
			when "0011" => hexj<="00110011";
			when "0100" => hexj<="00110100";
			when "0101" => hexj<="00110101";
			when "0110" => hexj<="00110110";
			when "0111" => hexj<="00110111";
			when "1000" => hexj<="00111000";
			when others => hexj<="00111001";
		end case;
```

```
                case citejd is
                        when "0000" => hexd<="00110000";
                        when "0001" => hexd<="00110001";
                        when "0010" => hexd<="00110010";
                        when "0011" => hexd<="00110011";
                        when "0100" => hexd<="00110100";
                        when others => hexd<="00110101";
                end case;
        end process;
end mujcitac;



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

entity clccitac is
        port( clkin: in std_logic;
                        settime: in std_logic;
                        clkout: buffer std_logic );
end clccitac;

architecture mujcasovac of clccitac is
begin
        runtime: process(clkin)
        variable citej:integer:=0;
        begin
                if settime='0' then
                        citej:=0;
                        clkout<='0';
                else
                        if clkin='1' then
                                if citej<25000000 then
                                        citej:=citej+1;
                                else
                                        citej:=0;
                                        clkout<=not clkout;
                                end if;
                        end if;
                end if;
        end process;
end mujcasovac;



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

entity clclcd is
        port( clkin: in std_logic;
                        clkout: buffer std_logic );
end clclcd;

architecture mujcasovac of clclcd is
begin
        runtime: process(clkin)
        variable citej:integer:=0;
        begin
                if rising_edge(clkin) then
                        if citej<24 then
                                citej:=citej+1;
                        else
                                citej:=0;
                                clkout<=not clkout;
                        end if;
                end if;
        end process;
end mujcasovac;
```

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

entity inicializace is
	port( clc,clktime,settime,setalarm: in std_logic;
		RW,EN,RS: out std_logic;
		DATA: out std_logic_vector(7 downto 0);
		time1,time2,time3,time4,time5,time6,alarm1,alarm2,alarm3,alarm4: in std_logic_vector(7 downto 0) );
end inicializace;

architecture mojeinicializace of inicializace is
shared variable cas:integer:=0;
shared variable ukazatel:integer:=-1;
signal endinit,ready:std_logic:='0';
begin
	casuj: process(clc,clktime,setalarm)
	begin
	if falling_edge(clc)then
		if endinit='0' then
			rs<='0';
		else
			rs<='1';
		end if;
		rw<='0';
		if cas/=0 then
			cas:=cas-1;
			en<='0';
		else
			if clktime='0' and ready='1' then
				ukazatel:=8;
			else
				ukazatel:=ukazatel+1;
			end if;
			case ukazatel is
				when 0 => cas:=16000;
				when 1 => cas:=4099;
						data<="00110000";
				when 2 => cas:=99;
						data<="00110000";
				when 3 => cas:=38;
						data<="00110000";
				when 4 => cas:=38;
						data<="00111000";
				when 5 => cas:=38;
						data<="00111000";
				when 6 => cas:=1529;
						data<="00000001";
				when 7 => cas:=38;
						data<="00000111";
						ready<='1';
				when 8 => cas:=38;
						data<="00001100";

				when 9 => cas:=1529;
						data<="00000001";
				when 10 => cas:=1529;
						data<="00000010";
				when 11 => cas:=40;
						data<=time6;
						rs<='1';
						endinit<='1';
				when 12 => cas:=40;
						data<=time5;
				when 13 => cas:=40;
						data<="00111010";
				when 14 => cas:=40;
						data<=time4;
				when 15 => cas:=40;
						data<=time3;
				when 16 => cas:=40;
						data<="00111010";
				when 17 => cas:=40;
						data<=time2;
				when 18 => cas:=40;
						data<=time1;
						if setalarm='1' then
```

```
                                                    ukazatel:=28;
                                            end if;
                                    when 19 => cas:=40;
                                            data<="11000000";
                                            rs<='0';
                                            endinit<='0';
                                    when 20 => cas:=40;
                                            data<=alarm4;
                                            rs<='1';
                                            endinit<='1';
                                    when 21 => cas:=38;
                                            data<=alarm3;
                                    when 22 => cas:=40;
                                            data<="00111010";
                                    when 23 => cas:=40;
                                            data<=alarm2;
                                    when 24 => cas:=40;
                                            data<=alarm1;
                                    when others => ukazatel:=25;
                                                    rs<='0';
                                                    endinit<='0';
                            end case;
                            en<='1';
                    end if;
            end if;
            end process;
end mojeinicializace;
```

Příloha C – VGA

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

entity monitor is
	port ( clock_50,clock_27: in std_logic;
				VGA_R,VGA_G,VGA_B: out std_logic_vector(9 downto 0);
				VGA_BLANK: out std_logic:='0';
				VGA_HS,VGA_VS,VGA_SYNC: out std_logic:='1';
				VGA_CLK: buffer std_logic;
				ledr: buffer std_logic_vector(0 downto 0) );
end monitor;

architecture displej of monitor is
component generaterow is
	Port (	clk : in std_logic;
				blank,hs,vs,sync: out std_logic;
				endrow: buffer std_logic;
				VGA_R,VGA_G,VGA_B: out std_logic_vector(9 downto 0) );
	end component;

signal endrow: std_logic;
begin
	hodinyVGA:process(clock_50)
	begin
		if rising_edge(clock_50) then
			vga_clk<=not vga_clk;
		end if;
	end process;

	drawrow:generaterow port map (vga_clk,vga_blank,vga_hs,vga_vs,vga_sync,ledr(0),VGA_R,VGA_G,VGA_B);

end displej;



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

entity generaterow is
	port ( clk: in std_logic;
				blank: out std_logic:='0';
				hs,vs,sync: out std_logic:='1';
				endrow: buffer std_logic;
				VGA_R,VGA_G,VGA_B: out std_logic_vector(9 downto 0) );
end generaterow;

architecture mujgeneraterow of generaterow is
signal drawrow:std_logic:='0';
begin
	row:process(clk)
	variable ukazatelrow,ukazatelframe,aframe,dframe:integer:=0;
	variable bframe:integer:=-1;
	variable jakyradek,citej:integer:=0;
	variable pomocna:integer:=1;
	begin
	if rising_edge(clk) and drawrow='1' then
		ukazatelrow:=ukazatelrow+1;
		if ukazatelrow<49 then
			BLANK<='0';
			HS<='1';
			SYNC<='1';
		elsif ukazatelrow<689 then
			BLANK<='1';
			if pomocna>0 then
				vga_r<="0000000000";
```

```
                                        vga_g<="0000000000";
                                        vga_b<="1111111111";
                                        pomocna:=pomocna-1;
                                elsif jakyradek<240 then
                                        vga_r<="1111111111";
                                        vga_g<="1111111111";
                                        vga_b<="1111111111";
                                elsif jakyradek<480 then
                                        vga_r<="1111111111";
                                        vga_g<="0000000000";
                                        vga_b<="0000000000";
                                end if;
                        elsif ukazatelrow<705 then
                                BLANK<='0';
                        elsif ukazatelrow<800 then
                                HS<='0';
                                SYNC<='0';
                        else
                                jakyradek:=jakyradek+1;
                                if jakyradek<241 then
                                        citej:=citej+1;
                                else
                                        citej:=citej-1;
                                end if;
                                pomocna:=citej;
                                ukazatelrow:=0;
                                if jakyradek=480 then
                                        endrow<=not endrow;
                                        jakyradek:=0;
                                        drawrow<='0';
                                end if;
                        end if;
                elsif rising_edge(clk) and drawrow='0' then
                        if ukazatelframe=0 then
                                bframe:=bframe+1;
                                if bframe=26400 then
                                        bframe:=0;
                                        drawrow<='1';
                                        ukazatelframe:=1;
                                end if;
                        elsif ukazatelframe=1 then
                                dframe:=dframe+1;
                                if dframe=8000 then
                                        dframe:=0;
                                        ukazatelframe:=2;
                                        sync<='0';
                                        vs<='0';
                                end if;
                        elsif ukazatelframe=2 then
                                aframe:=aframe+1;
                                if aframe=1600 then
                                        aframe:=0;
                                        ukazatelframe:=0;
                                        blank<='0';
                                        sync<='1';
                                        vs<='1';
                                end if;
                        end if;
                end if;
                end process;
end mujgeneraterow;
```