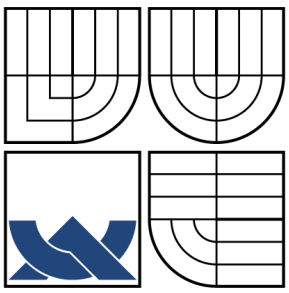
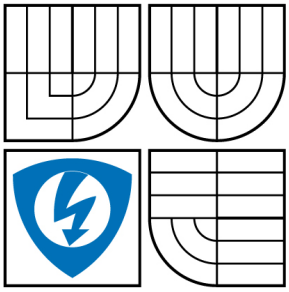


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF RADIO ELECTRONICS

DIGITÁLNÍ ZPRACOVÁNÍ SIGNÁLU Z TLAKOVÝCH SENZORŮ PROSTŘEDNICTVÍM CPLD

DIGITAL SIGNAL OF PRESSURE SENZORS PROCESSING USING CPLD

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL ZÁTURA

AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL KOVÁČ

SUPERVISOR

BRNO 2008

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Michal Zátura

Bytem: Sihotská 13, 920 01, Hlohovec, Slovenská Repoblika

Narozen/a (datum a místo): 23.01.1985 v Trnave

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií

se sídlem Údolní 244/53, 602 00, Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

prof. Dr. Ing. Zbyněk Raida, předseda rady oboru Elektronika a sdělovací technika.

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - □ diplomová práce
   - ■ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP:	Digitální zpracování signálu z tlakových senzorů prostřednictvím CPLD
Vedoucí/ školitel VŠKP:	Ing. Michal Kováč
Ústav:	Radioelektroniky
Datum obhajoby VŠKP:	

VŠKP odevzdal autor nabyvateli v*:

- ■ tištěné formě – počet exemplářů ……2……..
- ■ elektronické formě – počet exemplářů ……2……..

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - ■ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: ……06.06.2008…….

…….. ……
Nabyvatel Autor

Abstrakt

Cieľom tohto semestrálneho projektu je návrh elektronickej časti snímacieho systému na meranie výšky vo forme modulu k vývojovej doske XC2-XL a jeho následná realizácia v podobe plošného spoja. V projekte je popísaný princíp spracovania signálu z tlakového senzoru MPXH6250, jednotlivé bloky navrhnutého modulu a princíp barometrického merania výšky. Táto práca taktiež stručné oboznamuje s návrhovým prostredím Xilinx ISE, s jazykom VHDL, s princípom a aplikáciou tlakových senzorov, so štruktúrou programovateľných logických obvodov CPLD a FPGA, s vývojovými doskami XC2-XL, The Xilinx Spartan-3 Starter Kit, s programovateľnými logickými obvodmi CPLD Coolrunner-II XC2C256 a Spartan-3 XC3S200 FPGA použitými na vývojových doskách. Dôležitou časťou je návrh algoritmu na spracovanie logaritmu v digitálnej logike a zápis navrhnutého algoritmu vo VHDL jazyku.

Kľúčové slová:

CPLD, FPGA, PLD, programovateľný logický obvod, Coolrunner, Spartan, digitálne spracovanie signálu, tlakový senzor, VHDL, Xilinx ISE, výškomer

Abstract

The goal of this bachelor’s thesis is to design electronic part of the scanning system used to measure altitude. The electronic part is designed as module connected to the development platform XC2-XL. This module is realized as PCB. The principles of the signal processing of the signal from the pressure sensor MPXH6250, particular blocks of the designed module and principle of the barometrical altitude measuring are also described in this thesis. Thesis briefly informs about designing environment Xilinx ISE, VHDL language, the principle and the application of the pressure sensors, the structure of the programmable logic device CPLD and FPGA, the development platforms XC2-XL, the Xilinx Spartan-3 Starter Kit, the programmable logic devices CPLD Coolrunner-II XC2C256 and Spartan-3 XC3S200 FPGA used on the development platforms. The very important part is to design algorithm for processing the logarithm in the digital logic and it´s implementation in VHDL language.

Keywords:

CPLD, FPGA, PLD, programmable logic device, Coolrunner, Spartan, digital signal processing, pressure sensor, VHDL, Xilinx ISE, altimeter

ZÁTURA, M. *Digitální zpracování signálu z tlakových senzorů prostřednictvím CPLD.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 69 s. Vedoucí bakalářské práce Ing. Michal Kováč.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Digitální zpracování signálu z tlakových senzorů prostřednictvím CPLD jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 06.06.2008 .....
podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Michalovi Kováčovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne 06.06.2008 .....
podpis autora

1 Úvod

Bakalárska práca stručne popisuje štruktúru programovateľných obvodov CPLD a konkrétneho obvodu CPLD Coolrunner-II XC2C256, ktorý je použitý na vývojovej doske XC2-XL. Vzhľadom na zvolenú metódu programovania (metóda prevodovej tabuľky), bolo nutne použiť namiesto CPLD obvodu obvod FPGA ktorý obsahuje dvanásť 18 – Kbit-ových pamäti RAM (216K bit) a preto je v prací stručne popísaný obvod FPGA Spartan – 3 XC3S200 a vývojová doska s týmto obvodom. Vysvetľuje základný postup pri návrhu s obvodmi Xilinx vo vývojovom prostredí Xilinx ISE a stručne popisuje jazyk VHDL. Poukazuje na výhody a nevýhody tohto jazyka a oboznamuje so základnou syntaxou používanou v tomto jazyku. Taktiež oboznamuje so základnou štruktúrou v tomto jazyku, s dátovými typmi a operátormi, ktoré je možné používať pri práci s jazykom VHDL. V projekte sú v skratke spomenuté integrované tlakové senzory pre meranie rôznych druhov tlakov. Je popísaný princíp štruktúry snímacích elementov a prevedenie integrovaných senzorov tlaku. Poukazuje sa tu na možné oblasti použitia integrovaných tlakových senzorov a popisuje sa konkrétny integrovaný tlakový senzor zo série MPXH6250A (jeho merací rozsah, štruktúra snímacieho elementu, prevedenie integrovaného zapojenia atd.).

Jednou z oblastí, kde je možné tlakové senzory využívať, je oblasť barometrického merania výšky, ktorá je v práci taktiež popísaná (princíp a funkcia). Prevažná časť tohto projektu je venovaná návrhu modulu určeného na meranie výšky a algoritmu na prepočítanie zmeraného tlaku na nadmorskú výšku. Je navrhnutá bloková schéma a elektronická časť vo forme modulu k vývojovej doske. Opisuje sa princíp spracovania signálu v jednotlivých blokoch a postup pri návrhu celého modulu. Algoritmus je zapísaný vo VHDL jazyku a podrobne sú popísané jednotlivé časti tohto algoritmu. Každá časť programu, bola odsimulovaná v návrhovom prostredí ISE. Zaznamenaná dokumentácia jednotlivých simulácii, ako aj zdrojové texty programu sú priložené v prílohách na konci práce. Program spracováva digitálny signál a v digitálnej logike počíta logaritmus zo zmeraného tlaku. Zmeraný tlak vypočítavá z digitálneho signálu na výstupe AD prevodníku. Taktiež sa v programe vypočítava teplota z teplotného senzoru, ktorá sa dá spolu s vypočítanou nadmorskou výškou zobraziť na segmentovom displeji. Vzhľadom k tomu, že atmosférický tlak sa nepravidelné mení, je program zhotovený tak, aby bolo možne zariadenie kalibrovať. Zariadenie dosahuje rozlíšenie 1 m, čo je presnejší udaj ako sú schopné poskytnúť niektoré dnes používané navigačné systémy GPS.

2 CPLD

Programovateľné logické obvody (PLD) je skupina viacerých druhov digitálnych integrovaných obvodov, ktorých funkcia je určená užívateľom prostredníctvom predpisu (programu) definujúcemu prepoje jednotlivých blokov vo vnútri obvodu. Nemyslia sa tým však obvody postavené na mikroprocesoroch (mikrokontroléry). Jednoduchšie programovateľné logické obvody typu PAL/GAL sú schopné zastúpiť niekoľko štandardných digitálnych obvodov. Zložitejšie obvody CPLD sú v podstate integráciou niekoľkých jednoduchších obvodov typu GAL.

Prvé užívateľom programovateľné logické obvody (Programmable Logic Device, PLD) vznikli zo snahy nahradiť pomerne rozsiahle, avšak nekomplikované kombinatorické logické obvody (typicky adresné dekodéry) jedným obvodom aj tam, kde cena a počet výrobkov vylučovali použitie ASIC (Application Specific Integrated Circuit). Takéto obvody sa často realizovali pomocou bipolárnych PROM pamätí, toto však bolo pomerne nákladné a ťažkopádne riešenie. Keďže väčšina takýchto obvodov pozostáva z niekoľkých mnohovstupových hradiel typu AND s výstupmi spojenými hradlom OR, boli navrhnuté prvé PLD obvody ako matica AND/OR, kde sa prepaľovacími prepojkami (rovnakej technológie ako v PROM) určovali vstupy zapojené do AND matíc. Neskôr boli na výstupy OR hradiel pridané klopne obvody, čo umožnilo aj tvorbu jednoduchých sekvenčných obvodov.

Tieto jednorazovo programovateľné obvody boli nazvané PAL (Programmable Array Logic, programovateľná logika v tvare poľa) a v ich označení sa odzrkadľoval počet možných vstupov a výstupov (pričom však vstupy a výstupy zdieľali jednotlivé fyzické piny), napr. PAL16L8 bol obvod s max. 16 vstupmi a max. 8 výstupmi, ktorý realizoval čisto logickú funkciu (písmeno L v označení; písmeno R znamenalo PAL doplnený klopnými obvodmi).

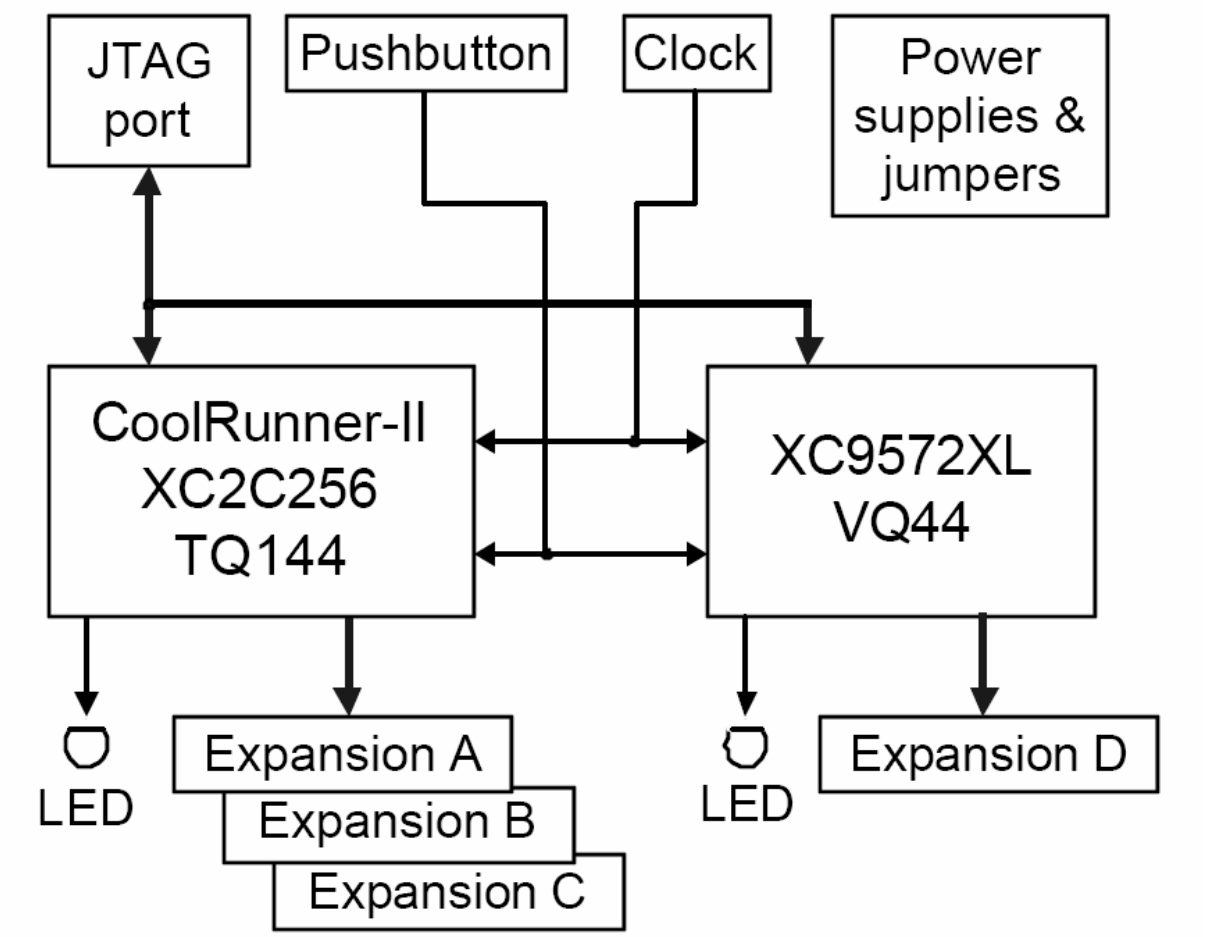
Neskôr boli prepaľovacie prepojky nahradené pamäťovými bunkami typu EEPROM, čím sa obvody stali preprogramovateľnými. Súčasne sa modifikovala ich štruktúra tak, že sa okolo výstupného klopného obvodu pridali ďalšie obvody umožňujúce napr. uvedenie výstupu do tretieho stavu alebo presmerovanie fyzického vstupu na iný vstup AND/OR matice. Tak sa vytvorila tzv. výstupná makrocela, ktorej presná funkcia je konfigurovaná ďalšími pamäťovými bunkami. Tieto obvody sa označujú GAL (Generic Array Logic), pričom je zachovaná schéma s číslovaním ako u PAL.

Zväčšovanie počtu vstupov a výstupov obvodov GAL narazilo na problém príliš veľkej AND/OR matice (ktorá rastie s počtom vstupov približne kvadraticky). Preto sa pre zložitejšie obvody navrhla štruktúra, kde niekoľko pomerne uzavretých blokov podobných GAL je prepojených pomerne jednoduchou prepojovacou sústavou. Takto je možné dosiahnuť obvody s niekoľkými stovkami vstupov a výstupov. Štruktúra GAL zostala v základnej podobe zachovaná, a CPLD sa kategorizujú podľa počtu výstupných makrociel - toto číslo je obvykle prítomné aj v označení obvodu, napr. XC95144 je obvod z radu XC95xx a obsahuje 144 makrociel. [21]

2. 1 Vývojová doska XC2-XL

XC2-XL je celistvá obvodová vývojová platforma, ktorá obsahuje Xilinx CoolRunner-II XC2C256 CPLD a Xilinx XC9572XL CPLD. Je vhodná pre základné návrhy obvodov CPLD pri používaní Xilinx CAD nástrojov. Poskytuje JTAG programovací port, napájanie, zdroj hodinového signálu a základne I/O zariadenia. Takže obvody môžu byť implementované ihneď bez potreby iných komponentov. Všetky piny z CPLD obvodov sú pripojené k expanzívnym konektorom na doske, čo dovoľuje pripojenie doprovodných

obvodov v podobe modulov. XC2-XL je kompatibilná so všetkými verziami Xilinx CAD nástrojov vrátane voľného WebPack-u dostupného na stránkach Xilinx.



Obr. 1. Bloková schéma zapojenia obvodov na doske XC2-XL [6]

Charakteristické rysy dosky XC2-XL:

- Obsahuje Xilinx CoolRunner-II XC2C256 CPLD v puzdre TQ144 a Xilinx XC9572XL CPLD v puzdre VQ44
- JTAG porty pre oba CPLD, ktoré môžu byť nezávisle povolené, alebo vyradené
- Napájanie môže byť zo sieťového transformátora, z batérii, alebo z externých zdrojov
- Oscilátor
- Úplné smerovanie I/O signálov od oboch CPLD na expanzívne konektory
- Tlačidlo a dve LED
- Not-volatility – ako u všetkých Xilinx CPLD, návrhy zostávajú v obvode uložene po odstránení napájania

2. 2 XC2C256 CoolRunner-II CPLD

CoolRunner-II obsahuje 256 makrociel. Toto zariadenie je skonštruované jak pre vysoko výkonové, tak pre nízko výkonové aplikácie. To poskytuje výkonové úspory na vysokej úrovni komunikačného vybavenia a dlhú výdrž batérie. Celková spoľahlivosť systému je veľmi pôsobivá kôli nízkej spotrebe stand-by a dynamickej operácii. Toto zariadenie je zostavené z 16 funkčných blokov vnútorné spojených do pokročilej vzájomne poprepájanej matice (AIM). AIM zásobuje 40 vstupov na každom funkčnom bloku. Funkčný

blok sa skladá zo 40 až 56 obvodov PLA a 16 makrociel, ktoré obsahujú početné nastavovacie bity, ktoré dovoľujú kombinačné režimové operácie. Tieto registre môžu byť globálne resetované, alebo predvolené a konfigurované ako klopný obvod typu D alebo T. Obvod obsahuje tiež viacnásobné hodinové signály. Obsadenie výstupných pinov zahrňuje sledovaciu rýchlosť, zbernicové svorky, programovateľné zeme atd.

Základné charakteristické vlastnosti obvodu XC2C256 CoolRunner-II CPLD (viď[5]):

- Obvod je optimalizovaný pre 1,8 V systémy
  - rýchle 5,7 ns oneskorenie
  - nízky kľudový prúd 13µA
- Priemyslovo najlepší 0,18 mikrón CMOS CPLD
  - Optimalizovaná architektúra pre efektívnu logickú syntézu
  - Multinapäťové I/O operácie – 1,5V až 3,3V
- Dostupný v rôznych puzdrách
- Najrýchlejší v systémovom programovaní
- Architektúra PLA
- Pokročilá návrhová bezpečnosť
- Voliteľné konfigurovatelné zeme na nepoužite I/O piny
- Zmiešané I/O napätia kompatibilne s 1,5 V; 1,8 V; 2,5 V a 3,3 V logickou úrovňou.

2. 3 FPGA Spartan-3 XC3S200 a vývojová doska

Doska Xilinx Spartan-3 Starter Kit, poskytuje nízku cenu a ľahko použiteľnú vývojovú platformu pre návrh obvodov Spartan-3 FPGA. Doska obsahuje praktické komponenty potrebné pre vývoj rôznych aplikácii (externe pamäte 256Kx16 SRAM, VGA port, expanzívne konektory, RS-232 Port, prepínače, tlačidlá, slot na oscilátor, 50Mhz oscilátor, LEDs, displej a.i.). Napájanie dosky je 5V zo sieťového adaptéra, toto napätie je privedené aj na každý expanzívny konektor, takže sa dá využiť, pri práci s doskou. Doska obsahuje štvorznakový sedem segmentový displej.

Spomenutý obvod FPGA má 4320 logických buniek a obsahuje dvanásť osemnásť kilobitových blokov pamäti RAM, čo tvorí spolu 216Kbits. Obvod ma 173 užívateľom definovaných pinov, štyri digitálne hodiny (DCM) a 200K systém gates.

3 Xilinx ISE

Pre vývoj aplikácii s CPLD a FPGA existuje niekoľko návrhových systémov. Pre vývoj je dôležité použiť minimálne dva nástroje. Prvým je nástroj pre syntézu, ktorý prevedie väčšinou textový popis návrhu v niektorom HDL jazyku na netlist využívajúci obecne logické bloky. Druhý nástroj zaistí konverziu obecného netlistu na netlist využívajúci prostriedky konkrétneho CPLD alebo FPGA a zisti ich najvhodnejšie rozmiestnenie a prepojenie. Nástroje pre rozmiestnenie a prepojenie obyčajne ponúkajú iba výrobcovia CPLD a FPGA. Prostriedky pre syntézu ponúkajú aj iné firmy. Firma Xilinx ponúka vývojový systém ISE.

ISE (Integrated Software Environment) je komplexný návrhový systém, ktorý slúži na programovanie CPLD a niektorých FPGA. Tento vývojový nástroj XILINX ISE (Webpack – je obmedzená verzia kompletného systému firmy Xilinx) je možné zadarmo stiahnuť na www.xilinx.com (viď [14]). Naprogramovanie CPLD a FPGA pomocou XILINX ISE je možné dosiahnuť rôznymi spôsobmi. Jednou z možností je zadanie obvodu formou schémy, alebo pomocou automatu. Klasický spôsob návrhu je použitie niektorého jazyka HDL. Okrem týchto možností ISE ponúka ešte niekoľko iných ciest. Výhodou tohto návrhového systému je, že máme k dispozícii hneď niekoľko nástrojov na jednom mieste. ISE ponúka nástroje pre:

- **vkladanie designu** – textový editor, editor schém, constrain editor, PACE pre zobrazenie a editáciu obmedzení (condtraints) a StateCAD pre editáciu stavových automatov.
- **syntézu –** pre syntézu je možné využiť integrovaný systém XST (Xilinx Synthesis Technology) alebo napojenie na komerčné produkty iných firiem.
- **simuláciu –** ISE ponuka automatickú generáciu testbench-ov.
- **implementáciu –** sem patria nástroje, ktoré pracujú s výsledkom syntézy a snažia sa ju namapovať do cieľovej platformy. Jedná sa o nástroje pre preklad (translate), mapovanie, analýzu, fitting a generovanie stimulačného modelu.
- **programovanie –** program pre generovanie konfiguračného bitstreamu a programovací nástroj iMPACT, ktorý umožňuje programovanie a testovanie napr. pomocou JTAG rozhrania (viď [15]).

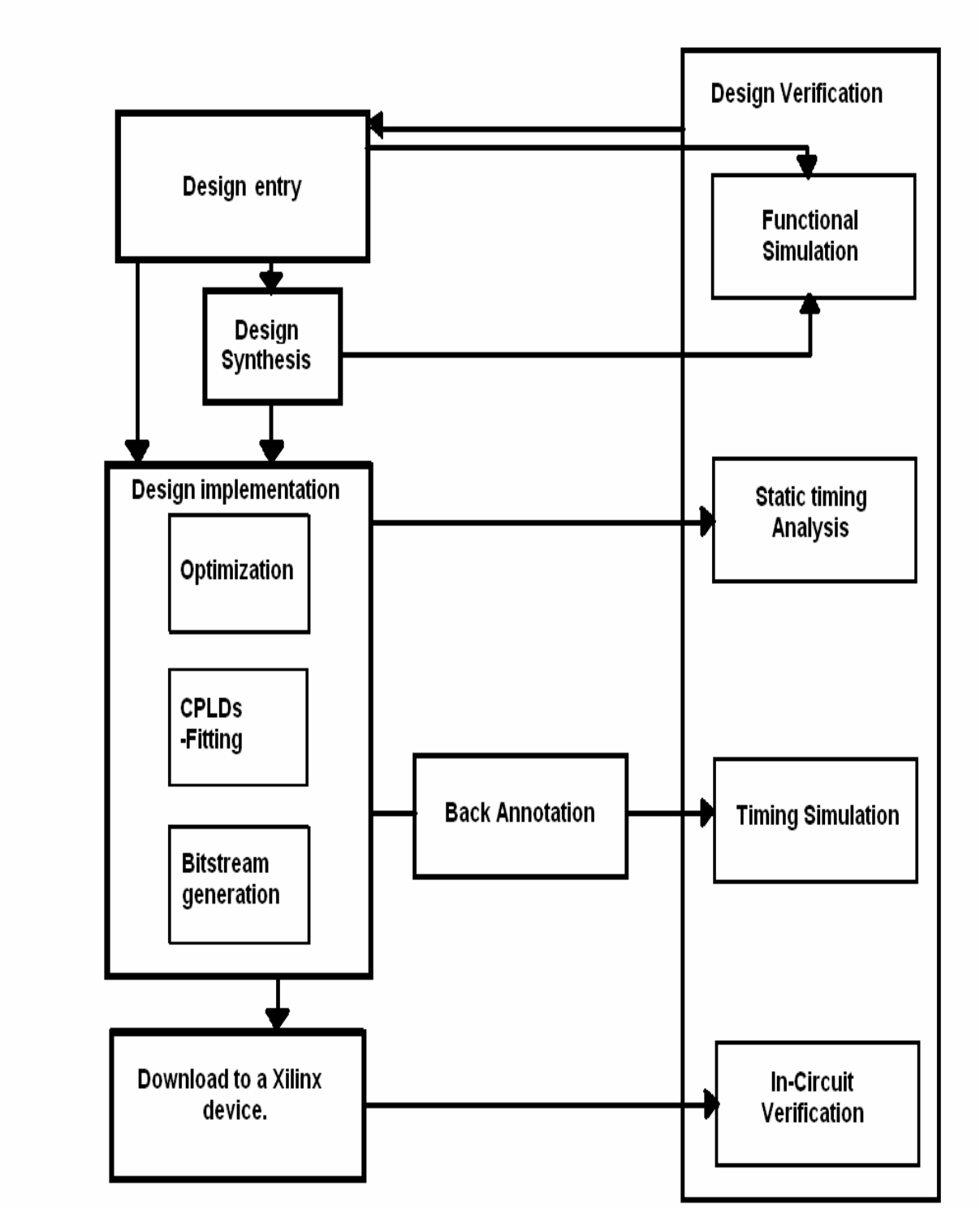
Programovanie je závislé na predchádzajúcich krokoch. Jednotlivé kroky však nemusíme vykonávať postupne, prakticky stačí spustiť posledný krok, tj. programovanie. Systém potom sám prevedie všetky potrebné kroky, ktoré musí predchádzať – tzn. syntézu a implementáciu. Každý krok návrhu sa spustí až po úspešnom ukončení kroku, na ktorom závisí. Ku každému kroku je možné vygenerovať podrobné reporty.

Vývojove prostredie ISE podporuje nasledujúce obvody Xilinx:

- FPGA Spartan-II
- FPGA Spartan-IIE (max. XC2S300E)
- FPGA Virtex-E (max. XCV300E)
- FPGA Virtex-II (max. XC2V250)
- FPGA Virtex-II Pro (pouze XC2VP2)
- CPLD CoolRunner2
- CPLD XC9500/XL/XV
- CPLD CoolRunner XPLA3

3. 1 Postup pri návrhu s obvodmi Xilinx vo vývojovom prostredí ISE

Podrobné blokové zobrazenie postupu pri návrhu s obvodmi Xilinx vo vývojovom prostredí ISE je na obr. 2.



Obr. 2. Postup pri návrhu s obvodmi Xilinx vo vývojovom prostredí ISE

Design entry (vstupný návrh) – je návrh vytvorený použitím niektorého z HDL jazykov formou schémy alebo pomocou automatu.

Design synthesis (syntéza návrhu) – je to vytvorenie netlistu z popisu zadaného niektorým z HDL jazykov, poprípade inou formou. Je to vlastne zapojenie obvodových prvkov (logických členov, klopných obvodov, registrov, ...). Je to vytvorenie schémy s obvodovými prvkami, ktoré obsahuje požadovaný CPLD, a ktorá bude vykonávať požadovanú funkciu.

Design implementation (implementácia návrhu) – patria sem nástroje, ktoré pracujú s výsledkom syntézy. Postupnými krokmi sa vytvorí popis prepojenia programovacích prepojek v cieľovom obvode. Tento popis je dôležitý a konečný pre naprogramovanie CPLD. Jedným z prvých krokov pri implementácii je optimization (optimalizácia). Optimalizácia má za úlohu minimalizovať popis aplikácie na ekonomickejšiu formu popisu (môžeme si ju predstaviť ako obdobu minimalizácie, ktorú vykonávame pomocou Karnaughových máp). Najdôležitejším krokom je fitting. Je to proces, pri ktorom sa vytvára konkrétna obvodová štruktúra. Konečným výsledkom implementácie je súbor, v ktorom je obsiahnutý stav jednotlivých prepojek a plán výslednej štruktúry. [22]

Ďalším výsledkom implementácie je Back Annotation. Je to simulačný model, ktorý je vytvorený v niektorom z jazykov HDL a je doplnený časovými parametrami o prepojených prvkoch a spojovacích štruktúrach použitého CPLD. Tento model je pre človeka nečitateľný a používa sa pre časovú simuláciu. Vstupným súborom simulátoru časovej simulácie je súbor zapísaný v jazyku HDL [2].

Pri navrhovaní projektu je veľmi pravdepodobné, že sa nepodarí vytvoriť popis aplikácie, ktorý by bol funkčný hneď na prvý pokus. Je možné, že pri zapisovaní projektu budú vytvorené nejaké logické chyby. Na overenie týchto chýb v návrhu slúži časť design verification (overovanie projektu) a v nej zahrnuté simulácie.

Functional simulation (funkčná simulácia) – používa sa pre overenie syntaxe zapísaného kódu a pre kontrolu, či po privedení vstupných signálov na vstup modelu sú na výstupe požadované výstupné signály. Teda pre kontrolu, či model správne funguje. Táto simulácia nie je závislá na architektúre cieľového CPLD. Táto simulácia sa nazýva aj simulácia pred fittingom alebo simulácia pred prepojením. Tieto názvy sú odvodené z toho, že simulácia používa model (text), ktorým je popísaná navrhovaná funkcia obvodu, a preto sa vykonáva pred ďalším spracovaním zdrojového textu.

Timing simulation (časová simulácia) – v prípade, že funkčná simulácia dopadla kladne, tj. na výstupe modelu sú požadované signály, je potrebné ešte overiť časové požiadavky návrhu, teda požiadavky na oneskorenie (rýchlosť reakcie) a kmitočet hodinového signálu. K tomu účelu slúži časová simulácia. Časová simulácia používa model vzniknutý pri implementácii (back annotation), ktorý obsahuje skutočné parametre cieľového CPLD a rešpektuje konkrétnu obvodovú štruktúru vzniknutú po fittingu. Z toho dôvodu sa táto simulácia nazýva aj simulácia po fittingu alebo simulácia po pripojení.

Pretože model funkčnej simulácie neobsahuje informácie o cieľovom obvode, býva funkčná simulácia omnoho rýchlejšia ako simulácia časová. U jednoduchých návrhov sa väčšinou časová simulácia vykonáva až na konci po prepojení. V prípade, keď je prevedenie jednotlivých krokov implementácie časovo náročnejšie je vhodné skontrolovať, aspoň orientačne, tieto jednotlivé kroky pomocou static timing analysis (statickej časovej analýzy). V prípade, že výsledok je z časového hľadiska nevyhovujúci, nemá zmysel pokračovať v implementácii a je potrebné vrátiť sa k predchádzajúcim krokom. [23]

4 VHDL

Skratka VHDL znamená VHSIC Hardware Description Language, pričom VHSIC je skratka pre Very High Speed Integrated Circuit. Jazyk VHDL patrí spolu s jazykom Verilog medzi najrozšírenejšie jazyky používané pri modelovaní číslicových zariadení. Tento jazyk podporuje množstvo komerčných aj nekomerčných aplikácií, ktoré sú prístupné na rôznych softvérových platformách.

Jazyk VHDL sa vyvíja už od roku 1981 na požiadavku amerického ministerstva obrany. Požiadavkou bolo vyvinutie jazyka so širokými možnosťami opisu, ktorý bude interpretovaný súhlasne rôznymi simulátormi a bude nezávislý na technológii a návrhovej metodike. Požadovali vyvinúť jazyk, ktorým by sa dala popísať funkcia (správanie sa) hardvéru a ktorý by bol zrozumiteľný pre ľudí a aj počítače. V rokoch 1993 až 1995 pracovalo na špecifikácii viacero firiem ako napríklad *Intermetrics*, *IBM*, *Texas Instruments* v rámci projektu VHSIC. Prvá norma bola formulovaná už v roku 1986, ale prvá rozšírená norma IEEE 1076-1987 sa objavila o rok neskôr. Najdôležitejším dôvodom pre použitie VHDL je to, že šetrí čas a peniaze. Medzi ďalšie výhody patrí zvýšenie produktivity práce pracovníkov.

Jazyk VHDL sa používa ako prostriedok na návrh, modelovanie, dokumentáciu, verifikáciu, simuláciu a syntézu číslicových obvodov, ale aj na návrh vložených diagnostických prostriedkov a testov. VHDL je technologicky nezávislý jazyk, ktorý sa nespája s konkrétnym simulátorom, a ktorý neurčuje návrhárovi metodológiu návrhu. Má možnosť naraz zachytiť operáciu na všetkých úrovniach opisu použitím rovnakej syntaxe a sémantiky vo všetkých úrovniach a umožňuje simulovať tento systém pri kombinácii rôznych úrovní opisu.

VHDL podporuje väčšiu časť z úrovní abstrakcie, pomocou ktorých je možné opísať funkciu zariadenia. Základnými podporovanými úrovňami sú: úroveň logických obvodov, úroveň medziregistrových prenosov a funkčná úroveň. Návrhár môže systém opísať dvoma základnými prístupmi a to: opisom správania a opisom štruktúry. Opis správania pripomína klasické programovanie softvérových aplikácií a umožňuje opis systému na funkčnej úrovni. Základným stavebným prvkom pre opis správania je proces, v ktorom sa všetky príkazy realizujú paralelne. Paralelne sú realizované aj príkazy, ktoré sa nachádzajú v bloku alebo procedúre. Opis systému pomocou funkčnej úrovne je vhodný ak nás nezaujímajú detaily implementácie a nepotrebujeme daný opis následne syntetizovať. Syntéza z funkčnej úrovne je komplikovaná a ak je realizovaná, zväčša výsledný produkt nespĺňa naše požiadavky. Pre potrebu syntézy obvodu je vhodná úroveň medziregistrových prenosov. Z tejto úrovne je možný automatický prechod do úrovne logických hradiel.

VHDL nie je ideálny prostriedok pre opis pomocou systémovej úrovne. V tejto oblasti môže byť síce VHDL s čiastočným úspechom použitý, ale vhodnejší je pre nižšie úrovne opisu.

Výhodami jazyka VHDL sú:

- *podpora viacerých úrovní abstrakcie* – možnosť prelínania opisu na rôznych úrovniach abstrakcie, kombinácia opisu na úrovni správania s opisom na úrovni logických obvodov
- *možnosť hierarchického návrhu systému* – podpora viacerých blokov a ich vzájomné previazanie, podpora balíkov a pod.
- *možnosť simulácie každého bloku systému zvlášť* – možná simulácia výlučne jedného bloku (entity), nezávisle od zostávajúceho systému
- *verifikácia špecifikácie* – špecifikáciu systému môžeme verifikovať simuláciou

- *oddelená funkcia od implementácie* – pri použití vyšších úrovní opisu je implementácia systému závislá na použitom nástroji pre syntézu
- *vyššia produktivita* – možnosť opätovného použitia už navrhnutých obvodov
- *technologická a nástrojová nezávislosť* – jazyk VHDL je štandardizovaný, takže by mal byť implementovaný všetkými nástrojmi rovnako
- *široká podpora* – jazyk VHDL je široko používaný a rozšírený, a preto ho podporuje väčšina softvérových a iných nástrojov
- *spojitý čas* – časovanie v opise VHDL je možné v spojitom čase

Nevýhodami jazyka VHDL sú:

- *zložitý spôsob opisu* – široké opisné možnosti jazyka komplikujú syntax a sémantiku jazyka
- *nevhodný pre systémovú úroveň* – oproti jazyku HSSL je nevhodný pre opis systému na systémovej úrovni, aj keď umožňuje čiastočné riešenia na tejto úrovni

Aj keď jazyk VHDL bol prvotne určený pre návrh číslicových systémov v súčasnej dobe existuje rozšírený štandard o analógové a zmiešané obvody. Ide o štandard IEEE 1076.1-1999 – IEEE Standard VHDL Analog and Mixed-Signal Extensions, skrátene označovaný ako VHDL-AMS.

4. 1 Základná syntax jazyka VHDL

- V jazyku VHDL sa nerozlišuje medzi veľkými a malými písmenami.
- Komentár začína symbolom "--" a je platný až po koniec riadka.
- Identifikátor vždy musí začínať písmenom a končiť písmenom alebo číslicou. Vo vnútri identifikátora sa môže vyskytovať podčiarnik "_".
- Vo VHDL je možná veľká rôznorodosť zápisu čísiel (napr.: 123, 12_234, 987E6, 0.5, 2.718_234, 12.4E-9)
- Znaky sa vkladajú do apostrofov: 'A'; '1' ….
- Reťazec sa vkladá medzi úvodzovky: "retazec". Ak chceme použiť úvodzovky v rámci reťazca, použijeme zdvojené úvodzovkovanie: "retazec""retazec2"".".
- Bitový reťazec je možné zapísať v binárnom, hexadecimálnom a osmičkovom formáte zápisu. Binárny zápis je označený znakom B, za ktorým bezprostredne nasleduje hodnota v binárnom tvare uvedená v úvodzovkách. Hexadecimálny formát je identifikovaný znakom X a osmičkový znakom O (napr.: B"1011", X"56", O"126").
- Oddelenie zoznamov zapíšeme pomocou čiarky ",".
- Priradenie do signálu sa zapíše pomocou symbolu "<=".
- Priradenie do premennej sa zapíše pomocou symbolu ":=".

4. 2 Popis základnej štruktúry návrhu

Prehľadný návrh je typicky rozdelený do niekoľkých blokov. Tieto bloky sú potom prepojené dohromady a tvoria kompletný návrh. Návrh vytvorený vo VHDL môže byť kompletne popísaný v jedinom bloku alebo rozložený do niekoľkých menších celkov. Každý blok vo VHDL je nazvaný "**entity**". Entita popisuje rozhranie k tomuto bloku.

Popis je špecifikovanie vstupov a výstupov k bloku. Hlavný kód kompletného programu môže byť zbierka mnohých blokov, pokiaľ zvolíme hierarchický návrh. Deklarácia entity sa skladá z definície novej entity, tj. z názvu entity, zo zoznamu deklarácii rozhrania, z typu dát, ktoré signál predstavuje a ukončenia definície. Každá deklarácia rozhrania definuje

jeden alebo viac signálov, ktoré sú vstupmi alebo výstupmi (popis režimu). Popis režimu špecifikuje, či je bit:

- Vstupný (IN) – vstup
- Výstupný (OUT) – výstup (výstupný signál nemôže byť použitý vo vnútri konštrukčnej entity)
- Obojsmerný (INOUT) – obojsmerný vývod
- Registrový výstup (BUFFER) – výstup so spätnou väzbou, z ktorého signál môže byť použitý vo vnútri entity

Typ špecifikuje aký druh hodnôt môže signál nadobúdať.

Ďalšia časť popisu návrhu je popis, čo sa má vykonať. Toto je definované "**architecture**" deklaráciou. Deklarácia architektúry sa skladá z definície novej architektúry (z názvu architektúry a k priradeniu k entite) z popisu operácie a ukončenia architektúry. Architektúra vždy patrí ku konkrétnej entite. Jedna entita môže mať viacej architektúr. Porty, ktoré sú definované v entite sú viditeľné aj vo vnútri architektúry a pracuje sa s nimi podobne ako s lokálnymi signálmi.

Medzi ďalšie objekty vyskytujúce sa vo VHDL patrí napr.: Process (Proces), Package (Balík), Library (knižnica).

- **Process –** výrazy vo VHDL sú vo všeobecnosti vykonávané paralelne. Ak chceme, aby nejaká skupina výrazov bola vykonávaná sekvenčne, tak ju „vložíme" do procesu. Kód obsiahnutý v procese nie je vždy vyhodnocovaný, ale čaká na konkrétne podmienky aktivácie.
- **Package –** obsahuje definície typov, konštánt, podprogramov a procedúr, ktoré sa používajú na rôznych miestach projektu. Vo všeobecnosti umožňuje sprehľadnenie zapísaného kódu rozsiahlejšieho projektu.
- **Library –** je adresár, v ktorom sú uložené skomprimované časti VHDL kódu.

Tab. 1. Umiestnenie deklarácii VHDL objektov

	Entiy (Entita)	**Architecture (Architektúra)**	**Process/Subprogram (Proces/Podprogram)**	**Package (Balík)**
Subprogram (Podprogram)	X	X	X	X
Component (Komponent)		X		X
Configuration (Konfigurácia)		X		
Constant (Konštanta)	X	X	X	X
Data type (Dátový typ)	X	X	X	X
Port (port)	X			
Signal (Signál)		X	X	X
Variable (Premenná)			X	

4. 3 Dátové typy – štandardné dátové typy

Aby bolo možné určiť akých hodnôt môže nadobúdať deklarovaný objekt a aké operácie s nim sú prístupné, je treba pred každým použitím nejakého objektu deklarovať jeho typ. Z toho dôvodu je jazyk VHDL deklarovaný ako prísne typový. Najčastejšie používané typy sú definované priamo v príslušných štandardoch a nazývajú sa štandardné dátové typy. Vo VHDL je možné definovať nové vymenované dátové typy. Principiálne tieto dátové typy môžu nadobúdať rozličné hodnoty. Používajú sa potom hlavne pri návrhu stavových automatov.

Vo VHDL existujú štyri **triedy dátových typov** a to:

- *Jednoduché* (sú to elementárne dátové jednotky, ktoré sa nedajú ďalej rozložiť)
    - integer (celočíselný)
    - real (reálny)
    - physical (fyzikálny)
    - enumeration (vymenovaný)
- *Zložené* (sú to dátové jednotky, ktoré sú zložené z iných jednoduchých dátových jednotiek)
    - array (polia)
    - record (záznamy)
- Typ *prístupu*
- Typ *súboru*

Pre syntézu sú najdôležitejšie jednoduché a zložené typy dát.

Medzi **štandardné dátové typy** patria:

- Boolean (false, true) – je preddefinovaný v balíku „Standard“ a je to vymenovaný dátový typ. Môže nadobúdať iba dve hodnoty: false a true.
- Bit (‘0’,’1’) – je preddefinovaný v balíku „Standard“ a je to vymenovaný dátový typ. Môže nadobúdať iba dve hodnoty: logickú 0 a logickú 1. Tento typ nie je možné použiť napríklad na modelovanie trojstavového budiča zbernice.
- Character – ASCI znaky (napr.: NUL, SOH, STX,....’0’, ’1’, ‘2’,….’A’, ‘B’, ‘C’,….’a’, ‘b’, ‘c’,…..’}’, ‘~’, DEL)
- Integer (-2 147 483 647 to 2 147 483 647) – 32 bits – je množina celých čísel v určitom rozsahu. VHDL štandard nedefinuje rozsah pre typ integer. Rozsah je závislý od konkrétneho návrhového prostriedku. Vo väčšine návrhových prostriedkov je jeho rozsah 32 bitov.
- Real (-1.0E38 to 1.0E38)
- Bit_vector – je neohraničený vektor (jeho veľkosť sa ohraničuje pri deklarácii), ktorého jednotlivé prvky sú typu bit.
- Time – je numerický dátový typ, ktorý špecifikuje časový údaj. Tento údaj je zapísaný použitím celého kladného čísla a časovej jednotky. (Časové jednotky môžu byť: fs – femtosekunda, ps – pikosekunda, ns – nanosekunda, us – mikrosekunda, ms – milisekunda, sec – sekunda, min – minúta, hr – hodina)

4. 4 Operátory

Operátory používané vo VHDL (viď tab. 2.) môžme rozdeliť na:

- Operátory používané vo výrazoch určených pre syntézu

- Logické operátory – sú definované pre typy bit alebo boolean a pre jednorozmerne polia tohto typu. Obidva operandy musia mať rovnakú dĺžku (okrem prípadu negácie)
- Aritmetické operátory – Operátory sčítanie, odčítanie, identita a negácia sú matematické operácie a vyžadujú operandy numerického typu. K aritmetickým operátorom sa počíta tiež operátor zjednotenia (concatenation) &
- Relačné operátory – oba operandy musia byť toho istého dátového typu a výsledok porovnania je vždy Boolean typ.
- Posuvné a rotačné operátory – sú definované pre jednorozmerné polia, ktorých elementy sú typu Bit alebo Boolean.

- Operátory používané vo výrazoch, ktoré nie sú určené pre syntézu (niektoré systémy ich v obmedzenom rozsahu podporujú aj pre syntézu)
  - Operátory násobenia a delenia – sú definované pre čísla typu integer a čísla s plávajúcou čiarkou. Operátory mod a rem sú definované iba pre čísla typu integer. Pri použití mod a rem sú oba operandy aj výsledok typu integer.
  - Operátory mocniny a absolútnej hodnoty

Tab. 2. Operátory používane vo VHDL

Logické operátory		**Aritmetické operátory**		**Relačné operátory**		**Posuvné a rotačné operátory**	
and	logický súčin	+	sčítanie	=	rovnosť	sll	logický posuv doľava
or	logický súčet	-	odčítanie	/=	nerovnosť	srl	logický posuv doprava
nand	inverzný logický súčin	+	identita	<	menší	sla	aritmetický posuv doľava
nor	inverzný logicky súčet	-	negácia	<=	menší rovný	sla	aritmetický posuv doprava
xor	exkluzívny or	*	násobenie	>	väčší	rol	rotácia logická doľava
xnor	inverzný exkluzívny or	/	delenie	>=	väčší rovný	rol	rotácia logická doprava
		mod	modulus				
		rem	zvyšok				
		abs	abs. hodnota				
		**	mocnina				

Medzi jednotlivými skupinami operátorov platí priorita (viď tab. 3.).

Tab. 3. Priorita operátorov

vyššia priorita	rôzne	**, abs, not
-	násobenie	*, /, mod, rem
-	zmena znamienka	+, -
-	súčtové	+, -, &
-	posuvne a rotačné	sll, slr, sla, sra, rol, ror
-	relačné	=, /=, <, <=, >, > =
nižšia priorita	logické	and, or, nand, nor, xor, xnor

5 Integrované senzory tlaku

Určovanie tlaku je jedno zo štandardných meraní v mnohých oblastiach bežného života a priemyslu. Meranie tlaku sa už v dnešnej dobe vykonáva prostredníctvom integrovaných tlakových senzorov, ktoré obsahujú nielen snímač, ale aj ďalšiu elektroniku (zosilnenie, filtráciu, A/D prevod atd.). Medzi popredných výrobcov integrovaných senzorov tlaku už dlhodobo patrí firma Freescale (viď [3]).

Integrovaný senzor tlaku v dnešnej dobe predstavuje komplexný, spoľahlivý, odolný a miniatúrny snímač, ktorý je síce zapuzdrený v špeciálnych puzdrách, ale často s vývodmi v klasických rozmeroch niektorého štandardu. Takýto integrovaný obvod obsahuje nielen na tlak citlivý snímací prvok, ale aj vyhodnocovaciu elektroniku a výstupné obvody , ktoré nameraný signál zo senzorovej časti spracovávajú, upravujú a následne poskytujú definovaný lineárny analógový alebo digitálny výstup. Naviac dnes sú bežné aj kompenzácie niektorých nechcených externých vplyvov. U tlakových senzorov je to najčastejšie kompenzácia teploty, ktorá sa obvykle vykonáva buď elektronicky podľa integrovaného teplotného snímača alebo na úrovni samotného snímača tlaku prostredníctvom referenčného prvku.

5. 1 Použitie

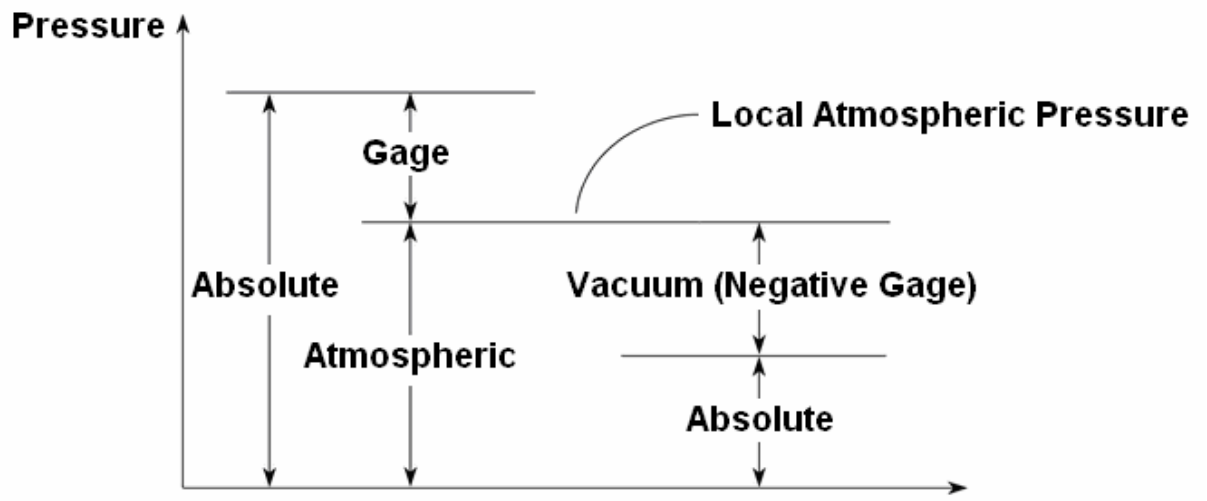
Integrované senzory tlaku sa prakticky využívajú všade tam, kde sa pracuje so vzduchom, kvapalinami a pneumatickými systémami.

- Zdravotníctvo – monitorovanie tlaku krvi, spirometre, respirátory, inhalátory apod.
- Meranie a regulácia tlaku plynov a kvapalín
- Meranie úrovne hladiny a množstva vody – studne, nádrže, rozvod vody, umývačky, práčky, apod.
- Barometre a altimetre
- Domáce spotrebiče – klimatizácia, vysávače, športové trenažéry apod.
- Meteorologické zariadenia a stanice
- Automobily – tlak v pneumatikách, nastaviteľné tlmiče a vzduchové pérovanie, posilňovače bŕzd apod.
- Pneumatické systémy – stavebné stroje, výrobné linky, kompresory apod.
- Robotika
- Atd.

5. 2 Druhy tlakov a princípy ich merania

V závislosti na tom, ku ktorej vzťažnej hodnote je tlak meraný, sa vyskytujú nasledujúce druhy tlakov:

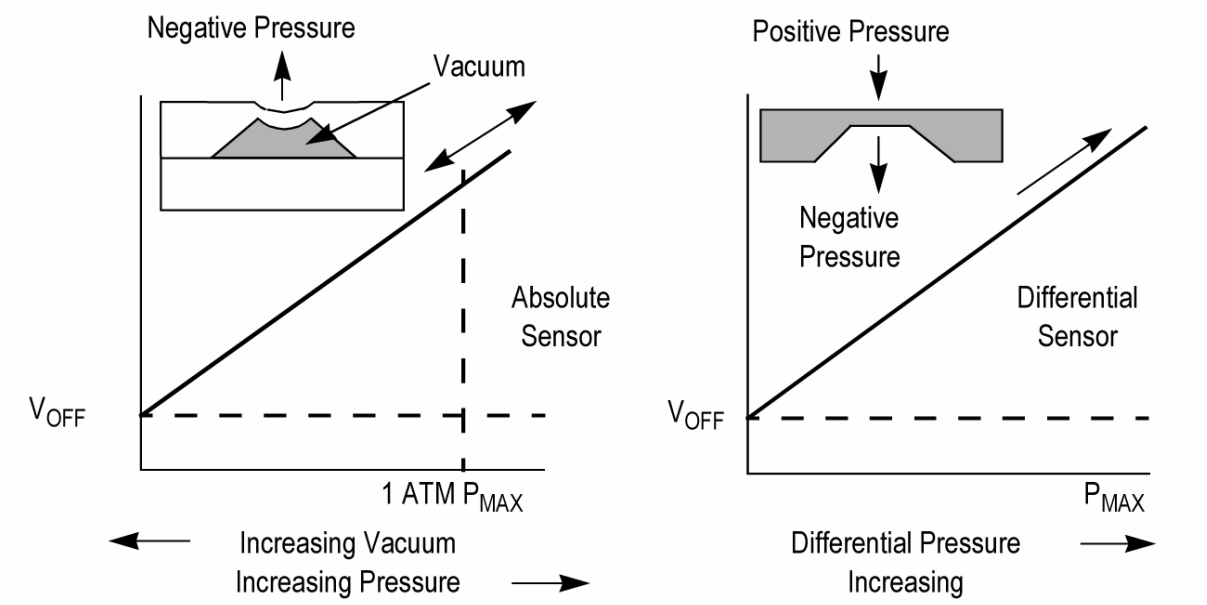
- **Absolútny tlak (Absolute pressure)** – tlak nezávislý na okolitom prostredí, pri jeho meraní je vzťažná hodnota tlak 0 Pa, tzn. vákuum
- **Atmosférický tlak (Atmospheric pressure)** – všadeprítomný okolitý vonkajší tlak prostredia, vzťažná hodnota je opäť tlak 0 Pa
- **Manometrický tlak (Gage pressure) –** tlak iného prostredia proti okolitému atmosférickému tlaku



Obr. 3. Rozdelenie a pomenovanie jednotlivých druhov tlakov [16]

Na obr.3. je zobrazené rozdelenie jednotlivých tlakov. Podľa tohto obrázku môžeme tlakové senzory rozdeliť do niekoľkých kategórii podľa princípu merania:

- **Senzory absolútneho tlaku (Absolute pressure sensor)** – senzory s vhodným rozsahom môžu merať atmosféricky tlak
- **Diferenčné/rozdielové senzory tlaku (Differential pressure sensor)** – merajú rozdiel tlaku vo dvoch prostrediach nezávisle na okolitom prostredí
- **Manometrické senzory tlaku (Gauge pressure sensor)** – merajú rozdielový tlak jedného prostredia proti hodnote atmosférického tlaku



Obr. 4. Princíp merania tlakov senzorov absolútneho tlaku (vľavo) a diferenčného tlaku (vpravo) [17]

Základné sú prvé dve skupiny, ktoré sa odlišujú samotnou funkciou senzorov (viď obr. 4.). Senzor absolútneho tlaku meria priamo externý tlak pôsobiaceho okolitého média (plynu alebo kvapaliny) vztiahnutý k nulovej hodnote tlaku vákua vo vnútornej komore senzoru (viď obr. 4. vľavo). V skutočnosti sa meria záporný tlak z pohľadu meracej strany senzoru.

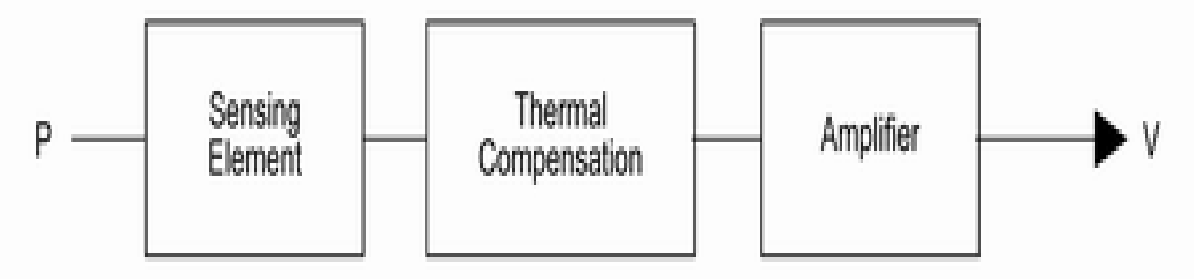
Diferenčný senzor tlaku potom vlastne meria rozdiel tlakov medzi dvoma vstupmi (viď obr. 4. vpravo), teda prehnutie kremíkovej steny je úmerné rozdielu tlakov na obidvoch stranách (vstup P1 a P2, obr. 7.).

Senzory manometrického tlaku sú špeciálnym prípadom senzorov diferenčných. Princíp je rovnaký. Iba s tým rozdielom, že sa meria rozdiel tlaku pôsobiaceho média na vstupe P1 a atmosférického tlaku skrz otvor P2 (viď obr. 7.).

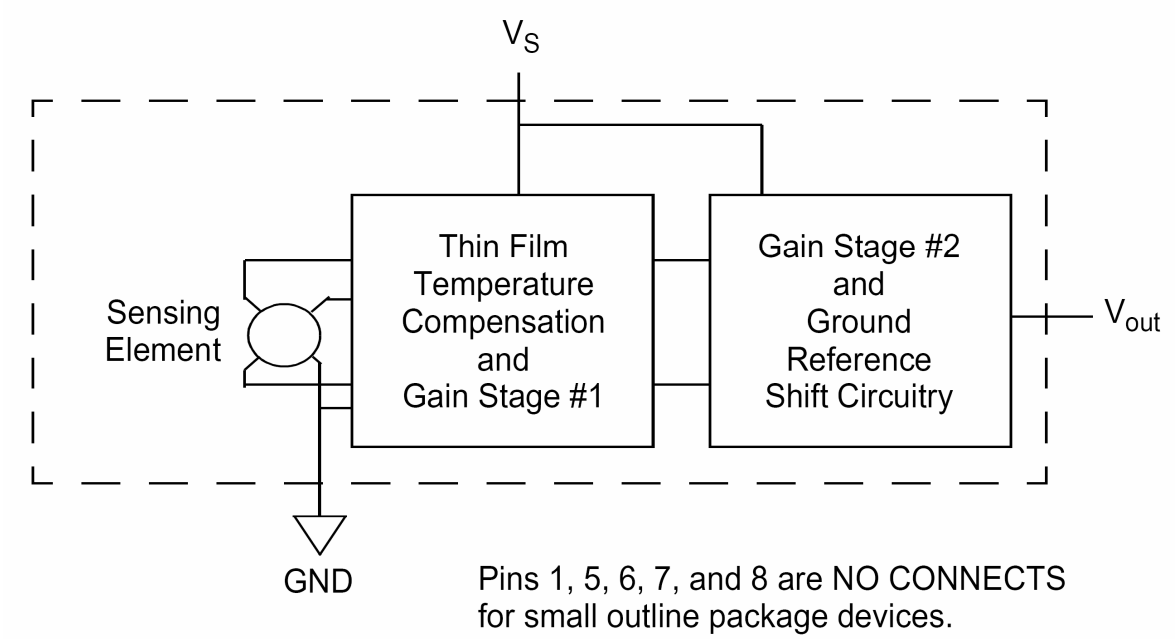
5. 3 Prevedenie integrovaného senzoru tlaku

Integrované senzory tlaku sú obvykle povrchovo integrované mikrozariadenia, ktoré sa skladajú z troch častí:

- Tlakový senzor – povrchovo integrovaná meracia bunka (sensing element)
- Teplotný senzor (len v niektorých prípadoch)
- Analógové alebo digitálne integrované obvody pre spracovanie signálov zo senzorov



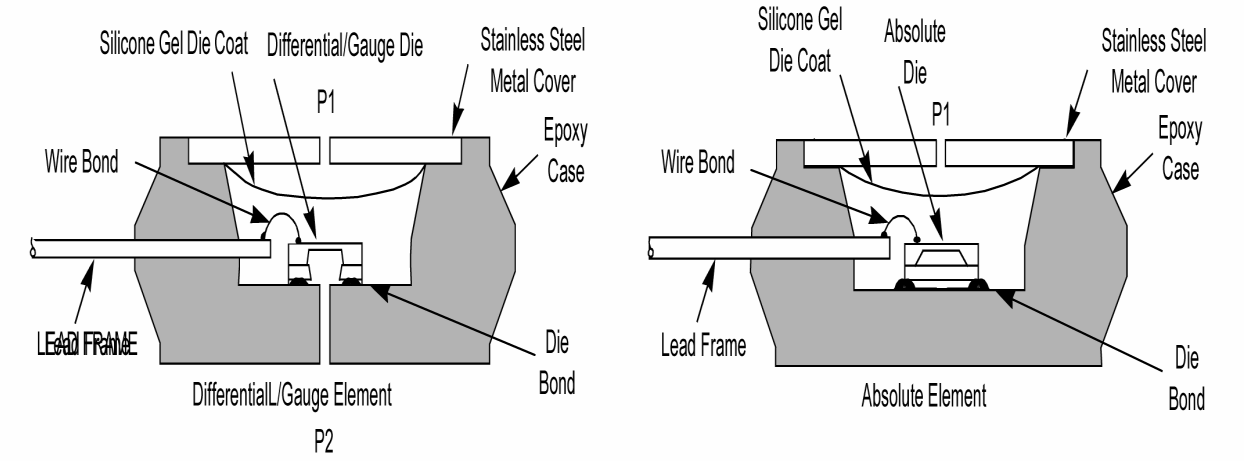
Obr. 5. Postupnosť blokov senzorov prevádzajúcich tlak P na hodnotu napätia V [16]



Obr. 6. Bloková schéma štruktúry implementovaná na čipe senzoru [8]

Na obr. 5. a 6. je príklad blokov, ktoré sa na čipe senzoru najčastejšie objavujú. Niekedy sa na čipe objavujú dve meracie bunky, kedy jedna bunka meria žiadaný pôsobiaci tlak a druhá funguje ako referenčná pre elimináciu vplyvu teploty (poprípade iných nežiaducich vplyvov). V prípade digitálneho výstupu na senzore, druhý blok zapracovania signálu obsahuje prevod analógového signálu na nejaký druh digitálneho, resp. diskrétneho.

Z hľadiska použitia je citlivý merací senzor nutné chrániť krytom na meracej strane a pevnou epoxidovou schránkou proti mechanickému poškodeniu. (viď obr. 7.). Aby nedošlo k poškodeniu meracieho elementu, je chránený z meracej strany silikónovým dielektrickým gélom.



Obr. 7. Príklad mechanického riešenia puzdra chrániaceho citlivý senzor [17]

Ďalej je možne rozdeliť senzory do dvoch skupín, podľa vplyvu vnútornej logiky a spracovávania na samotné meranie na:

- Teplotne kompenzované (compensated)
- Teplotne nekompenzované (uncompensated)

5. 4 Princíp a štruktúra snímacích elementov

Celá mechanická štruktúra citlivej časti senzorov je dnes vytváraná z kremíku a integrovaná na jednom monolitickom obvode s ostatnou elektronikou. Princíp merania, resp. prevod pôsobiaceho tlaku na elektrickú veličinu, je často založený na jednom z nasledujúcich princípov:

- Zmena kapacity kondenzátoru pôsobením tlaku
- Zmena odporu dráhy, na ktorú pôsobí tlak (tenzometre)
- Piezoelektrický jav, tzn. vznik napätia s pôsobiacim tlakom

Prvý prípad, integrovaná kapacitná meracia bunka, resp. bunky, sa dnes stále využíva pre meranie veľkých tlakov a v absolútnych senzoroch tlaku. Tu sa využíva porovnanie kapacít meracieho kondenzátoru, tvoreného jednou pevnou a jednou pružnou kremíkovou elektródou a referenčného kondenzátoru pre elimináciu nežiaducich okolitých vplyvov (napr. teploty).

Druhý prípad je tradičný systém, ktorý sa vyznačuje integrovaným Wheatstonovým mostíkom zloženým zo štyroch tenzometrov (odporových ciest). Tie menia svoj odpor so zmenou svojich rozmerov pri prehýbaní membrány vplyvom pôsobiaceho tlaku a tak prevádzajú pôsobiaci tlak na elektrickú veličinu.

Tretí prípad, ten najnovší, využíva kremíkové piezorezistory, ktoré pri pôsobení tlaku priamo generujú elektrické napätie na dvoch kontaktných plôškach.

5. 5 Senzor MPXH6250A od firmy Freescale

Tlakové senzory patriace do série MPXH6250A obsahujú integrované kremíkové snímače tlaku pre meranie absolútneho tlaku v rozmedzí 20 kPa až 250 kPa, ktoré majú vysokú tepelnú presnosť. Signál spracovávaný v čipe je kalibrovaný a tepelne kompenzovaný. Integrované senzory tlaku z tejto série obsahujú na jednom čipe dvojpólový operačný zosilňovač a tenkú tepelne odporovú vrstvu pre poskytnutie vysokého výstupného signálu a tepelnej kompenzácie. Senzory majú malý tvarový činiteľ a vysokú spoľahlivosť zapojenia na čipe. Tlakové senzory zo série MPXH6250 využívajú najnovšiu metódu prevodu tlaku na elektrické napätie piezoelektrický jav. Piezorezistorový snímač a obvody pre spracovanie signálu (zosilnenie a tepelná kompenzácia) integrované do jedného monolitického obvodu zaraďujú tuto sériu medzi najmodernejšie kremíkové snímače tlaku. Tieto snímače kombinujú pokročilé mikromechanické techniky, tenké metalizovanie a dvojpólove polovodičové spracovanie signálu, ktoré poskytujú presný analógový výstupný signál s vysokou úrovňou, ktorý je úmerný k aplikovanému tlaku. Blokové zapojenie obvodov integrovaných v tlakovom senzore MPXH6250 ukazuje obr. 6..

6 Snímací modul tlaku – výškomer

6. 1 Barometrické meranie výšky

V praxi sa používa veľa prístrojov založených na zmene atmosférického tlaku. Jedným z týchto prístrojov je tlakový výškomer. Tlakový výškomer sa používa v leteckej doprave a v prenosnej verzii ho používajú napr. horolezci a potápači počas svojich výprav. Nevýhodou tohto výškomeru je závislosť na momentálnom tlaku vzduchu, a preto je výškomer potrebné pred použitím kalibrovať. Tlakový výškomer je v skutočnosti barometer, ktorý nameraný tlak prepočítava (na základe matematického modelu štandardnej atmosféry) a zobrazuje v jednotkách výšky. Barometrický výškomer využíva pokles tlaku so zvyšujúcou sa nadmorskou výškou. Závislosť hodnoty tlaku na výške je popísaná barometrickou rovnicou (viď [13])**:**

$$p = p_0 . e^{\left(-\frac{M_m . g . h}{R_m . T} \right)} . \qquad (1)$$

p_0 – atmosférický tlak na nulovej výške
p – atmosférický tlak v danej výške
M_m – molárna hmotnosť vzduchu
R_m – molárna plynová konštanta
g – tiažové zrýchlenie
h – výška
T – teplota v Kelvinoch

Rovnicu (1) môžme zjednodušiť pomocou vzorca na výpočet univerzálnej plynovej konštanty

$$R = \frac{R_m}{M_m} = \frac{8{,}314472}{29.10^{-3}} = 286{,}706 \quad \left[\frac{m^2}{s^2 . K} \right] \qquad (2)$$

a dosadením konštánt.
R – univerzálna plynová konštanta

Po zjednodušení rovnice (1) dostaneme:

$$p = p_0 . e^{\left(-\frac{h}{T} . 0{,}034216 \right)} \quad [\text{Pa}].$$

Upravením rovnice (1) a tiež dosadením do nej, môžme odvodiť vzťah pre výpočet výšky v závislosti na zmene tlaku:

$$h = -\ln\left(\frac{p}{p_0} \right) . \frac{R.T}{g} = -\ln\left(\frac{p}{p_0} \right) . T . 29{,}2259 \quad [\text{m}].$$

Za predpokladu, že výškový rozdiel uvažovaných hladín nie je príliš veľký, je možné použiť Babinetov vzorec:

$$h = 16000.(1 + 0{,}004.t_m).\frac{p_0 - p}{p_0 + p}. \qquad \textbf{(3)}$$

t_m – priemerná teplota medzi dvoma hladinami [C°]

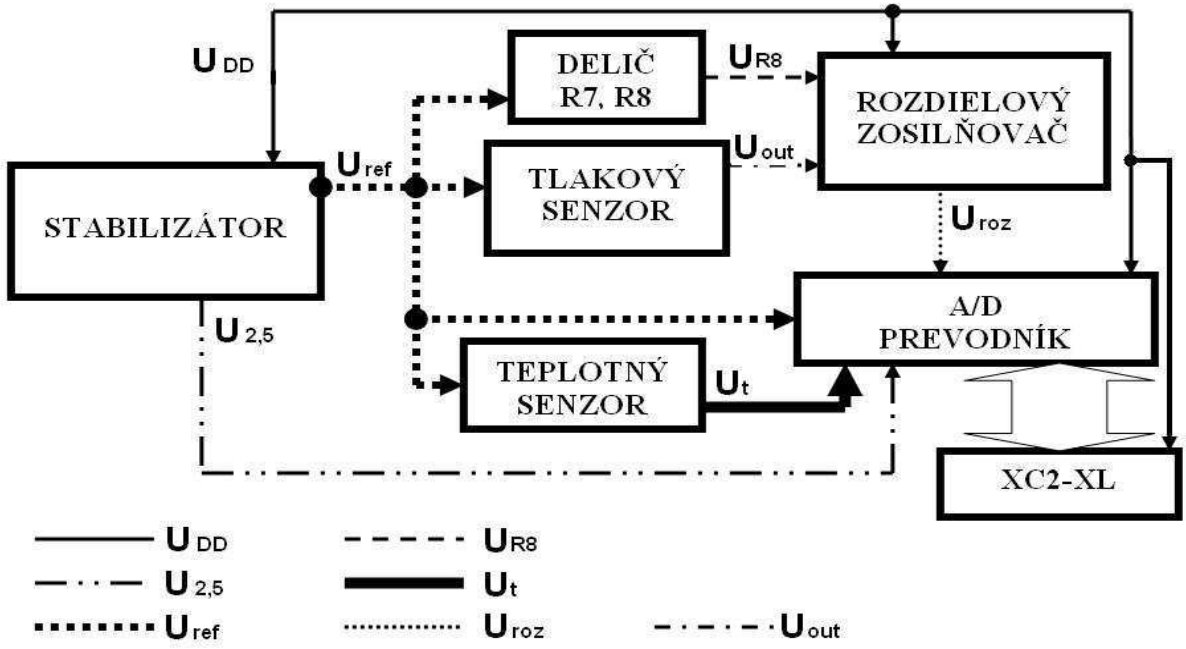
$p_0 > p$

Presnosť tohto vzorca je nepriamo úmerná vzdialenosti uvažovaných hladín. Z toho dôvodu je možné tento vzorec uspokojivo použiť iba v prípadoch, keď vzdialenosť obidvoch hladín neprekročí 1000 m (viď [18]).

6. 2 Princíp spracovania signálu v snímacom module

Zapojenie elektronickej časti snímacieho modulu sa skladá zo šiestych blokov (stabilizátor, odporový delič, tlakový senzor, teplotný senzor, rozdielový zosilňovač a A/D prevodník) (viď obr. 8.). Napájanie rozdielového zosilňovača, A/D prevodníka a stabilizátora napätím U_{DD} je privedené z vývojovej dosky. Vzhľadom k tomu, že aj malá zmena vstupného napätia U_{ref} na senzoroch, odporovom deliči a referenčného napätia U_{ref} na A/D prevodníku spôsobí zmenu výstupného napätia na senzoroch a veľkú zmenu výsledného vyhodnotenia výšky, bolo potrebné stabilizovať napätie, ktoré je potom privádzané na jednotlivé bloky v zapojení.

Stabilizované napätie U_{ref} privedené do tlakového senzoru sa v tlakovom senzore zmení v závislosti od aktuálneho atmosférického tlaku. Výstupné napätie U_{out} tlakového senzoru je privedené na jeden zo vstupov rozdielového zosilňovača. Na druhý vstup rozdielového zosilňovača je privedené výstupné napätie U_{R8} z odporového deliča (odporový delič upravuje stabilizovane napätie U_{ref} na napätie rovné najmenšiemu možnému napätiu U_{R8}, ktoré sa môže vyskytnúť na výstupe tlakového senzoru, vzhľadom k atmosférickému tlaku). V rozdielovom zosilňovači je zosilnený rozdiel týchto dvoch napätí (U_{out} a U_{R8}). Výstupné napätie rozdielového zosilňovača U_{roz} (zosilnený rozdiel napätí U_{out} a U_{R8}) je privedené na jeden zo vstupov A/D prevodníka. Úlohou rozdielového zosilňovača je vybrať a zosilňovať iba užitočné napätie zo senzora, ktoré sa pohybuje v intervale $<U_{out}(\min), U_{out}(\max)>$, tj. rozdiel napätia na výstupe tlakového senzoru U_{out} a najmenšieho možného napätia U_{R8}, ktoré sa môže na tlakovom senzore objaviť vzhľadom k atmosférickému tlaku.



Obr. 8. Bloková schéma zapojenia snímacieho modulu

Stabilizované napätie U_{ref} privedené na vstup teplotného senzoru sa v teplotnom senzore mení v závislosti od zmeny aktuálnej okolitej teploty. Výstupné napätie U_t teplotného senzora je privedené na druhý vstup A/D prevodníku. V A/D prevodníku je napätie z teplotného senzora U_t a napätie z rozdielového zosilňovača U_{roz} prevedené z analógovej formy signálu do digitálnej formy signálu (paralelný dátový tok). Napätie U_{ref} privedené zo stabilizátora k A/D prevodníku nastavuje hodnotu napätia najnižšieho platného bitu LSB. Vlastne nastavuje v akých napäťových intervaloch bude signál na vstupoch kvantovaný (nastavuje hodnotu jednej kvantovacej hladiny). Napätie $U_{2,5}$ privedené taktiež zo stabilizátora na A/D prevodník nastavuje s akou napäťovou logikou bude A/D prevodník pracovať (nastavuje s akým napätím bude pracovať paralelné rozhranie A/D prevodníka).

6. 3 Popis a funkcia blokov snímacieho modulu

6. 3. 1 Stabilizátor

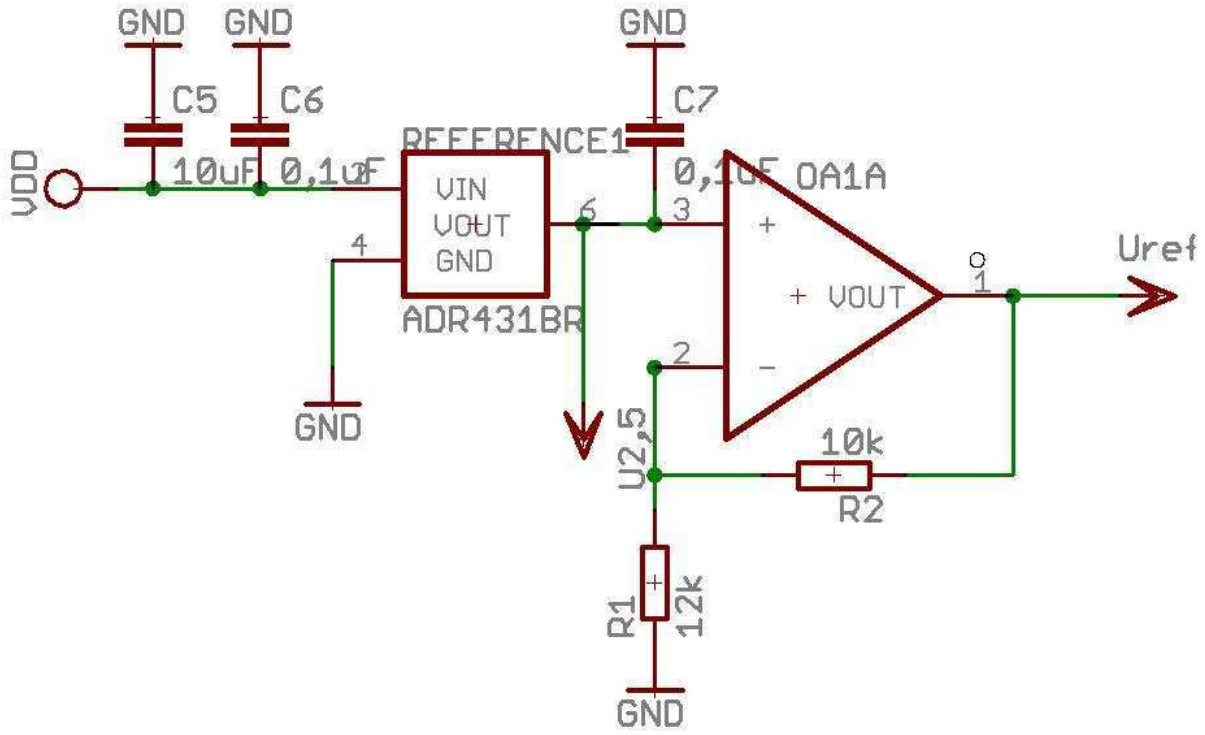
Zapojenie stabilizátora (viď obr. 9.) je tvorené z 2,5 V referencie ADR431BR a operačného zosilňovača AD8552AR, ktorý je zapojený ako neinvertujúci zosilňovač. Hodnoty rezistorov R2=10k a R1=12k boli zvolené tak, aby U_{ref} bolo menej než 5V.

U_{ref} vypočítame podľa vzťahu:

$$U_{ref} = \left(1 + \frac{R_2}{R_1}\right).2{,}5V\,, \qquad \textbf{(4)}$$

$$U_{ref} = \left(1 + \frac{10}{12}\right).2{,}5V = 4.583\bar{3}V\,.$$

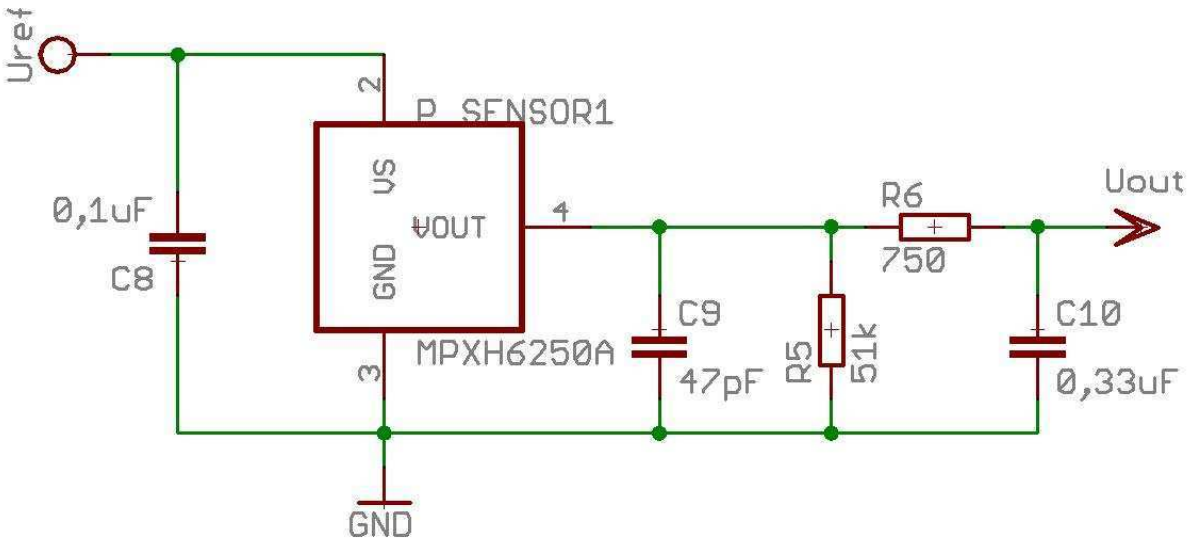
Vstupné napätie stabilizátora je napätie U_{DD} zo zdroja a výstupne napätia sú napätie $U_{2,5}$ a U_{ref}.



Obr. 9. Zapojenie stabilizátora s OZ [12]

6. 3. 2 Tlakový senzor

Zapojenie tlakového senzoru (obr. 10.) je typická aplikácia obvodu MPXH6250 (viď [8]), plus dolná priepust R6 a C10, ktorá filtruje šum a zlepšuje stabilitu signálu zo senzoru (viď [9]). Napätie ktoré napája senzor je stabilizované napätie U_{ref} zo stabilizátora.



Obr. 10. Zapojenie tlakového senzoru

Zapojenie bolo navrhnuté pre bežný atmosférický tlak v rozsahu 750 – 1100 hPa. Pomocou Barometrickej rovnice (1) bol vypočítaný pokles tlaku *p* do výšky 2000 m. n. m. pri teplote T=298,15°K=25°C:

$$p = 750.e^{\left(-\frac{2000\ .\ 9,81}{286,706\ .\ 298,15}\right)},$$

$$\underline{\underline{p = 596,184hPa \cong 600hPa}}.$$

Takže meraný atmosférický tlak bude v rozsahu 600 hPa – 1100 hPa. Z toho dôvodu bol vybraný senzor MPXH6250, ktorého merací rozsah je v intervale (20 kPa – 250 kPa) a výstupné napätie tohto senzoru je popísane vzťahom:

$$U_{out} = U_{ref}.(0,004.p - 0,04) \text{ (viď [8]).} \qquad (5)$$

U_{out} – výstupné napätie
U_{ref} – referenčné napätie
p – tlak v kPa

Dosadením spodnej a hornej hranice bežného atmosférického tlaku vo výške 2000 m. n. m. do vzťahu (5) dostaneme spodnú a hornú hranicu rozsahu, v ktorom sa bude výstupné napätie senzoru U_{out} pohybovať:

$$U_{out} = U_{\min} = 4,583\overline{3}.(0,004.60 - 0,04) = 0,916\overline{6}V,$$

$$U_{out} = U_{\max} = 4,583\overline{3}.(0,004.110 - 0,04) = 1,83\overline{3}V.$$

Z vypočítaných hodnôt vieme, že napätie na výstupe tlakového senzoru U_{out} sa bude pohybovať v intervale <0,9166V;1,833V>.

Základným parametrom, ktorý bude určovať výsledné rozlíšenie je citlivosť tlakového senzoru, ktorá je uvedená v datasheet MPXH6250. Pre senzor MPXH6250 je to 20,4 mV/kPa = 2,04 mV/hPa. Pomocou barometrickej rovnice (1) môžme vypočítať zmenu výšky *h* ak sa zmení tlak o 1 hPa, čiže výstupné napätie sa zmení o 2,04 mV:

$$h = -\ln\left(\frac{p}{p_0}\right).\frac{R.T}{g},$$

$$h = -\ln\left(\frac{999}{1000}\right).\frac{286{,}706\ .\ 273{,}15}{9{,}81},$$

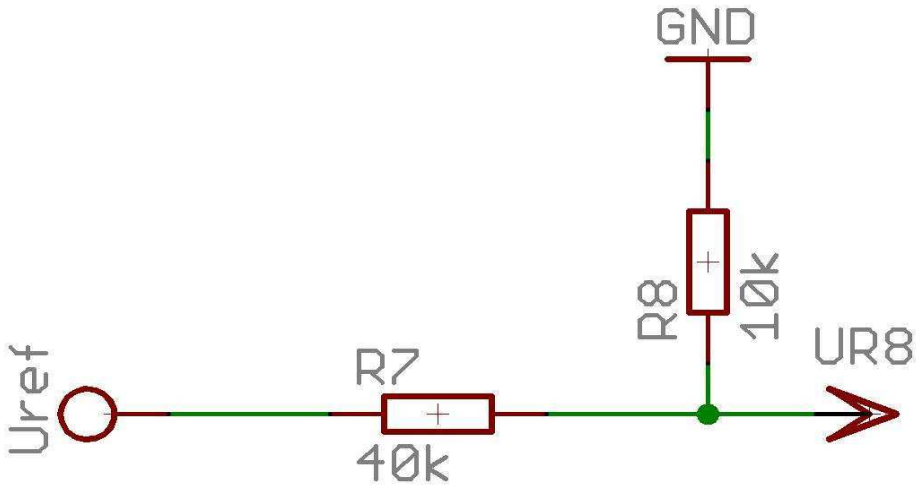
$$\underline{\underline{h = 7{,}987 \cong 8\ \ m}}.$$

Pokles tlaku o 1 hPa spôsobí zmenu výšky o 8 m a zároveň sa zníži výstupné napätie na senzore U_{out} o 2,04 mV. Podobne môžme určiť, že zmena o 1 m výšky vyvolá zmenu výstupného napätia na senzore U_{out} približne o 2,04/8=0,255 mV.

6. 3. 3 Odporový delič

Na vstup odporového deliča (obr. 11.) je privedené referenčné napätie U_{ref} zo stabilizátora a na výstupe je napätie U_{R8}. Hodnota napätia U_{R8} je rovná napätiu U_{min}. Teda platí:

$$U_{R8}=U_{min}=0{,}9167\ \text{V}.$$



Obr. 11. Odporový delič *R7, R8*

Ak chceme, aby platilo $U_{R8}=U_{min}$ musí platiť $R7=4R8$. Pretože:

$$Uref = (R7 + R8).I,$$

$$I = \frac{Uref}{R7 + R8},$$

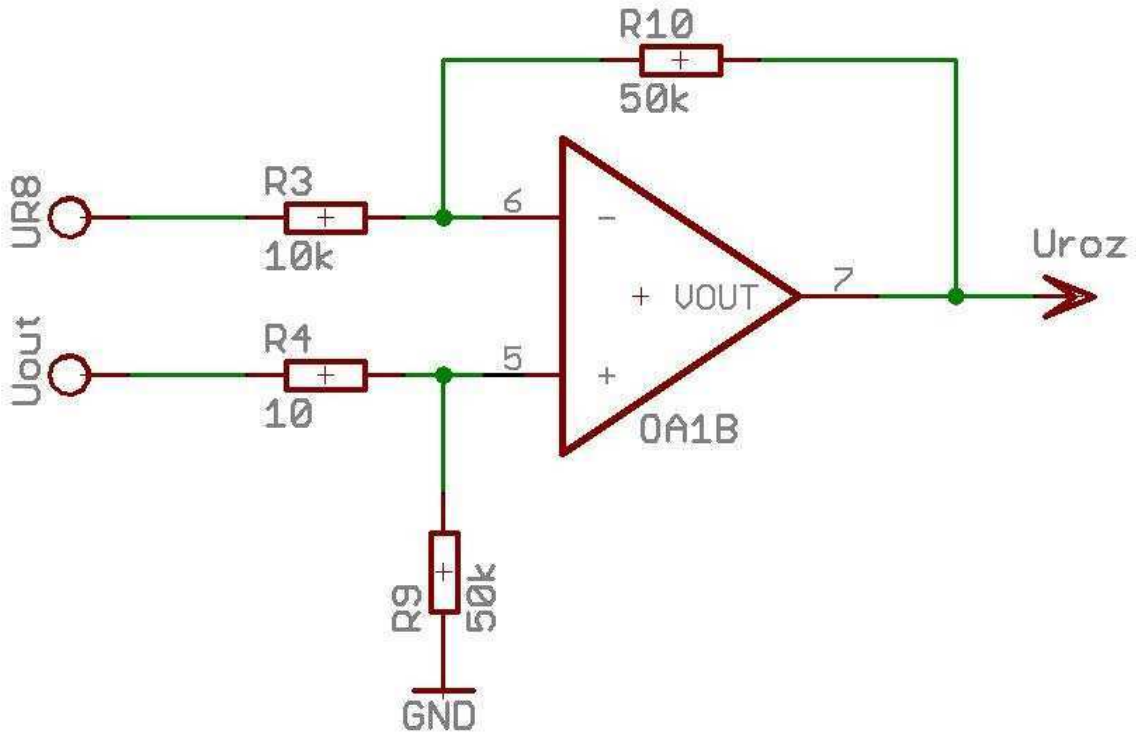
$$U_{R8} = R8.I = R8.\frac{Uref}{R7 + R8},$$

$$0{,}916\overline{6} = R8.\frac{4{,}583\overline{3}}{R7 + R8},$$

$$\underline{\underline{R7 = 4R8}}.$$

6. 3. 4 Rozdielový zosilňovač

Rozlíšenie výškomeru záleží aj na veľkosti zosilnenia zosilňovača. Vzhľadom k tomu, že užitočné napätie na výstupe senzoru sa pohybuje v rozsahu od U_{min}=0,9166V do U_{max}=1,833V bolo na zosilnenie výstupného signálu zo senzora zvolené zapojenie rozdielového zosilňovača s OZ (obr.12.). Úlohou rozdielového zosilňovača je zosilniť rozdiel výstupného napätia U_{out} z tlakového senzora a najnižšej možnej hodnoty napätia U_{min}, ktorú je senzor schopný vyprodukovať vzhľadom k atmosférickému tlaku (600 – 1100 hPa). Zosilňovanie tohto rozdielu umožňuje nastaviť väčšie zosilnenie na zosilňovači, pretože výstupné napätie zosilňovača je obmedzené napájacím napätím U_{DD} (U_{max}=1,833V je možné zosilniť 2,5-krát, ale U_{max}-U_{min}=0,9166V je možné zosilniť 5-krát). Väčšie zosilnenie zosilňovača zabezpečí väčšie rozlíšenie výškomeru.



Obr. 12. Zapojenie rozdielového zosilňovača s OZ

Ak platí: *R3=R4* a *R9=R10*, potom platí:

$$U_{roz} = A.(U_{out} - U_{R8}), \tag{6}$$

$$A = \frac{R10}{R3} = \frac{R9}{R4} \quad \text{(viď [19]).} \tag{7}$$

Výstupné napätie na senzore U_{out} sa bude pohybovať v intervale $U_{out} \in (U_{min}; U_{max})$. Toto napätie zo senzora U_{out} privedieme na neinvertujúci vstup OZ cez rezistor *R4* respektíve cez delič *R4*, *R9* (priamo na neinvestujúcom vstupe OZ bude napätie U_{R9}). Ďalej privedieme cez rezistor *R3* na invertujúci vstup OZ napätie U_{R8}, ktoré bude odvodené z referenčného

napätia U_{ref} pomocou odporového deliča *R7, R8* a jeho hodnota bude rovná napätiu U_{min}. Teda bude platiť:

$$U_{R8}=U_{min}$$
$$U_{roz} = A.(U_{out} - U_{\min}) .$$

Zo vzťahu (6) vyplýva, že zosilňovač bude zosilňovať iba rozdiel medzi aktuálnou hodnotou napätia na výstupe senzoru U_{out} a najnižšou možnou hodnotou napätia U_{min}, ktorú je senzor schopný vyprodukovať vzhľadom k atmosférickému tlaku (600 – 1100 hPa). Ak je na výstupe tlakového senzoru maximálne napätie $U_{out}=U_{max}=1,833$ V. Dosadením U_{max} do vzťahu (6) dostaneme na výstupe zosilňovača U_{roz} maximálne možné napätie vzhľadom k atmosférickému tlaku (600 – 1100 hPa):

$$U_{roz}(\max) = A.(1,83\bar{3} - 0,916\bar{6}) = A.0,916\bar{6}V .$$

Zosilnenie *A* môžme vypočítať pomocou vzťahu (7). Vieme, že referenčné napätie na A/D prevodníku je $U_{ref}=4,5833$V. Z toho dôvodu môžme na vstup A/D prevodníku priviesť napätie s maximálnou veľkosťou U_{ref}. Musí platiť:

$$U_{roz} \le U_{ref} .$$

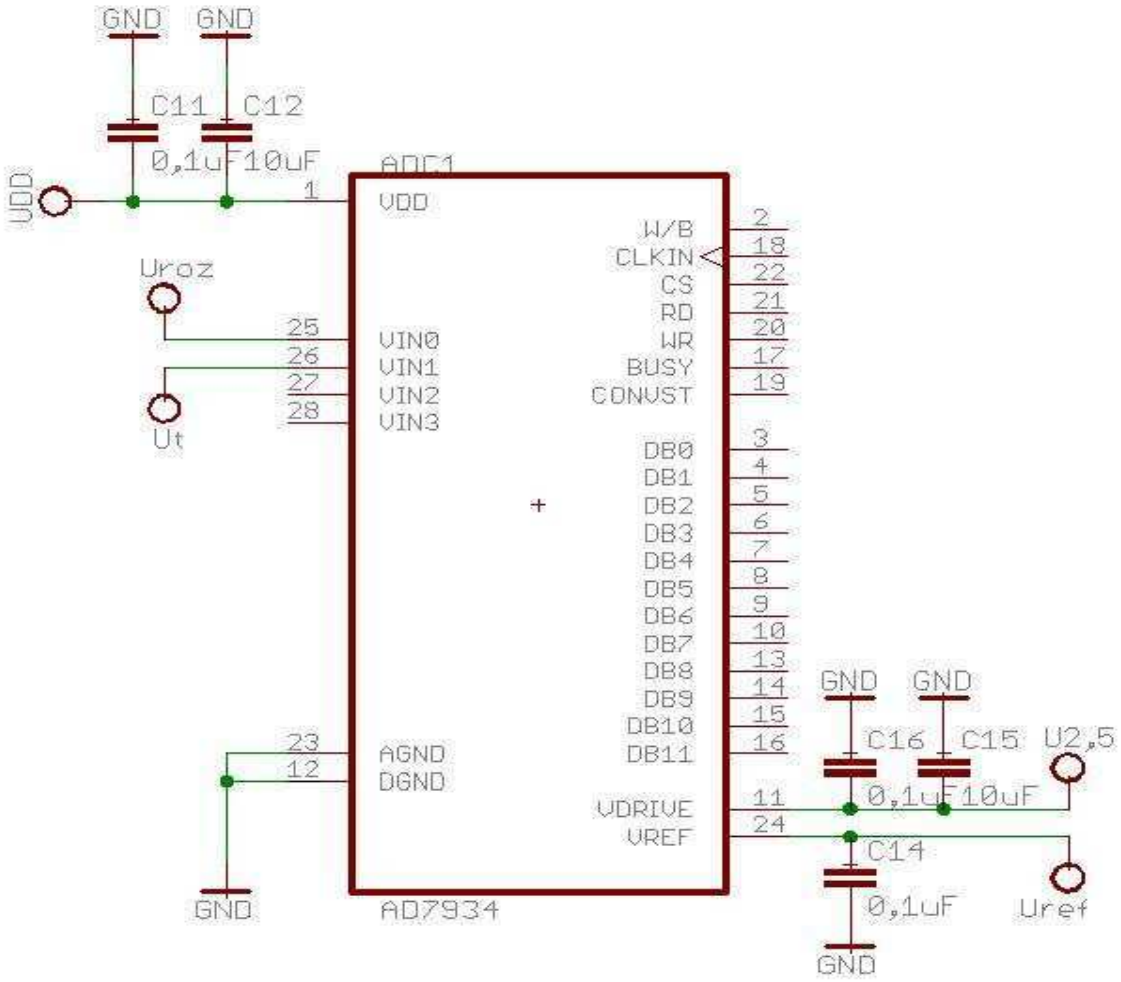
Z toho vyplýva, že zosilnenie rozdielového zosilňovača môže byť maximálne *A(max)*=5, pretože:

$$A(\max) = \frac{U_{ref}}{U_{roz}(\max)} = \frac{4,583\bar{3}}{0,916\bar{6}} = 5 .$$

Ak poznáme veľkosť zosilnenia *A* môžme vypočítať pomocou vzťahu (7) hodnoty rezistorov *R3, R4, R9, R10:*

$$\text{napr.: } A = \frac{R10}{R3} = \frac{R9}{R4} = \frac{50k\Omega}{10k\Omega} = 5 .$$

6. 3. 5 A/D prevodník



Obr. 13. Zapojenie A/D prevodníka AD7934

Bol vybraný 12 bitový prevodník AD7934, ktorý má 4 vstupy a na výstupe má paralelný dátový tok. Zapojenie A/D prevodníka AD7934 na (obr.13.) je typická aplikácia zapojenia A/D prevodníka AD7934 (viď [4]).Na vstupy A/D prevodníka sú privádzané napätia z teplotného senzora a z rozdielového zosilňovača. Na pin V_{drive} je privedené napätie $U_{2,5}$= 2,5V priamo zo stabilizátora. Toto napätie nastavuje s akým napätím bude paralelne rozhranie ADC pracovať. Ďalšie napätie, ktoré je privádzane na ADC, je U_{ref}=4,5833 V. Toto napätie je privedené na pin V_{ref}. Napätie U_{ref} nastavuje hodnotu najnižšieho platného bitu (LSB). Vlastne nastavuje v akých napäťových intervaloch bude signál na vstupoch kvantovaný (nastavuje hodnotu jednej kvantovacej hladiny):

$$LSB = \frac{U_{ref}}{2^{12}} = \frac{U_{ref}}{4096} = \frac{4{,}5833V}{4096} = 0{,}001119V\ .$$

Rozlíšenie výškomeru záleží hlavne na rozlíšení A/D prevodníka. Vieme, že zmena o 1 m výšky vyvolá zmenu výstupného napätia na senzore U_{out} približne o 2,04/8=0,255 mV. Keďže toto napätie bude zosilnené 5-krát, tak na výstupe zosilňovača vyvolá zmena výšky o 1 m zmenu napätia na výstupe rozdielového zosilňovača U_{roz} približne o 0,000255.5=0,001275V/m. Teda celkové rozlíšenie merania výšky bude približne 0,9m:

$$\frac{LSB}{0{,}001275\frac{V}{m}} = \frac{0{,}001119V}{0{,}001275\frac{V}{m}} = 0{,}8776m \cong 0{,}9m\,.$$

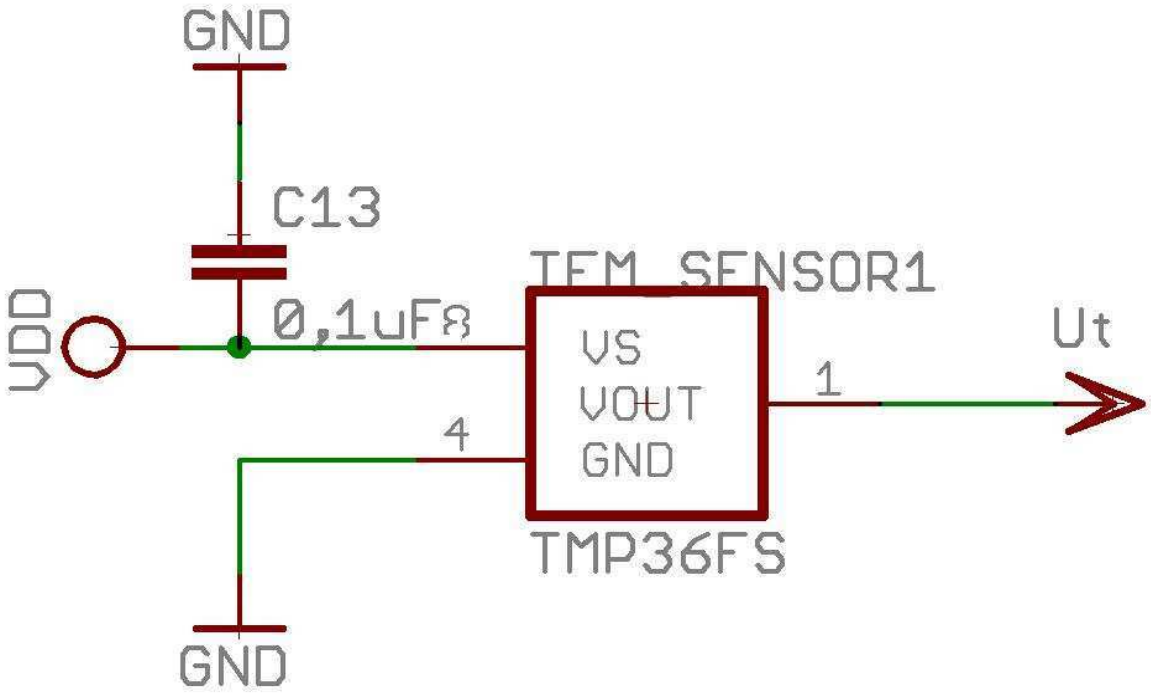
Referenčné napätie na A/D prevodníku je U_{ref}=4,5833 V. V prípade, že by nebol použitý referenčný zosilňovač, tak pre rozlíšenie aspoň 1 m, by bolo treba A/D prevodník, ktorý ma 4,5833V/0,000255V=17973,7 úrovní (t. j. aspoň 15 bitový prevodník, pretože 2^14=16384 –málo a 2^15=32768 – dosť úrovni).

6. 3. 6 Teplotný senzor

Úlohou tohto senzoru je merať teplotu. Zmeraná teplota sa využije v barometrickej rovnici na výpočet výšky. Výstupné napätie sa lineárne mení v závislostí na teplote. Tato závislosť senzoru TMP36FS je na obr. 16. a v datasheet [10]. Vzťah (8) je vzťah ktorým sa dá táto závislosť aproximovať :

$$T = Ut.0{,}1 + 223 \quad [°\mathrm{K}] \qquad (8)$$

Napätie U_t privedieme na jeden zo vstupov na ADC. Okamžitú teplotu potom vyjadríme pomocou algoritmov na výpočet teploty a napätia U_t.



Obr. 14. Zapojenie teplotného senzoru

Senzor ma citlivosť výstupného napätia 10mV na 1° C. ADC ma hodnotu najnižšieho platného bitu:

$$LSB = \frac{U_{ref}}{2^{12}} = \frac{U_{ref}}{4096} = \frac{4{,}5833V}{4096} = 0{,}001119V\,.$$

Rozlíšenie merania teploty môže dosahovať 0,1119°C:

$$\frac{0{,}001119V}{0{,}010V/°C} = 0{,}1119°C\,.$$

Vzhľadom k tomu, že algoritmus, ktorý počíta aktuálnu teplotu zaokrúhľuje na celé čísla, tak v tejto aplikácii nie je možne dosiahnuť vypočítane rozlíšenie. Ak by sme chceli dosiahnuť kvalitnejšie rozlíšenie než jeden stupeň tak by bolo vhodne použiť iný teplotný senzor, vzhľadom na kolísanie výstupného napätia, ale pre aplikáciu v ktorej je použitý nadštandardne postačuje.

7 Algoritmus spracovania digitálneho signálu

Bloková schéma algoritmu ja zobrazená v prílohe F. Skladá sa z jedenástich blokov (Konfigurácia ADC, tabuľka pre prevod tlakového signálu, blok pre prevod teplotného signálu, blok pre výpočet nekalibrovanej výšky, kalibrácia – skladá sa z nastavenia kalibračnej výšky a výpočtu rozdielu nameranej a kalibračnej výšky, blok pre výpočet aktuálnej výšky, blok zabezpečujúci výber zobrazovanej hodnoty, blok pre prevod binárneho reťazca na BCD kód, blok pre zobrazenie BCD kódu na displeji a zobrazovací displej). V prílohách B-D sú zobrazené výsledky niektorých simulácii jednotlivých blokov.

Konfigurácia ADC, ma za úlohu nastaviť riadiace signály na ADC a uložiť požadované bity do riadiacich registrov. Okrem toho táto časť programu rozdelí výstupný signál z ADC na dva signály reprezentujúce tlak a teplotu. Ďalej sa signály spracovávajú v dvoch vetvách. V tabuľke pre prevod tlaku, ktorá ma za úlohu previesť tlakový signál na číslo, ktoré bude potrebné v ďalších výpočtoch a v bloku pre prevod teplotného signálu, ktorý ma za úlohu vypočítať teplotu v kelvinovej stupnici. Po získaní týchto hodnôt sa tieto hodnoty vynásobia. Vynásobením teploty a hodnoty získanej z tabuľky pre prevod tlaku vypočítame nekalibrovanú výšku.

V časti kalibrácia sa nastaví kalibračná výška a vypočíta sa rozdiel medzi nekalibrovanou výškou a kalibračnou výškou. V ďalšom bloku pre výpočet aktuálnej výšky sa rozdiel vypočítaný v časti kalibrácia odpočíta od nekalibrovanej výšky a tým dostávame výsledok v binárnom tvare, ktorý reprezentuje aktuálnu skalibrovanú nadmorskú výšku. Blok zabezpečujúci výber zobrazovanej hodnoty vyberie podľa toho, ktoré tlačidlo je stlačené, respektíve nestlačene, binárne číslo reprezentujúce skalibrovanú nadmorskú výšku, alebo kalibračnú výšku, alebo teplotu. V ďalšom kroku sa táto vybraná hodnota prevedie na BCD kód a potom sa zobrazí na segmentovom displeji.

7. 1 Konfigurácia ADC

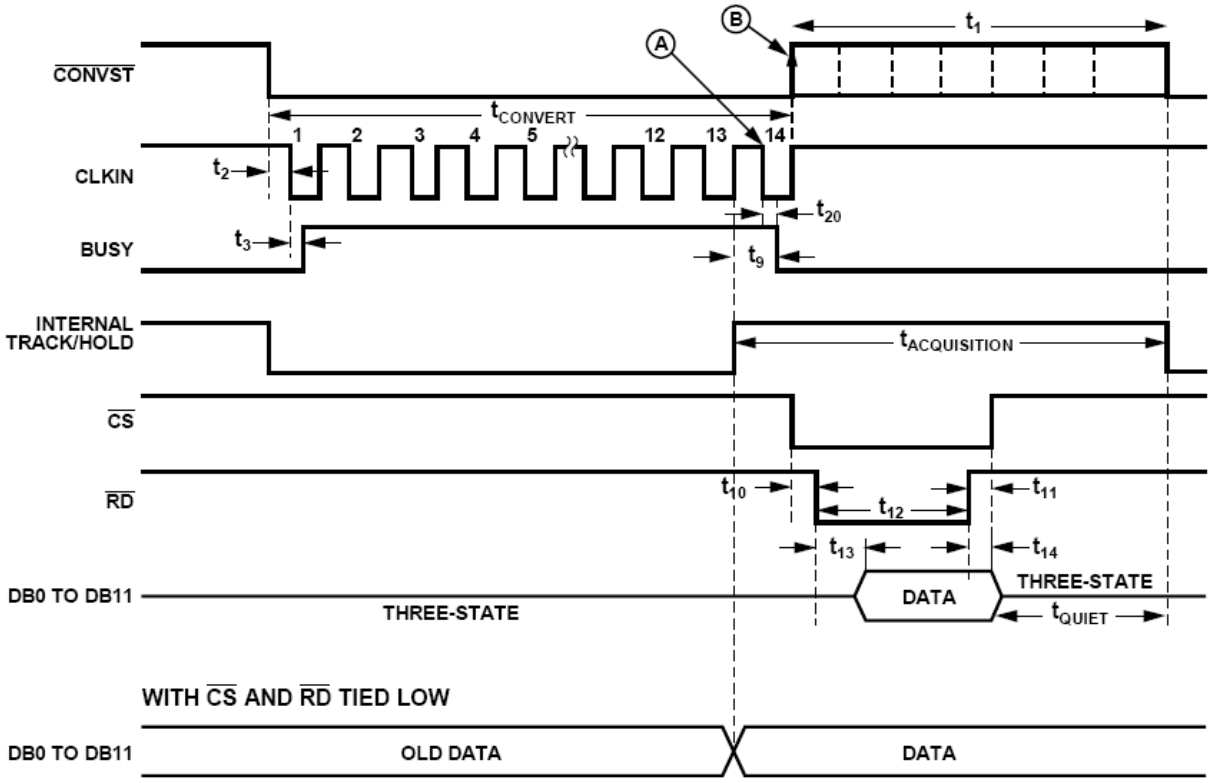
Navrhnutý modul na meranie nadmorskej výšky používa na prevod analógového signálu analógovo - digitálny prevodník AD7934 od firmy Analog Devices. Podrobné vlastnosti tohto ADC nájdeme v literatúre [4]. Výstup ADC je 12 bitový, takže budeme spracovávať 12 bitové binárne číslo reprezentujúce tlak, alebo teplotu. Podľa nastavenia riadiaceho registru, je možne využívať viacero pracovných režimov tohto ADC. Ďalej je dôležité správne nastaviť riadiace signály AD prevodníka. Priebehy riadiacich signálov pri konverzii vidíme na obr. 15..

AD7934 môže pracovať s 12 bitovým slovom (W/B – vysoká úroveň), alebo v bajtovom režime (W/B – nízka úroveň). CONVST signál sa používa pre iniciáciu prevodu. Padajúca hrana tohto signálu spusti konverziu analógového signálu na digitálny. Medzi dvoma za sebou nasledujúcimi konverziami musí byt CONVST vo vysokej úrovni minimálne po dobu 10ns. Konverzia musí trvať najmenej 14 padajúcich hrán. V prípade, že konverzia bude trvať menej, než 14 padajúcich hrán, bude prevod analógového signálu na digitálny neúspešný. Pre čítanie dát z ADC je potrebne paralelne aktivovať nízkou úrovňou signály RD a CS. Pre zápis do kontrolného registra, je potrebne paralele aktivovať nízkou úrovňou signály WR a CS.

V kontrolnom registri sa nastavuje:

- pracovný režim ADC
- vstup, ktorý bude aktívny
- interne, alebo externe referenčné napätie
- nastavenie sekvenčného prevodu vstupov

- kódovanie výstupu v prirodzenom binárnom tvare alebo v dvojkovom doplnku
- nastavenie vstupného rozsahu napätia



Obr. 15 Priebehy konverzie a čítania v 12 bitovom režime [4]

Pre správnu funkciu aplikácie je treba ovládať päť riadiacich signálov (W/B, CONVST, RD, WD, CS) a do riadiaceho registru nahrávať dve kombinácie 12 bitových slov. Prvá kombinácia nastaví AD prevodník na prevod signálu z prvého vstupu a druha kombinácia nastaví prevod signálu z druhého vstupu.

V texte TXT1 vidíme časť zápisu konfigurácie ADC vo VHDL jazyku. Proces reaguje na padajúcu hranu hodinového signálu clk. Ak bude premenná `write_read_int` a `sensor_int` mat úroveň nula, tak na výstup `drive_signals` sa uloží "110010", (`drive_signals` = "W/B; CONVST; WR; RD; CS") a na obojsmerný vývod `convert_signals` sa uloží "000000000000". Signál `counter1_int` sa inkrementuje o 1, dokiaľ sa nebude rovnať binárnej dvojke ("10"). Ak sa `counter1_int` rovná binárnej dvojke ("10"), tak `write_read_int` sa nastaví na hodnotu jedna a tým sa ukončí nastavovanie riadiaceho registra. Počas inkrementácie signálu `counter1_int` riadiace signály (`drive_signals`) nastavujú ADC tak, aby bol schopný zápisu do registra a `counter_signals` nastavujú register na prevod signálu z prvého vstupu AD prevodníka.

Ak sa `write_read_int` rovná jednej a `sensor_int` sa rovná nule, tak riadiace signály (`drive_signals`) su nastavené na čítanie a začiatok konverzie. Signál `counter2_int` sa inkrementuje do hodnoty binárnej 14 ("1110") a potom uloží hodnoty z obojsmerného vývodu do signálu `pressure_signal_int`. Ďalej sa vynulujú všetky pomocné premenné a signály. Premenná `sensor_int` sa nastaví na hodnotu jedna, čo začne nastavenie registra a následný prevod pre teplotný signál. Princíp pre prevod a nastavenie registra teplotného signálu je zhodný s prevodom a nastavením registra tlakového signálu (v riadiacom registri sa zmení nastavenie vstupu na ADC) viď prilohu A. so zdrojovým textom.

```
process (clk)                                                    --TXT1
  variable write_read_int: STD_LOGIC := '0';
  variable write_int: STD_LOGIC := '0';
  variable sensor_int: STD_LOGIC := '0';
------------------------------------------------------------------------
 begin
    if (clk'event and clk = '0' and off_switch='1') then
------------------------------------------------------------------------
-- zaciatok nastavenia registrov pre signal z tlakoveho senzoru
           if (write_read_int='0' and sensor_int='0') then
                    drive_signals <= "11010";
                    convert_signals <= "000000000000";
                    counter1_int <= counter1_int + 1;
                    if (counter1_int="10") then
                       write_read_int := '1';
                    end if;
           end if;
-- koniec nastavenia registrov pre signal z tlakoveho senzoru
------------------------------------------------------------------------
-- zaciatok prevodu signalu z tlakoveho senzoru
           if (write_read_int='1' and sensor_int='0') then
                    drive_signals <= "10100";
                    counter2_int <= counter2_int + 1;
                    if (counter2_int="1110") then
                       write_int := '1';
                       if (write_int='1') then
                                pressure_signal_int <= convert_signals;
                                sensor_int := '1';
                                write_read_int := '0';
                                counter1_int <= "00";
                                counter2_int <= "0000";
                                write_int := '0';
                       end if;
                    end if;
           end if;
```

7. 2 Tabuľka pre prevod tlakového signálu

Vzhľadom k tomu, že počítať logaritmus vo VHDL jazyku je prakticky nemožne a použitím rôznych metód aproximácie (Taylorova rada, algoritmus CORDIC) veľmi zložité, bola na prevod tlaku na nadmorskú výšku zvolená metóda prevodovej tabuľky. Zápis prevodovej tabuľky vo VHDL jazyku je jednoduchý (viď TXT2). Slovom „`case`“ sa definujú tabuľky. Vstupná premenná je uvedená hneď za príkazom „`case`“. Výraz: `when"000000000000"=>altitude_temperature_int<="0000111101001011";` sa dá preložiť asi takto. Keď (when) `pressure_signal_int` sa rovná „000000000000“ tak presuň hodnotu „0000111101001011“ do signálu `altitude_temperature_int`.

```
case pressure_signal_int is                                      --TXT2
when"000000000000"=>altitude_temperature_int<="0000111101001011";
when"000000000001"=>altitude_temperature_int<="0001101101011000";
when"000000000010"=>altitude_temperature_int<="0001101100111111";
when"000000000011"=>altitude_temperature_int<="0001101100100111";
when"000000000100"=>altitude_temperature_int<="0001101100001111";
when"000000000101"=>altitude_temperature_int<="0001101011110110";
when"000000000110"=>altitude_temperature_int<="0001101011011110";
when"000000000111"=>altitude_temperature_int<="0001101011000110";
when"000000001000"=>altitude_temperature_int<="0001101010101101";
when"000000001001"=>altitude_temperature_int<="0001101010010101";
when"000000001010"=>altitude_temperature_int<="0001101001111101";
```

```
when"000000001011"=>altitude_temperature_int<="10001101001100100";
when"000000001100"=>altitude_temperature_int<="10001101001001100";
when"000000001101"=>altitude_temperature_int<="10001101000110100";
when"000000001110"=>altitude_temperature_int<="10001101000011100";
when"000000001111"=>altitude_temperature_int<="10001101000000011";
when"000000010000"=>altitude_temperature_int<="10001100111101011";
when"000000010001"=>altitude_temperature_int<="10001100111010011";
when"000000010010"=>altitude_temperature_int<="10001100110111010";
when"000000010011"=>altitude_temperature_int<="10001100110100010";
when"000000010100"=>altitude_temperature_int<="10001100110001010";
--
--
-- dalsie hodnoty
--
--
when others=>null;
                end case;
```

Prevodová tabuľka obsahuje 2^12 = 4096 prvkov, čo sú vlastne všetky kombinácie jednotiek a núl, ktoré sa môžu na výstupe ADC objaviť. Prevod binárneho čísla ktoré sa objaví na výstupe ADC vychádza z navrhnutého modulu a z barometrickej rovnice (1). Binárne číslo na výstupe ADC vlastne reprezentuje hodnotu napätia na vstupe ADC. Pre výpočet nadmorskej výšky, potrebujeme poznať hodnotu napätia na vstupe ADC. Z hodnoty napätia na vstupe ADC zistime hodnotu napätia na výstupe tlakového senzoru a z hodnoty napätia na výstupe tlakového senzoru vieme vypočítať aktuálny atmosférický tlak. Z atmosférického tlaku, vieme podľa barometrickej rovnice vypočítať nekalibrovanú nadmorskú výšku. Pretože v barometrickej rovnici sa počíta s aktuálnou teplotou, nemohol byť celý výpočet nadmorskej výšky zahrnutý v prevodovej tabuľke. Vzhľadom k tomu, že pri vynásobení aktuálnej teploty s číslom z prevodovej tabuľky majú vplyv na rozlíšenie merania hodnoty za desatinou čiarkou tohto čísla, je toto číslo v prevodovej tabuľke vynásobené hodnotou 2^12. V ďalších výpočtoch sa toto číslo vynásobí teplotou a podeli číslom 2^12.

Ak vieme, že hodnota LSB na ADC je:

$$LSB = \frac{U_{ref}}{2^{12}} = \frac{U_{ref}}{4096} = \frac{4{,}5833\overline{3}V}{4096} = 0{,}001119V\,.$$

Tak vieme vypočítať hodnotu napätia na vstupe ADC:

$$Vin_{ADC} = U_{roz} = pressure\_signal\_\mathrm{int}.LSB\,.$$

Napätie na vstupe ADC, je vlastne napätie na výstupe rozdielového zosilňovača. Takže $Vin_{ADC}=U_{roz}$ a $U_{roz}= 5\,.\,(U_{out} - U_{R8})$. U_{out} je napätie na výstupe tlakového senzoru a U_{R8} je napätie upravene odporovým deličom R7, R8. Hodnota napätia U_{R8} je rovná U_{min}. U_{min} je najmenšie možné napätie, ktoré sa môže objaviť na tlakovom senzore vzhľadom k možnému atmosférickému tlaku. Napätie na výstupe senzoru je podľa rovnice (6):

$$U_{out} = \frac{U_{roz}}{5} + U_{R8}\,.$$

Upravením a dosadením do rovnice (5) dostávame:

$$p = \frac{\frac{U_{out}}{U_{ref}} + 0{,}04}{0{,}004} \quad [\mathrm{kPa}],$$

$$p = \frac{\frac{pressure\_signal\_\text{int}.LSB.}{A} + U_{R8}}{\frac{U_{ref}}{0{,}004}} + 0{,}04 \quad [\text{kPa}].$$

Upravením rovnice (1) a tiež dosadením do nej, môžme odvodiť vzťah pre výpočet výšky v závislosti na zmene tlaku:

$$h = -\ln\left(\frac{p}{p_0}\right).\frac{R.T}{g} = -\ln\left(\frac{p}{p_0}\right).T.29,2259 \quad [\text{m}].$$

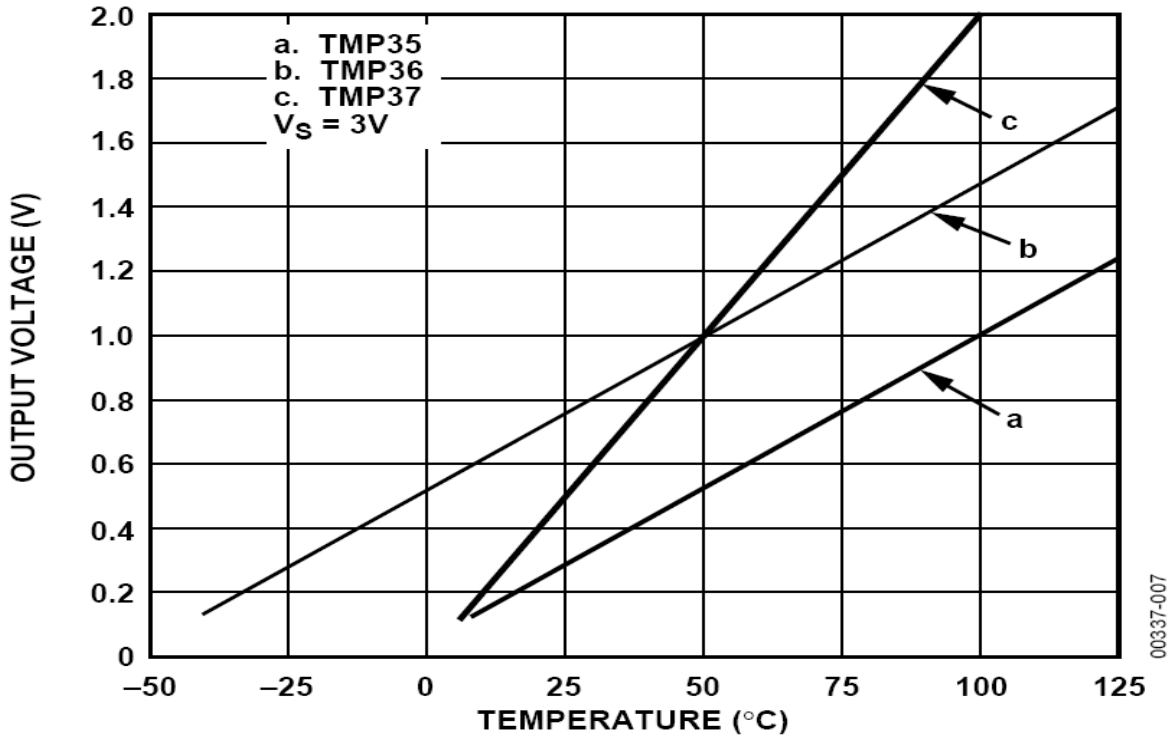
Z takto upravenej rovnice, môžme získať rovnicu pre výpočet prevodových prvkov v tabuľke:

$$altitude\_temperature\_\text{int} = -\ln\left(\frac{p}{p_0}\right).29{,}2259.2\text{^}12$$

Tlak p_0 volíme 110 kPa aby nám pri kalibrácii pri odčítaní kalibračnej výšky od nekalibrovanej výšky nevychádzali záporne hodnoty. Tabuľka s príkladmi výpočtov je v prilohe G. Samozrejme výpočty sú zaokrúhlene na celé čísla, čo spôsobuje odchýlku merania 1 meter. Spôsobená odchýlka je akceptovateľná a čiastočne potlačená kalibráciou.

7. 3 Prevod teplotného signálu

Na meranie teploty je použitý teplotný senzor TMP36 od firmy Analog Devices. Podrobné vlastnosti tohto senzoru nájdeme v literature [7]. Senzor meria v rozsahu -40°C až +125°C. Citlivosť senzoru je 10mV/°C a výstupne napätie pri 25°C je 750mV viď obr. 16. .



Obr. 16. zavislost vystupneho napatia na teplote. Viď lit[7]

Výstupné napätie na senzore nie je nijak upravované a je priamo privedené na vstup ADC. Výpočet teploty zo signálu reprezentujúceho teplotu je zapísaný priamo vo VHDL jazyku (viď TXT3). Výpočet teploty v kelvinovej stupnici sa vypočíta podľa vzorca, ktorý sa dá ľahko odvodiť z citlivosti senzora aj z obr.16.

Vzorec pre výpočet teploty:

$$T = Ut.0{,}1 + 223 \quad [°\mathrm{K}].$$

U_t je napätie na výstupe senzoru a vypočítame ho podľa vzťahu:

$$U_t = temperature\_signal\_\mathrm{int}.LSB \quad [\mathrm{mV}].$$

`Temperature_signal_int` je teplotný signál z výstupu ADC. Vzhľadom k tomu, že hodnota LSB je desatine číslo, je tato hodnota vynásobená číslom 2^12. Takže signál z analógovo digitálneho prevodníka v podobe binárneho reťazca je vynásobený číslom LSB.2^12=1,119.2^12=4583_{10}=1000111100111_2 a výsledok je uložený v signále `calculation1_int`. V ďalšom kroku je vytvorený akoby bitový posuv doprava o 12 bitov a výsledná hodnota (napätie na výstupe senzoru v mV) je uložená v signále `voltage_on_adc_int` (bitový posuv o 12 bitov reprezentuje delenie číslom 2^12). Potom je hodnota napätia vynásobená číslom 0,1.2^12=410_{10}=110011010 a výsledok je uložený v signále `calculation2_int`. Do signálu `calculation3_int` je uložený bitový reťazec reprezentujúci signál `calculation2_int` posunutý o 12 bitov doprava. Nakoniec je k signálu `calculation_3` pripočítane číslo 223_{10}=11011111_2 a výsledok je uložený do signálu `temperature_int`. Signál `temperature_int` je binárne číslo reprezentujúce aktuálnu teplotu. Tabuľka s príkladmi výpočtov je v prilohe H. Presnosť výpočtu je znova obmedzená zaokrúhľovaním nadol na celé čísla, pretože sa pri výpočtoch používa bitový posuv. Maximálna chýba, ktorá sa pri výpočtoch vyskytuje je 1°K, čo je pre aplikáciu altimeter akceptovateľné.

```
calculation1_int<=temperature_signal_int*"1000111100111          --TXT3
voltage_on_adc_int<=calculation1_int (24 downto 12);
calculation2_int<=voltage_on_adc_int*"110011010";
calculation3_int<=calculation2_int (21 downto 12);
temperature_int<=calculation3_int+"011011111";
```

7. 4 Výpočet nekalibrovanej výšky

V tomto bloku sa počíta neskalibrovaná nadmorská výška. Vypočítame ju vynásobením aktuálnej teploty, ktorú sme vypočítali v časti prevod teplotného signálu s číslom z prevodovej tabuľky, ktoré sme získali v časti tabuľka pre prevod tlakového signálu. Samozrejme nesmieme zabudnúť na bitový posuv o 12 bitov doprava, pretože hodnota čísla z tabuľky pre prevod tlakového signálu je vynásobená číslom 2^12. Zápis výpočtu je zobrazený v texte TXT4.

```
calculation4_int<=altitude_temperature_int*temperature_int;      --TXT4
altitude_notactual_int<= calculation4_int (26 downto 14);
```

V prvom riadku textu TXT4 je vynásobený prevodový prvok z tabuľky pre prevod tlakového signálu s aktuálnou teplotou. V druhom riadku je vykonaný akoby bitový posuv o 12 bitov (delenie 2^12). Ďalej je do signálu altitude_notactual_int uložená hodnota signálu, ktorá reprezentuje nekalibrovanú nadmorskú výšku v podobe binárneho čísla.

7. 5 Kalibrácia

Kalibrácia sa skladá z dvoch častí:

- nastavenie kalibračnej nadmorskej výšky
- výpočet rozdielu nekalibrovanej nadmorskej výšky a kalibračnej nadmorskej výšky

Princíp kalibrácie spočíva v nastavení kalibračnej výšky (presná nadmorská výška v ktorej sa modul v danom čase nachádza) a odčítaní tejto výšky od nekalibrovanej zmeranej výšky. Získaný rozdiel výšok sa potom odpočítava od zmeranej nekalibrovanej výšky a tým dostávame aktuálnu skalibrovanú nadmorskú výšku.

7. 5. 1 Nastavenie kalibračnej nadmorskej výšky

Nastavenie kalibračnej nadmorskej výšky sa bude obsluhovať tromi tlačidlami. Jedno tlačidlo bude aktivovať binárny čítač nahor a druhé tlačidlo aktivuje binárny čítač nadol. Tretie tlačidlo zobrazí aktuálnu hodnotu kalibračnej výšky na displeji. Pri stlačení všetkých troch tlačidiel naraz sa hodnota kalibračnej výšky prepíše hodnotou neskalibrovanej nadmorskej výšky. Zápis vo VHDL jazyku je ukázaný v texte TXT5.

```
process (clk_int)                                                    --TXT5
begin
  if (clk_int'event and clk_int='0') then
-- start up counter
     if (up_button = '1' and down_button = '0') then
        calibration_altitude_int <= calibration_altitude_int + 1;
        if (calibration_altitude_int > "0011111111111") then
           calibration_altitude_int <= "0000000000000";
        end if;
-- end up counter
-- start down counter
     elsif (up_button = '0' and down_button = '1') then
        calibration_altitude_int <= calibration_altitude_int - 1;
        if (calibration_altitude_int = "0000000000000") then
           calibration_altitude_int <= "0011111111111";
        end if;
-- end down counter
     elsif ( up_button='1' and down_button='1' and display_button='1') then
        calibration_altitude_int <= altitude_notactual_int;
     end if;
  end if;
end process;
```

Binárny čítač nahor počíta do hodnoty 2047_{10}=0011111111111_2 a potom začne počítať znova od nuly. Binárny čítač nadol počíta do hodnoty 0_{10}=0000000000000_2 a potom začne počítať od hodnoty 2047_{10}=0011111111111_2. Hodinový signál `clk_int` ma nízku frekvenciu, aby bolo možne pozorovať zmenu kalibračnej nadmorskej výšky na displeji.

7. 5. 2 Rozdiel nekalibrovanej a kalibračnej nadmorskej výšky

Pri nastavení kalibračnej výšky, je táto hodnota uložená v signále calibration_altitude, kde nemá na meranie nadmorskej výšky v aplikácii žiaden vplyv. V prípade, že sa prepne prepínač calibration_switch, tak sa kalibračná hodnota (calibration_altitude) odpočíta od nekalibrovanej výšky (altitude_notcalibration) a uloží sa do signálu differential_altitude a automaticky je táto hodnota započítavaná do algoritmu výpočtu aktuálnej kalibrovanej nadmorskej výšky. Jednoduchá ukážka zápisu tohto bloku vo VHDL jazyku je zobrazená v texte TXT6.

```
process (clk5m)                                                    --TXT6
begin
  if (clk5m'event and clk5m='0' and off_switch = '1') then
   if (calibration_switch = '1') then
      differential_altitude <= altitude_notactual - calibration_altitude;
   end if;
  end if;
end process;
```

7. 6 Výpočet skalibrovanej nadmorskej výšky

Táto časť programu počíta jednoduchý výpočet, kde sa od nekalibrovanej nadmorskej výšky odpočíta rozdiel nekalibrovanej a kalibračnej výšky, ktorý bol uložený do signálu `differential_altitude` pri kalibrácii a prepnutí prepínača `calibration_switch`. Zápis vo VHDL kóde je zobrazený v texte TXT7.

```
process (clk5m)                                                    --TXT7
begin
  if (clk5m'event and clk5m='0' and off_switch = '1') then
         altitude_actual <= altitude_notactual - differential_altitude;
     end if;
end process;
```

Spôsob kalibrácie je založený na podobnom princípe aký sa používa v niektorých typoch lietadiel. Meraná výška v lietadlách je vzťažená k jednej hodnote atmosférického tlaku, bez ohľadu na to, či sa atmosférický tlak mení, alebo nemení. Tým je dosiahnuté, že lietadlo neletí v presnej vo výške ako mu meria altimeter, ale je zaručene, že nepríde k zrážke s iným lietadlom, pretože rozdiel výšok medzi dvoma lietadlami bude presný, bez ohľadu na zmenu atmosférického tlaku.

7. 7 Výber zobrazovanej hodnoty na displeji

Táto časť programu vyberie z troch možnosti, ktoré program môže zobraziť na displeji. Výber závisí na stlačenej kombinácii tlačidiel. Zdrojový text je zobrazený v texte TXT8.

```
Begin                                                              TXT8
process (clk50m)
begin
 if (off_switch = '1' and clk50m = '0') then
    if (up_button ='0' and down_button ='0' and  display_button='0' and
temperature_button='0')then
           bin<=altitude_actual;
       elsif (up_button='0' and down_button='0'  and display_button='0' and
temperature_button='1')then
           bin<="000" & temperature;
       else bin<=calibration_altitude;
    end if;
 end if;
end process;
```

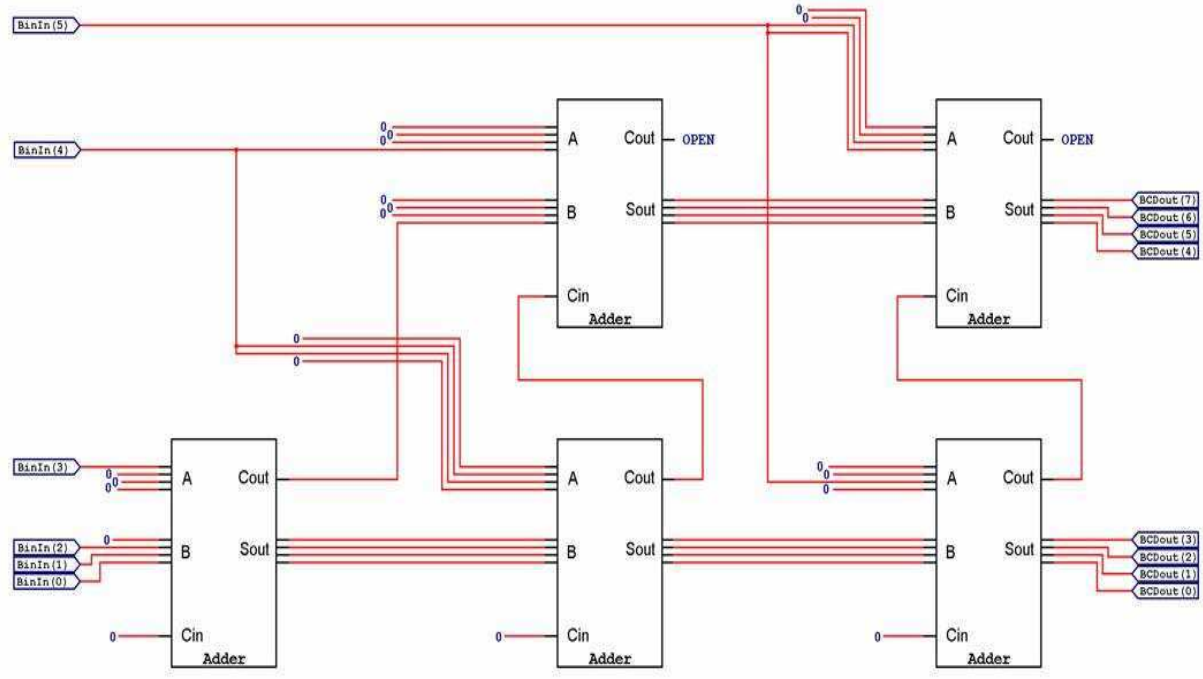
Pokiaľ nie je stlačené žiadne tlačidlo, tak zobrazuje aktuálnu nadmorskú výšku, ak je stlačene tlačidlo `temperature_button`, tak zobrazí teplotu a ak je stlačená nejaká iná kombinácia, tak zobrazí kalibračnú výšku.

7. 8 Prevodník binárneho čísla na BCD kód

Všetky počítane výsledky v programe sú v tvare bitového reťazca, ktorý reprezentuje nejaké číslo v binárnom tvare. Vzhľadom k tomu, že prevodník signálov na segmentový displej prevádza signál z BCD kódu na segmenty displej, bolo potrebne vložiť blok programu, ktorý prevedie binárny reťazec na BCD kód. Prevod funguje na princípe sčítačiek. Zápis sčítačky vo VHDL jazyku je zobrazený v texte TXT9.

```
LIBRARY ieee;                                                    --TXT9
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
ENTITY AddBCD IS
  PORT(a,b:  IN  std_logic_vector(3 DOWNTO 0);
       Sout: OUT std_logic_vector(3 DOWNTO 0);
       Cin:  IN  std_logic;
       Cout: OUT std_logic);
END AddBCD;

ARCHITECTURE AddArch OF AddBCD IS BEGIN
  BCDadd: PROCESS (a,b,Cin)
    VARIABLE SwCvar: std_logic_vector(4 DOWNTO 0); -- Sum with Carry
  BEGIN
    SwCvar := ('0'&a) + ('0'&b) +  ("0000"&Cin);
    IF (SwCvar > 9) THEN SwCvar := SwCvar + 6; END IF;
    Cout <= SwCvar(4);  Sout <= SwCvar(3 DOWNTO 0);
  END PROCESS BCDadd;
END AddArch;
```

Pre prevod 13 bitového signálu bolo potrebne použiť 28 sčítačiek. Poprepájanie signálov medzi sčítačkami je zapísane v zdrojovom texte programu v prílohe A. Princíp funkcie sčítačiek je zrejmý z obrázku obr. 17.. V prípade, že vstupný signál ma 5 bitov, tak sa v prvom stupni sčítačiek pripočítava 16. Ak ma vstupný signál 6 bitov, tak sa v druhom stupni sčítačiek pripočíta 32 (atd. sa pripočítava 64, 128, 256, 512....).



Obr. 17. Princip sčítania signálov pri prevode bin2BCD.

7. 9 Zobrazovanie na displej

Program v tomto bloku ma za úlohu previesť BCD signál na segmentový displej a zobraziť tak na displeji správnu hodnotu (nadmorskú výšku [m], teplotu [°K], kalibračnú výšku [m]). Na zobrazenie hodnoty na displeji je využitá zotrvačnosť ľudského oka. Program používa dva hodinové signáli. Signál s menšou frekvenciou slúži na aktivovanie a zobrazenie jednotlivých digitov. Zápis programu vo VHDL jazyku je v texte TXT10.

```
if clk50'event and clk50 = '1' then                                    --TXT10
   if khertz_en = '1' then
     cd <= cd + 1;
   end if;
   case cd(1 downto 0) is  -- curr je soucasne zobrazena cifra, digit
                           -- jejeji pozice na displeji
     when "00" =>curr <= bcdint(3 downto 0);digit <= "1110"; dp <= dp0;
     when "01" =>curr <= bcdint(7 downto 4);digit <= "1101"; dp <= dp1;
     when "10" =>curr <= bcdint(11 downto 8);digit <= "1011"; dp <= dp2;
     when others => curr <= bcdint(15 downto 12); digit <= "0111"; dp <=
dp3;
   end case ;
     case curr is
      when "0000" => seg <= "0000001" & dp; -- 0
      when "0001" => seg <= "1001111" & dp; -- 1
      when "0010" => seg <= "0010010" & dp; -- 2
      when "0011" => seg <= "0000110" & dp; -- 3
      when "0100" => seg <= "1001100" & dp; -- 4
      when "0101" => seg <= "0100100" & dp; -- 5
      when "0110" => seg <= "0100000" & dp; -- 6
      when "0111" => seg <= "0001111" & dp; -- 7
      when "1000" => seg <= "0000000" & dp; -- 8
      when others => seg <= "0000100" & dp; -- 9
     end case;
   end if;
  end process;
```

Proces je aktivovaný nábežnou hranou signálu `clk50`. Ak je signál `khertz` na úrovni logickej jednotky, tak sa signál `cd` inkrementuje o jednotku. Ďalej je vytvorená akási tabuľka, ktorá podľa aktuálnej hodnoty signálu `cd` nakopíruje do pomocných signálov `digit` a `curr` potrebné hodnoty (digit – určuje polohu na dipleji, curr – zobrazovaná cifra). V ďalšej tabuľke sa podľa cifry, ktorá je uložená v signále `curr` uloží do signálu `seg` binárny reťazec. Signály `digit` a `seg` sú pripojené na displej a rozsvecujú jednotlivé segmenty na displeji.

8 Záver

Zadaním tejto prace bolo zoznámenie sa s obvodmi CPLD, s vývojovým prostredím Xilinx ISE, s princípom a aplikáciou tlakových senzorov. Taktiež bolo požadované navrhnúť blokové zapojenie snímacieho systému - výškomer a jeho elektronickú časť vo forme modulu k vývojovej doske a algoritmus na prepočet zmeraného tlaku na nadmorskú výšku. Ďalej bolo požadované tento algoritmus zapísať v jazyku VHDL.

V prvej kapitole bola rozobratá štruktúra programovateľných obvodov CPLD a konkrétny obvod CPLD Coolrunner-II XC2C256, ktorý je použitý na vývojovej doske XC2-XL. Vzhľadom na použitú metódu prepočítania tlaku na nadmorskú výšku (metóda prevodovej tabuľky), bolo nutné použiť FPGA obvod obsahujúci pamäte. Z toho dôvodu je v práci stručne popísaný obvod FPGA Spartan – 3 XC3S200 a vývojová doska s týmto obvodom. Ďalej bol vysvetlený základný postup návrhu v komplexnom návrhovom systéme Xilinx ISE. V skratke bol popísaný jazyk VHDL jeho výhody a nevýhody, syntax, základná štruktúra, dátové typy a operátory používané v jazyku VHDL. Tiež boli spomenuté integrované senzory tlaku pre meranie rôznych druhov tlakov. Bol popísaný ich princíp a aplikácia.

Najdôležitejšou časťou tejto práce je samotný návrh elektronickej časti modulu k vývojovej doske a návrh algoritmu na výpočet nadmorskej výšky. Najprv bola navrhnutá bloková schéma systému a potom boli rozpísane podrobnosti o jednotlivých blokoch, tj. funkcia, postup návrhu a výpočet konkrétnych súčiastok. Výsledkom práce je navrhnutá elektronická časť modulu a algoritmu na výpočet nadmorskej výšky. Navrhnutý algoritmus je zapísaný v jazyku VHDL a je odsimulovaný vo vývojovom prostredí Xilinx ISE. Pôvodným cieľom bakalárskej práce, bolo použiť obvod CPLD Coolrunner-II XC2C256 a naprogramovať ho algoritmom zapísaným vo VHDL jazyku. Vzhľadom k výhodnejším vlastnostiam obvodu FPGA bolo v priebehu riešenia práce zhodnotené, že bude oveľa výhodnejšie použiť namiesto CPLD obvodu obvod FPGA. Táto zmena priniesla so sebou rôzne výhody, ako napríklad rozlíšenie merania 1 m, možnosť, jednoduchej a relatívne presnej kalibrácie, ako aj meranie a zobrazovanie teploty.

Zmena cieľového obvodu prácu nezjednodušila, skôr naopak, ale dovolila vytvoriť aplikáciu, ktorá je pre konštruktéra zaujímavá a v praxi viac využiteľná. Dôležitým aspektom je, že pôvodná myšlienka prace, oboznámiť sa s jazykom VHDL a vytvoriť aplikáciu na meranie výšky prostredníctvom programovateľných logických obvodov, zostala zachovaná.

9 Literatúra

[1] DUECK, K. R. Digital Design with Cpld Applications and VHDL. Thomson Delmar Learning, 2001. ISBN 9780766811614

[2] KOLOUCH, J. Programovatelné logické obvody - přednášky. Skriptum. Brno: FEKT VUT v Brně, 2002

[3] FREESCALE SEMICONDUCTOR, Inc. Austin, Texas. Pressure sensors. 2007. Firemní stránky. Dostupné na WWW: www.freescale.com.

[4] ANALOG DEVICES: *AD7933/AD7934 datasheet*

[5] XILINX: *XC2C256 CoolRunner-II CPLD datasheet*

[6] DIGILENT: Digilent XC2-XL system board reference manual. Dostupné na WWW: www.digilent.com

[7] ANALOG DEVICES: TMP36 *datasheet*

[8] FREESCALE SEMICONDUCTOR: *MPXH6250A datasheet*

[9] FREESCALE SEMICONDUCTOR: AN1646 Noice considerations for integrated pressure sensors, aplication note

[10] ANALOG DEVICES: TMP36FS *datasheet*

[11] PHILIPS: *KTY81 datasheet*

[12] Digitální výškoměr. [cit. 23.10.2007]. Dostupné na WWW: http://hw.cz/docs/vyskomer/vyskomer.html

[13] Barometrická rovnice. [cit. 10.10.2007]. Dostupné na WWW: http://www.volny.cz/jtomsa/barometr.htm

[14] Vývojové prostředí Xilinxu. [cit. 01.11.2007]. Dostupné na WWW: http://gw.asix.cz/zuban/skola-fpga/kap3_1.htm

[15] Xilinx ISE. [cit. 29.10.2007]. Dostupné na WWW: http://www.stud.fit.vutbr.cz/~xvasic11/cl.cpld/#ise

[16] Princip a struktura integrovaných senzorů tlaku freescale. [cit. 21.11.2007]. Dostupné na WWW: http://hw.cz/teorie-praxe/art1941-princip-struktura-integrovanych-senzoru-tlaku-freescale.html

[17] Senzory pro měření tlaku v integrovaném provedení. [cit. 22.11.2007]. Dostupné na WWW: http://automatizace.hw.cz/clanek/2005081401

[18] Wikipedie otevřená encyklopetdie – Barometrické měření výšky. [cit. 30.11.2007]. Dostupné na WWW: http://cs.wikipedia.org/wiki/Barometrick%C3%A9_m%C4%9B%C5%99en%C3%AD_v%C3%BD%C5%A1ky

[19] Opera4né zosilňovače. [cit. 02.12.2007]. Dostupné na WWW: http://alzat.szm.sk/Zosil/Jednosmr/oz/oz.htm

[20] Meření teploty – polovodičové odporové senzory teploty. [cit. 21.12.2007]. Dostupné na WWW: http://hw.cz/Teorie-a-praxe/Dokumentace/ART1141-Mereni-teploty---polovodicove-odporove-senzory-teploty.html

[21] Wikipédia slobodná enciklopédia – Programovatelný logický obvod. [cit. 23.10.2007]. Dostupné na WWW: http://sk.wikipedia.org/wiki/Programovate%C4%BEn%C3%BD_logick%C3%BD_obvod

[22] MUSIL, V. a kol.: Navrhování digitálních integrovaných obvodů. Jazyk VHDL.[Skriptum FEKT VUT v Brně.] Brno, 2000

[23] KOLOUCH, J.: Programovatelné logické obvody a návrh jejich aplikací v jazycích ABEL a VHDL – počítačové cvičení. [Skriptum FEKT VUT v Brně.] Brno, 2005

10 Zoznam použitých skratiek

A/D – Analog/Digital
ADC – Analog-to-Digital Convertor
AIM – Advanced Interconnect Matrix
ASIC - Application Specific Integrated Circuit
CAD – Computer-Aided Design
CMOS – Complementary metal-oxide semiconductor
CPLD – Complex Programmable Logic Device
EEPROM – Electrically-Erasable Programmable Read-Only Memory
FPGA – Field Programmable Gate Array
GAL - Generic Array Logic
HDL – Hardware description language
I/O – Input/Output
IEEE – Institute of Electrical and Electronics Engineers
ISE - Integrated Software Environment
JTAG – Joint Test Action Group
LED – Light Emitting Diode
OZ – Operačný Zosilňovač
PAL- Programmable Array Logic
PLA – Programmable Logic Array
PLD - Programmable Logic Device
PROM – user-Programable Read-Only Memory
VHDL - VHSIC Hardware Description Language
VHDL-AMS - VHDL Analog and Mixed-Signal Extensions
VHSIC - Very High Speed Integrated Circuit
XST - Xilinx Synthesis Technology

11 Zoznam príloh

A. Zdrojový text celého programu
B. Simulácia prevodníku bin2bcd
C. Simulácia výberu zobrazovanej hodnoty na displeji
D. Simulácia celého programu + vnútorné signály
E. Schéma zapojenia navrhnut0ho modulu
F. Bloková schéma algoritmu
G. Tbuľka výpočtov tlaku
H. Tabuľka výpočtov teploty

Prílohy

A. Zdrojový text celého programu

-- Konfigurácia ADC

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    07:43:47 06/06/2008
-- Design Name:
-- Module Name:    konfiguracia - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity konfiguracia is
Port ( clk5m : in  STD_LOGIC;
           off_switch : in  STD_LOGIC;
           drive_signals : out  STD_LOGIC_VECTOR (4 downto 0);
           convert_signals : inout  STD_LOGIC_VECTOR (11 downto 0);
           pressure_signal : out  STD_LOGIC_VECTOR (11 downto 0);
           temperature_signal : out  STD_LOGIC_VECTOR (11 downto 0));
end konfiguracia;

architecture Behavioral of konfiguracia is
signal counter1_int: STD_LOGIC_VECTOR (1 downto 0) := "00";
signal counter2_int: STD_LOGIC_VECTOR (3 downto 0) := "0000";
signal pressure_signal_int: STD_LOGIC_VECTOR (11 downto 0) := "000000000000";
signal temperature_signal_int: STD_LOGIC_VECTOR (11 downto 0) := "000000000000";

begin

process (clk5m)
  variable write_read_int: STD_LOGIC := '0';
  variable write_int: STD_LOGIC := '0';
  variable sensor_int: STD_LOGIC := '0';

begin
    if (clk5m'event and clk5m = '0' and off_switch='1') then
---------------------------- zaciatok nastavenia registrov pre signal z tlakoveho
senzoru
        if (write_read_int='0' and sensor_int='0') then
                  drive_signals <= "11010";
                      convert_signals <= "000110111110";
```

```
                    counter1_int <= counter1_int + 1;
                    if (counter1_int="10") then
                       write_read_int := '1';
                    end if;
                end if;
---------------------------- koniec nastavenia registrov pre signal z tlakoveho
senzoru
---------------------------- zaciatok prevodu signalu z tlakoveho senzoru

            if (write_read_int='1' and sensor_int='0') then
                   drive_signals <= "10100";
                    counter2_int <= counter2_int + 1;
                    if (counter2_int="1110") then
                       write_int := '1';
                       if (write_int='1') then
                              pressure_signal<= convert_signals;
                                  sensor_int := '1';
                                  write_read_int := '0';
                                  counter1_int <= "00";
                                  counter2_int <= "0000";
                                  write_int := '0';
                           end if;
                     end if;
        end if;
---------------------------- zaciatok nastavenia registrov pre signal z teplotneho
senzoru
        if (write_read_int='0' and sensor_int='1') then
                   drive_signals <= "11010";
                    convert_signals <= "000000100000";
                    counter1_int <= counter1_int + 1;
                    if (counter1_int="10") then
                       write_read_int := '1';
                    end if;
                end if;
---------------------------- koniec nastavenia registrov pre signal z teplotneho
senzoru
---------------------------- zaciatok prevodu signalu z teplotneho senzoru

            if (write_read_int='1' and sensor_int='1') then
                   drive_signals <= "10100";
                    counter2_int <= counter2_int + 1;
                    if (counter2_int="1110") then
                       write_int := '1';
                       if (write_int='1') then
                              temperature_signal <= convert_signals;
                                  sensor_int := '0';
                                  write_read_int := '0';
                                  counter1_int <= "00";
                                  counter2_int <= "0000";
                                  write_int := '0';
                           end if;
                     end if;
        end if;
      end if;
end process;
end Behavioral;
```

--Tabuľka pre prevod tlakového signálu

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:13:20 06/04/2008
-- Design Name:
-- Module Name:    table - Behavioral
-- Project Name:
```

```
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity table is
    Port ( pressure_signal : in  STD_LOGIC_VECTOR (11 downto 0);
           altitude_temperature : out  STD_LOGIC_VECTOR (16 downto 0);
           clk5m : in  STD_LOGIC ;
                     off_switch : in STD_LOGIC);
end table;

architecture Behavioral of table is

begin
process (clk5m)

begin
    if (clk5m'event and clk5m = '0' and off_switch='1') then
case pressure_signal is
when"000000000000"=>altitude_temperature<="00010001100101100";
when"000000000001"=>altitude_temperature<="10001101101011000";
when"000000000010"=>altitude_temperature<="10001101100111111";
when"000000000011"=>altitude_temperature<="10001101100100111";
when"000000000100"=>altitude_temperature<="10001101100001111";
when"000000000101"=>altitude_temperature<="10001101011110110";
when"000000000110"=>altitude_temperature<="10001101011011110";
when"000000000111"=>altitude_temperature<="10001101011000110";
when"000000001000"=>altitude_temperature<="10001101010101101";
----
----vlozit dalsie hodnoty----
----
when others=>null;
                   end case;

end if;
end process;
end Behavioral;
```

-- Prevod teplotného signálu

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:21:26 06/04/2008
-- Design Name:
-- Module Name:    temperature - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
```

```
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity temperature is
    Port ( temperature_signal : in  STD_LOGIC_VECTOR (11 downto 0);
           temperature : out  STD_LOGIC_VECTOR (9 downto 0);
           clk5m : in  STD_LOGIC;
           off_switch : in  STD_LOGIC);
end temperature;

architecture Behavioral of temperature is
signal calculation1_int : STD_LOGIC_VECTOR (24 downto 0);
signal calculation2_int : STD_LOGIC_VECTOR (21 downto 0);
signal calculation3_int : STD_LOGIC_VECTOR (9 downto 0);
signal voltage_on_adc_int : STD_LOGIC_VECTOR (12 downto 0);
begin
process (clk5m)

begin
    if (clk5m'event and clk5m = '0' and off_switch='1') then
        calculation1_int<=temperature_signal*"1000111100111";
        voltage_on_adc_int<=calculation1_int (24 downto 12);
        calculation2_int<=voltage_on_adc_int*"110011010";
        calculation3_int<=calculation2_int (21 downto 12);
        temperature<=calculation3_int+"011011111";
        -- vzorec je odvodeny z datasheet pre talkovy senzor
    -- temperature_int={[(temperature_signal_int*LSB*2^12)/2*12]*0,1*2^12}+223
        end if;
end process;
end Behavioral;
```

-- Výpočet nekalibrovanej výšky

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:30:13 06/04/2008
-- Design Name:
-- Module Name:    notcalibration_altitude - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
```

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

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

entity notcalibration_altitude is
    Port ( altitude_temperature : in  STD_LOGIC_VECTOR (16 downto 0);
           temperature : in  STD_LOGIC_VECTOR (9 downto 0);
           altitude_notactual : out  STD_LOGIC_VECTOR (12 downto 0);
           clk5m : in  STD_LOGIC;
           off_switch : in  STD_LOGIC);
end notcalibration_altitude;

architecture Behavioral of notcalibration_altitude is
signal calculation4_int : STD_LOGIC_VECTOR (26 downto 0);
begin
process (clk5m)

begin
    if (clk5m'event and clk5m = '0' and off_switch='1') then
        calculation4_int<=altitude_temperature*temperature;               --
    altitude_notactual<= calculation4_int (26 downto 14);
    end if;
end process;
end Behavioral;
```

-- Nastavenie kalibračnej nadmorskej výšky

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    05:12:17 06/05/2008
-- Design Name:
-- Module Name:    nastavenie_kal - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity nastavenie_kal is
    Port ( up_button : in  STD_LOGIC;
           down_button : in  STD_LOGIC;
           clk1 : in  STD_LOGIC;
```

```
           on_switch : in  STD_LOGIC;
           calibration_altitude : out  STD_LOGIC_VECTOR (12 downto 0);
           display_button : in  STD_LOGIC;
           altitude_notactual : in  STD_LOGIC_VECTOR (12 downto 0));
end nastavenie_kal;

architecture Behavioral of nastavenie_kal is
signal calibration_altitude_int :  STD_LOGIC_VECTOR (12 downto 0):="0000110010000";
begin
process (clk1)
begin
  if (clk1'event and clk1='0') then
-- start up counter
     if (up_button = '1' and down_button = '0') then
        calibration_altitude_int <= calibration_altitude_int + 1;
        if (calibration_altitude_int > "0011111111111") then
           calibration_altitude_int <= "0000000000000";
        end if;
-- end up counter
-- start down counter
     elsif (up_button = '0' and down_button = '1') then
        calibration_altitude_int <= calibration_altitude_int - 1;
        if (calibration_altitude_int = "0000000000000") then
           calibration_altitude_int <= "0011111111111";
        end if;
-- end down counter
     elsif ( up_button = '1' and down_button = '1' and display_button = '1') then
        calibration_altitude_int <= altitude_notactual;
     end if;
  end if;

end process;
calibration_altitude <= calibration_altitude_int;
end Behavioral;
```

-- Rozdiel nekalibrovanej a kalibračnej nadmorskej výšky

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    05:47:11 06/05/2008
-- Design Name:
-- Module Name:    switch_calibration - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity switch_calibration is
```

```
    Port ( calibration_switch : in  STD_LOGIC;
           calibration_altitude : in  STD_LOGIC_VECTOR (12 downto 0);
                     altitude_notactual : in  STD_LOGIC_VECTOR (12 downto 0);
           differential_altitude : out  STD_LOGIC_VECTOR (12 downto
0):="0000000000000";
           off_switch : in  STD_LOGIC;
           clk5m : in  STD_LOGIC);
end switch_calibration;

architecture Behavioral of switch_calibration is

begin
process (clk5m)
begin
  if (clk5m'event and clk5m='0' and off_switch = '1') then
   if (calibration_switch = '1') then
      differential_altitude <= altitude_notactual - calibration_altitude;
   end if;
  end if;
end process;

end Behavioral;
```

-- Výpočet skalibrovanej nadmorskej výšky

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    06:00:34 06/05/2008
-- Design Name:
-- Module Name:    actual - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity actual is
    Port ( clk5m : in  STD_LOGIC;
           differential_altitude : in  STD_LOGIC_VECTOR (12 downto 0);
           altitude_notactual : in  STD_LOGIC_VECTOR (12 downto 0);
           altitude_actual : out  STD_LOGIC_VECTOR (12 downto 0);
                     off_switch : in  STD_LOGIC );
end actual;

architecture Behavioral of actual is

begin
process (clk5m)
begin
```

```
  if (clk5m'event and clk5m='0' and off_switch = '1') then

      altitude_actual <= altitude_notactual - differential_altitude;

  end if;
end process;

end Behavioral;
```

-- Výber zobrazovanej hodnoty na displeji

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    07:31:20 06/06/2008
-- Design Name:
-- Module Name:    distributer - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity distributer is
    Port ( up_button : in  STD_LOGIC;
           down_button : in  STD_LOGIC;
           temperature_button : in  STD_LOGIC;
           display_button : in  STD_LOGIC;
           off_switch : in  STD_LOGIC;
           calibration_altitude : in  STD_LOGIC_VECTOR (12 downto 0);
           temperature : in  STD_LOGIC_VECTOR (9 downto 0);
           altitude_actual : in  STD_LOGIC_VECTOR (12 downto 0);
           bin : out  STD_LOGIC_VECTOR (12 downto 0);
           clk50m : in  STD_LOGIC);
end distributer;

architecture Behavioral of distributer is

begin
process (clk50m)
begin
 if (off_switch = '1' and clk50m = '0') then
    if (up_button ='0' and down_button ='0' and  display_button='0' and
temperature_button='0')then
           bin<=altitude_actual;
       elsif (up_button='0' and down_button='0'  and display_button='0' and
temperature_button='1')then
           bin<="000" & temperature;
        else bin<=calibration_altitude;
    end if;
```

```
 end if;
end process;

end Behavioral;
```

-- Prevodník binárneho čísla na BCD kód

-- ADDBCD

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
ENTITY AddBCD IS
  PORT(a,b:  IN  std_logic_vector(3 DOWNTO 0);
       Sout: OUT std_logic_vector(3 DOWNTO 0);
       Cin:  IN  std_logic;
       Cout: OUT std_logic);
END AddBCD;

ARCHITECTURE AddArch OF AddBCD IS BEGIN
  BCDadd: PROCESS (a,b,Cin)
    VARIABLE SwCvar: std_logic_vector(4 DOWNTO 0); -- Sum with Carry
  BEGIN
    SwCvar := ('0'&a) + ('0'&b) +  ("0000"&Cin);
    IF (SwCvar > 9) THEN SwCvar := SwCvar + 6; END IF;
    Cout <= SwCvar(4);  Sout <= SwCvar(3 DOWNTO 0);
  END PROCESS BCDadd;
END AddArch;
```

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    07:55:27 06/06/2008
-- Design Name:
-- Module Name:    bin2bcd - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity bin2bcd is
    Port ( binin : in  STD_LOGIC_VECTOR (12 downto 0);
           BCDout : out  STD_LOGIC_VECTOR (15 downto 0);
           off_switch : in  STD_LOGIC);
end bin2bcd;

architecture Behavioral of bin2bcd is
```

```
COMPONENT AddBCD
	PORT(
		a : IN std_logic_vector(3 downto 0);
		b : IN std_logic_vector(3 downto 0);
		Cin : IN std_logic;
		Sout : OUT std_logic_vector(3 downto 0);
		Cout : OUT std_logic
		);
	END COMPONENT;

signal A0, B0 : std_logic_vector (3 DOWNTO 0):="0000";
signal A1H, B1H, A1L, B1L : std_logic_vector (3 DOWNTO 0):="0000";
signal A2H, B2H, A2L, B2L : std_logic_vector (3 DOWNTO 0):="0000";
signal A3H, B3H, A3L, B3L : std_logic_vector (3 DOWNTO 0):="0000";
signal A4H, B4H, A4M, B4M, A4L, B4L : std_logic_vector (3 DOWNTO 0):="0000";
signal A5H, B5H, A5M, B5M, A5L, B5L : std_logic_vector (3 DOWNTO 0):="0000";
signal A6H, B6H, A6M, B6M, A6L, B6L : std_logic_vector (3 DOWNTO 0):="0000";
signal A7H, B7H, A7M1, B7M1, A7M2, B7M2, A7L, B7L : std_logic_vector (3 DOWNTO
0):="0000";
signal A8H, B8H, A8M1, B8M1, A8M2, B8M2, A8L, B8L : std_logic_vector (3 DOWNTO
0):="0000";
signal A9H, B9H, A9M1, B9M1, A9M2, B9M2, A9L, B9L : std_logic_vector (3 DOWNTO
0):="0000";

signal BCD1, BCD2, BCD3, BCD4 : std_logic_vector (3 DOWNTO 0):="0000";

signal C1H, C2H, C3H : std_logic :='0';
signal C4H, C5H, C6H : std_logic :='0';
signal C4M, C5M, C6M1, C6M2 : std_logic :='0';
signal C7H, C8H, C9H : std_logic :='0';
signal C7M1, C8M1, C9M1 : std_logic :='0';
signal C7M2, C8M2, C9M2 : std_logic :='0';
signal C3, C5, C10, C13,C20, C24, C28 : std_logic :='0';


begin
ADD1: AddBCD PORT MAP(a =>A0 ,b => B0, Sout =>B1L ,Cin =>'0',Cout =>B1H (0));
ADD2: AddBCD PORT MAP(a =>A1L ,b => B1L,Sout =>B2L ,Cin => '0' ,Cout => C1H    );
ADD3: AddBCD PORT MAP(a =>A1H ,b =>B1H ,Sout =>B2H ,Cin => C1H,Cout =>C3);
ADD4: AddBCD PORT MAP(a =>A2L ,b =>B2L ,Sout =>B3L ,Cin =>'0' ,Cout =>C2H      );
ADD5: AddBCD PORT MAP(a =>A2H ,b =>B2H ,Sout =>B3H ,Cin =>C2H ,Cout =>C5);
ADD6: AddBCD PORT MAP(a =>A3L ,b =>B3L ,Sout =>B4L ,Cin =>'0' ,Cout =>C3H      );
ADD7: AddBCD PORT MAP(a =>A3H ,b =>B3H ,Sout =>B4M ,Cin =>C3H ,Cout =>B4H (0)  );
ADD8: AddBCD PORT MAP(a =>A4L ,b =>B4L ,Sout =>B5L ,Cin =>'0' ,Cout =>C4M      );
ADD9: AddBCD PORT MAP(a =>A4M ,b =>B4M ,Sout =>B5M ,Cin =>C4M ,Cout =>C4H      );
ADD10: AddBCD PORT MAP(a =>A4H ,b =>B4H ,Sout =>B5H ,Cin =>C4H ,Cout =>C10     );
ADD11: AddBCD PORT MAP(a =>A5L ,b =>B5L ,Sout =>B6L ,Cin =>'0' ,Cout =>C5M     );
ADD12: AddBCD PORT MAP(a =>A5M ,b =>B5M ,Sout =>B6M ,Cin =>C5M ,Cout =>C5H     );
ADD13: AddBCD PORT MAP(a =>A5H ,b =>B5H ,Sout =>B6H ,Cin =>C5H ,Cout =>C13     );
ADD14: AddBCD PORT MAP(a =>A6L ,b =>B6L ,Sout =>B7L ,Cin =>'0' ,Cout =>C6M1 );
ADD15: AddBCD PORT MAP(a =>A6M ,b =>B6M ,Sout =>B7M1 ,Cin =>C6M1 ,Cout =>C6M2 );
ADD16: AddBCD PORT MAP(a =>A6H ,b =>B6H ,Sout =>B7M2 ,Cin =>C6M2 ,Cout =>B7H (0));
ADD17: AddBCD PORT MAP(a =>A7L ,b =>B7L ,Sout =>B8L ,Cin =>'0' ,Cout =>C7M1 );
ADD18: AddBCD PORT MAP(a =>A7M1 ,b =>B7M1 ,Sout =>B8M1 ,Cin =>C7M1 ,Cout =>C7M2 );
ADD19: AddBCD PORT MAP(a =>A7M2 ,b =>B7M2 ,Sout =>B8M2 ,Cin =>C7M2 ,Cout =>C7H );
ADD20: AddBCD PORT MAP(a =>A7H ,b =>B7H ,Sout =>B8H ,Cin =>C7H ,Cout =>C20     );
ADD21: AddBCD PORT MAP(a =>A8L ,b =>B8L ,Sout =>B9L ,Cin =>'0' ,Cout =>C8M1 );
ADD22: AddBCD PORT MAP(a =>A8M1 ,b =>B8M1 ,Sout =>B9M1 ,Cin =>C8M1 ,Cout =>C8M2 );
ADD23: AddBCD PORT MAP(a =>A8M2 ,b =>B8M2 ,Sout =>B9M2 ,Cin =>C8M2 ,Cout =>C8H );
ADD24: AddBCD PORT MAP(a =>A8H ,b =>B8H ,Sout =>B9H ,Cin =>C8H ,Cout =>C24     );
ADD25: AddBCD PORT MAP(a =>A9L ,b =>B9L ,Sout =>BCD1 ,Cin =>'0' ,Cout =>C9M1 );
ADD26: AddBCD PORT MAP(a =>A9M1 ,b =>B9M1 ,Sout =>BCD2 ,Cin =>C9M1 ,Cout =>C9M2 );
ADD27: AddBCD PORT MAP(a =>A9M2 ,b =>B9M2 ,Sout =>BCD3 ,Cin =>C9M2 ,Cout =>C9H );
ADD28: AddBCD PORT MAP(a =>A9H ,b =>B9H ,Sout =>BCD4 ,Cin =>C9H ,Cout =>C28 );

A0<= binin(3)&"000";
B0<='0'& binin (2 downto 0);
A1H<= "000" & binin (4);
```

```
BCDout<= BCD4 & BCD3 & BCD2 & BCD1;

A1L<='0' & binin (4) & binin (4) & '0';
A2H<="00" & binin (5) & binin (5);
A2L<="00" & binin (5) & '0';
A3H<='0' & binin (6)&binin (6)&'0';
A3L<='0' & binin (6) & "00";



A4H<= "000" & binin (7);
A4M<= "00" & binin (7) & '0';
A4L<= binin (7) & "000";
A5H<= "00" & binin (8) & '0';
A5M<= '0' & binin (8) & '0' & binin (8);
A5L<= '0' & binin (8) & binin (8) & '0';
A6H<= '0' & binin (9) & '0' & binin (9);
A6M<= "000" & binin (9);
A6L<= "00" & binin (9) & '0';
A7H<= "000" & binin (10);
A7M2<= "0000";
A7M1<= "00" & binin (10) & '0';
A7L<= '0' & binin (10) & "00";
A8H<= "00" & binin (11) & '0';
A8M2<= "0000";
A8M1<= '0' & binin (11) & "00";
A8L<= binin (11) & "000";
A9H<= '0' & binin (12) & "00";
A9M2<= "0000";
A9M1<= binin (12) & "00" & binin (12);
A9L<= '0' & binin (12) & binin (12) & '0';


end Behavioral;
```

-- Zobrazovanie na displej

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    08:19:22 06/06/2008
-- Design Name:
-- Module Name:    display - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity display is
```

```
    Port ( clk50m : in  STD_LOGIC;
           khertz_en: in std_logic;
                  bcdint:    in std_logic_vector(15 downto 0);

           digit:     out std_logic_vector(3 downto 0);
           seg:       out std_logic_vector(7 downto 0));
end display;

architecture Behavioral of display is

signal cd:   std_logic_vector(1 downto 0);
signal curr: std_logic_vector(3 downto 0);
signal dp:   std_logic;
signal dp3,dp2,dp1,dp0: std_logic;
begin

--dp3 <= '1'; -- zadna des.tecka nesviti pro dp = '1'
--dp2 <= '1';
--dp1 <= '1';
--dp0 <= '1';
 process (clk50m) begin
dp3 <= '1'; -- zadna des.tecka nesviti pro dp = '1'
dp2 <= '1';
dp1 <= '1';
dp0 <= '1';
--  if rst = '1' then -- reset neni potrebny
--   seg <= (others => '1');
--   digit <= (others => '1');
--   cd <= (others => '0');
--   curr <= (others => '0');
  if clk50m'event and clk50m = '1' then
   if khertz_en = '1' then
     cd <= cd + 1;
   end if;
   case cd(1 downto 0) is -- curr je soucasne zobrazena cifra, digit je jeji pozice
na displeji
     when "00" =>   curr <= bcdint(3 downto 0);   digit <= "1110"; dp <= dp0;
     when "01" =>   curr <= bcdint(7 downto 4);   digit <= "1101"; dp <= dp1;
     when "10" =>   curr <= bcdint(11 downto 8);  digit <= "1011"; dp <= dp2;
     when others => curr <= bcdint(15 downto 12); digit <= "0111"; dp <= dp3;
   end case ;
     case curr is
      when "0000" => seg <= "0000001" & dp; -- 0
      when "0001" => seg <= "1001111" & dp; -- 1
      when "0010" => seg <= "0010010" & dp; -- 2
      when "0011" => seg <= "0000110" & dp; -- 3
      when "0100" => seg <= "1001100" & dp; -- 4
      when "0101" => seg <= "0100100" & dp; -- 5
      when "0110" => seg <= "0100000" & dp; -- 6
      when "0111" => seg <= "0001111" & dp; -- 7
      when "1000" => seg <= "0000000" & dp; -- 8
      when others => seg <= "0000100" & dp; -- 9
     end case;
   end if;
  end process;

end Behavioral;
```

-- Časovanie displeja

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    08:07:25 06/06/2008
-- Design Name:
-- Module Name:    timing_display - Behavioral
```

```
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity timing_display is
    Port ( clk50m : in  STD_LOGIC;
           khzen : out  STD_LOGIC);
end timing_display;

architecture Behavioral of timing_display is

SIGNAL mhertz_count,khertz_count: std_logic_vector(9 downto 0):="0000000000";
signal mhertz_en,khertz_en: std_logic;

begin

process (clk50m) begin
  if clk50m'event and clk50m = '1' then
   mhertz_count <= mhertz_count + 1 ;
   if mhertz_count = "110010" then
     mhertz_en <= '1' ;
     mhertz_count <= (others => '0') ;
   else
     mhertz_en <= '0' ;
   end if ;
  end if ;
 end process ;

-- generates a 1 kHz signal from a 1Mhz signal - khertz_en
 process (clk50m) begin
  if clk50m'event and clk50m = '1' then
   if mhertz_en = '1' then
     khertz_count <= khertz_count + 1 ;
     if khertz_count = "1111101000" then
       khertz_en <= '1' ;
       khertz_count <= (others => '0') ;
     else
       khertz_en <= '0' ;
     end if ;
   else
     khertz_en <= '0' ;
   end if ;
  end if ;
 end process ;

 KHzEn <= khertz_en;


end Behavioral;
```

-- Delička hodinového signálu

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    11:36:14 06/04/2008
-- Design Name:
-- Module Name:    delicka5m - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity delicka5m is
    Port ( clk50 : in  STD_LOGIC;
           clk5m : out  STD_LOGIC;
           off_switch : in  STD_LOGIC);

end delicka5m;

architecture Behavioral of delicka5m is
signal count_int: std_logic_vector (2 downto 0):="000";
begin
process (clk50)
variable help_int: STD_LOGIC := '0';
begin
    if (clk50'event and clk50 = '0' and off_switch='1') then
          if (help_int='0')then
           clk5m <= '1';
        count_int <= count_int+1;
              if count_int = "100" then
               help_int := '1';
                clk5m <= '0';
           count_int <= "000";
              end if;
      elsif ( help_int ='1')then
             clk5m <= '0';
              count_int <= count_int+1;
              if count_int = "100" then
               help_int := '0';
                    clk5m <= '1';
           count_int <= "000";
          end if;
      end if;

        end if;
end process;
end Behavioral;
```

-- Top modul

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    10:56:27 06/04/2008
-- Design Name:
-- Module Name:    altimeter - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

entity altimeter is
    Port ( clk50m : in  STD_LOGIC;
           up_button : in  STD_LOGIC;
           down_button : in  STD_LOGIC;
           display_button : in  STD_LOGIC;
           temperature_button : in  STD_LOGIC;
           off_switch : in  STD_LOGIC;
           calibration_switch : in  STD_LOGIC;
           digit: out std_logic_vector(3 downto 0);
                     seg: out std_logic_vector(7 downto 0);
           drive_signals : out  STD_LOGIC_VECTOR (4 downto 0);
           convert_signals : inout  STD_LOGIC_VECTOR (11 downto 0));
end altimeter;

architecture Behavioral of altimeter is
COMPONENT konfiguracia
      PORT(
            clk5m : IN std_logic;
            off_switch : IN std_logic;
            convert_signals : INOUT std_logic_vector(11 downto 0);
            drive_signals : OUT std_logic_vector(4 downto 0);
            pressure_signal : OUT std_logic_vector(11 downto 0);
            temperature_signal : OUT std_logic_vector(11 downto 0)
            );
      END COMPONENT;
signal clk5m_int :STD_LOGIC;
signal pressure_signal_int :STD_LOGIC_VECTOR (11 downto 0);
signal temperature_signal_int :STD_LOGIC_VECTOR (11 downto 0);
COMPONENT delicka5m
      PORT(
            clk50 : IN std_logic;
            off_switch : IN std_logic;
            clk5m : OUT std_logic
            );
      END COMPONENT;

COMPONENT table
```

```
	PORT(
		pressure_signal : IN std_logic_vector(11 downto 0);
		clk5m : IN std_logic;
		off_switch : IN std_logic;
		altitude_temperature : OUT std_logic_vector(16 downto 0)
		);
	END COMPONENT;
signal altitude_temperature_int :STD_LOGIC_VECTOR (16 downto 0);

COMPONENT temperature
	PORT(
		temperature_signal : IN std_logic_vector(11 downto 0);
		clk5m : IN std_logic;
		off_switch : IN std_logic;
		temperature : OUT std_logic_vector(9 downto 0)
		);
	END COMPONENT;
signal temperature_int :STD_LOGIC_VECTOR (9 downto 0);

COMPONENT notcalibration_altitude
	PORT(
		altitude_temperature : IN std_logic_vector(16 downto 0);
		temperature : IN std_logic_vector(9 downto 0);
		clk5m : IN std_logic;
		off_switch : IN std_logic;
		altitude_notactual : OUT std_logic_vector(12 downto 0)
		);
	END COMPONENT;
signal altitude_notactual_int :STD_LOGIC_VECTOR (12 downto 0);

COMPONENT delicka1
	PORT(
		up_button : IN std_logic;
		down_button : IN std_logic;
		off_switch : IN std_logic;
		clk5m : IN std_logic;
		clk1 : OUT std_logic
		);
	END COMPONENT;
signal clk1_int :STD_LOGIC;

COMPONENT nastavenie_kal
	PORT(
		up_button : IN std_logic;
		down_button : IN std_logic;
		clk1 : IN std_logic;
		on_switch : IN std_logic;
		display_button : IN std_logic;
		altitude_notactual : IN std_logic_vector(12 downto 0);
		calibration_altitude : OUT std_logic_vector(12 downto 0)
		);
	END COMPONENT;
signal calibration_altitude_int :STD_LOGIC_VECTOR (12 downto 0);

COMPONENT switch_calibration
	PORT(
		calibration_switch : IN std_logic;
		calibration_altitude : IN std_logic_vector(12 downto 0);
		altitude_notactual : IN std_logic_vector(12 downto 0);
		off_switch : IN std_logic;
		clk5m : IN std_logic;
		differential_altitude : OUT std_logic_vector(12 downto 0)
		);
	END COMPONENT;
signal differential_altitude_int : std_logic_vector(12 downto 0);

COMPONENT actual
	PORT(
```

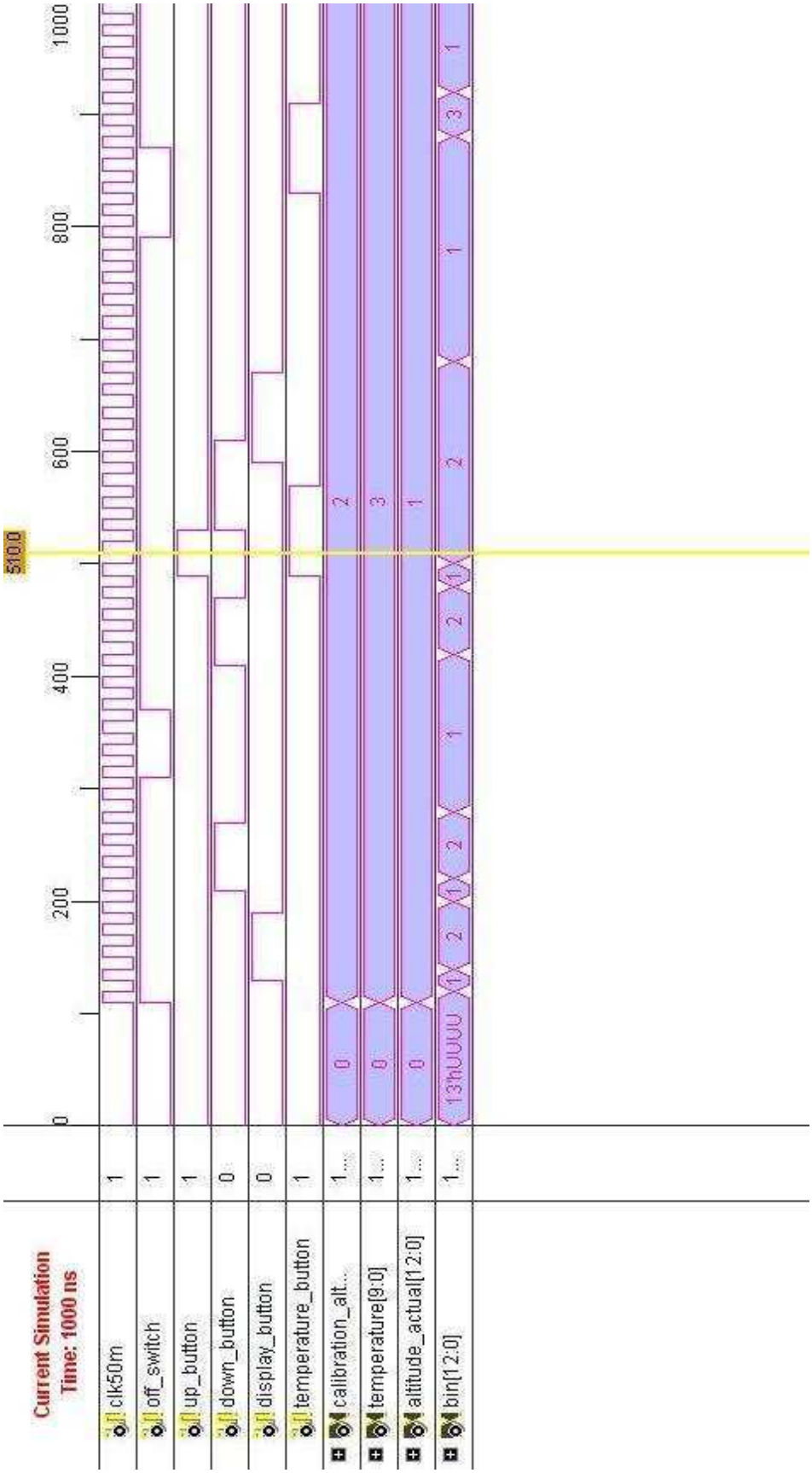
```
            clk5m : IN std_logic;
            differential_altitude : IN std_logic_vector(12 downto 0);
            altitude_notactual : IN std_logic_vector(12 downto 0);
            off_switch : IN std_logic;
            altitude_actual : OUT std_logic_vector(12 downto 0)
            );
      END COMPONENT;
signal altitude_actual_int : std_logic_vector(12 downto 0);

COMPONENT distributer
      PORT(
            up_button : IN std_logic;
            down_button : IN std_logic;
            temperature_button : IN std_logic;
            display_button : IN std_logic;
            off_switch : IN std_logic;
            calibration_altitude : IN std_logic_vector(12 downto 0);
            temperature : IN std_logic_vector(9 downto 0);
            altitude_actual : IN std_logic_vector(12 downto 0);
            clk50m : IN std_logic;
            bin : OUT std_logic_vector(12 downto 0)
            );
      END COMPONENT;
signal bin_int : std_logic_vector(12 downto 0);

COMPONENT bin2bcd
      PORT(
            binin : IN std_logic_vector(12 downto 0);
            off_switch : IN std_logic;
            BCDout : OUT std_logic_vector(15 downto 0)
            );
      END COMPONENT;
signal BCDout_int : std_logic_vector(15 downto 0);

COMPONENT timing_display
      PORT(
            clk50m : IN std_logic;
            khzen : OUT std_logic
            );
      END COMPONENT;
signal khzen_int : std_logic;

COMPONENT display
      PORT(
            clk50m : IN std_logic;
            khertz_en : IN std_logic;
            bcdint : IN std_logic_vector(15 downto 0);
            digit : OUT std_logic_vector(3 downto 0);
            seg : OUT std_logic_vector(7 downto 0)
            );
      END COMPONENT;

begin
Inst_konfiguracia: konfiguracia PORT MAP(
            clk5m =>clk5m_int ,
            off_switch =>off_switch ,
            drive_signals =>drive_signals ,
            convert_signals =>convert_signals ,
            pressure_signal =>pressure_signal_int ,
            temperature_signal => temperature_signal_int
      );

Inst_delicka5m: delicka5m PORT MAP(
            clk50 => clk50m ,
            clk5m => clk5m_int,
            off_switch => off_switch
      );
```

```
Inst_table: table PORT MAP(
             pressure_signal => pressure_signal_int ,
             altitude_temperature =>altitude_temperature_int ,
             clk5m => clk5m_int ,
             off_switch => off_switch
      );

Inst_temperature: temperature PORT MAP(
             temperature_signal => temperature_signal_int,
             temperature => temperature_int,
             clk5m => clk5m_int,
             off_switch => off_switch
      );

Inst_notcalibration_altitude: notcalibration_altitude PORT MAP(
             altitude_temperature => altitude_temperature_int,
             temperature => temperature_int,
             altitude_notactual => altitude_notactual_int,
             clk5m => clk5m_int ,
             off_switch => off_switch
      );

Inst_delicka1: delicka1 PORT MAP(
             up_button => up_button,
             down_button => down_button,
             off_switch => off_switch,
             clk5m => clk5m_int,
             clk1 => clk1_int
      );


Inst_nastavenie_kal: nastavenie_kal PORT MAP(
             up_button => up_button,
             down_button => down_button ,
             clk1 => clk1_int,
             on_switch => off_switch,
             calibration_altitude => calibration_altitude_int,
             display_button => display_button,
             altitude_notactual => altitude_notactual_int
      );

Inst_switch_calibration: switch_calibration PORT MAP(
             calibration_switch => calibration_switch,
             calibration_altitude => calibration_altitude_int,
             altitude_notactual => altitude_notactual_int,
             differential_altitude => differential_altitude_int,
             off_switch => off_switch,
             clk5m => clk5m_int
      );

Inst_actual: actual PORT MAP(
             clk5m => clk5m_int,
             differential_altitude => differential_altitude_int,
             altitude_notactual => altitude_notactual_int,
             altitude_actual => altitude_actual_int,
             off_switch => off_switch
      );

Inst_distributer: distributer PORT MAP(
             up_button =>up_button ,
             down_button =>down_button ,
             temperature_button => temperature_button,
             display_button => display_button,
             off_switch => off_switch,
             calibration_altitude => calibration_altitude_int,
             temperature => temperature_int,
             altitude_actual => altitude_actual_int,
             bin => bin_int,
```

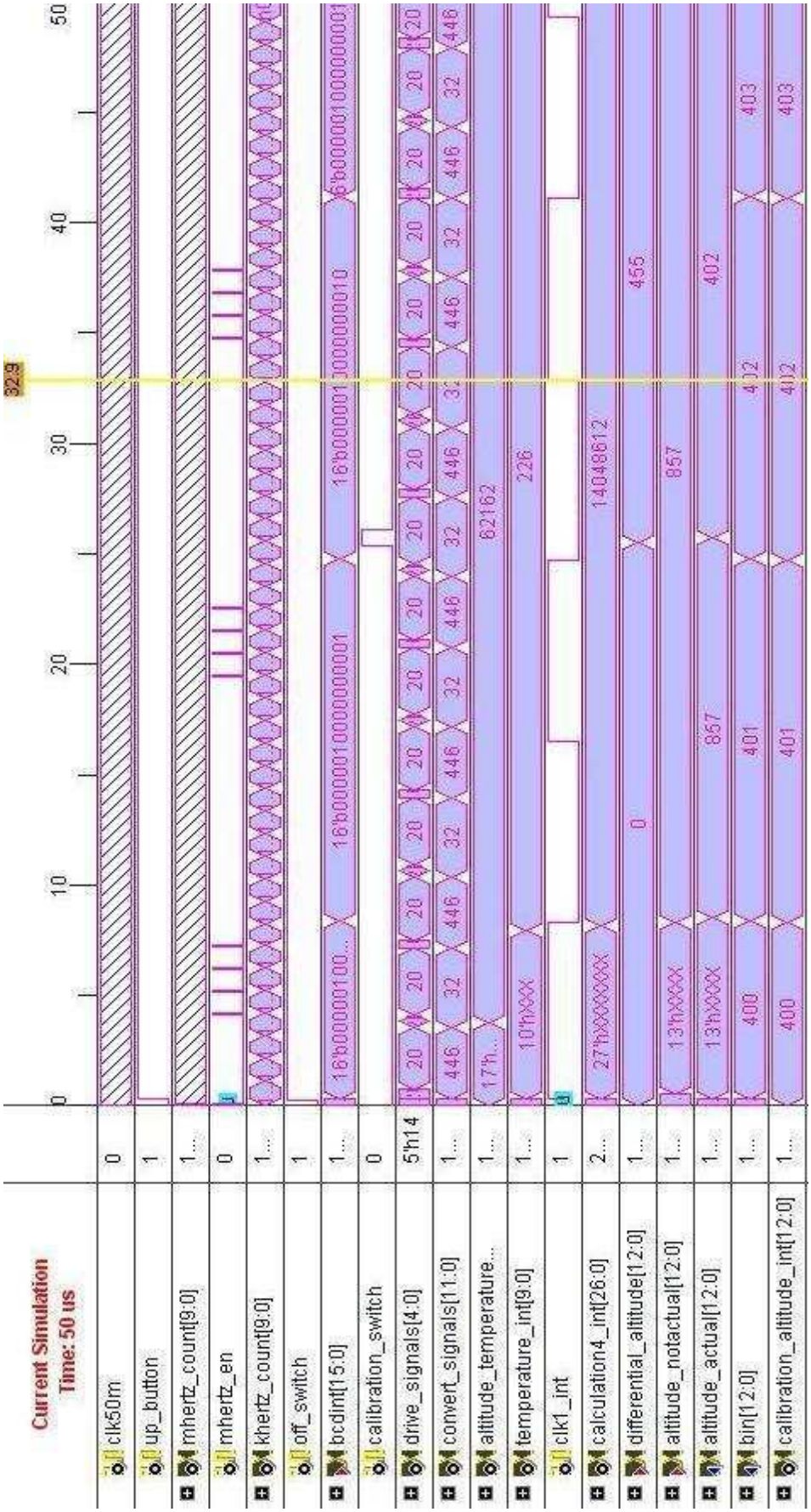
```
            clk50m => clk50m
      );

Inst_bin2bcd: bin2bcd PORT MAP(
            binin =>bin_int ,
            BCDout => BCDout_int,
            off_switch => off_switch
      );

Inst_timing_display: timing_display PORT MAP(
            clk50m => clk50m ,
            khzen => khzen_int
      );

Inst_display: display PORT MAP(
            clk50m => clk50m,
            khertz_en => khzen_int,
            bcdint => BCDout_int,
            digit => digit,
            seg => seg
      );

end Behavioral;
```

B. Simulácia prevodníku bin2bcd

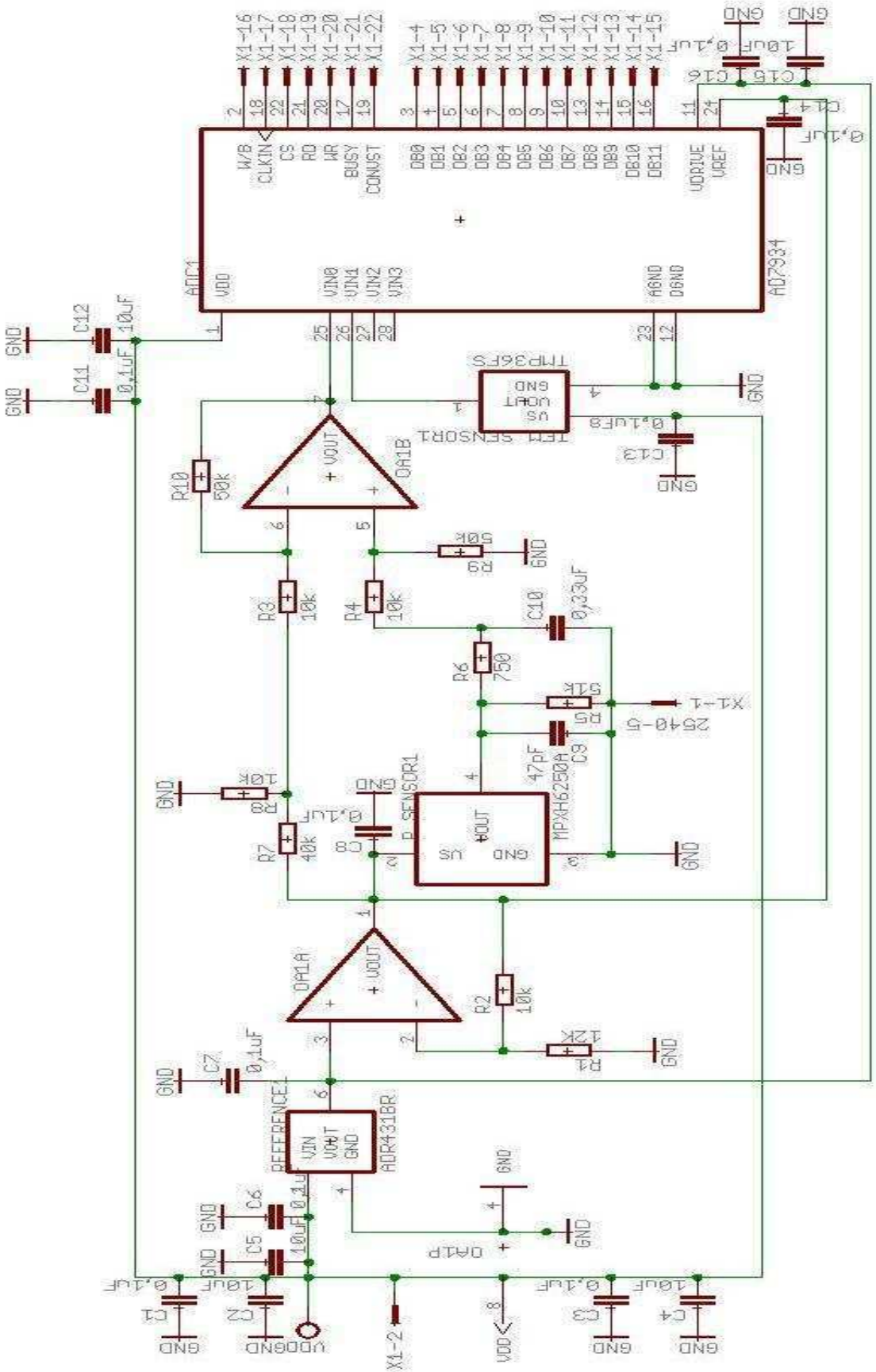
C. Simulácia výberu zobrazovanej hodnoty na displeji



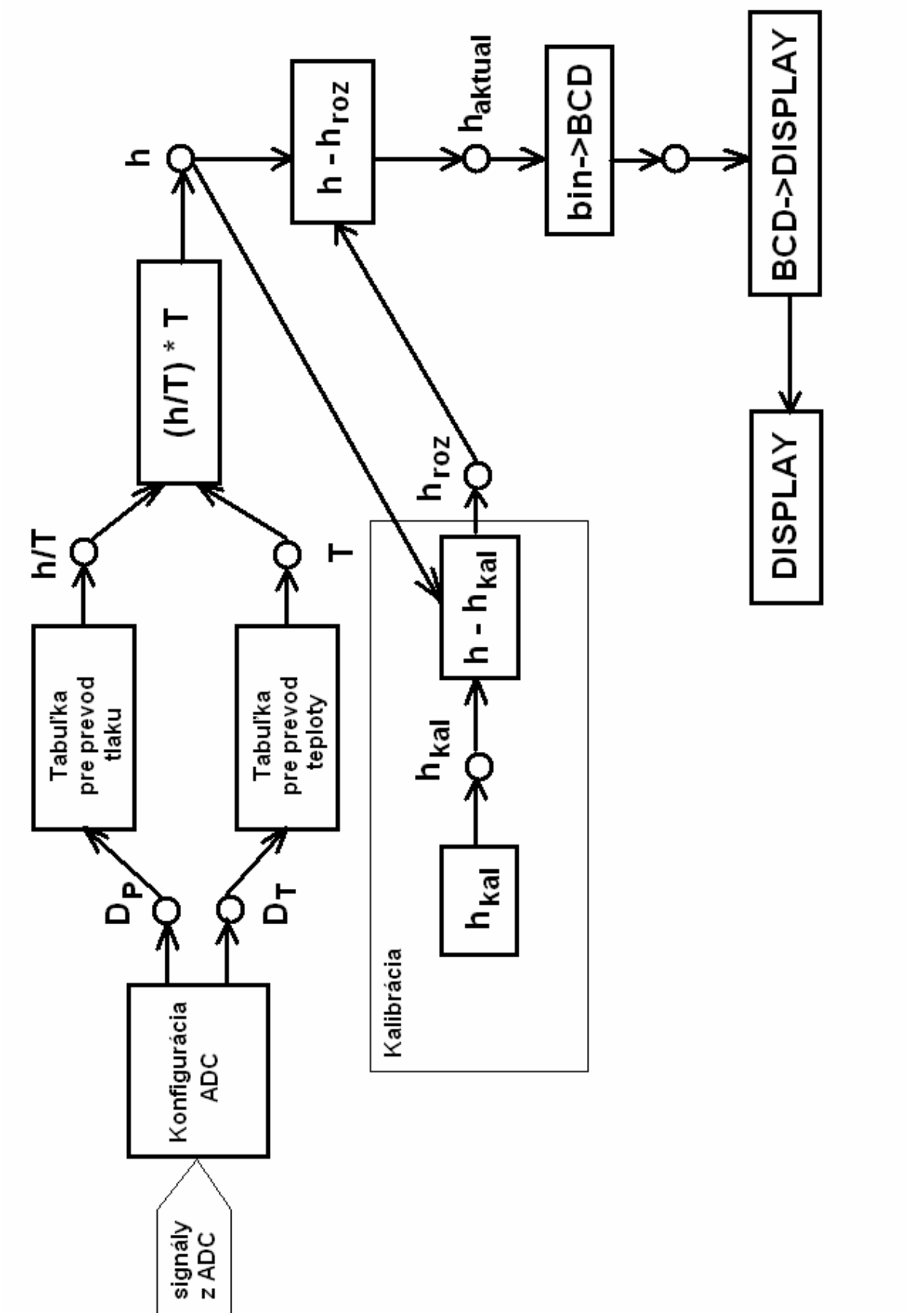
D. Simulácia celého programu



E. Schema zapojenia navrhnutého modulu



F. Bloková schéma algoritmu



G Tbuľka výpočtov tlaku

DB	P	h/T	h	h/T*2^12	zaokr_h/T*2^12*273	zaokr_h
1	60,01220727	17,70891884	4834,535	72535,7316	19802328	4835
2	60,02441455	17,70297452	4832,912	72511,3836	19795503	4833
3	60,03662182	17,6970314	4831,29	72487,0406	19788951	4831
4	60,04882909	17,69108949	4829,667	72462,7025	19782399	4830
5	60,06103636	17,68514878	4828,046	72438,3694	19775574	4828
6	60,07324364	17,67920929	4826,424	72414,0412	19769022	4826
7	60,08545091	17,673271	4824,803	72389,718	19762470	4825
8	60,09765818	17,66733391	4823,182	72365,3997	19755645	4823
9	60,10986545	17,66139804	4821,562	72341,0864	19749093	4822
10	60,12207273	17,65546337	4819,941	72316,7779	19742541	4820
11	60,13428	17,6495299	4818,322	72292,4745	19735716	4818
12	60,14648727	17,64359764	4816,702	72268,1759	19729164	4817
13	60,15869455	17,63766658	4815,083	72243,8823	19722612	4815
14	60,17090182	17,63173672	4813,464	72219,5936	19716060	4813
15	60,18310909	17,62580807	4811,846	72195,3099	19709235	4812
16	60,19531636	17,61988062	4810,227	72171,031	19702683	4810
17	60,20752364	17,61395437	4808,61	72146,7571	19696131	4809
18	60,21973091	17,60802933	4806,992	72122,4881	19689306	4807
19	60,23193818	17,60210548	4805,375	72098,2241	19682754	4805
20	60,24414545	17,59618284	4803,758	72073,9649	19676202	4804
21	60,25635273	17,59026139	4802,141	72049,7107	19669650	4802
22	60,26856	17,58434115	4800,525	72025,4613	19662825	4801
23	60,28076727	17,5784221	4798,909	72001,2169	19656273	4799
24	60,29297455	17,57250425	4797,294	71976,9774	19649721	4797
25	60,30518182	17,5665876	4795,678	71952,7428	19643169	4796
26	60,31738909	17,56067215	4794,063	71928,5131	19636617	4794
27	60,32959636	17,55475789	4792,449	71904,2883	19629792	4792
28	60,34180364	17,54884484	4790,835	71880,0684	19623240	4791
29	60,35401091	17,54293297	4789,221	71855,8535	19616688	4789
30	60,36621818	17,53702231	4787,607	71831,6434	19610136	4788
31	60,37842545	17,53111284	4785,994	71807,4382	19603311	4786
32	60,39063273	17,52520456	4784,381	71783,2379	19596759	4784
33	60,40284	17,51929747	4782,768	71759,0425	19590207	4783
34	60,41504727	17,51339159	4781,156	71734,8519	19583655	4781
35	60,42725455	17,50748689	4779,544	71710,6663	19577103	4780
36	60,43946182	17,50158339	4777,932	71686,4855	19570278	4778
37	60,45166909	17,49568108	4776,321	71662,3097	19563726	4776
38	60,46387636	17,48977996	4774,71	71638,1387	19557174	4775

G Tbuľka výpočtov tlaku

Vout(mV)_vyp	DB_dec	vypocet1	napatie	napatie_zaokr	vypocet2	teplota	teplota_zaokr	vystup K
199,1780599	178	815774	199,163574	199	81590	19,919	19	242
200,2970378	179	820357	200,282471	200	82000	20,02	20	243
201,4160156	180	824940	201,401367	201	82410	20,12	20	243
202,5349935	181	829523	202,520264	202	82820	20,22	20	243
203,6539714	182	834106	203,63916	203	83230	20,32	20	243
204,7729492	183	838689	204,758057	204	83640	20,42	20	243
205,8919271	184	843272	205,876953	205	84050	20,52	20	243
207,0109049	185	847855	206,99585	206	84460	20,62	20	243
208,1298828	186	852438	208,114746	208	85280	20,82	20	243
209,2488607	187	857021	209,233643	209	85690	20,92	20	243
210,3678385	188	861604	210,352539	210	86100	21,021	21	244
211,4868164	189	866187	211,471436	211	86510	21,121	21	244
212,6057943	190	870770	212,590332	212	86920	21,221	21	244
213,7247721	191	875353	213,709229	213	87330	21,321	21	244
214,84375	192	879936	214,828125	214	87740	21,421	21	244
215,9627279	193	884519	215,947021	215	88150	21,521	21	244
217,0817057	194	889102	217,065918	217	88970	21,721	21	244
218,2006836	195	893685	218,184814	218	89380	21,821	21	244
219,3196615	196	898268	219,303711	219	89790	21,921	21	244
220,4386393	197	902851	220,422607	220	90200	22,021	22	245
221,5576172	198	907434	221,541504	221	90610	22,122	22	245
222,6765951	199	912017	222,6604	222	91020	22,222	22	245
223,7955729	200	916600	223,779297	223	91430	22,322	22	245
224,9145508	201	921183	224,898193	224	91840	22,422	22	245
226,0335286	202	925766	226,01709	226	92660	22,622	22	245
227,1525065	203	930349	227,135986	227	93070	22,722	22	245
228,2714844	204	934932	228,254883	228	93480	22,822	22	245
229,3904622	205	939515	229,373779	229	93890	22,922	22	245
230,5094401	206	944098	230,492676	230	94300	23,022	23	246
231,628418	207	948681	231,611572	231	94710	23,123	23	246
232,7473958	208	953264	232,730469	232	95120	23,223	23	246
233,8663737	209	957847	233,849365	233	95530	23,323	23	246
234,9853516	210	962430	234,968262	234	95940	23,423	23	246
236,1043294	211	967013	236,087158	236	96760	23,623	23	246
237,2233073	212	971596	237,206055	237	97170	23,723	23	246
238,3422852	213	976179	238,324951	238	97580	23,823	23	246
239,461263	214	980762	239,443848	239	97990	23,923	23	246