FAKULTA
ELEKTROTECHNICKÁ
ČVUT V PRAZE

Bakalářská práce

VYUŽITÍ PŘÍPRAVKU TERASIC DE10-LITE V JAZYCE VHDL

Josef Šebánek

Duben 2021

Vedoucí práce: Ing. Pavel Lafata, Ph.D.
Katedra telekomunikační techniky FEL

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šebánek** Jméno: **Josef** Osobní číslo: **478259**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Využití přípravku Terasic DE-10 Lite v jazyce VHDL

Název bakalářské práce anglicky:

Using Terasic DE-10 Lite in VHDL

Pokyny pro vypracování:

Seznamte se s přípravkem Terasic DE-10 Lite založeném na FPGA Intel. Navrhněte a v jazyce VHDL vytvořte nezbytné knihovny a kódy pro využití jednotlivých integrovaných periferií přípravku. Využijte a demonstrujte čítač s výstupem na LED přípravku a 7segmentový displej, pomocí integrovaného PLL vytvořte čítač času – stopky s výstupem na displej, vytvořte generátor tónů (syntezátor) s využitím přepínačů a výstupem na připojený reproduktor, využijte vestavěný ADC převodník pro měření napětí a teploty z vnitřního čidla, využijte akcelerometr na přípravku, navrhněte aplikaci a využijte VGA výstup přípravku. Výstupem budou knihovny v jazyce VHDL pro ovládání jednotlivých částí přípravku a jejich popis.

Seznam doporučené literatury:

[1] Dokumantace DE10-Lite User Manual dostupná na
https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=2
18&No=1021&PartNo=1
[2] Ashenden, P.J.: The VHDL Cookbook (First Edition), ISBN 978-0-12088-7-859.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Lafata, Ph.D., katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **26.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Pavel Lafata, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

PROHLÁŠENÍ

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Ve Vraném nad Vltavou, 5. dubna 2021

ANOTACE

Tato práce má za cíl seznámit s možnostmi využití vývojového kitu Terasic DE10-Lite, postaveném na technologii FPGA. První část práce seznamuje obecně s technologií FPGA v porovnání s architekturou Intel MAX10. Další části práce je seznámení se všemi komponentami na vývojovém kitu, které jsou využity. V praktické části je vytvořeno sedm knihoven, které slouží jako ukázka, k jakému účelu se dá tento kit využít.

Klíčová slova: FPGA, VHDL, Intel MAX10, DE10-Lite

ANNOTATION

This thesis aims to acquaint with the possibilities of using the Terasic DE10-Lite kit, which is based on the FPGA technology. The first part of the thesis generally introduces FPGA technology in comparison with the Intel MAX10 architecture. The next part presents all the components on the development kit, that are used. In the practical part, seven categories are created as an example of what purpose this kit can be used for.

Keywords: FPGA, VHDL, Intel MAX10, DE10-Lite

PODĚKOVÁNÍ

V prvé řadě bych chtěl poděkovat vedoucímu bakalářské práce Ing. Pavel Lafata, Ph.D. za odborné vedení a cenné rady při vypracování této práce. Dále Bych rád poděkoval celé mé rodině, která mě během studia podporovala.

OBSAH

OBRÁZKY

TABULKY

1 ÚVOD

Programovatelná logika *Programmable Logic Devices* (PLD) je v dnešní době ve velkém rozvoji. Výhodou PLD je její multifunkčnost oproti *Application Specific Integrated Circuit (*ASIC*)* a zároveň její rychlost oproti mikrokontrolerům *microcontroller* (MCU). V dnešní době je nejrozšířenější PLD právě v podobě programovatelných hradlových polí *Field Programmable Gate Array* (FPGA), pomocí kterých jsme téměř schopní nahradit jakoukoliv integrovanou digitální část zařízení. K naprogramování FPGA se využívají jazyky hardwarového popisu *Hardware Description Language* (HDL). Hlavní zastupitelé HDL jsou Verilog a VHDL (*VHSIC Hardware Description Language – Very High Speed Integrated Circuit Hardware Description Language)*. V práci budeme využívat zejména VHDL.

Hlavními výrobci FPGA jsou dnes Altera a Xilinx. Tito výrobci mají celou řadu produktů od samotných FPGA až po mnoho druhů desek, které krom samotného FPGA obsahují i samotné periferie. Jedním ze zástupců takové desky je přípravek Altera Terasic DE10-Lite, který obsahuje spoustu základních i pokročilých periferií.

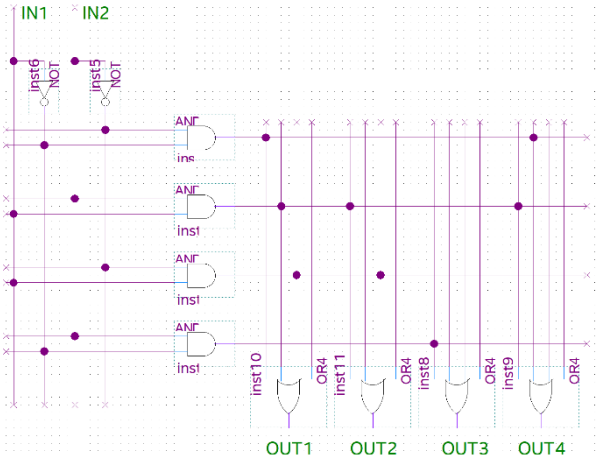
K této desce lze na internetu sehnat program, který dokáže ovládat veškeré periferie pomocí uživatelského prostředí. Tento program je sice vhodný pro demonstraci použití desky, ale není vhodný pro ty, kteří chtějí veškeré periferie využít do svého projektu. Tato bakalářská práce je zaměřena na vytvoření sady knihoven využívajících většinu důležitých periferií. Vzhledem k množství periferií na DE10-Lite jsou vybrány jen některé, a to konkrétně: ovládání led diod, sedmi segmentový displej, ADC převodník, akcelerometr a VGA výstup. Veškeré knihovny jsou vytvořené pomoci VHDL a řádně komentovány pro pozdější využití.

Nejprve práce pojednává o samotné struktuře FPGA a veškerých vestavěných funkcích, které budou pro praktickou část důležité. Dále se zaměřuje na jazyk VHDL, kde popisuje jeho strukturu a syntaxi. Důležitou částí je také popis vývojového studia Quartus Prime, které bylo na veškeré projekty použito. Další část popisuje samotnou desku DE10-Lite, veškeré periferie a parametry.

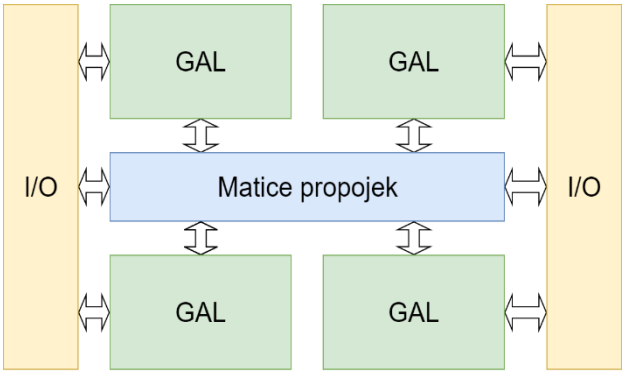
2 FPGA

2.1 HISTORIE

Historie programovatelných polí sahá do roku 1970, kdy byl vytvořen první PLA obvod viz Obrázek 2.2. Tento obvod obsahoval programovatelnou matici AND a OR hradel, které bylo možno naprogramovat pouze jednou. Vylepšená verze je GAL (Generic Array Logic), která má stejný design jako PLA jen s výhodou přeprogramování. To však nevylepšilo možnost složitějších logických struktur. Toto omezení vyřešila firma Altera v roce 1988 vyvinutím CPLD (*Complex Programmable Logic Device*) viz Obrázek 2.3. Jedná se o spojení více obvodů GAL nebo PLA, které jsou propojeny programovatelnou maticí spojů. To umožňuje také tvoření složitější sekvenční logiky, která je v mnoha případech nezbytná. [1]



Obrázek 2.2 struktura PLA, GAL [2, s. 4]

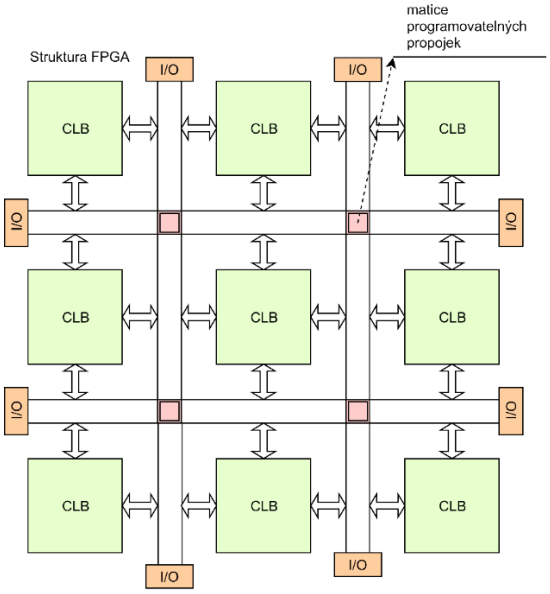


Obrázek 2.1 Struktura zapojení CPLD [2, s. 9]

První FPGA (EP300) bylo vyvinuto firmou Altera již v roce 1984. Tento čip byl založen na paměti EPROM, která byla mazatelná pomocí ultrafialového světla skrze okénko. Na konci 80. let si Stev Casselman nechal patentovat návrh FPGA s více než 600 000 přeprogramovatelnými hradly. V polovině 90. let FPGA bylo využíváno zejména v telekomunikacích, ale postupně začalo expandovat i do dalších odvětví jako například do automobilového průmyslu, domácích spotřebičů a podobně, kde způsobilo velkou revoluci [3].

2.2 ARCHITEKTURA FPGA

Struktura FPGA na Obrázek 2.3 je založena na matici CLB (*Configurable logic blocks*), ve kterých se nachází veškerá logika. Veškeré CLB jsou propojeny pomocí horizontálních a vertikálních sběrnic, mezi kterými se CLB nachází, sběrnice jsou dále přepojovány přes přepínací matice (Switch cell). Nedílnou součástí FPGA jsou vstupy a výstupy I/O cell (*input/output*). Tyto buňky zajištují komunikaci a ochranu na vstupně-výstupních pinech. V samotném čipu se může nacházet spoustu dalších integrovaných bloků jako je například PLL (*Phase-locked loop*), ADC (*analog digital converter*), FLASH paměť a mnoho dalších. Některé z těchto bloků potřebují vlastní I/O například jako ADC. [4]



Obrázek 2.3 struktura FPGA

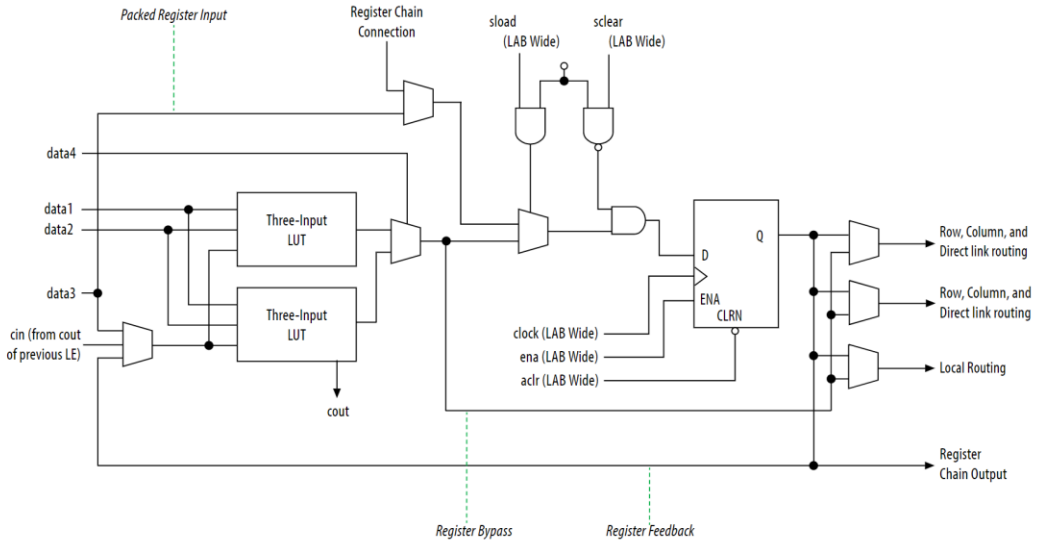
2.2.1 ARCHITEKTURA CLB

Srdcem každého FPGA je CLB, které primárně tvoří veškerou kombinační a sekvenční logiku. CLB se u různých výrobců liší a také různě nazývají. I jeden výrobce může mít ve svých produktových řadách různou strukturu těchto buněk. Příkladem je zrovna architektura MAX10 od Altery, která používá buňky s názvem LAB (Logic Array Block). Základem každého bloku jsou pokaždé tři základní prvky: jedna nebo více *look-up table* (LTU), sčítačka a výstupní registr. LUT realizuje logickou funkci pomocí logické hodnoty 0 nebo 1 podle vstupní kombinace. LUT může mít různý počet vstupů, obvykle má vstupy čtyři, ale CLB můžou obsahovat i šesti, osmi nebo deseti bitové LUT. Počet řádků s binárními hodnotami se odvíjí od počtu vstupů, pro tabulku s n vstupy musí být 2^n řádků. To vyplývá z počtu řádků libovolné logické funkce. Výstup může být dále přiveden do sčítačky s výstupem z druhé LUT anebo společně mohou tvořit více bitovou LUT pomocí multiplexoru. Poslední částí je výstupní registr, který zajištuje synchronizaci dat s ostatními CLB. [5]

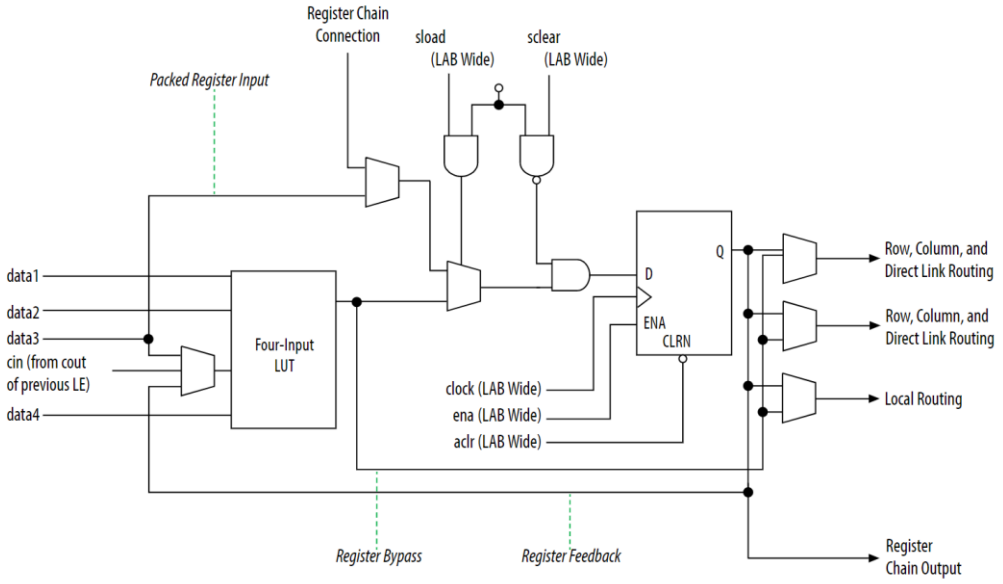
Struktura CLB u řady Altera MAX10 je velmi komplikovaná a dala by se na ni napsat samostatná práce. Zjednodušeně funguje na principu dvou přepínatelných módů. Aritmetický mód viz Obrázek 2.4

funguje na bázi dvou LUT, kde každá z nich má tři vstupy. Výstup z tabulek je připojen do sčítačky, ze které jsou data vyvedeny buď přes registr nebo přímo ven. [6]

V normální módu viz Obrázek 2.5 má LUT tabulka čtyři vstupy, které jsou následně vyvedeny stejně jako v aritmetickém módu. U obou těchto módů je možné data vyvést přes *chain register* do vedlejší buňky na sčítačku.



Obrázek 2.4 Altera MAX10 CLB v aritmetickém módu – Intel [6, s. 11]



Obrázek 2.5 Altera MAX10 CLB v normálním módu – Intel [6, s. 10]

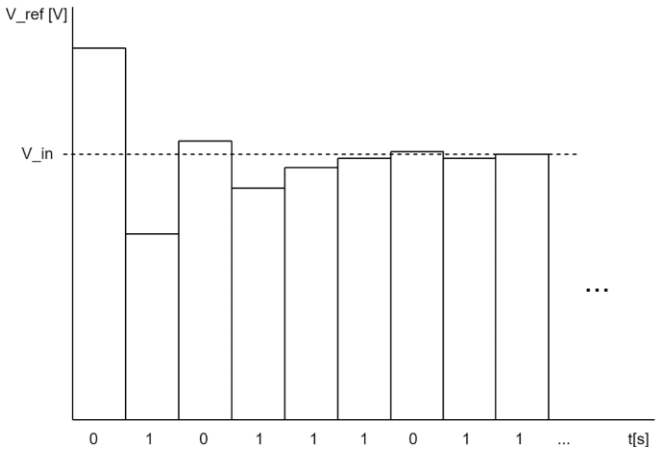
2.2.2 PLL

Jak již bylo zmíněno dříve, tak do dnešních FPGA jsou přidávány různé přídavné periferie. Dost častým problémem u FPGA je tvoření různých hodinových frekvencí, které musí být tvořeny pomocí děličeček. Tyto děličky ale zabírají určitý počet logických elementů. Altera MAX10 tento problém řeší pomocí periferie fázového závěsu PLL. Na čipu 10M50, který je používaný na DE10-lite nalezneme čtveřici PLL, kde každý z nich je schopen generovat až čtyři nezávislé hodinové takty. [6]

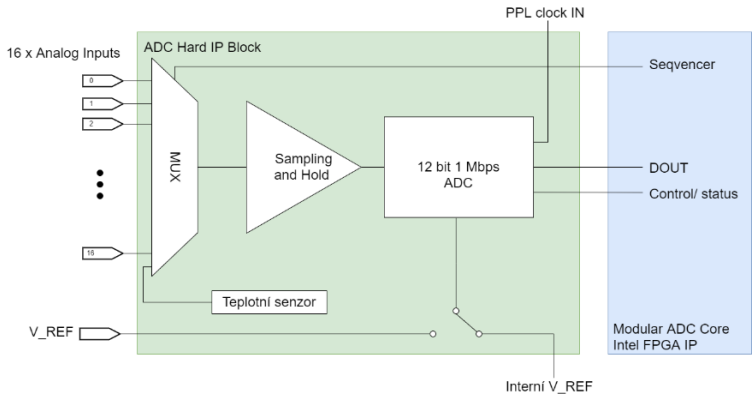
2.2.3 ADC

Další komponentou je analog digitální převodník (ADC). 10M50 v sobě obsahuje dva 12bitové ADC s postupnou aproximací. Tento převodník podle [4] funguje na principu porovnávání vstupní hodnoty napětí V_{in} s referenční hodnotou napětí V_{ref} která je nastavována pomocí aproximačního registru. V čase t_0 převodník nastaví nejvyšší bit aproximačního registru na hodnotu log. 1, která odpovídá $V_{ref} = \frac{1}{2}V_r$, kde V_r je rozsah ADC. Tato hodnota se pomocí komparátoru porovná s V_{in}. Pokud hodnota V_{in} je vyšší než V_{ref} , aproximační registr ponechá nejvyšší hodnotu na 1. Pokud je hodnota V_{in} nižší, registr nastaví na nejvyšší bit hodnotu 0. V čase t_1 se přejde na další bit, který již zvyšuje v_{ref} pouze o polovinu bitu předchozího $V_{ref} = \frac{1}{4}V_r$, dále se znovu provádí stejné porovnávání. Tento celý postup je vidět na Obrázek 2.6. Po převedení všech dvanácti bitů převodník vyšle signál pro signalizaci dokončeného převodu, data uloží na výstup a čeká na další převod. Rozlišení převodníku je závislé na napájecím napětí V_r a počtu bitů n vztahem [7]

$$LBS = \frac{V_r}{2^n} \ . \qquad 2.1$$



Obrázek 2.6 ukázka postupné aproximace u ADC předodníku – [7]



Obrázek 2.7 Vnitřní zapojení ADC core u architektury Altera MAX 10 – Intel [8]

Referenční napětí je možné využívat buď interní nebo externí viz Obrázek 2.7. Vstupní hodnoty do ADC jsou řízeny pomocí multiplexoru (MUX), který má 16 vstupů. Pokud bychom chtěli převádět všech 16 vstupů musíme MUX přepínat a dané hodnoty převádět postupně. Tím se nám snižuje rychlost převádění. Tento převodník můžeme spustit ve dvou módech. Singl mode využívá pouze jeden ADC v 12bit režimu, z toho vyplývá, že takovéto převodníky můžeme vnitřně implementovat dva, které budou fungovat na sebe nezávisle. Pomocí obou převodníků můžeme vstupy rozdělit a tím zvýšit rychlost převodu na polovinu. [8]

V režimu Dual mode ovládáme oba ADC jako jeden blok, který synchronně převádí 2 hodnoty najednou. Tyto hodnoty jsou následně uloženy do vnitřní RAM paměti, ze které jsme schopni číst. Vstupní piny do ADC jsou pevně připojeny a nedají se vnitřně přepojit. V uživatelském manuálu [9] se dočteme, že na desce DE10-lite je připojeno do ADC core pouze osm analogových vstupů, které jsou vyvedeny pomocí Arduino Header. Referenční napětí V_{ref} je napájeno $2.5\ V$, z tohoto důvodu je podle rovnice 2.1 minimální rozlišovací napětí $610{,}35\ \mu V$. Poslední z pinů je připojen do interního teplotního senzoru, díky kterému jsme schopni podle převodní tabulky zjistit teplotu jádra.

2.3 VHDL

VHDL (VHSIC Hardware Description Language) patří do skupiny jazyků HDL (Hardware Description Language). Tato skupina slouží pro popis hardware a jeho implementaci. Dalším jazykem řadícím se do této kategorie je například Verilog. Jazyk VHDL byl původně vytvořen ministerstvem obrany USA s cílem popisu chování zákaznických integrovaných obvodů. Specifikace jazyka byla přijata jako standard IEEE 1076-1987 v roce 1987. [10]

Struktura VHDL je založena na modulech. Každý jednotlivý soubor s koncovou *.vhd* funguje jako samostatný obvod, který dále můžeme do dalších kódů implementovat a vytvořit tak složitější zapojení. Struktura modulu je rozdělena do dvou základních částí **entity** a **architecture.**

2.3.1 ENTITY

Entity nám definuje vnější pohled na modul. v části **Port** jsou zapsány veškeré vstupní a výstupní porty, jejich definice módu, šířky sběrnic, definice jejich základní hodnoty v případě nepřipojení dané

sběrnice. Názvy veškerých těchto portů musí být jednoslovné, ale místo mezery lze použít podtržítko. Mód nám označuje, jestli se jedná o vstupní nebo výstupní port, základní typy jsou:

- OUT – definice výstupního portu
- IN – definice vstupního portu
- INOUT – vstupně výstupní port
- BUFFER – výstupní port, ze kterého lze zapsaná hodnota číst

2.3.2 ARCHITECTURE

Část architecture popisuje funkce daného modulu. Ve VHDL můžeme popisovat třemi způsoby abstrakce:

Behaviorální popis je nejvyšší úrovní abstrakce. Jedná se o nejrychlejší a nejjednodušší popis daného modulu. Modul si představíme jako „black box“, u kterého programátorským způsobem popíšeme jeho chování. Tento způsob je výhodný pro popis složitějších zapojení, kdy nám až tolik nejde o efektivitu, jelikož do syntézy nevíme, jaké bude výsledné zapojení. Na druhou stranu kód bude lépe čitelný. Nevýhodou je minimální kontrola nad syntézou. Do jisté míry jsme schopni syntezátoru říct, jestli nám jde o rychlost obvodu nebo o minimalizaci, ale to je celé.

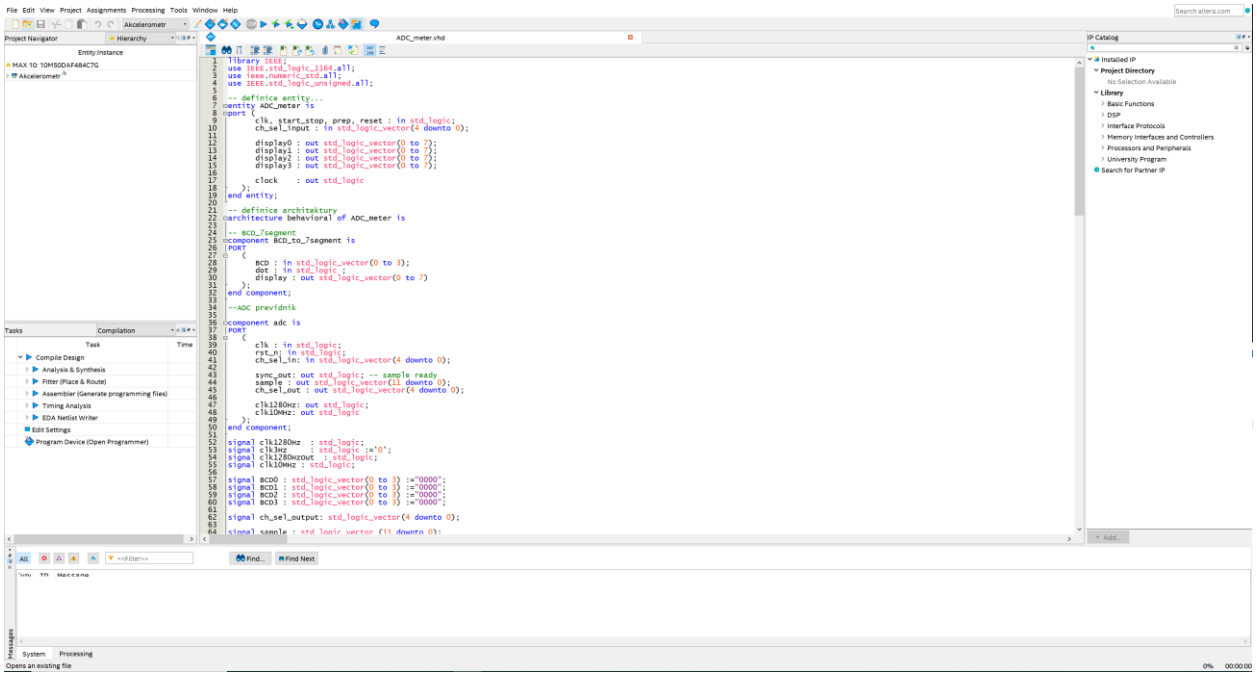
Dataflow (RTL) jedná se o střední úroveň abstrakce na úrovni RTL (register transfer level), modelování na úrovni registrů. V této části jsme obvod schopni popsat pomocí boolových rovnic. Výhodou je již vyšší kontrola nad výsledným zapojením. Budeme vědět jaké operace je potřeba zajistit. Nevýhodou je opět nejasné přesné zapojení. Syntezátor stále za nás vytvoří zapojení (netlist).

Strukturální popis je nejnižší úrovní abstrakce, kde máme maximální kontrolu nad výsledkem zapojení. V této části může být použit i schématický editor, ve kterém obvod sami nakreslíme z elementárních obvodů. Nebo jej stejným způsobem můžeme napsat pomocí VHDL. Syntezátor tento obvod již neupravuje ale pouze ho převede na netlist. Strukturální popis je vhodný na jednodušší obvody, u kterých jsme schopni zapojení vymyslet, nebo jej máme vytvořené.

3 QUARTUS PRIME

3.1 EDITOR

Pro práci s FPGA je obecně možné použít jakékoli vývojové prostředí, ale nejlepší variantou je vždy vyhledat přímo prostředí pro dané FPGA, které má v sobě uloženou celou architekturu a veškeré IP bloky, pomocí kterých jsme pak schopni efektivněji využívat FPGA. Altera poskytuje vývojové prostředí s názvem Quartus Prime, které vidíme na Obrázek 3.1. Prostředí je pro práci přehledné. IP bloky jsme schopni implementovat pomocí jednoduchých uživatelsky přístupných oken, ve kterých si jednoduše nastavíme, veškeré požadované vlastnosti. Další velkou výhodou tohoto prostředí je zpětná kompatibilita projektů. Po nainstalování nové verze není žádný problém otevřít jakýkoli starší projekt. Prostředí po otevření staršího projektu nabídne i aktualizaci některých prvků na novější verzi.



Obrázek 3.1 Prostředí Quartus Prime

3.2 ZOBRAZENÍ A PRÁCE S OBVODEM

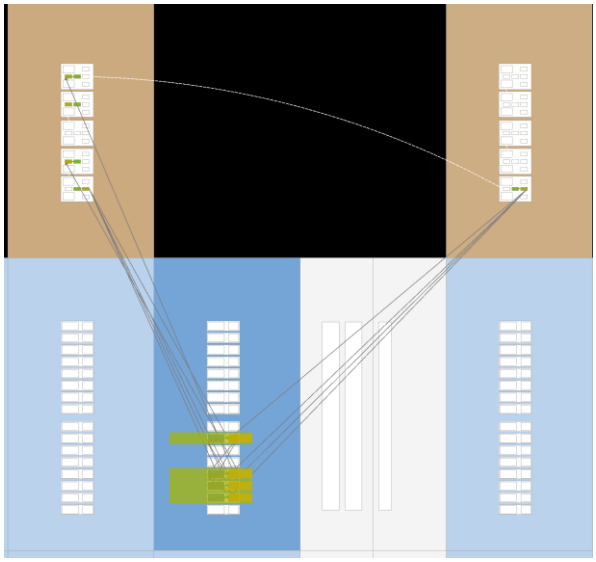
V prostředí Quartus Prime máme mnoho možností zobrazení našeho obvodu při samotném vývoji. Základní zobrazení je samozřejmě možnost otevřít obvod v některém z jazyků HDL. Vzhledem k vývoji se předpokládá, že budeme pracovat právě ve VHDL nebo Verilogu, takže s tímto zobrazením se budeme setkávat nejčastěji. Další možností jak už vývoje, tak zobrazení, je schématický editor. V Quartus Prime existuje mnoho vytvořených funkčních bloků jak základních, tak komplikovanějších. Tyto bloky můžeme v schématickém editoru využít a společně je propojovat, a tvořit tak funkční celek. Výhodou tohoto „zapojování“ je, že budeme s jistotou vědět, jak bude celkový obvod vypadat.

Pokud budeme mít obvod navrhnutý, a projde analýzou a syntézou, tak je možné se podívat, jestli vše proběhlo v pořádku pomocí **RTL Viewer**. Tento nástroj nám zobrazí pouze přesný překlad našeho kódu. K tomuto kroku není potřeba znát ani FPGA, na které později toto zapojení budeme aplikovat. **RTL Viewer** nám pouze vytváří schématické zobrazení s jednoduchými funkčními bloky, jako například sčítačky, násobičky, čítače, paměti, ale také i základní prvky jako jsou AND, OR a další logická hradla. Tento pohled nám dokáže jednoduše ukázat chyby, jako jsou například statické hodnoty u výstupů, kde by statické být neměly. Nebo popřípadě zapomenuté propojky.

Dalším užitečným zobrazením je **State machine Viewer**. Zde si můžeme prohlédnout grafické vyjádření přechodů pomocí jednoduchého orientovaného grafu. Z tohoto grafu lze lépe pochopit chování našeho obvodu při různých stavech a jsme tak schopni zjistit, jestli nemůže nastat nějaká neočekávaná situace, která nám obvod dostane například do nestabilního stavu.

Při aplikaci na určité FPGA nás může zajímat, jak vlastně náš vytvořený obvod bude opravdu zapojen. K tomuto pohledu nám slouží zobrazení **Technology Map Viewer**. Toto zobrazení je dále děleno na dvě části **Post-Mapping** a **Post-Fitting**. U jednodušších obvodů budou zobrazení téměř totožná, jelikož se budou lišit pouze v definici pár pinů, které nejsou využity pro náš obvod, jako například piny pro programování. **Post-Mapping** nám ukáže ideální realizaci našeho obvodu složenou z prvků vyskytujících se přímo v FPGA. U **Post-Fitting** může dojít ke změně oproti prvnímu případu u složitějších obvodů. Kdybychom například potřebovali více osmibitových sčítaček než se jich fyzicky vyskytuje v FPGA, tak bychom v tomto zobrazení viděli jejich nahrazení jinými obvody. Další změnou může být využití jiných obvodů kvůli vzdálenosti na čipu.

Posledním zobrazením je **Chip Planner**. Na tomto zobrazení, jíž uvidíme celé FPGA a jednotlivé jeho funkční bloky. V tomto zobrazení se můžeme přímo podívat, jak bude náš obvod realizován. Tady již nejsme schopni kontrolovat naše zapojení jako takové, ale může nám posloužit například pro optimalizaci zátěže z pohledu teploty. Na Obrázek 3.2 vidíme jednoduché zapojení čítače implementované na FPGA.



Obrázek 3.2 Chip Planner realizace čítače

3.3 SIGNAL TAP

Užitečnou funkcí v programu Quartus Prime je Signal Tap. Pomocí tohoto programu můžeme v reálném čase testovat vnitřní a výstupní signály v samotném FPGA. Principiálně funguje na vytvoření speciálního zapojení, ve kterém signály, které si předem stanovíme, vyvede do vnitřní paměti a následně je po paketech posílá pomocí USB do programu. Při používání tohoto programu u složitějších obvodů si musíme dát pozor na počet využitých logických členů, jelikož samotné přeposílání využívá také vnitřní logiku. Program nám následně hodnoty vnitřních signálů vykreslí do grafu logických hodnot.

Jelikož snímání funguje na bázi ukládání do paměti, tak příchozí data přichází jako balíček hodnot za nějakou danou dobu od spuštění. Samotné snímání může být nastaveno různě. Jednou z možností je **Continuous,** kdy nám program snímá předem určenou velikost datového balíčku od spuštění snímání přímo v programu. Samotná data jsou vzorkována dle hodinového taktu, který se dá také nastavit jako vnitřní proměnná. Další možností je **Input signál**. Tento mód funguje stejně jako **Continuous,** jedinou změnou je spuštění snímání, které funguje na vstupním signálu. Zajímavým módem je **Transitional**. Tento mód snímá opět od spuštění snímání přímo v programu. Rozdílem oproti ostatním módům je ovšem samotné vzorkování, které probíhá pouze v době, kdy některý z našich zkoumaných signálů změní svou hodnotu. Tento mód je užitečný, pokud chceme zkoumat například některý konkrétní přechodový děj nebo hazardní stav.

4 TERASIC DE10-LITE

Tato deska je jedna z šesti vydaných desek od firmy Terasic, která obsahuje FPGA vystavěné na 55nm architektuře MAX10. Jak již bylo v úvodu zmíněno, tak každý výrobce si svoje FPGA tvoří podle svého, a jednotlivé architektury se mohou od sebe dost lišit. Na Obrázek 4.1 vidíme, že tato architektura je vystavěná na LAB (logic Array Block), ve kterých je tvořena veškerá logika. Dále vidíme velký blok FLASH paměti, který je rozdělen do UFM (User Flash Memory) aneb paměť volně přístupná pro vlastní data, a CFM (Configuration Flash Memory). Z této paměti si FPGA načte veškeré zapojení logiky po připojení napájení. [11]

Dalším zajímavým blokem přidaným v této architektuře je Embedded memory (vestavěná paměť) nebo také M9K. Jedná se o paměťové bloky typu RAM (Random Access Memory), ROM (Read Only Memory), shift register a FIFO (First In First Out). Tyto bloky mají výhodu, že jsou rozmístěny všude po FPGA a umožnují nám rychlejší přístup. Další velkou výhodou je, že je již nemusíme sami vytvářet a nezabírají nám tak ostatní logické členy. [11]

Posledním zajímavým typem bloku je clock. Tyto obvody slouží jako rozvod hodinových signálů vytvořených pomocí PLL popsaných v kapitole 2.2.2. Tyto bloky jsou výhodné u vyšších kmitočtů, kde by již mohly delší trasy hodinových signálů vytvořit v čipu problémy.

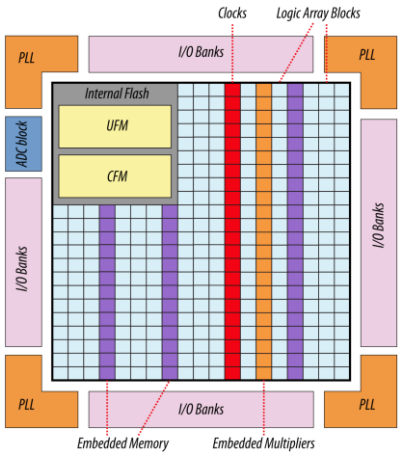
Na desce Terasic DE10-Lite je využito FPGA s označením 10M50DAF484C7G toto označení nám popisuje důležité parametry daného FPGA. Podle Obrázek 4.2 můžeme vyčíst že:

- **10M** značí architekturu Intel MAX 10
- **50** znamená 50 000 logických elementů
- **DA** obsahuje 2x ADC a flash
- **F** menší verze kontaktů typu BGA (ball grid array)
- **484** 484 I/O kontaktů
- **C** provozní teplota mezi 0 °C – 85 °C
- 7 střední rychlost
- G specifikace produktu s označením RoHS6

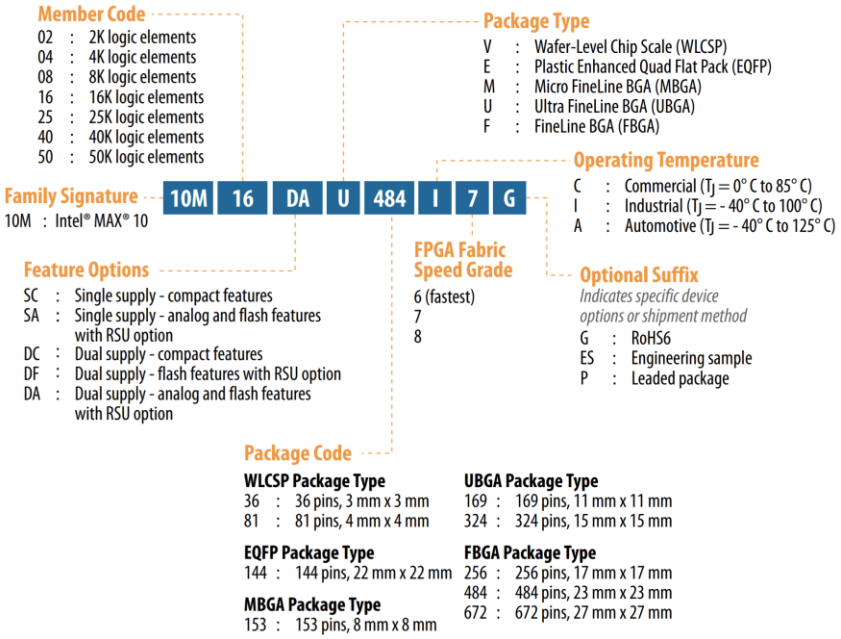
Dalšími důležitými specifikacemi 10M50, které se můžeme dočíst od prodejce jsou:

- 1638 kb Embedded memory (M9K)
- 5888 kb User flash memory (UFM)
- 144 násobiček 18x18
- 4 PLL
- 2 ADC
- Napájení 3,3 V

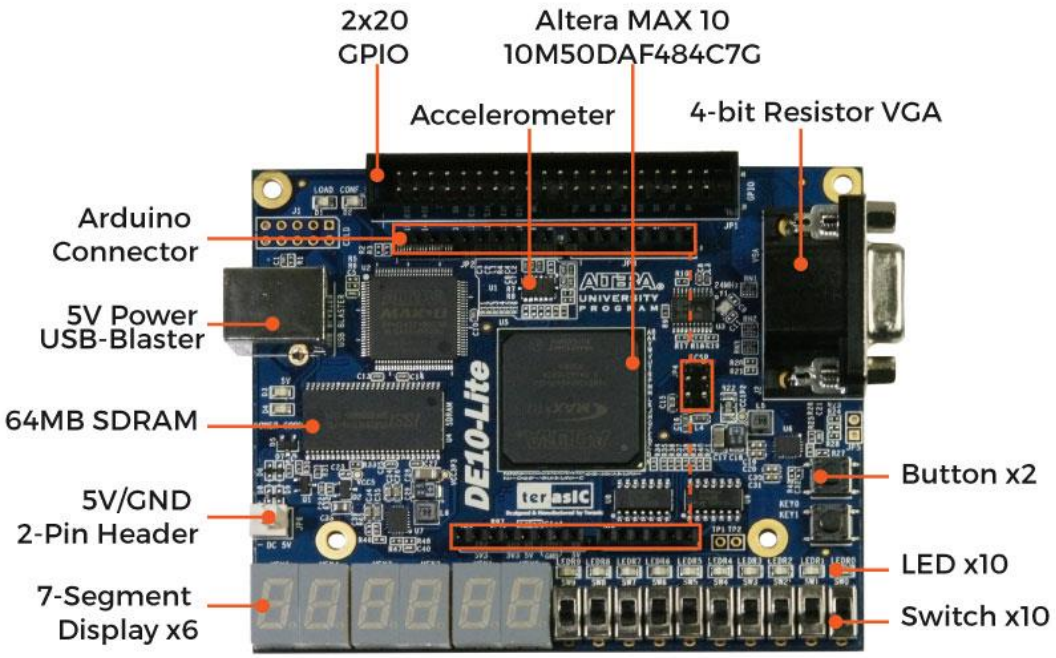
Z popisku je patrné, že se jedná o nejvyšší verzi na architektuře MAX10, která nám umožnuje vytvářet složitá zapojení, které pomocí čtyři PLL mohou pracovat až na šestnácti různých hodinových taktech bez nutnosti použití děliček. Díky dvěma ADC jsme schopni pracovat s analogovými signály s rychlostí dohromady 2 Mbps.



Obrázek 4.1 bloková ukázka architektury MAX 10 – Intel [6, s. 4]



Obrázek 4.2 Popis typového označení od firmy Terasic – Intel [11, s. 5]



Obrázek 4.3 Deska Terasic De10-lite [9 s. 4]

4.1 LED DIODY

Na desce DE10-lite můžeme dohromady nalézt třináct led diod, viz Obrázek 4.3. Tři diody jsou pouze signalizační, jejich popis je v Tabulka 4.1. Dalších deset červených diod (LEDR0 – LEDR9) nalezneme na spodu desky vedle přepínačů. Těchto deset led diod lze libovolně používat v našich aplikacích. Doporučená konfigurace výstupů na tyto led je 3.3-V LVTTL. Jedná se o 3,3 V stavovou logiku, u které platí, že: úroveň log. 0 musí být menší než 0,4 V a úroveň log. 1 musí být větší než 2,4 V. [9]

Tabulka 4.1 signalizace diod

Označení diody	Název signalizace
D1 (load)	Svítí, pokud pracuje USB programátor
D2 (conf)	Svítí, pokud je FPGA úspěšně nakonfigurováno
D4 (power)	Svítí, pokud je deska správně napájena.

4.2 PŘEPÍNAČE

Ve spodní části desky pod led diodami nalezneme deset dvoustavových přepínačů (SW0 – SW9). Tyto přepínače můžeme libovolně využít v našich aplikacích. Doporučená stavová logika je stejně jako u led diod 3.3-V LVTTL. Jelikož se jedná o vstupní periferii tentokrát hodnota log. 0 musí být maximálně 0,8 V a hodnota log.1 minimálně 2 V. Lze také použít pro vyšší bezpečnost stavu logiku 3.3 V SCHMITT TRIGGER. Schmitt trigger nám stabilizuje přechodový děj přepínače. [9]

4.3 TLAČÍTKA

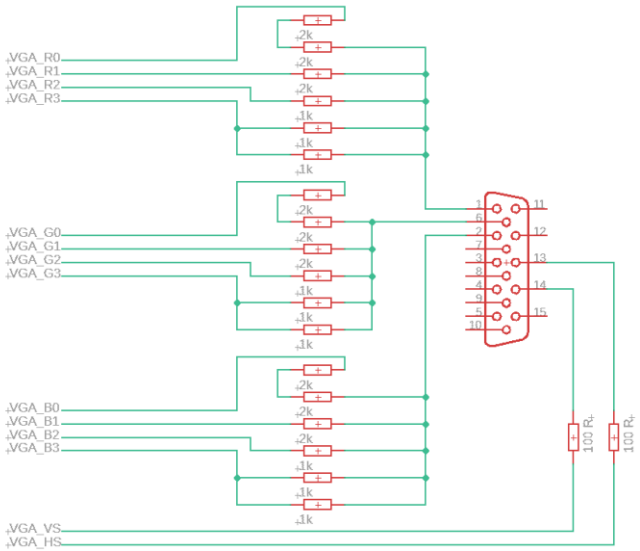
Na pravé části desky nalezneme dvojici tlačítek (KEY0 a KEY1). Tlačítka jsou připojena k zemi a vstup do FPGA je připojen přes odpory na napájení. Z toho vyplývá, že klidová hodnota tlačítek je log. 1 a hodnota při podržení tlačítka je log. 0. Vzhledem k tomu, že tlačítka mají kmitavý přechodový děj, tak vstupní pin musíme nastavit na 3.3 V SCHMITT TRIGGER. Toto nastavení nám zajistí plynulý přechod z log. 1 na log. 0. a naopak. Pokud bychom to nenastavili, tak by nám mohlo docházet k vícenásobnému falešnému zmáčknutí. [9]

4.4 SEGMENTOVÝ DISPLEJ

Další užitečnou periferií na desce je segmentový displej o velikosti 6 znaků (HEX0 – HEX5). Tyto znaky mají společnou katodu. Veškeré anody jsou zapojeny do FPGA, takže lze indexovat každý segment zvlášť. Indexace segmentů je číslována podle čísel jednotlivých znaků. Tedy například pro znak s označením HEX0 máme indexaci kontaktů (HEX00 – HEX06) kde HEX06 je tečka. Konfigurace těchto pinů je stejná jako u led diod v kapitole 4.1. [9]

4.5 VGA

Konektor VGA přesněji DE-15 nalezneme na pravé části desky. Tento konektor se standardně používá pro přenos analogového obrazu. Do konektoru vede pouze pět vodičů: červený kanál, modrý kanál, zelený kanál, horizontální synchronizace a vertikální synchonizace. Jelikož každá barva je reprezenována analogovou hodnoutou 0 V až 0,7 V, je potřeba před každý kanál dát DA převodník. Tento převodník je čtyř bitový, což nám umožní každou barvu vysílat v šestnácti intenzitách. Převodník je vytvořen pomocí šesti rezistorů o hodnotách $1\ \mathrm{k\Omega}$ a $2\ \mathrm{k\Omega}$. Přesné zapojení vidíme na schématu na Obrázek 4.4. Další dva vodiče VGA_HA a VGA_VS slouží pro vertikální a horizonální sinchonizaci. Tyto vodiče mají předřadné rezistory $100\ \Omega$. Maximální rozlišení obrazu může dosahovat 640x480 při frekvenci 25 MHz což nám dává obnovovací frekvenci 60 Hz. [9]



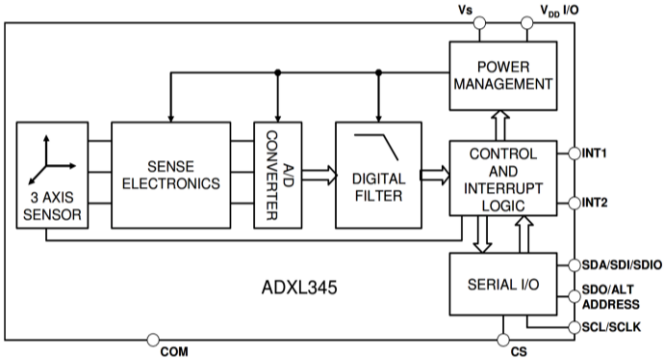
Obrázek 4.4 Zapojení VGA do FPGA [9, s. 35]

4.6 AKCELEROMETR

Těsně nad FPGA přibližně v polovině desky můžeme najít malý integrovaný obvod s názvem ADCL345, jedná se o tříosý akcelerometr, který nám umožnuje měřit náklon a zrychlení. K FPGA je připojen pomocí šesti pinů. Piny GSENZOR_INT1 a GSENZOR_INT2 slouží jako signalizace přerušení, kterou můžeme v ADCL345 nastavit. Další čtyři piny slouží pro sériovou komunikaci. V tomto případě leze použít, jak komunikaci pomocí SPI, tak komunikaci pomocí I2C. Nastavení požadované komunikace určuje pin GSENSOR_SC_n, který při log. 0 nastaví komunikaci SPI a naopak. Strukturu vnitřního blokového zapojení vidíme na Obrázek 4.5. [9]

Samotné ADCL345 ovládáme a čteme pomocí vnitřních osmibitových registrů, kterých obsahuje 58. K těmto registrům přistupujeme pomocí jejich adresy. První bit této adresy udává, jestli chceme číst nebo zapisovat. Maximální měřitelné zrychlení je 16 g při 13bit vzorkováním, to nám dává rozlišovací schopnost [12]

$$LBS = \frac{G_{max}}{2^{13}} = 1{,}95\ mg\ . \qquad 4.4.1$$



Obrázek 4.5 Vnitřní zapojení ADXL345 – [12, s. 1]

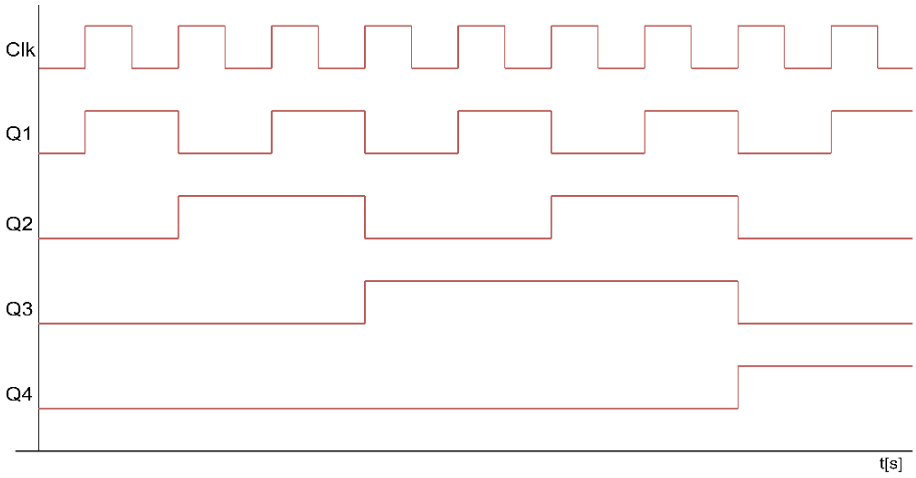
5 TVORBA DEMO KNIHOVEN

5.1 ČÍTAČ S VÝSTUPEM NA LED

Čítač je obecně popsán jako sekvenční logický obvod, který nám čítá vstupní pulsy. Čítač lze realizovat jak sestupný, tak i vzestupný. Výstupem je N-bitový výstup, který reprezentuje v binární soustavě počet načítaných pulsů. Dalším signálem vstupujícím do čítače je reset, který nám po sepnutí čítač nuluje. Pokud bychom nepožadovali nijak vysokou rychlost čítače, tak lze postavit jednoduše pomocí sedmi J-K obvodů zapojených do série jako asynchronní čítač. Toto zapojení ovšem není vhodné pro rychlejší obvody, kde zpoždění jednotlivých hradel se postupně sčítá, proto je vhodnější použít zapojení podle Obrázku 5.3. realizující synchronní čítač.

5.1.1 CÍLE

V tomto případě bude realizován 4-bit synchronní vzestupný čítač, u kterého vstupní pulsy budou simulovány pomocí tlačítka KEY0 s výstupem na led diody LEDR0-LEDR3. Reset bude ovládán tlačítkem KEY1. Dalším krokem bude vytvořit čtyřbitové číslo, které nám bude určovat do kolika má čítač maximálně počítat. Toto číslo bude realizováno binárním zápisem pomocí přepínačů SW0 až SW3. Jako poslední část bude vytvořen BCD převodník na 7-segmentový displej, pomocí kterého zobrazíme výstup z čítače v dekadickém zápise na displeji.



Obrázek 5.1 časový průběh čítače

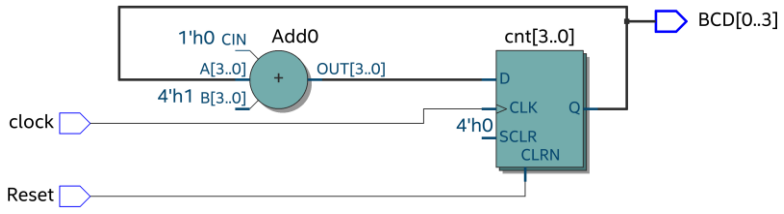
5.1.2 ČÍTAČ POMOCÍ VHDL

Čítač lze v jazyce VHDL realizovat více způsoby. V tomto případě je čítač realizován pomocí procesu. To znamená, že použijeme nejvyšší abstrakci. Jak již bylo popsáno v kapitole 2.3 VHDL, nejvyšší abstrakce nám umožní jednoduchou orientaci v kódu, ale do syntézy nemůžeme přesně vědět, jak bude vypadat finální zapojení. V tomto případě nás zapojení ani tak nemusí zajímat, jelikož se nepočítá s tím, že by vstupní pulsy byly rychlejší než 10 Hz. Pomocí Behaviorálního popisu můžeme čítač popsat například takto:

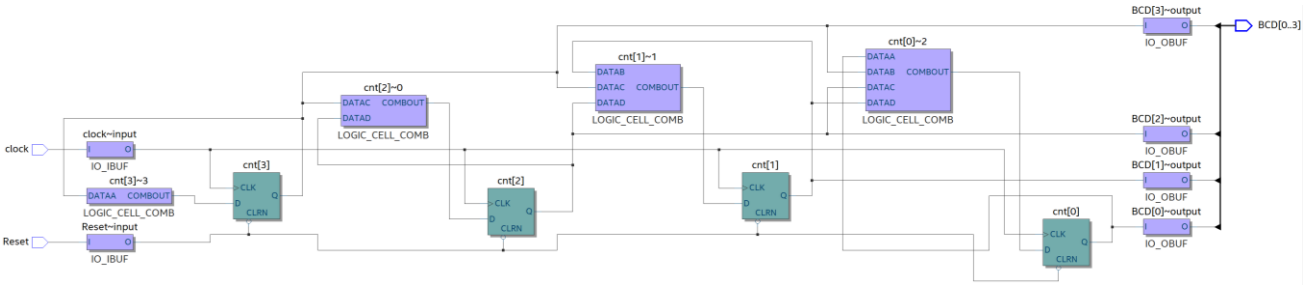
```
1  signal cnt : std_logic_vector(0 to 3);
2  begin
3  process(clock,Reset)begin
5     if (Reset='1') then
6           cnt <= "0000";
7     elsif rising_edge(clock) then
8           cnt <= cnt + 1;
9     end if;
10 end process;
```

Signál `clock` je napojen na naše vstupní pulsy. Signál `Reset` representuje naše resetovací tlačítko. Vektorový signál `cnt` reprezentuje výstupní binární číslo. Tento vektor může reprezentovat číslo, kvůli použití knihovny **ieee.std_logic_unsigned,** která nám v základu umožní pracovat s vektorem jakožto s číselnou hodnotou unsigned.

Po spuštění syntézy se můžeme podívat na vygenerované zapojení z pohledu RTL viz Obrázek 5.2, tento pohled, jak se můžeme dočíst v kapitole 3.2, nám zobrazí pouze přeložení našeho kódu bez ohledu na funkční bloky, které reálně můžeme využít. Tučná čára s popisem výstupu BCD[0..3] nám reprezentuje čtyřbitovou výstupní sběrnici. Jak si můžeme všimnout analýza obvodu nám vytvořila pouze sčítačku s pamětí typu D. Technologické zobrazení nám už ukazuje reálné vnitřní zapojení. Toto zapojení viz Obrázek 5.3 se podobá synchronnímu čítači. Syntéza obvodu je nastavena na poloviční optimalizaci rychlosti obvodu. Vzhledem k jednoduchosti tohoto zapojení není v podstatě žádný problém přidat pár obvodů navíc. Proto bylo zvoleno právě synchronní zapojení. Toto zapojení obsadí celkově pouze 5 logických elementů, což je vzhledem k 50 000 možných zanedbatelné množství.



Obrázek 5.2 RTL zobrazení čítače

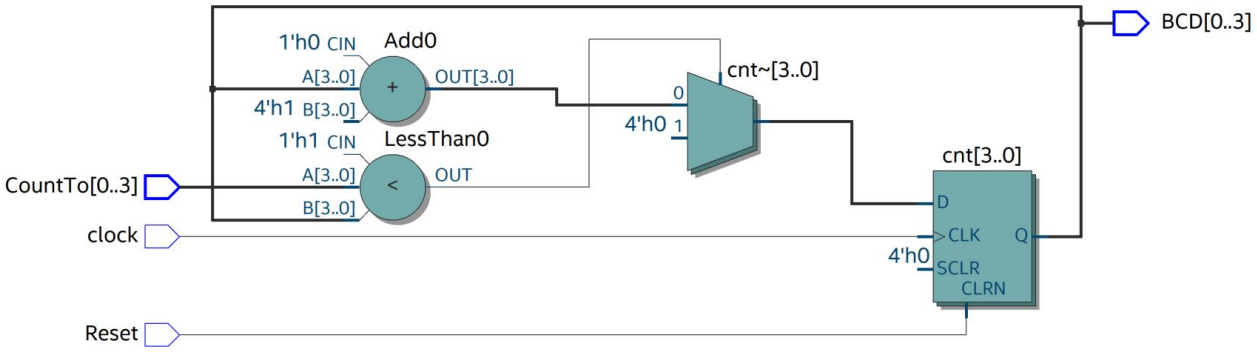


Obrázek 5.3 technologické zobrazení čítače

5.1.3 ČÍTAČ DO URČENÉ HODNOTY

Čítač, který umí počítat do předem určené hodnoty, se hodí například do děličky frekvencí. Realizace je jednoduchá, stačí vytvořit nový signál v toto případě s názvem `CountTo`. Tento signál je možné nastavit přímo jako konstantu, nebo jej lze měnit ručně pomocí čtveřice přepínačů. Tímto lze v binárním kódu nastavit číslo, při kterém se má čítač vynulovat.

```
1  begin
2  process(clock,Reset)begin
3     if (Reset='1') then
4           cnt <= "0000";
5     elsif rising_edge(clock) then
6           if(cnt >= CountTo) then
7                 cnt <= "0000";
8           else
9                 cnt <= cnt + 1;
10          end if;
11    end if;
12 end process;
```



Obrázek 5.4 RTL Zobrazení čítače do určité hodnoty

5.1.4 REALIZACE NA DESCE TERASIC

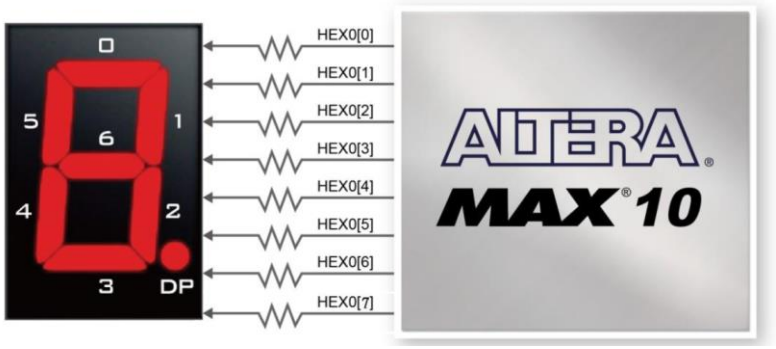
Funkce tohoto čítače je realizována pomocí čtyř led diod (LEDR0 – LEDR3), kde LEDR0 symbolizuje nejnižší bit, který se přepíná jednou za hodinový takt. Jak již bylo dříve zmíněno, vstupní signál je realizován pomocí tlačítka, konkrétně KEY0. Vstupní omezující signál `CountTo` je ovládán pomocí čtveřice přepínačů (SW0 – SW3).

Výstupní hodnoty odpovídají hodnotám na Obrázek 5.1 časový průběh čítače. Čítač reaguje na změnu maximální hodnoty až po dalším hodinovém taktu. Ve výsledku to znamená, že při změně hodnoty `CountTo` na hodnotu nižší dojde ke kontrole až následující takt. Z tohoto důvodu může dojít ke kolizi, kdy `CountTo` ukazuje nižší číslo, než má napočítán čítač.

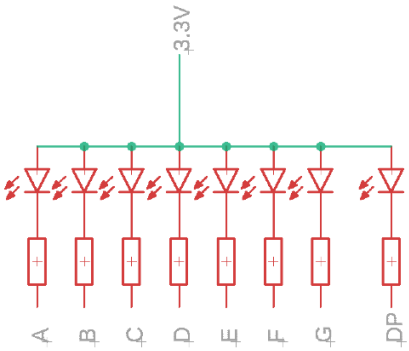
5.2 OVLÁDÁNÍ SEDMISEGMENTOVÉHO DISPLEJE

Sedmisegmentový displej je v dnešní době jeden z nejjednodušších a velmi využívaných displejů pro komunikaci přístroje s uživatelem. Tento displej je složen z osmi led diod (sedm na číslici, plus jedna jako tečka). Tyto displeje se vyrábí v mnoha variantách. Dají se sehnat buď jednotlivě pouze s jednou číslicí, nebo také jako řádek, například čtyř číslic za sebou. Další rozdíly jsou v samotném vnitřním zapojení. Displeje můžeme najít buďto se společnou katodou nebo anodou. Na přípravku DE10-Lite nalezneme displej, který obsahuje šest sedmisegmentů které mají společnou katodu.

U těchto displejů můžeme ovládat všechny segmenty zvlášť. To je pro nás u mnoha aplikací zbytečné, jelikož potřebujeme na displeji většinou vykreslovat pouze hexadecimální číslice 0 - F plus prázdný znak zobrazený jako pomlčka. Ovládání všech segmentů je v tomto případě neefektivní, jelikož pro vykreslení všech těchto znaků potřebujeme pouze 16 kombinací signálů, které zvládneme vytvořit pomocí čtyř signálů. Z tohoto důvodu je mnohem jednodušší vytvořit převodník.



Obrázek 5.5 Nákres zapojení z manuálu [9 s. 28]



Obrázek 5.6 Zapojení segmentů do FPGA

5.2.1 REALIZACE PŘEVODNÍKU

Převodník jako vstup využívá kód BCD (čtyř signálová sběrnice), který bude v binárním vyjádření reprezentovat čísla, která budeme chtít na displeji vykreslit. Dále tento převodník připojíme na čítač popsaný v kapitole 5.1, pomocí kterého ověříme jeho funkčnost.

Samotný převodník lze realizovat několika způsoby s různou úrovní abstrakce. U takto snadných úloh by se dal využít i schématický editor, kde bychom s využitím Karnaughových map sestavili schéma. Další možností je vzniklé rovnice zjištěné z Karnaughových map vepsat jako popis závislostí vstupů a výstupů. V tomto případě byla použita nejvyšší úroveň abstrakce, kde je popsáno, jak má vypadat výstup při daném vstupu. Tato varianta je nejjednodušší a také nejvíce přehledná. U neplatných stavů (10-15) nastavíme prázdný znak. Pro realizaci je využita funkce `with select when`, která při určitém předem definovaném vstupu nastaví předem definovaný výstup.

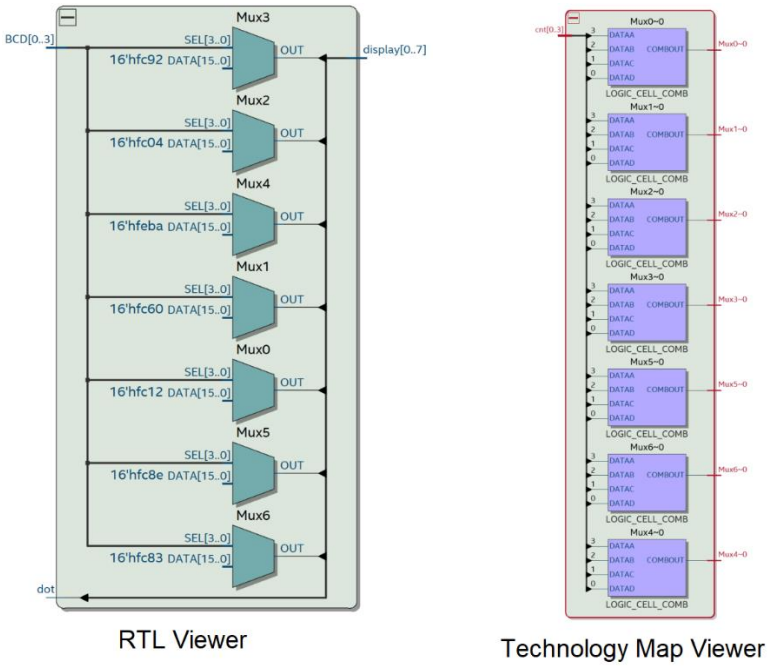
```
1 with BCD select display(0 to 6) <=
2     "0000001" when "0000",
3     "1001111" when "0001",
4     "0010010" when "0010",
5     "0000110" when "0011",
6     "1001100" when "0100",
7     "0100100" when "0101",
8     "0100000" when "0110",
9     "0001111" when "0111",
10    "0000000" when "1000",
11    "0000100" when "1001",
12    "1111111" when others ;

display(7) <= dot;
```

V kódu si můžeme všimnout, že veškeré výstupy do displeje jsou realizovány invertně. Důvodem je inverzní zapojení samotných led diod, které jsou pevně připojeny na 3,3 V. Z toho důvodu při připojení logické jedničky dioda svítit přestane a naopak. Tento kód vykresluje pouze čísla od 0 do 9, při jakékoliv jiné hodnotě nezobrazí nic. Tečka je realizována separátně kvůli jednoduššímu ovládání.

5.2.2 ZÁVĚR ČÍTAČE S PŘEVODNÍKEM

Po syntéze kódu v RTL zobrazení na Obrázek 5.7 vidíme, že samotný převodník byl realizován pomocí sedmi multiplexorů. Každý multiplexor ovládá jeden kanál. Do všech multiplexorů jsou přivedena data, ze kterých je vybírána hodnota pro danou vstupní kombinaci. Při pohledu na Technology map Viewer vidíme, že převod je realizován pomocí Logických tabulek. Stejným způsobem by byl převodník realizován i při ostatních popisech převodníku.



Obrázek 5.7 Převodník BCD na sedmisegment

5.3 STOPKY VYUŽÍVAJÍCÍ PLL

Měření přesného času je dnes nedílnou součásti každodenního života. Čas měříme mnoha způsoby. Nejčastěji to bývá pomocí hodin, kde u většiny z nich jsme schopni měřit s přesností na minuty. Dalším způsobem měření času, respektive časových úseků, jsou stopky, které dokážou měřit klidně na setiny sekundy. Stopky mohou být realizovány jako interní komponenta nějakého většího celku, jako je třeba radar. Nebo mohou být jako samostatné zařízení, jako například sportovní stopky.

Tato část je zaměřena na realizaci sportovních stopek s přesností na desetinu sekundy, s délkou časového úseku maximálně šedesát sekund. Tyto stopky budou obsahovat také dvě základní tlačítka. Tlačítko STOP/START spouští a pozastavuje počítání času a tlačítko reset, které hodnotu stopek vynuluje. Časový údaj bude vypsán na trojici segmentových displejů.

5.3.1 REALIZACE STOPEK

Stopky jsou již komplexnější zařízení, které je potřeba rozdělit na několik částí. První částí je zdroj hodinového signálu. Každé zařízení, které počítá čas, musí mít předem daný nejmenší časový úsek, který měří. V tomto případě se jedná o desetinu sekundy. Druhou částí je soustava čítačů, které počítají počet příchozích signálů. Jako výstup mají tyto čítače BCD kódy pro dané číslice. Třetí částí je převodník z BCD na segmentový displej, stejný jako v kapitole 5.2.1. Poslední částí je ovládání stopek pomocí tlačítek.

5.3.1.1 HODINOVÝ SIGNÁL

Pro realizaci stopek s přesností na desetiny sekundy je potřeba zajistit hodinový takt 10 Hz. DE10-Lite však nabízí pouze dva hodinové takty o frekvenci 10 MHz a 50 MHz. Proto je potřeba využít fázový závěs (PLL). Ovšem i samotné PLL nedokáže tvořit nekonečně dlouhé časové úseky. Nejmenší hodinový takt, který zvládne vytvořit je 1200 Hz. Proto je potřeba realizovat také děličku frekvencí. Tato dělička bude realizována pomocí podobného čítače jako v kapitole 5.1.2. Pro zjednodušení vytvoříme čítač, který nám zajistí střídu signálu 1:1. Při využití takovéhoto děliče je potřeba přesně vypočítat požadovanou frekvenci z PLL. Pomocí rovnice 5.1, kde f_1 symbolizuje požadovanou hodnotu, tedy 10 Hz. Hodnotu n můžeme zjistit iterativně, kdy budeme požadovat výstupní hodnotu $f_{PLL} > 1200\ Hz$.

$$f_{PLL} = f_1 \cdot 2^n = 10 \cdot 2^7 Hz = 1280\ Hz \qquad 5.1$$

Z výpočtu vyšlo, že při využití sedmibitového čítače potřebujeme výstupní frekvenci z PLL 1280 Hz. Jako vstupní frekvenci do PLL budeme používat 10MHz hodinový signál. Při zadání těchto parametrů do programu Quartus Prime, dostaneme výsledné parametry fázového závěsu. Ten nejprve hodinový signál vynásobí dvakrát na hodnotu 20 MHz, a následně vydělí 15625 na požadovaných $f_{PLL} = 1280\ Hz$.

5.3.1.2 SOUSTAVA ČÍTAČŮ

Tato soustava je složena z trojice čítačů, napojených na sebe, podobných jako v kapitole 5.1.3, kde signál reset při aktivaci vynuluje všechny čítače. Vstupní signál `CountTo` je přímo vepsán do podmínky v čítači. První dva z těchto čítačů fungují také jako děličky frekvencí, které dělí deseti. Tyto děličky vytvářejí dva další signály o frekvencích $f_1 = 1Hz$ a $f_2 = 0{,}1\ Hz$. Třetí čítač je nastaven tak, aby počítal pouze do šesti, tím zajistíme, aby se stopky při požadovaných šedesáti sekundách samy restartovaly.

5.3.1.3 OVLÁDÁNÍ

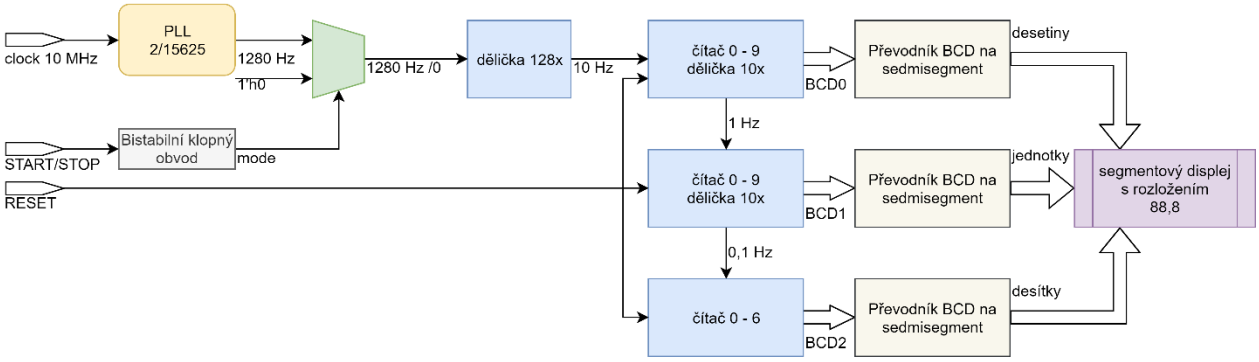
Ovládání je realizováno pomocí dvou tlačítek RESET a START/STOP. Tlačítko reset není nijak komplikované, jelikož při zmáčknutí tohoto tlačítka signál reset vynuluje veškeré vnitřní čítače. Z tohoto důvodu není potřeba přidávat žádnou zvláštní logiku.

Tlačítko START/STOP funguje na bázi odpojování a připojování signálu f_{PLL}. Jelikož toto tlačítko funguje ve dvou módech, tak je potřeba přidat logiku. Tato logika se dá jednoduše popsat kódem:

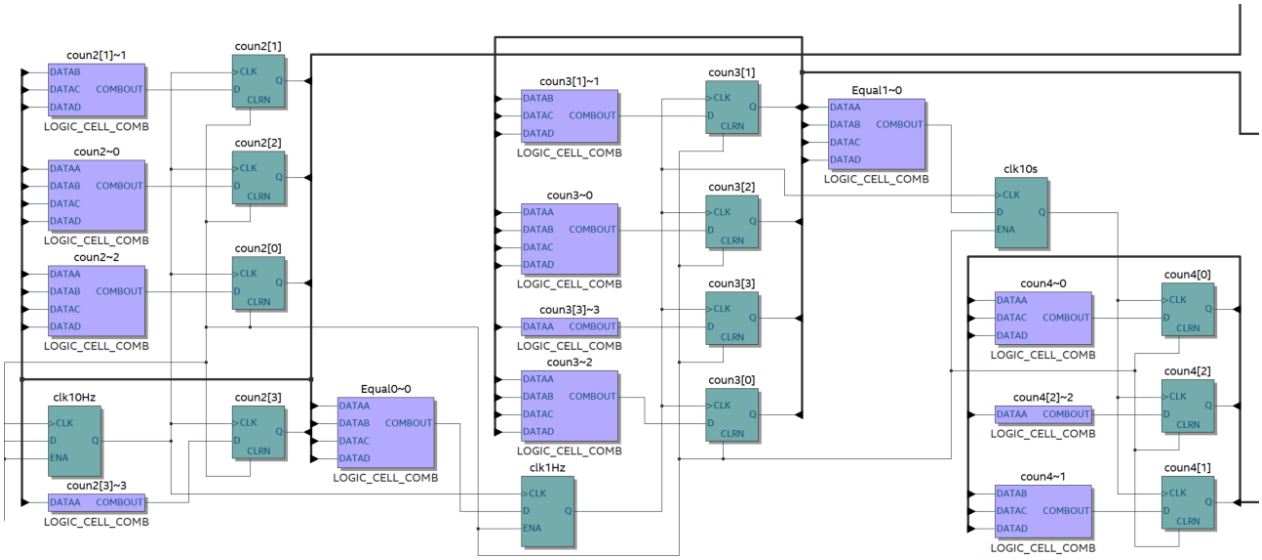
```
1  process(start)
2  begin
3     if (start'event and start='0') then
4           mode <= NOT mode;
5     end if;
6  end process;
7  with mode select clk1280Hz <=
8     '0' when '0',
9     clk1280HzOut when '1';
```

5.3.2 ZAPOJENÍ

Celkové zapojení stopek je vidět pomocí blokového schéma na Obrázku 5.8. Největší částí celých stopek je soustava čítačů. Na Obrázku 5.9 čítače s výstupem na displej. vidíme trojici čítačů, které mají výstup k převodníkům, které by se nacházely vpravo.



Obrázek 5.8 Blokové schéma stopek



Obrázek 5.9 Čítače s výstupem na displej.

5.3.3 ZÁVĚR

Samotná logika stopek zabírá v FPGA pouze 45 logických elementů ze 49760 možných. To přibližně odpovídá využití na 0,09 %, což je velmi zanedbatelné množství. Obsazenost pinů je 27 ze 360. Samotný segmentový displej požaduje 24 pinů. V programu Quartus Prime se můžeme dočíst, že využíváme 25 % PLL. Tato hodnota odpovídá pouze počtu bloků PLL, které právě využíváme. Pokud ale tuto hodnotu přepočítáme na obsazenost výstupních hodinových signálů, tak se dostaneme k číslu 6,25 %.

Tyto stopky umějí počítat pouze do šedesáti vteřin s přesností na desetinu vteřiny, ovšem je možné přidat další část čítačů pro realizaci minut či hodin. Pokud bychom chtěli vytvořit stopky s využitím všech segmentů displeje, mohly by tyto stopky fungovat až do deseti hodin neboli do 9:60:60,9.

5.4 SYNTEZÁTOR VYUŽÍVAJÍCÍ PLL

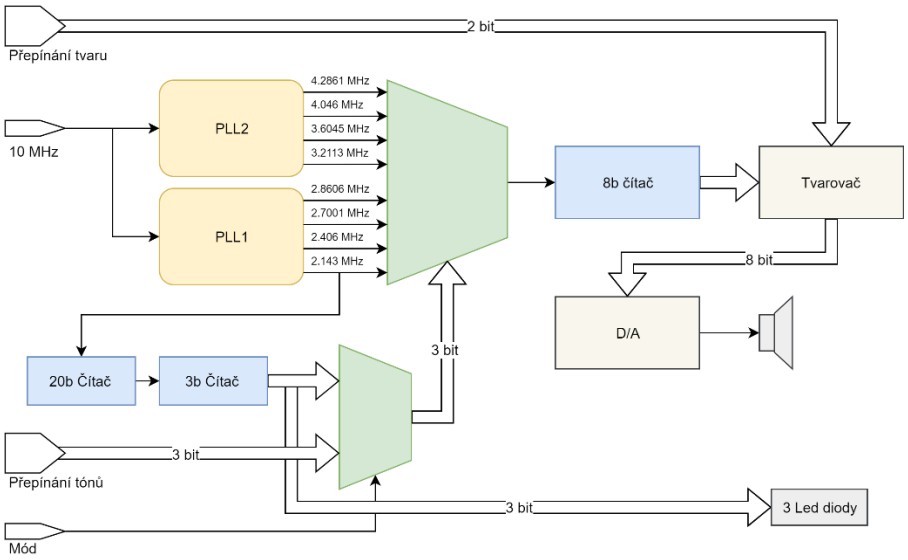
Syntezátor je hudební nástroj, který byl vynalezen v roce 1895. Jedná se o nástroj, který generuje zvuk pomocí syntézy. Syntezátory mohou fungovat na analogové bázi, kdy původní signál, vytvořený generátorem, prochází různými filtry, nebo je míchán s dalšími zvuky. U syntezátorů digitálních je požadovaný zvuk většinou tvořen softwarově a následně interpretován pomocí zvukové karty nebo určitého D/A převodníku.

Tato část je zaměřena na tvorbu syntezátoru s osmibitovým zvukovým výstupem, který bude přehrávat jednotlivé tóny stupnice C od C1 do C2. Tvary signálů budou moci být nastaveny buďto jako obdélník, nebo trojúhelník. Výstupní komponentou bude přídavný D/A převodník s Piezo reproduktorem. Druhý výstup bude optický, na kterém bude vidět výstup z čítače, přepínající jednotlivé tóny při automatickém módu.

5.4.1 REALIZACE SYNTEZÁTORU

Základem každého syntezátoru je generátor signálů. V tomto zapojení jsou signály generovány pomocí dvojice PLL s celkově osmi frekvencemi. Tyto signály jsou přivedeny do multiplexoru, který vybere jeden signál, který bude přehráván. Dále signál pokračuje do osmibitového čítače. Tento čítač načítá svou maximální hodnotu vždy každou jednu periodu výstupního tónu. Díky tomu lze tón různě modulovat.

Dalším důležitým prvkem je řídící část. Ovládání funguje ve dvou módech. První mód je ruční výběr výstupních tónů, ve kterém výstupní tón ovládáme pomocí pěti přepínačů. Při aktivování automatického módu je přehrávána stupnice C, ve které můžeme měnit pouze tvar tónu. Blokové schéma celého syntezátoru je na Obrázku 5.10.



Obrázek 5.10 Blokové schéma zapojení syntezátoru

5.4.2 NASTAVENÍ PLL

Základem celého syntezátoru je generátor, pomocí kterého se vytvářejí frekvence, odpovídající požadovaným tónům. V tomto případě se jedná o dvojici fázových závěsů, které dohromady generují osm na sobě nezávislých signálů. Vstup do těchto dvou fázových závěsů je napojen na frekvenci $f_{in} = 10\ MHz$. Vzhledem k požadavku na tvarování výstupního signálu, je potřeba, aby výstupní signál byl násobně vyšší než požadované frekvence. V tomto případě se jedná o výstupní průběh, jehož jedna perioda bude mít 8b vzorkování. To znamená, že se bude jednat o periodu složenou z 256 vzorků. Při navrhování PLL tedy je potřeba, aby výstupní frekvence každého tónu byla 256násobně vyšší. Tyto frekvence jsou v Tabulce 5.1.

Tabulka 5.1 Potřebné hodnoty frekvencí k jednotlivým tónům.

Název tónu	$f_{tónu}\ [Hz]$	$f_{PLL} = f_{tónu} \cdot 256\ [MHz]$
C1	262	2,143
D1	294	2,406
E1	330	2,4001
F1	349	2,8606
G1	392	3,2113
A1	440	3,6045
H1	494	4,046
C2	523	4,2861

Výstupní frekvence je vybírána pomocí multiplexoru, ze kterého je signál dále zpracován pomocí osmibitového čítače. V tomto projektu by mohl být čítač již součástí tvarovače. Je to proto, že požadované dva tvary při osmibitovém kvantování tvarovače dokážeme přímo pomocí čítače vytvořit. Pilový signál je čistý výstup čítače, který je následně přiveden do D/A převodníku. Obdélník je ve své

podstatě řízen pomocí nejvyššího bitu čítače, takže lze pouze vytvořit jednoduché zapojení, kdy při hodnotě nejvyššího bitu nastavíme D/A převodníku hodnotu maximální a při nule naopak hodnotu minimální. Pro přehlednost je tvarovač napsán zvlášť. Jako vstup výběru tvaru je zvolen dvoubitový signál. Tímto máme možnost při nastavení `shape_sel` na hodnotu „00“ k vypnutí přehrávání.

```
process(act_clk,rst,shape_sel)
    begin
          if shape_sel = "00" then
                func_out <="00000000";
          elsif shape_sel = "01" then
                func_out <= tone_count;
          elsif shape_sel = "10" then
                if tone_count < "10000000" then
                      func_out <= "11111111";
                else
                      func_out <= "00000000";
                end if;
          end if;
    end process;
end Generator;
```

Další částí je přehrávání stupnice. K přehrávání stupnice je potřeba vytvořit další hodinový signál, který musí být o frekvenci, která nám stačí k odlišení jednotlivých tónů od sebe. Ideální přesná frekvence neexistuje, tudíž tuto frekvenci lze přizpůsobit co nejvíce k jednoduchosti zapojení. Aby se nezahlcoval zbytečně další PLL, tak je tato frekvence vytvořena pomocí děličky z nejnižší frekvence, která již existuje, přesněji 2,143 MHz. Tento hodinový signál je dále přiveden do 22b čítače. Tento čítač je rozdělen na dvě části, 0-19 bit slouží čistě jako dělička frekvencí s koeficientem dělení $2^{20} = 1048576$. Z toho vyplývá, že při přivedení 2,143 MHz získáme výstup o frekvenci přibližně 2 Hz. Další trojice bitů 20-22 v tomto čítači slouží jako tónová adresace. Pro názornost je zobrazeno horních pět bitů na (LEDR0 – LEDR4).

K ovládání tohoto syntezátoru slouží jak přepínače, tak tlačítka. Tlačítko KEY0 slouží jako reset, který nám vynuluje veškeré čítače. Pomocí přepínačů SW0-SW1 přepínáme tvar výstupního signálu mezi obdélníkem a pilou. Popřípadě můžeme pomocí kombinace 00 přehrávání vypnout. Přepínač SW2 slouží pro přepínání módů mezi přehráváním stupnice a přehráváním jednotlivých tónů. Poslední část ovládání slouží pro přepínání tónů v tónovém režimu. K tomu slouží kombinace přepínačů SW9-SW7. Toto přepínání funguje na bázi binární kombinace těchto přepínačů, kde SW9 je nejnižší bit, tudíž jsme schopni vytvořit až 8 kombinací.

Jako výstupní periferie je použit piezo reproduktor, který je napojen na 8b D/A převodník. Jelikož tento převodník není součásti desky, tak ho je potřeba připojit. Pro připojení jsou použity výstupní piny na patici GPIO přesněji GPIO_(2) – GPIO_(9). Jelikož D/A převodník potřebuje také napájení, je z této patice využíváno i napájení 5V a GND.

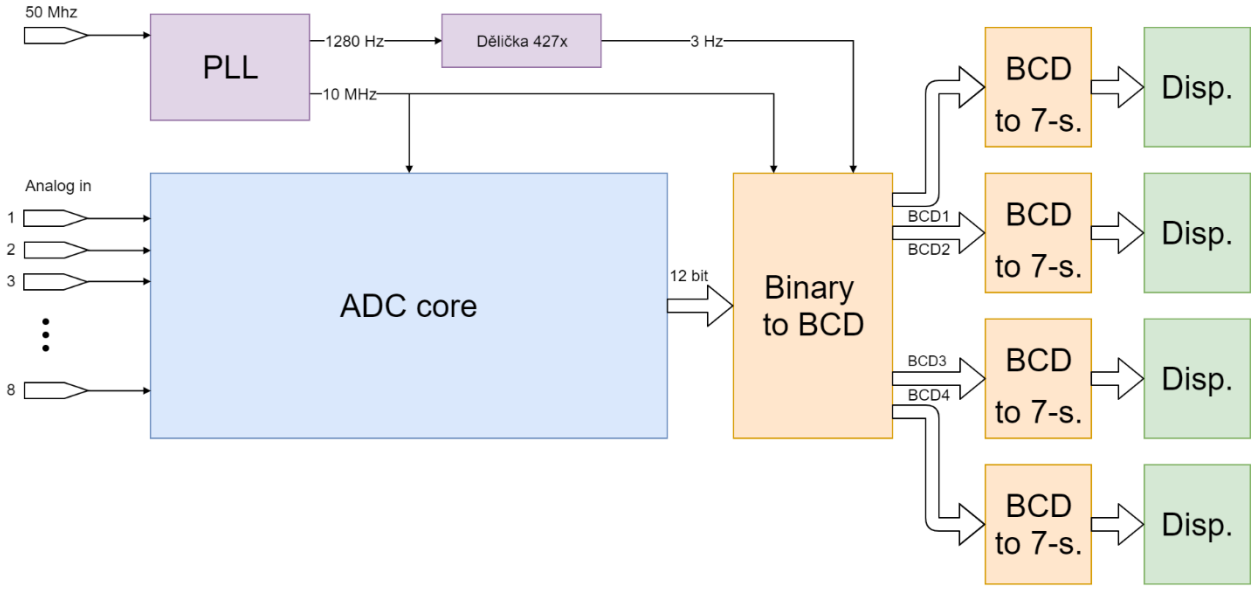
5.4.3 ZÁVĚR

Ač se zdá, že je toto zapojení již složitější a musí zabrat nějaký větší počet logických elementů, tak tomu tak není. Toto zapojení využívá pouze 70 Total logic elements, což není více než 0,15 % obsazenosti elementů. Obsazenost pinů je v tomto případě 21, to nám dává 6 % obsazenost. U PLL jsme tímto zabrali již polovinu. Z toho důvodu byl signál pro přehrávání stupnice vytvořen již pomocí děliček frekvencí. Pokud bychom tento signál tvořili pomocí dalšího PLL, tak bychom již obsadili 75 % veškerých PLL.

5.5 VYUŽITÍ ADC PRO REALIZACI VOLTMETRU

Dnešní moderní technika pracuje převážně na digitální úrovni, to je ovšem velký problém pokud tato technika má reagovat na okolní analogové signály. K tomuto účelu je potřeba využít A/D převodník, který vstupní analogový signál převede na signál digitální. Už z principu je jasné, že vstupní signál bude muset být při převodu nějakým způsobem kvantován, a proto nedosáhneme přesné hodnoty, ale jen hodnoty přibližné s absolutní nejistotou μ_u. A/D převodníky nalezneme dnes skoro v každém zařízení, které je digitální a zpracovává okolní signály. Příkladem mohou být teplotní senzory, mikrofon, senzory na měření vlhkosti vzduchu a mnoho dalších.

V této části se zaměříme na využití ADC core, které je integrováno přímo v FPGA. Cílem je vytvořit převodník s výpisem na čtveřici sedmisegmentových displejů. Struktura tohoto zapojení je znázorněna na Obrázku 5.11. Jádrem celého zapojení je samotné ADC core. Výstupy jsou realizovány binárně, a proto musíme vytvořit převodník z binárního kódu na BCD. Dále je potřeba realizovat trojici převodníků BCD na sedmisegment. Jedna z nejdůležitějších částí je hodinový signál. V tomto případě budeme využívat tři hodinové signály. Jeden bude synchronizace celé logiky na frekvenci 10 MHz. Druhý bude sloužit pouze jako mezisignál, který bude následně připojen do děličky. Poslední signál nám bude obnovovat displeje na frekvenci 3 Hz.



Obrázek 5.11 Blokové schéma zapojení AD převodníku s výpisem na displej

5.5.1 REALIZACE VOLTMETRU

Jádrem celého voltmetru je samotný převodník z analogového vstupu na digitální výstup. Tento převodník má osm různých analogových vstupů. Tyto vstupy nejsou převáděny najednou, ale jsou přivedeny do multiplexoru. Tento multiplexor podle vstupní adresy vybere jeden vstup, který se následně začne převádět. Výstupem celého převodu je 12bitová hodnota, která reprezentuje vstupní hodnotu napětí. Vzhledem k tomu, že převodník pracuje na principu postupné aproximace, tak převod není okamžitý a trvá nějakou dobu. Proto převodník mimo vzorky má výstupní porty, které nám oznamují konec převodu. Tyto signály je třeba sledovat a data zpracovávat až po oznámení konce převodu.

Jako každý digitální obvod potřebuje i tento obvod hodinový signál. V tomto obvodu jsou zapotřebí dva základní hodinové takty. První, který běží na frekvenci 10 MHz, udržuje veškeré komponenty v synchronizaci. Dále také zajišťuje fungování čítače uvnitř A/D převodníku, který zajišťuje fungování postupné aproximace. Dále je potřeba zajistit obnovovací kmitočet displeje fungující přibližně na 3 Hz. Tato frekvence nemusí být naprosto přesná, a proto stačí vytvořit signál s kmitočtem, který je této frekvenci blízký. Pro účely prezentace využití PLL vytvoříme obě tyto frekvence pomocí vstupního hodinového signálu 50 MHz. Toto zapojení se může hodit v případě, kdy je velmi důležité mít stabilní kmitočet na přesné hodnotě, jelikož dochází k menšímu rušení okolím.

Stejně jako v kapitole **Chyba! Nenalezen zdroj odkazů.** nemůžeme signál 3 Hz vytvořit přímo pomocí PLL. V tomto případě budeme postupovat naopak. Nejprve je potřeba vytvořit nejnižší kmitočet, jaký lze vytvořit pomocí PLL. V tomto případě jde o signál s frekvencí 1280 Hz. Tento signál je následně přiveden do děličky 427x, která je navržena tak, aby se k požadovaným 3 Hz pouze přiblížila. Výsledná frekvence je tedy 2,998 Hz. Tato odchylka od frekvence požadované je pro lidské oko zanedbatelná.

ADC nám dává hodnoty, které jsou vyjádřeny pomocí 12bitového čísla. Tato hodnota odpovídá kolikrát musíme vynásobit nejmenší dílek, abychom dostali napětí ve voltech. Pro přepočet je tedy potřeba využít rovnici 5.2. V této rovnici je referenční hodnota vynásobena tisícem z důvodu vynechání desetinných čísel a pozdějšího jednoduššího převádění. Hodnota V_{ref} je referenční napětí ADC_Core, které je nastaveno na interní napětí $2{,}5\ V$. Kvůli využití předřadného vnitřního děliče napětí můžeme i s touto referenční hodnotou měřit až do $3\ V$. Tímto krokem ovšem zvětšíme nejmenší dílek na dvojnásobek, proto je nutné celou hodnotu vynásobit dvakrát.

$$V_{in} = ADC_{Out} \cdot \frac{V_{ref} \cdot 1000 \cdot 2}{4096} \qquad 5.2$$

Výsledné číslo je ovšem v binárním tvaru, který je sice vhodný pro přenos v digitální technice, ale už ne tak vhodný pro vyjádření pomocí segmentových displejů, u kterých používáme BCD vstup. Pro převod mezi těmito soustavami slouží převodník, který je na Obrázku 5.11 označen jako **Binary to BCD**. Převodník funguje na principu postupného dělení se zbytkem. Zápis pomocí VHDL vypadá takto:

```
vstup := vstup*5000;
vstup := vstup/4096;
vysledek := vstup/1000;
BCD3 <= std_logic_vector(to_unsigned(vysledek, BCD3'length));
vstup := vstup - 1000*vysledek;
vysledek := vstup /100;
BCD2 <= std_logic_vector(to_unsigned(vysledek, BCD2'length));
vstup := vstup - 100*vysledek;
vysledek := vstup /10;
BCD1 <= std_logic_vector(to_unsigned(vysledek, BCD1'length));
vstup := vstup - 10*vysledek;
BCD0 <= std_logic_vector(to_unsigned(vstup, BCD0'length));
```

Jako výstupy z tohoto převodníku jsou hodnoty BCD0 – BCD3, ve kterých jsou uloženy příslušné hodnoty v dekadickém zápisu.

Tento voltmetr také umožnuje zobrazovat hodnotu nejnižší, nejvyšší, nebo popřípadě u sinusového průběhu hodnotu střední. Tyto hodnoty jsou počítány každý hodinový takt, tedy s frekvencí 10 MHz. Proto nejsou nijak ovlivněny hodnotami zobrazenými. Snímání maxima a minima je vytvořeno jednoduše pomocí bufferu, který je vždy porovnán s aktuální převedenou hodnotou. Tyto hodnoty lze jednoduše vynulovat pomocí signálu reset. Příkladem je tento kód, který zjišťuje hodnotu minimální.

```
process(clk10MHz, reset)
begin
   if rising_edge(clk10MHz) then
        if (reset = '0') then
             sampleL <= "000000000000";
        elsif (sample < sampleL or sampleL = "000000000000") then
             sampleL <= sample;
        end if;
   end if;
end process;
```

Střední hodnota je tvořena pomocí průměrování s každou příchozí hodnotou. Tento výpočet samozřejmě není přesný a zároveň trvá relativně dlouho než se ustálí. Na druhou stranu je jednoduchý a nezabírá více signálových linek než výpočet maxima a minima. Ovládání tohoto výpočtu je stejné jako při předchozí ukázce. Proto tento příklad je uvedena ukázka výpočtu pomocí VHDL.

```
soucet := to_integer(unsigned(sample)) + to_integer(unsigned(sampleS));
soucet := soucet/2;
sampleS <= std_logic_vector(to_unsigned(soucet, sampleS'length));
```

Jako výsledek je 12bitová sběrnice `sampleS`, která obsahuje průměrnou hodnotu všech vzorků pracující na klouzavém průměrování. Mezi těmito třemi hodnotami plus hodnotou aktuální je následně přepínáno a jsou pomocí předchozího převodníku vypisovány na displeji.

Ovládání celého převodníku je mírně komplikovanější než u předchozích zapojení. Kromě ovládání, výstupních hodnot a přepínání mezi vstupními kanály, je potřeba zajistit ovládání samotného ADC Core. Převodník využívá kromě hodinového portu několik důležitých portů pro převod.

Tabulka 5.2 Vybrané důležité porty pro převod. [8 s.19]

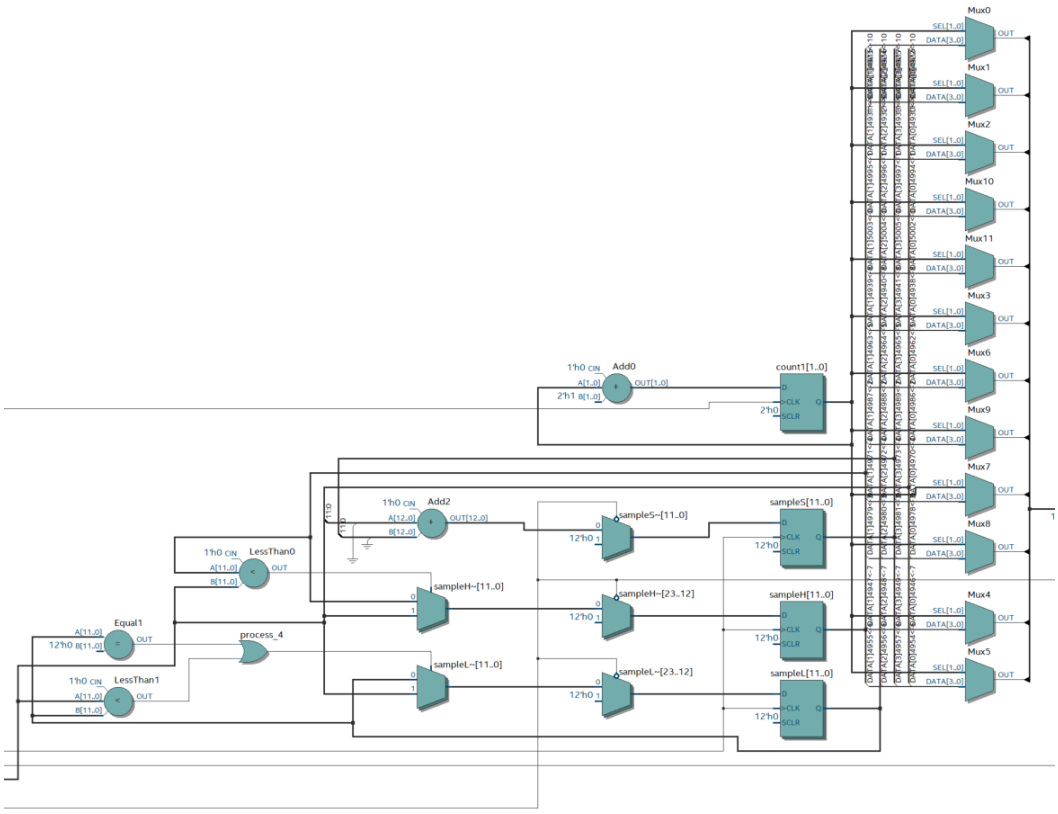
Název portu	typ	funkce
command_valid	In	Při hodnotě 1 povoluje ovládání ADC_Core
command_channel[4:0]	In	Nastavení vstupu, který má být převáděn
command_startofpackt	In	Zahájení převádění
command_endofpacket	In	Ukončení převádění
command_ready	out	Přijímání instrukcí.
response_valid	out	Hodnoty připraveny ke čtení
response_data[11:0]	out	Převedení hodnoty
response_channel[4:0]	out	Odpovídající adresa převedeného kanálu.

Tento převodník používá i další hodnoty, které pro tuto úlohu není potřeba využívat. Úkolem je převodník spustit a nechat ho spuštěný po celou dobu převodu. Toho lze dosáhnout poměrně jednoduše pomocí spuštění portu **command_valid** a následně **command_startofpack**. Převodník ovšem nelze ovládat pořád. Je potřeba vždy vyčkat na port **command_ready**. Pouze pokud je tento port aktivní, lze přenastavit vstupní kanál, popřípadě zadávat jiné příkazy. Dokončení převodu nastává, pokud je souběžně sepnut port **command_ready** a **response_valid**. Tato data se nacházejí na sběrnici **response_data[11:0]**, tato data odpovídají hodnotě, která se nachází na vstupu s adresou **response_channel[4:0]**. V průběhu převodu mezi porty, které byly zmíněny výše, jsou tyto sběrnice vynulovány. Z tohoto důvodu je potřeba vytvořit nějaký buffer, který bude hodnoty udržovat.

Ovládání celkového zapojení je již jednoduché, obsahuje čtyři základní prvky. Tlačítko KEY1 funguje jako zapínání a vypínání vypisování. Proto pomocí tohoto tlačítka lze také zamrazit právě převedenou hodnotu. Tlačítko KEY0 slouží pro přepínání módů mezi hodnotou minimální, maximální, průměrnou a aktuální. Reset je v tomto případě realizován pomocí přepínače SW9. Pokud tento přepínač přepneme do hodnoty 1, celé zapojení se vynuluje a zůstane vynulováno, dokud přepínač není přepnut zpátky. Posledním prvkem jsou přepínače SW0–SW4, pomocí kterých se vybírá převáděný kanál. K zobrazování hodnot je využita čtveřice segmentových displejů, které vypisují požadovanou hodnotu. LEDR0 blikne vždy, když je obnovena vypsaná hodnota na displeji.

5.5.2 ZÁVĚR VOLTMETRU

Realizace voltmetru již potřebovala více komponent a složitější ovládání, toto je vidět již na RTL Viewer. Ovšem při využití ve větším celku bychom například neřešili převádění na displeje, ale sběrnici s daty dále zpracovávali jinak. Toto zapojení využívá 1595 logických elementů což odpovídá přibližně 3 % využití. Vzhledem k složitosti toto zapojení využívá tři 9bitové násobičky z 288 možných, což odpovídá přibližně 1 %. Pro toto zapojení byly využity také paměťové bloky. Veškeré tyto složitější komponenty jsou využity pro realizaci dělení a násobení. V tomto případě by se možná více vyplatilo vytvořit Aritmeticko-logickou jednotku, která by veškeré tyto výpočty řešila sama. Ovšem pokud FPGA nepotřebujeme využít na nic jiného než tento převodník, tak je to zbytečné, a toto zapojení je ideální variantou.



Obrázek 5.12 RTL Viewer: část přepínání hodnot maximu, minimum, průměr a aktuální

5.6 AKCELEROMETR

Akcelerometry dnes již najdeme v každém mobilu, tabletu nebo třeba i hodinkách. Pomocí akcelerometru se nemusí snímat jen zrychlení, ale také náklon nebo určitý druh pohybu. V dnešní době existuje již spousta aplikací, které akcelerometry využívají například k ovládání. Na desce DE10-Lite se nachází akcelerometr ADCL345, který má tři osy. Jelikož se jedná o separátní čip, musí být mezi FPGA a tímto čipem vytvořena komunikace.

V této části bude vytvořena komunikace pomocí SPI s akcelerometrem, kde FPGA bude typu master. Po navázání komunikace je potřeba tento čip nastavit a spustit snímání dat. Data budou snímána jen z jedné osy, přesněji z osy X. Dále tato data budou zpracována pomocí FPGA a vypsána na displeji.

5.6.1 REALIZACE SPI

Jelikož samotné FPGA v sobě nemá modul pro komunikační protokol SPI, bude potřeba jej vytvořit. Toto zapojení již je samozřejmě dávno vymyšlené a sepsané pomocí VHDL na internetu. Jelikož účelem této úlohy není samotné spuštění SPI, ale spíše využití akcelerometru, tak je tato část stažena od uživatele nandland [14] a upravena k využití pro akcelerometr.

SPI je sériová master-slave komunikace, využívaná pro sériové propojení různých mikrokontrolerů, pamětí, A/D převodníku a dalších čipů na jednom plošném spoji. SPI využívá čtyři druhy vodičů popsané v Tabulce 5.3. U hodinového signálu je potřeba dále řešit jeho mód, který řeší klidovou úroveň, a také kdy je hodnota čtena. Tento mód se nastavuje podle periferie, se kterou chceme komunikovat. Dále je také potřeba nastavit komunikační rychlost.

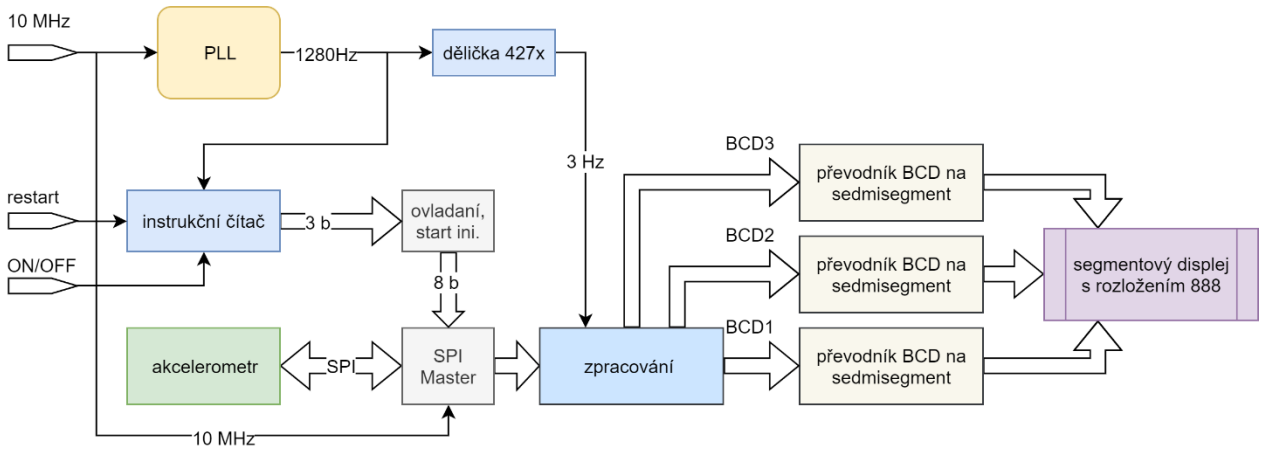
Tabulka 5.3 Porty využívané SPI.

Název	funkce
SCLK	Komunikační hodinový port.
MOSI	Master out, slave in (data)
MISO	Master in, slave out (data)
SS1-SSN	Adresační porty.

Pro ADCL345 je potřeba nastavit klidovou úroveň hodinového portu na log. 1 a úroveň čtení dat na náběžnou hranu. Tomu odpovídá nastavení módu 3. Maximální frekvence, na která dokáže tento akcelerometr komunikovat pomocí SPI, je 5 MHz. Pro naší aplikaci je tato rychlost zbytečně vysoká, ale z důvodu jednoduchosti tvorby této frekvence využijeme tuto rychlost.

Akcelerometr nastavíme pomocí konfiguračních registrů tak, aby snímal maximálně do $\pm 2\ g$ s 10bitovou kvantizací. Konfigurační registry nastavujeme tím, že nejdříve pošleme osm bitů, které adresují registr, který chceme upravovat, kde první bit značí, jestli chceme zapisovat nebo číst. Následně v dalších osmi bitech pošleme hodnotu k zápisu do registru, nebo v případě čtení obdržíme hodnotu registru.

Pro jednoduchost budeme snímat jen jeden registr s daty, princip snímání více registrů je principiálně stejný. Při použití pouze jednoho registru dostaneme při tomto nastavení možnost snímat zrychlení do $5\ m \cdot s^{-2}$. s rozlišením $0{,}0195\ m \cdot s^{-2}$. Na displeji se budou vypisovat data, která přímo přijdou od akcelerometru. Tato data tedy budou odpovídat přímo hodnotě obsáhlé ve spodním registru. Výpis dat bude fungovat stejně, jako v kapitole 5.5 u voltmetru. Celkové zapojení je vidět na Obrázku 5.13Obrázek 5.13 Blokové schéma ovládání akcelerometru.



Obrázek 5.13 Blokové schéma ovládání akcelerometru

5.6.2 ZÁVĚR AKCELEROMETRU

Samotná část ovládání a zpracování není tak složitá. Nejvíce logických členů v tomto zapojení zabírá část komunikace pomocí SPI. Výpis dat byl volen tak, aby data byla co nejméně upravovaná, důvodem je, abychom zjistili, kolik logických členů zbývá například pro složitější zpracování. V tomto případě bylo celkově využito 376 logických elementů, což je přibližně 0,76 % to je opravdu zanedbatelné množství. Dále byl použit jeden PLL. V případě pokud bychom využívali jiné zpracování dat, nebyl by problém použít jakýkoliv jiný hodinový signál. Důvodem je, že pro komunikaci SPI využíváme takt 10 MHz, který je uvnitř SPI Master dělen na 5 MHz.

5.7 OVLÁDÁNÍ VGA

Pokud bychom potřebovali zobrazovat větší množství dat, které bude FPGA zpracovávat, a segmentový displej by již nestačil, lze použít obrazový výstup v podobě VGA. Tím, že VGA je ovládáno přímo pomocí FPGA, máme možnost si vytvořit vlastní grafický akcelerátor, který můžeme uzpůsobit přímo pro výpis našich dat. V této části vytvoříme pouze ovladač na VGA, který nám zajistí základní komunikaci s displejem a nastaví jej na rozlišení 640x480 při obnovovací frekvenci 60 Hz. Na tento ovladač se dá následně jednoduše vytvořit jakékoliv grafické rozhraní. V tomto případě, pro ukázku funkčnosti, budou na displeji vykresleny tři barevné pruhy.

5.7.1 REALIZACE OVLADAČE VGA

Základem celé komunikace pomocí VGA jsou signály pro vertikální a horizontální synchronizaci (V_synch. a H_synch.). Kde H_synch určuje časovou délku jednoho řádku a V_synch časovou délku jednoho snímku. Frekvence těchto dvou signálů je vypočítána z obrazového rozlišení a obnovovací frekvence. Jako obrazové rozlišení se nepoužívá rozlišení zobrazené na obrazovce, ale rozlišení, které v sobě zahrnuje i vypočtené přechodové děje viz Tabulka 5.4, které jsou pozůstatkem CRT obrazovek.

Tabulka 5.4 Přechodové děje pro rozlišení 640x480, 60Hz (13)

Horizontal (in Pixels)				Vertical (in Lines)			
Active Video	Front Porch	Sync Pulse	Back Porch	Active Video	Front Porch	Sync Pulse	Back Porch
640	16	96	48	480	11	2	33

Z těchto parametrů se dají následně vypočítat požadované frekvence změn signálů H_synch a V_synch pomocí:

$$f_H = 60 \cdot (480 + 11 + 2 + 33) = 31{,}5\ kHz \quad (5.3)$$

$$f_V = f_H \cdot (640 + 16 + 96 + 48) = 25{,}2\ MHz \quad (5.4)$$

Tyto frekvence nemusí být zcela přesné, obrazovka si již dokáže obraz srovnat tak, aby nedocházelo k chybám. Tyto Porty dále povedou do horizontálního a vertikálního čítače, který nám již podle Tabulka 5.4 vytvoří požadované signály. Příklad podmínek tvorby H_synch:

```
1  if (H_counter < "0000001111") then -- front 15 px
2      H_stop_RGB <= '1';
3      H_synch <= '1';
4  elsif (H_counter < "0001100000") then
5      H_stop_RGB <='1';
6      H_synch <= '0';
7  elsif (H_counter < "0010011111") then
8      H_stop_RGB <='1';
9      H_synch <= '1';
10 else
11     H_stop_RGB <='0';
12     H_synch <= '1';
```

Jelikož mimo zobrazovací plochu nesmí být vysílán žádný signál, tak je vytvořen signál `H_stop_RGB`, který při aktivaci vynuluje veškeré výstupní hodnoty.

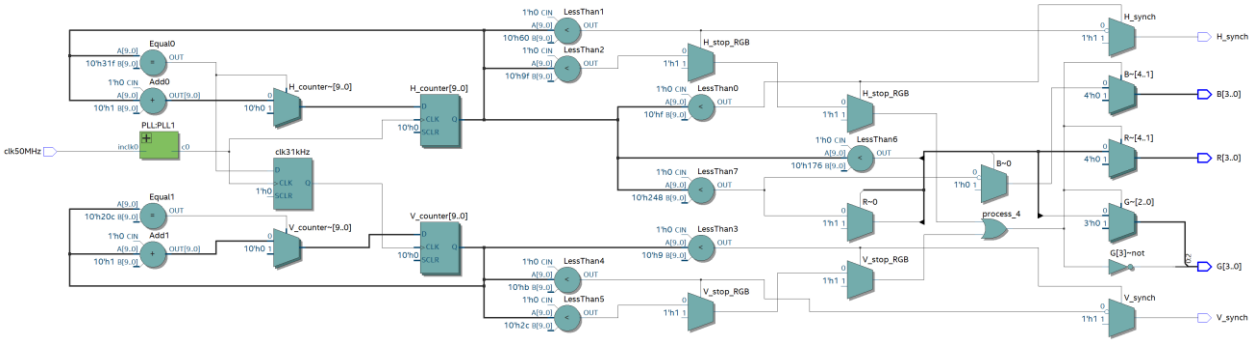
Barvy jsou řešeny analogově, pomocí jednoduchého D/A převodníku popsaného v kapitole **Chyba! Nenalezen zdroj odkazů.** VGA. Jelikož v této úloze jsou vykresleny pouze 3 barevné svislé pruhy, dá se tato část vyřešit jednoduše pomocí podmínek na horizontální čítač. Příkladem je zobrazení žluté, kde sběrnice `R,G,B` slouží jako výstupní hodnoty těchto barev:

```
1     elsif (H_counter < "0101110110" ) then
2           R <= "1111";
3           G <= "1111";
4           B <= "0000";
```

Jako vstup je potřeba jediný signál o frekvenci 50 MHz, jehož frekvence bude dále snižována na 31,5 MHz pomocí PLL.

5.7.2 ZÁVĚR OVLADAČE VGA

Samotné ovládání VGA není nijak složité na realizaci, a proto se ani neočekává žádná složitost výsledného zapojení, jehož RTL View je vidět na Obrázku 5.14. Celé toto zapojení využívá pouze 51 logických elementů, což odpovídá přibližně 0,1 % z celku. Jednoduchost tohoto zapojení je důležitá, kvůli pozdějšímu využití pro možné složitější grafické rozhraní.



Obrázek 5.14 RTL View Ovladače VGA

6 ZÁVĚR

V rámci tohoto projektu se mi povedlo osvojit si práci s programováním FPGA od firmy Terasic, s pomocí jejich vývojového prostředí Quartus Prime. Začátek této práce byl zaměřen na historii a současnost okolo FPGA. Velký důraz jsem kladl na vysvětlení architektury Terasic MAX10, na které je vystavěno FPGA na desce DE10-Lite. Dále jsem přibližně seznámil čtenáře s veškerými komponentami, které jsou v rámci jednotlivých knihoven využívány. Jelikož tato práce má sloužit jako představení této vývojové desky, bylo také potřeba se seznámit s vývojovým prostředím a jeho možnostmi využití.

V praktické části bylo navrženo sedm knihoven, které se zaměřují zejména na ovládání hlavních komponent vývojové desky: tlačítka, segmentový displej, led diody, přepínače, analogový vstup, akcelerometr, obrazový analogový výstup. Dále jsou také tyto knihovny zaměřeny na přednosti architektury MAX10, jako jsou například vnitřní PLL nebo ADC_core. Veškeré knihovny jsou psány tak, aby dostatečně popisovaly způsob ovládání těchto komponent, ale také aby byly dostatečně jednoduché, například pro jejich implementaci do složitějších zapojení. Z tohoto důvodu v každém závěru úlohy bylo zmíněno, kolik logických či dalších elementů z FPGA toto zapojení zabírá. Výsledkem jsou tedy knihovny, kde většina má využití logických elementů v FPGA do 1 %. Toto se liší pouze u realizace voltmetru, která zabrala 3 %.

Tato deska má na sobě i další užitečné komponenty jako je například Arduino Header nebo SDRAM, na které v této práci již nebylo místo. Je vidět, že DE10-Lite je ideální nástroj pro ty, kteří chtějí zkoušet jednotlivé komponenty a chtějí se s nimi naučit.

BIBLIOGRAFIE

[1] KAITLYN Franz *History of the FPGA* [online] [cit. 2021-01-06]. Dostupné z: https://blog.digilentinc.com/history-of-the-fpga/

[2] WANG, Haibo. CPLD and FPGA Architectures [online]. In: . Southern Illinois: University Carbondale [cit. 2021-01-06]. Dostupné z: https://www.engr.siu.edu/haibo/ece428/notes/ece428_fpgaarch.pdf

[3] *Field Programmable Gate Array (FPGA) History* [online]. hardwarebee, 2018 [cit. 2021-01-06]. Dostupné z: https://hardwarebee.com/field-programmable-gate-array-fpga-history-applications/

[4] Fundamentals of FPGAs: What Are FPGAs and Why Are They Needed? [online]. North Americ: Digi-Key, 2019 [cit. 2021-01-06]. Dostupné z: https://www.digikey.com/en/articles/fundamentals-of-fpgas-what-are-fpgas-and-why-are-they-needed

[5] HIDEHARU, Amano. Principles and Structures of FPGAs [online]. 1. Singapore: Springer, 2018 [cit. 2020-12-03]. ISBN 978-981-13-0824-6. Dostupné z: https://link.springer.com/book/10.1007%2F978-981-13-0824-6

[6] Intel [online katalogový list]. MAX 10 FPGA Device Architecture. [online] 2017 [cit. 30-11-2020]. Dostupnost (https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/max-10/m10_architecture.pdf

[7] *Analogové Vstupy [online].* [cit. 2021-01-06]. Dostupné z: http://plc-automatizace.cz/knihovna/periferie/analogove/analogove-vstupy.htm

[8] Intel [online katalogový list]. MAX® 10 Analog to Digital Converter User Guide. [online] 2017 [cit. 30-11-2020]. Dostupnost (https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/max-10/archives/ug_m10_adc-17-0.pdf).

[9] Intel [online katalogový list]. DE10-Lite: User Manual. [online] 2016 [cit. 30-11-2020]. Dostupnost (https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-2912030810549-de10-lite-user-manual.pdf).

[10] ASHENDEN, Peter. The VHDL Cookbook. South Australia: Computer Science University of Adelaide, 1990, 111 s. ISBN 978-0120887859. Dostupné z: https://www.ics.uci.edu/~alexv/154/VHDL-Cookbook.pdf

[11] Intel [online katalogový list]. Intel® MAX® 10 FPGA Device Overview. [online] 2017 [cit. 06-01-2021]. Dostupnost (https://www.infinity-component.hk/datasheet/6f-10M04DCU324A7G.pdf).

[12] Intel [online katalogový list]. ADXL345 Datasheet. [online] 2017 [cit. 06-01-2021]. Dostupnost (https://pdf1.alldatasheet.com/datasheet-pdf/view/254714/AD/ADXL345.html).

[13] VGA Timing *[online] Dostupné z (http://martin.hinner.info/vga/timing.html)*

[14] spi-master/SPI_Master.vhd at 02ae8b34b1182a04fdece05a29c5db0892ca59aa · nandland/spi-master · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2021 GitHub, Inc. [cit. 02.05.2021]. Dostupné z: https://github.com/nandland/spi-master/blob/02ae8b34b1182a04fdece05a29c5db0892ca59aa/VHDL/source/SPI_Master.vhd

PŘÍLOHA A

SEZNAM ZKRATEK

- ASIC - Application Specific Integrated Circuit
- BGA - Ball Grid Array
- CFM - Configuration Flash Memory
- CLP - Configurable logic blocks
- CPLD - Complex Programmable Logic Device
- DAC - Analog digital converter
- EPROM- Erasable programmable read-only memory
- FPGA - Field Programmable Gate Array
- FIFO - First In First Out
- GAL - Generic Array Logic
- HDL - Hardware Description Language
- I/O - Input/Output
- LAB - Logic Array Block
- LUT - Lookup Table
- MCU - microcontroller
- M9K - Embedded Memory
- PLA - Programmable logic device
- PLD - Programmable Logic Devices
- PLL - Phase-locked loop
- RAM - Random Access Memory
- RTL - Register Transfer Level
- ROM - Read Only Memory
- UFM - User Flash Memory
- VHSIC - Very High Speed Integrated Circuit

VYTVOŘENÉ PROJEKTY

V této příloze se nachází veškeré vytvořené projekty, které se dají přímo otevřít v programu Quartus Prime. Výčet důležitých souborů:

- Priloha B
  - Akcelerometr
    - Akcelerometr.qpf — *projekt akcelerometru*
    - 7-seg.vhd — *ovladač na sedmisegment VHDL*
    - Akcelerometr.vhd — *ovladač akcelerometru VHDL*
    - pll.vhd — *fázový závěs VHDL*
    - SPI_master.vhd — *komunikace SPI_master* [14]
  - Counter
    - Counter.qpf — *projekt čítače*
    - Counter.vhd — *čítač VHDL*
  - Displey_driver
    - Display_driver.qpf — *projekt ovladače pro displej*
    - BCD_TO_7segment.vhd — *ovladač na sedmisegment VHDL*
    - Display_driver.vhd — *čítač pro displej VHDL*
  - Timer
    - Timer.qpf — *projekt stopek*
    - BCD_TO_7segment.vhd — *ovladač na sedmisegment VHDL*
    - TImer.vhd — *zapojení stopek VHDL*
    - clock.chd — *fázový závěs VHDL*
  - Tone_synt
    - Tone_synt.qpf — *projekt sntezátoru*
    - Tone_synt.vhd — *syntezátor VHDL*
    - clock1.vhd — *fázový závěs 1 VHDL*
    - clock2.vhd — *fázový závěs 2 VHDL*
  - VGA
    - VGA.qpf — *projekt VGA*
    - VGA_control.vhd — *ovládání VGA VHDL*
    - PLL.vhd — *fázový závěs VHDL*
  - Voltmetr
    - ADC_contol.pqf — *ovládání převodníku*
    - ADC_control.vhd — *ovládání převodníku VHDL*
    - ADC_meter.vhd — *zpracování dat z převodníku VHDL*
    - BCD_TO_7segment.vhd — *ovladač na sedmisegment VHDL*
    - PLL.vhd — *fázový závěs VHDL*

MANUÁLY

V této příloze se nachází veškeré stažené manuály:

- Priloha C
  - ADXL345.pdf *ovládání akcelerometru* [12]
  - DE10-Lite_User_Manual.pdf *manuál pro DE10-Lite* [9]
  - m10_architecture.pdf *popis architektury MAX10* [6]
  - ug_m10_adc-17-0.pdf *ovládání ADC_core* [8]