ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

KATEDRA TELEKOMUNIKAČNÍ TECHNIKY

Monitor teploty a vlhkosti vzduchu s výstupem na displej realizovaný pomocí přípravku DE10-Lite a jazyka VHDL

Bakalářská Práce

Studijní program: Elektronika a Komunikace

Vedoucí práce: Ing. Pavel Lafata, Ph.D.

Filip Žourek

Praha 2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Žourek** Jméno: **Filip** Osobní číslo: **499198**

Fakulta/ústav: **Fakulta elektrotechnická**

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

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Monitor teploty a vlhkosti vzduchu s výstupem na displej realizovaný pomocí přípravku DE10-Lite a jazyka VHDL

Název bakalářské práce anglicky:

Air, Temperature and Humidity Monitor with Display Output Using DE10-Lite Kit in VHDL

Pokyny pro vypracování:

Seznamte se s přípravkem DE10-Lite (FPGA MAX10) a jeho obsluhou pomocí jazyka VHDL. K přípravku připojte alespoň 2 dostupné digitální senzory teploty a vlhkosti vzduchu, např. DS18B20, DHT22, DHT11 aj. dostupné. Využijte rovněž ovládací prvky přípravku – přepínače, tlačítka, segmentový displej. Dále navrhněte a realizujte připojení znakového displeje k přípravku pro zobrazování naměřených hodnot. Vytvořte VHDL kódy a knihovny pro ovládání čidel a senzorů a vyčítání naměřených veličin. Využijte segmentový displej jako pomocný způsob zobrazení naměřených hodnot. Hlavní zobrazení realizujte pomocí samostatného znakového displeje. Využijte také přepínače a tlačítka na přípravku pro přepínání zobrazených hodnot a výběr čidla. K přípravku připojte jednoduchý piezo měnič a využijte jej jako jednoduchý alarm pro případ, že dojde k překročení hodnoty měřené veličiny mimo nastavený interval (např. teploty apod.).

Seznam doporučené literatury:

[1] Lafata, P. - Hampl, P. - Pravda, M.: Digitální technika. 1. vyd. Praha: Česká technika - nakladatelství ČVUT, 2011. 164 s. ISBN 978-80-01-04914-3.
[2] Pinker, J. - Poupa, M.: Číslicové systémy a jazyk VHDL. Praha : BEN - technická literatura, 2006. 349 s. ISBN 80-7300-198-5.
[3] Ashender, P., J.: The VHDL Cookbook [online]. Dostupné z: https://tams-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf.
[4] Terasic: DE10-Lite User Manual [online]. Dostupné z: https://www.intel.com/content/dam/www/programmable/us/en/portal/dsn/42/doc-us-dsnbk-42-2912030810549-de10-lite-user-manual.pdf

Jméno a pracoviště vedoucí(ho) bakalářské práce:

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

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

Datum zadání bakalářské práce: **27.01.2023** Termín odevzdání bakalářské práce: **26.05.2023**

Platnost zadání bakalářské práce: **22.09.2024**

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

III. PŘEVZETÍ ZADÁNÍ

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

Datum převzetí zadání	Podpis studenta

Prohlášení

Prohlašuji, že jsem zadanou bakalářskou práci zpracoval sám s přispěním vedoucího práce a konzultanta a používal jsem pouze literaturu v práci uvedenou. Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé bakalářské práce nebo její části se souhlasem katedry.

V Praze dne 26. května 2023 ……..

podpis autora práce

Poděkování

Chtěl bych poděkovat svému vedoucímu práce Ing. Pavlu Lafatovi, Ph.D. za odborné vedení práce, rady a vypůjčení potřebného vybavení a periferii potřebných pro splnění práce. Také bych rád poděkoval své rodině a přátelům za podporu.

Abstrakt

Tato bakalářská práce se zabývá návrhem a realizací monitoru teploty a vlhkosti v jazyce VHDL s využitím přípravku DE10-Lite. K přípravku jsou připojeny senzory teploty DHT11 a DS18B20 a integrovaný obvod pro udržování reálného času DS1302. Změřené hodnoty jsou zobrazeny na znakovém LCD displeji a sedmi segmentovém displeji. Naměřené hodnoty přípravek posílá po sériové sběrnici do počítače, kde jsou tyto hodnoty ukládány a graficky zobrazeny pomocí navržené aplikace v jazyce Python. Na přípravku lze nastavit rozsah hodnot pro jednotlivé měřené veličiny. Pokud se měřená veličina nachází mimo rozsah, spustí se zvuková signalizace.

Klíčová slova: VHDL, FPGA, DHT11, DS18B20, DS1302, LCD displej, SPI, I2C, 1-Wire, UART, Python, DE10-Lite

Abstract

This bachelor thesis deals with the design and implementation of temperature and humidity monitor in VHDL with DE10-Lite kit. Kit is connected to DHT11 and DS18B20 temperature sensors and to DS1302 real time clock integrated circuit. Measurements are displayed on a LCD character display and on a seven segment display. Simultaneously the measurements are sent to a computer via the serial port where the data is logged and plotted by a custom application written in Python. User can setup a range of allowed values for each of the measturements. If any measurement is outside of the allowed range, a buzzer will notify the user.

Keywords: VHDL, FPGA, DHT11, DS18B20, DS1302, LCD display, SPI, I2C, 1-Wire, UART, Python, DE10-Lite

Obsah

1. Úvod

Tato práce se zabývá návrhem monitoru teploty a vlhkosti v jazyce VHDL. Práce je realizována na přípravku DE10-Lite. Přípravek komunikuje se senzory DHT11, DS18B20 a obvodem reálného času DS1302. Naměřené hodnoty jsou zobrazovány na displejích a odesílány do počítače. V počítači jsou naměřená data ukládána a graficky zpracována vlastní aplikací vytvořenou v jazyce Python.

V teoretické části práce je uveden stručný úvod do jazyka VHDL, do hradlových (FPGA) polí a je zde popsáno co je přípravek DE10-Lite. Dále je zde popsaný princip komunikačních protokolů (SPI, I^2C, UART, 1-Wire, AOSONG Single Wire) potřebných pro komunikaci s periferiemi použitých v této práci. Dále je zde vysvětleno, jak se pracuje s digitálními senzory teploty DHT11 a DS18B20 a jak z nich získat naměřené hodnoty. Na závěr teoretické části je popsáno, jak se pracuje s obvodem reálného času DS1302, displeji a jak lze odesílat naměřená data do počítače.

V praktické části práce je uvedena samotná realizace práce. Na začátku je uvedené blokové schéma vyhotovené práce. Dále jsou zde popsány principy a algoritmy jednotlivých dílčích bloků a jejich konkrétní realizace v jazyce VHDL. Dále je zde nastíněná realizace PC aplikace v jazyce Python. Ke konci je uveden popis výsledné práce a jak s jí uživatel může ovládat.

2. Teoretická část

2.1 Jazyk VHDL

Jazyk VHSIC Hardware Description Language (zkráceně VHDL) řadíme do skupiny tzv. popisovacích jazyků. Na rozdíl od programovacích jazyků, které generují program pro procesory, popisovací jazyky popisují chování reálného obvodu [1]. Kód napsaný v jazyce VHDL lze následně syntetizovat syntetizátorem, který automaticky vygeneruje reálný obvod. Vygenerovaný obvod lze například implementovat do FPGA (hradlových) polí.

2.2 Hradlová pole

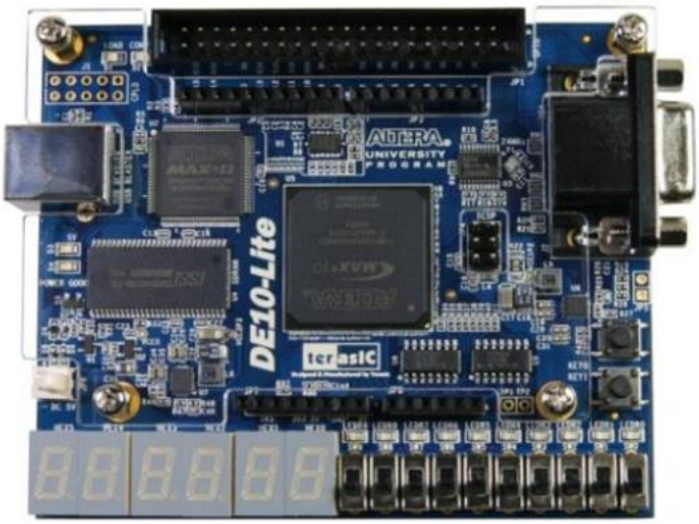
Programovatelná hradlová pole FPGA (field programmable gate array) jsou integrované obvody, které lze naprogramovat pomocí popisovacích jazyků (VHDL, Verilog). Pole se skládají z jednotlivých bloků, které se propojují dle syntetizovaného HDL kódu. Architektura FPGA pole se liší dle výrobce, ale ve většině polí lze najít následující bloky [1]:

- I/O bloky – bloky pro připojení vstupních a výstupních periferií
- LAB (logic array block) – logické bloky, které provádí logické operace
- PSM (programmable switch matrix) – programovatelná matice propojů
- RAM (random access memory) – volatilní paměť
- Násobičky – bloky určené pro aritmetické operace
- Bloky hodinových signálů – obsahují fázové závěsy
- ADC a DAC (analog digital/digital analog convereter) – převodníky analogového signálu na digitální nebo digitálního na analogový
- Flash paměť – nevolatilní paměť pro ukládání konfigurace a HDL kódu

Oproti procesorům, které jsou limitovány instrukční sadou, lze FPGA pole využít například k optimalizaci výpočtů, díky možnosti vytvoření hardwaru tak, aby co nejefektivněji výpočet prováděl. Další velká oblast využití FPGA polí je návrh prototypů zákaznických integrovaných polí (ASIC – aplication specific integrated circuit). Návrh obvodu probíhá pomocí HDL kódu implementovaného na FPGA pole. Z naprogramovaných FPGA polí lze vytvořit ASIC čip. Díky tomu dojde k ušetření nákladů, protože není potřeba vytvářet několik ASIC prototypů.

2.3 Přípravek DE10-Lite

Vývojový přípravek DE10-Lite od firmy Terasic je hradlové pole založené na rodině hradlových polí MAX10 od firmy Intel [2]. Součástí přípravku je několik periferií, jako je 10 přepínačů, 10 diod, 2 tlačítka, VGA výstup, 6 sedmi segmentových displejů, akcelerometr a 40 GPIO pinů. Do přípravku je také vestavěný programátor USB Blaster [2].

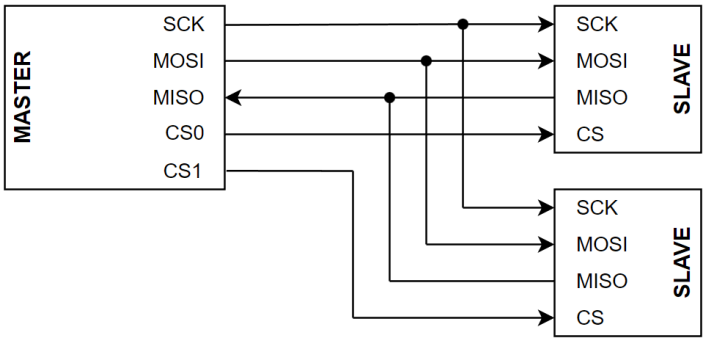


Obrázek 1: Přípravek DE10-Lite, převzato z [1]

2.4 Sériová rozhraní

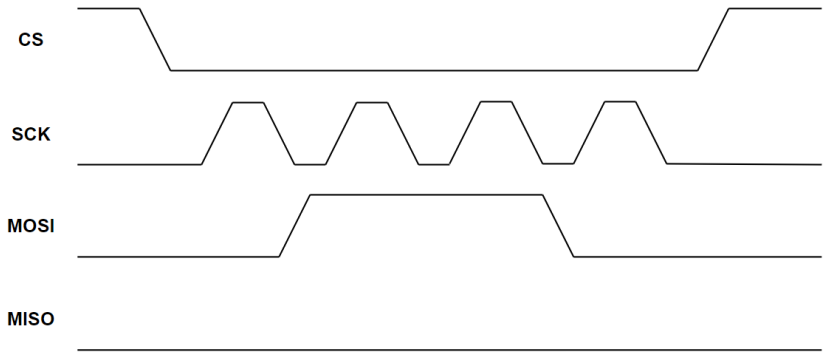
2.4.1 SPI

Sběrnice SPI je synchronní sériová 4 vodičová sběrnice. Sběrnice se řadí mezi typ s jedním řídícím zařízením (single master) a více řízenými zařízeními (multiple slaves). Pro komunikaci používá vodiče označené SCK, MOSI, MISO a CS (SS). Vodič SCK je vodič hodinového signálu, který master sám generuje. Hodinový signál se používá pro synchronizaci posílaných/přijatých dat. Vodič MOSI (master out slave in) je vodič, po kterém master posílá data a slave je odtud přijímá. Vodič MISO (master in slave out) je vodič, po kterém slave posílá data a master je odtud přijímá. Poslední vodič CS (chip select, občas označováno SS – slave select), je vodič, kterým si master vybírá slave zařízení, se kterým chce komunikovat. Pro každé slave zařízení je potřeba zvlášť CS vodič. Nevýhodou sběrnice SPI je, že pro každé slave zařízení je potřeba zvlášť CS vodič. Schématické zapojení sběrnice je na obrázku 2.



Obrázek 2: Zapojení sběrnice SPI

Logické úrovně a komunikace na úrovni bitů nejsou pro sběrnici SPI přesně definována. Každé zařízení může mít trochu jinak nastavené parametry přenosu dle dokumentace. Na obrázku 3 je uveden příklad jednoho typu komunikace.



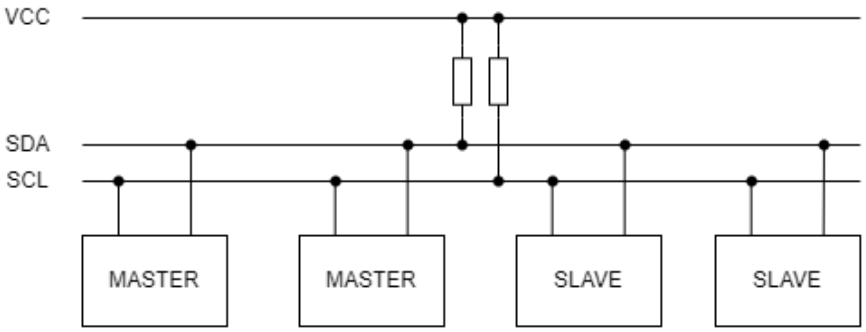
Obrázek 3: Příklad komunikace na sběrnici SPI

Na výše uvedeném příkladě master zahájil komunikaci se slave zařízením stažením vodiče CS do logické 0. Během komunikace master generoval hodinový signál. Uvažujme, že v příkladě se bity přenáší na vzestupnou hranu hodinového signálu – tedy během přenosu master přenesl na slave zařízení poslouplost „0110“. Slave na mastera přenesl posloupnost „0000“.

Sběrnice SPI má i méně používanou 3 vodičovou variantu, která spojuje datové vodiče (MISO, MOSI) do jednoho. Výhodou této varianty je ušetření jednoho vodiče, nevýhodou je, že komunikace nemůže probíhat plně duplexně, ale pouze v half-duplex režimu.

2.4.2 I²C

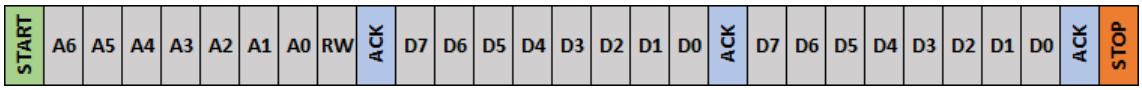
Sběrnice I²C je synchronní sériová sběrnice typu multiple masters multiple slaves. Sběrnice má pouze dva datové vodiče SDA a SCL. Vodič SDA se používá pro datovou komunikaci. Vodič SCL se používá pro hodinový signál. Každé zařízení na sběrnici je adresováno 7bitovou adresou.



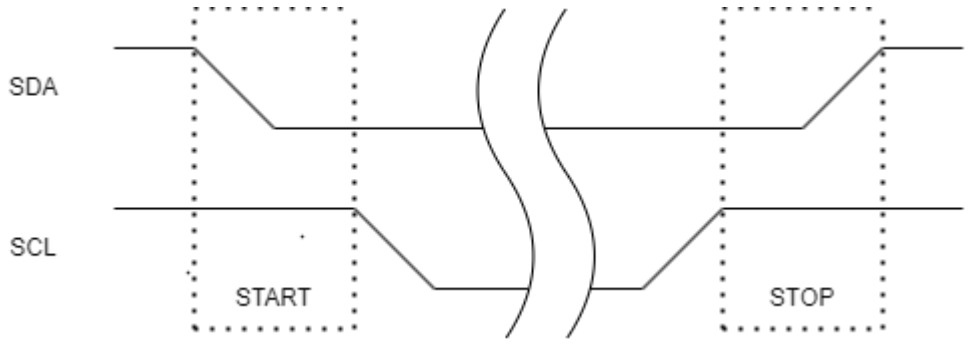
Obrázek 4: Zapojení sběrnice I²C

Přenos začíná tak, že je masterem sběrnice SDA nastavena na log. 0 (označováno jako start podmínka). Po startu master odešle 7 bitů adresy, kterým si zvolí zařízení, se kterým chce

komunikovat. 8. bit slouží jako R/W bit. Pokud adresované zařízení rozpoznalo, že s ním chce master komunikovat, odešle zpět na sběrnici tzv. acknowledge bit (ACK). Dále komunikace probíhá podle specifikace slave zařízení, v 8bitových datových blocích potvrzovaných ACK bity. Pro ukončení komunikace master nastaví SDA do log. 1 po tom, co SCL byl nastaven do log. 1 (tzv. stop podmínka).



Obrázek 5: Příklad datové komunikace na sběrnici I^2C



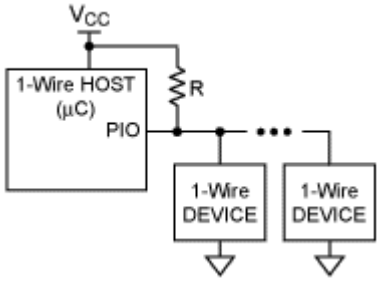
Obrázek 6: Zobrazení start a stop podmínky na sběrnici I^2C

Sběrnice je vhodná na krátké vzdálenosti, její výhodou oproti SPI je menší počet vodičů a možnost více master zařízení. Nevýhodou oproti SPI je složitější realizace komunikace díky adresaci a komunikace pouze v half-duplex režimu.

2.4.3 1-Wire

Sběrnice 1-Wire byla vytvořena společností Maxim Integrated [3]. Jedná se o sériovou komunikační sběrnici typu single master multiple slaves. Sběrnice byla vytvořena pro komunikaci s jednoduchými zařízeními na krátké vzdálenosti [3]. Každé řízené zařízení má svoji unikátní adresu, což umožňuje připojení několika slave zařízení na sběrnici.

2.4.3.1 Zapojení sběrnice 1-Wire



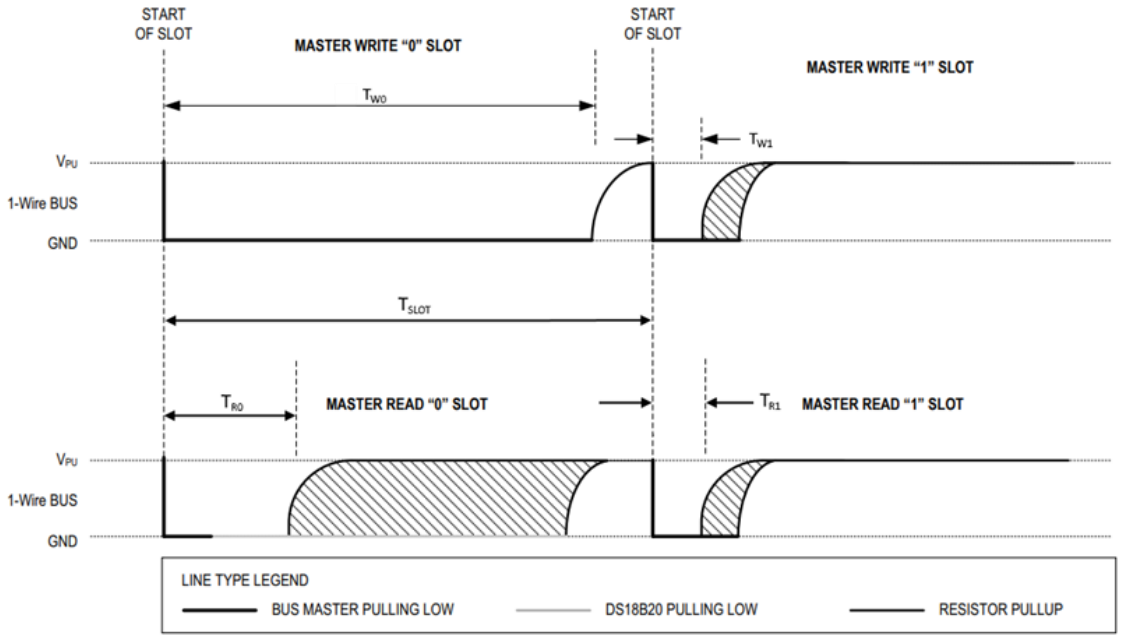
Obrázek 7: Zapojení sběrnice 1-Wire, převzato z [3]

Na obr. 7 je znázorněno zapojení sběrnice. Ze schématu je vidět, že sběrnice je přidržována ve stavu log. 1 pomocí pull-up rezistoru. Doporučená hodnota pull-up rezistoru R je

4,7 kΩ. Většina slave zařízení podporující 1-Wire technologii nepotřebuje externí napájení, jelikož se dokáží napájet přímo ze sběrnice.

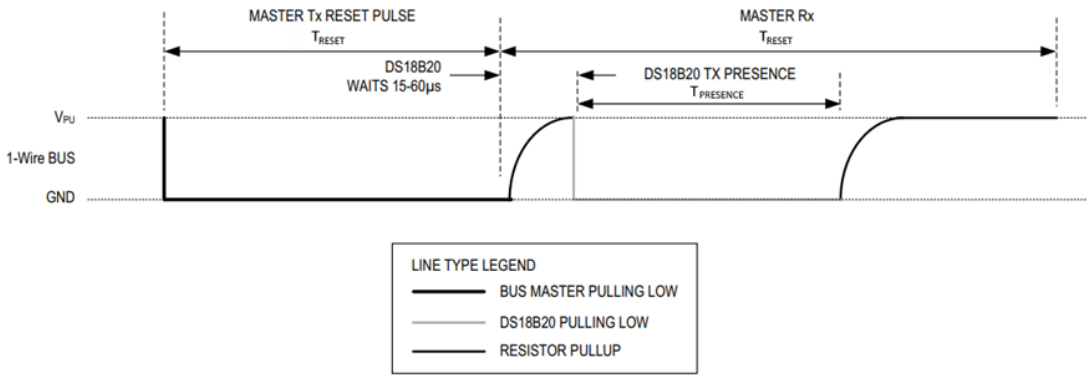
2.4.3.2 Komunikace na sběrnici 1-Wire

Komunikace po sběrnici probíhá v tzv. časových slotech. V tomto vyhrazeném časovém slotu fixní délky může vysílací zařízení stáhnout sběrnici do log. 0. Délka tohoto pulzu indikuje, zda zařízení odeslalo log. 0 nebo log. 1. Časování je znázorněno na obr. 8.



Obrázek 8: Časování sběrnice 1-Wire [4], upraveno

Pro reset veškerých slave zařízení na sběrnici se využívá reset pulz. Po resetu slave odpovídá presence pulzem.



Obrázek 9: Reset sběrnice 1-Wire [4], upraveno

Délky jednotlivých pulzů jsou popsány v tabulce 1.

Tabulka 1: Délky pulzů [4]

	Délka pulzu (µs)	
	min	max
T_{SLOT}	60	120
T_{W0}	60	120
T_{W1}	1	15
T_{R0}	15	60
T_{R1}	1	15
T_{RESET}	480	-
$T_{PRESENCE}$	60	240

2.4.3.3 Příkazy na sběrnici 1-Wire

V kapitole 2.5.2. byla popsána komunikace mezi zařízeními na úrovni bitů. Master může odeslat posloupnost 16 bitů, které tvoří příkaz. Posloupnost se odesílá od LSB (bit s nejmenším významem) po MSB (bit s největším významem. Slave zařízení na tyto příkazy můžou reagovat. Ne každé slave zařízení podporuje všechny příkazy. V tabulce 2 je uvedený výtah důležitých příkazů pro komunikaci se senzorem použitým v této práci.

Tabulka 2: Hexadecimální hodnota příkazů na sběrnici 1-Wire [4]

Název	Hodnota
Search ROM	0xF0
Read ROM	0x33
Match ROM	0x55
Skip ROM	0xCC
Convert T	0x44
Read Scratchpad	0xBE
Write Scratchpad	0x4E

Příkazy mají následující funkce:

- Search ROM – Master vyzve všechny slave zařízení, aby odeslaly svoji adresu.
- Read ROM – Master vyzve pouze jedno slave zařízení, aby odeslalo svoji adresu. Lze použít pouze, pokud je jen jedno slave zařízení připojeno ke sběrnici
- Match ROM – Master zahájí komunikaci s vybraným slave. Výběr probíhá odesláním adresy cílového zařízení po odeslání příkazu.
- Skip ROM – Master zahájí komunikaci se všemi slave zařízeními.
- Convert T – Master vyzve senzor, aby si uložil aktuální teplotu.
- Read Scratchpad – Master vyzve senzor, aby mu odeslal uloženou teplotu.

- Write Scratchpad – Master tímto příkazem zapisuje do paměti slave zařízení. Master musí ukládaná data poslat hned po tomto příkazu.

2.4.3.4 Adresace na sběrnici 1-Wire

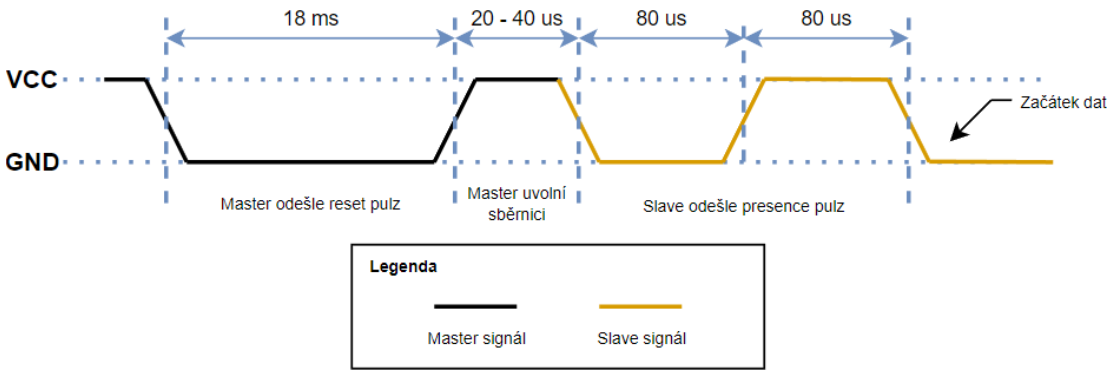
Každé slave zařízení na sběrnici má svoji unikátní 64bitovou adresu. Pro zjištění adresy lze využít příkazy *Search ROM* nebo *Read ROM*. Pro zahájení komunikace s určitým zařízením je potřeba odeslat příkaz *Match ROM* a následně 64bitovou adresu vybraného zařízení. Pokud je na sběrnici připojeno pouze jedno slave zařízení, lze adresování vynechat a zahájit komunikaci příkazem *Skip ROM*.

2.4.4 AOSONG Single Wire

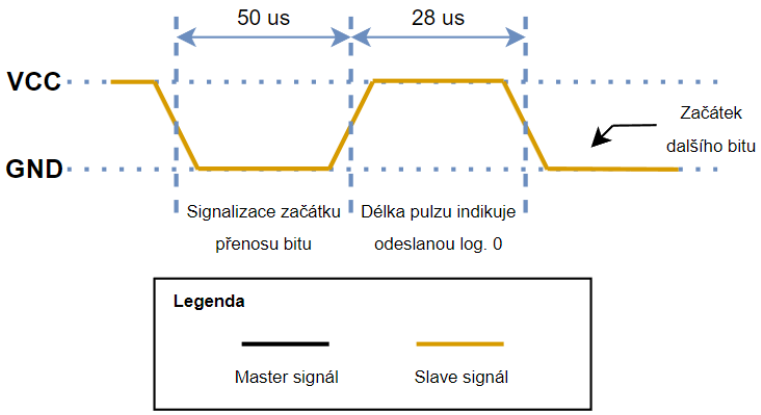
Sběrnice Single Wire od firmy AOSONG je zjednodušená verze sběrnice 1-Wire [5]. Slave zařízení na této sběrnici nejsou adresována. Díky tomu ke sběrnici může být připojeno jen jedno slave zařízení.

2.4.4.1 Komunikace po sběrnici Single Wire

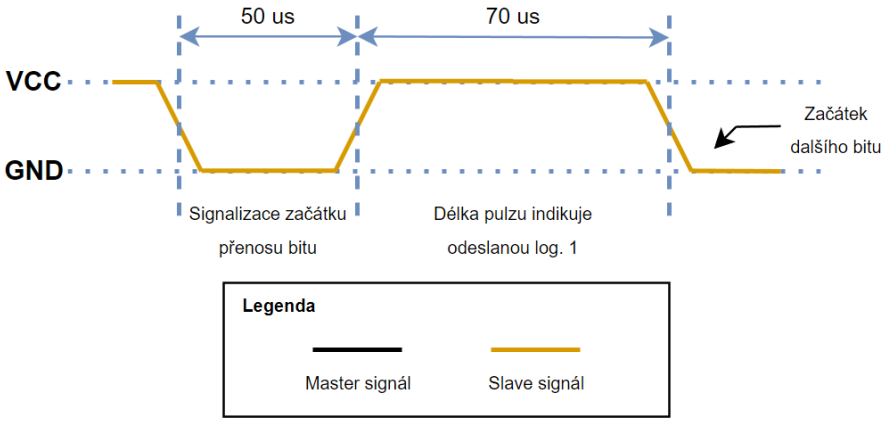
Komunikace začíná reset pulzem vyslaným od master zařízení. Slave zařízení na tento reset odpoví tzv. „presence“ pulzem a poté odešle data (viz. obrázek 10). Data jsou reprezentována délkou pulzů logických 0 a 1 na sběrnici dle obrázků 7 a 8.



Obrázek 10: Začátek komunikace po sběrnici [5], překresleno



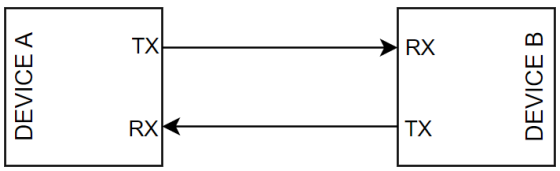
Obrázek 11: Reprezentace log. 0 na sběrnici [5], překresleno



Obrázek 12: Reprezentace log. 1 na sběrnici, převzato z [5]

2.4.5 UART

UART (z anglického universal asynchronous receiver transmitter) je univerzální sériový protokol často využívaný pro komunikaci mezi mikrokontroléry a sériovým portem počítače. Komunikace probíhá pouze mezi dvěma zařízeními v režimech simplex, half-duplex nebo duplex. Zařízení pro komunikaci využívají vodiče TX (transmitter – vysílač) a RX (reciever – přijímač).



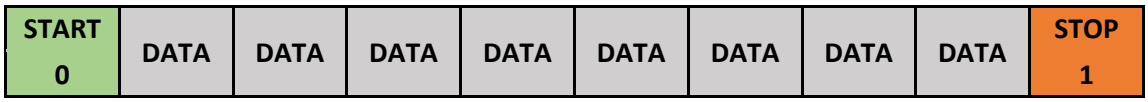
Obrázek 13: Zapojení sběrnice UART v duplex režimu

2.4.5.1 Rámce UART

Před samotnou komunikací na sběrnici je potřeba, aby zařízení byla stejně nakonfigurována. Před komunikací je potřeba definovat následující parametry:

- Rychlost přenosu (rychlost v baudech)
- Počet datových bitů (5-9)
- Počet paritních bitů (0-1, lichá/sudá parita)
- Počet stop bitů (1-2)

Pokud obě zařízení nejsou nakonfigurovány na stejné parametry přenosu, pak nelze data přenášet. Nejčastěji se používá konfigurace o rychlosti 9600 baudů, 8 datových bitů, 0 paritních bitů a 1 stop bit (tato konfigurace se označuje 8N1). Přenos začínající start bitem a končící stop bitem se označuje jako rámec.



Obrázek 14: Rámec 8N1

2.5 Senzor teploty DHT11

Senzor teploty a vlhkosti vzduchu DTH11 je vyroben firmou AOSONG Electronics. Senzor komunikuje po sběrnici AOSONG Single Wire. Jeho napájecí napětí je 3–5 V [6]. Senzor má čtyři vývody – VDD, DQ, NC a GND. Vývod VDD se používá pro připojení napájení a vývod GND se využívá pro elektrickou zem. Vývod DQ se využívá pro komunikaci. Vývod NC není zapojen. Senzor měří teploty v rozsahu 0 až 50 °C s přesností ± 2 °C, relativní vlhkost v rozsahu 20 až 90 % s přesností ± 5 %.

Pro zahájení komunikace se senzorem je potřeba resetovat sběrnici. Senzor po resetu na sběrnici odešle presence pulz. Po odeslání tohoto pulzu odešle 40 bitů dat. Data jsou posílána od MSB po LSB. První byte odpovídá vlhkosti vzduchu. Druhý byte odpovídá hodnotě desetinného místa vlhkosti vzduchu (pro tento senzor by tato hodnota měla být vždy nulová). Třetí byte odpovídá hodnotě teploty. Čtvrtý byte odpovídá hodnotě desetinného místa teploty (opět pro tento senzor by tato hodnota měla být rovna 0). Poslední byte odpovídá kontrolnímu součtu. Struktura dat je naznačena na obr.: 15.

Byte 0	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4**
Vlhkost	0	Teplota	0	Kontrolní součet

Obrázek 15: Struktura přijatých dat ze senzoru DHT11

Kontrolní součet se vypočítá dle následujícího vztahu:

$$SUM = Byte0 + Byte1 + Byte2 + Byte3$$

Master si může tento součet vypočítat a porovnat posledních osm bitů výsledku s přijatým součtem a získat tak základní kontrolu nad přenesenými daty.

2.6 Senzor teploty DS18B20

Senzor teploty DS18B20 je vyroben firmou Maxim Integrated. Senzor komunikuje po sběrnici 1-Wire. Jeho napájecí napětí je 3–5 V s rozlišením teploty až 12 bitů v rozsahu teplot −55 až +125 °C [4]. Senzor má tři vývody – VDD, DQ a GND. Vývod VDD se používá pro připojení napájení a vývod GND se využívá pro elektrickou zem. Vývod DQ se využívá pro komunikaci.

Výčet teploty ze senzoru se provádí ve dvou krocích. Nejprve je potřeba odeslat senzoru příkaz „Convert T“. Po přijetí tohoto příkazu senzor změří aktuální teplotu a zapíše si ji do paměti. Pro přečtení teploty z paměti je potřeba odeslat příkaz „Read Scratchpad“. Senzor na tento příkaz odešle 8 bytů – první dva byty obsahují informace o teplotě. Byte 9 obsahuje kontrolní cyklický součet. Struktura dat je na následujícím na obr. 16.

	SCRATCHPAD
BYTE 0	TEPLOTA LSB (50h)
BYTE 1	TEPLOTA MSB (05h)
BYTE 2	UŽIVATELSKÝ BYTE 1
BYTE 3	UŽIVATELSKÝ BYTE 2
BYTE 4	KONFIGURAČNÍ REGISTR
BYTE 5	VYHRAZENO PRO SENZOR (FFh)
BYTE 6	VYHRAZENO PRO SENZOR
BYTE 7	VYHRAZENO PRO SENZOR (10h)
BYTE 8	CRC

Obrázek 16: Paměť senzoru DS18B20 [4], upraveno

Bity obsahující informace o teplotě jsou v datovém formátu „signed“. Poslední 4 bity obsahují informaci o desetinném místě, dalších 12 bitů obsahuje hodnotu teploty. Příklad převodu dat na hodnotu teploty je uveden v následující tabulce.

Tabulka 3: Převod dat na teplotu, převzato z [4], přeloženo

TEPLOTA (°C)	DIGITÁLNÍ VÝSTUP (BINÁRNÍ)
+125	0000 0111 1101 0000
+85*	0000 0101 0101 0000
+25.0625	0000 0001 1001 0001
+10.125	0000 0000 1010 0010
+0.5	0000 0000 0000 1000
0	0000 0000 0000 0000
-0.5	1111 1111 1111 1000
-10.125	1111 1111 0101 1110
-25.0625	1111 1110 0110 1111
-55	1111 1100 1001 0000

Senzor lze konfigurovat zápisem do konfiguračního registru pomocí příkazu „Write Scratchpad“. Po přijetí tohoto příkazu senzor očekává 3 byty dat na zapsání. První dva byty jsou

vyhrazená pro uživatelská data a třetí byte je určen pro konfiguraci. Konfigurační registr je vyznačen na obrázku 17.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
0	R1	R0	1	1	1	1	1

Obrázek 17: Konfigurační registr senzoru DS18B20, převzato z [4]

Pomocí proměnných bitů R1 a R0 můžeme volit rozlišení sezoru, ostatní bity v registru jsou vyhrazeny pro senzor a nemůžou být přepsány. Volbou R1 a R0 měníme rozlišení senzoru. Pokud senzor měří s větším rozlišení, trvá déle ukládání teploty do paměti po přijetí příkazu „Convert T“. Nastavitelné rozlišení senzoru je 9 až 12 bitů, což odpovídá rozlišení 0,5 °C až 0,0625 °C. Konfigurace bitů je zachycena v tabulce 4.

Tabulka 4: Konfigurace rozlišení senzoru [4], upraveno

R1	R0	ROZLIŠENÍ (V BITECH)	MAXIMÁLNÍ DOBA PŘEVODU
0	0	9	93.75ms
0	1	10	187.5ms
1	0	11	375ms
1	1	12	750ms

Ve výchozí konfiguraci je senzor nastaven s rozlišením 12 bitů.

2.7 Znakový LCD displej

Znakový LCD displej o velikosti 16 x 2 (16 znaků na řádek, 2 řádky) je vybaven univerzálním LCD ovladačem HD44780U od firmy Hitachi. Ovladač umožňuje základní funkce (jako např. vykreslování či mazání znaků) pomocí jednoduché paralelní komunikace.

2.7.1 Zapojení displeje

Displej má 16 vývodů, vývody jsou popsány v následující tabulce číslo 5.

Tabulka 5: Vývody LCD displeje

Číslo	Označení	Funkce
1	VSS	Napájení (GND)
2	VDD	Napájení (5 V)
3	VO	Kontrast displeje
4	RS	Register Select
5	RW	Read/Write
6	E	Enable
7	D0	Data pin 0
8	D1	Data pin 1
9	D2	Data pin 2
10	D3	Data pin 3
11	D4	Data pin 4
12	D5	Data pin 5
13	D6	Data pin 6
14	D7	Data pin 7
15	A	Anoda LED podsvícení
16	K	Katoda LED podsvícení

Displej je napájen 5 V. Na vývod *VO* lze přivést jakoukoliv hodnotu mezi 0 V až 5 V. Velikost napětí na tomto pinu mění kontrast znaků vykreslovaných na displeji. Vhodné je na tento pin zapojit potenciometr, aby si uživatel mohl kontrast měnit. Vývody *A* a *K* slouží k připojení napájení LED podsvícení displeje. Piny 4 – 14 jsou datové piny a jsou obsluhovány přímo LCD ovladačem. Tyto piny podporují 5 V i 3,3 V logiku [6].

2.7.2 Komunikace s ovladačem

Vývod *RS* určuje, jak mají být data zpracována. Pokud je *RS* v log. 0, data jsou interpretována jako příkazy. Pokud je *RS* v log. 1, data jsou interpretována jako ASCII znaky a vykreslována na displej. Vývod *RW* určuje zápis, nebo čtení dat z displeje. Log. 0 označuje zápis dat, log. 1 čtení.

Vývod *E* slouží k odesílání dat na ovladač. Pokud je přivedena na vývod *E* náběžná hrana, data z pinů *RS*, *RW*, *D0* až *D7* jsou zachycena a zpracována ovladačem.

Ovladač může komunikovat v 8bitovém nebo 4bitovém režimu. Pokud pracuje v 8bitovém, jsou data přivedena na piny *D0* až *D7* a tato data jsou zpracována na jeden Enable pulz. Pokud pracuje v 4bitovém režimu, využívají se pouze piny *D4* až *D7*. Nejprve se na datové piny přivedou první 4 bity s nejvyšší váhou a odešlou se Enable pulzem. Poté se přivedou na piny *D4* až *D7* poslední 4 bity a znovu se odešlou Enable pulzem. Tímto je 8bitová informace rozložena do 2×4bitů. Režim je na ovladači možné přepnout pomocí příkazů.

2.7.3 Příkazy ovladače HD44780U

V tabulce 6 jsou zobrazeny základní příkazy potřebné pro ovládání displeje.

Tabulka 6: Vybrané instrukce LCD ovladače [6]

Instrukce	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0
Clear display	0	0	0	0	0	0	0	0	0	1
Display on/off cntrl	0	0	0	0	0	0	1	*D*	*C*	*B*
Entry mode set	0	0	0	0	0	0	0	0	*I/D*	*S*
Function set	0	0	0	0	1	*DL*	*N*	*F*	X	X
Cursor set	0	0	1	*L*	DATA					

- Instrukce *Clear display* smaže všechny znaky na displeji
- Instrukce *Display on/off control* zapíná (*D* = 1), nebo vypíná displej. Dále tato instrukce umožňuje zapnout kurzor (*C* = 1) a povolit blikání kurzoru (*B* = 1).
- Instrukce *Entry mode set* nastavuje, jestli se má kurzor pohybovat po zapsání znaku vpravo (*I/D* = 1) nebo vlevo. Nastavením *S* = 1 se po zapsání znaku místo posunutí kurzoru, posune celý displej.
- Instrukce *Function set* nastavuje 8bitový (*DL* = 0) nebo 4bitový datový přenos. Parametr N určuje počet řádků displeje (N = 0 odpovídá 1 řádkovému displeji, N = 1 pro 2 řádkový displej). *F* přepíná mezi znaky o velikosti 5×8 pixelů (F = 0) nebo 5×10 pixelů [6]. Nastavení parametrů je potřeba přizpůsobit displeji, který je ovladačem ovládán. (Pozn.: na logické úrovni X nezáleží.)
- Instukce *Cursor set* nastaví kurzor na příslušnou pozici. Parametr *L* určuje řádek, na který chceme kurzor nastavit (*L* = 0: první řádek; *L* = 1: druhý řádek). Na pozici DATA binárně zapíšeme číslo sloupce, na který se má kurzor nastavit.

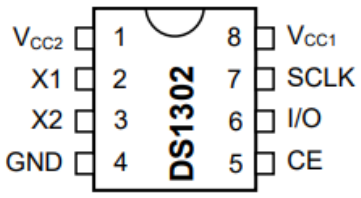
2.7.4 Inicializace a zápis znaků na displej

Po zapnutí napájení je potřeba inicializovat displej následujícími příkazy: *Function set*, *Display on/off control*, *Entry mode set* a *Clear display*. Po této inicializaci se kurzor nastaví na pozici 0 (první řádek, první sloupec). Pro zápis znaku stačí nastavit RW do logické 1 a datové piny nastavit dle požadovaného znaku. Ovladač používá modifikovanou (a zároveň limitovanou) ACII tabulku viz. tabulka 7.

Tabulka 7: Tabulka znaků ovladače [6]

2.8 Obvod reálného času DS1302

Obvod DS1302 byl vyvinut společností Maxim Integrated [7], který slouží k udržování reálného času a data díky připojené záložní baterii. Obvod má 8 vývodů. Na obr. 18 je zobrazeno pouzdro obvodu.



Obrázek 18: Obvod DS1302, převzato z [7]

Vývod VCC_2 slouží jako primární zdroj napájení, VCC_1 je určen pro záložní baterii. Na piny X1 a X2 je potřeba připojit krystal o frekvenci 32,768 kHz. Piny SCLK, I/O a CE jsou vývody 3 vodičové SPI sběrnice (častěji označované jako CLK, DATA a CS).

Nejprve je potřeba do obvodu zapsat aktuální datum a čas. Zápis se provádí tak, že se nejprve odešle 8bitová adresa, do které chceme zapisovat a poté 8 bitů s daty. Pro výčet dat je potřeba odeslat adresu na kterou senzor odešle 8bitů s daty.

Tabulka 8: Paměť obvodu DS1302 [7], upraveno

<table>
<tr><th>ZÁPIS</th><th>ČTENÍ</th><th>BIT 7</th><th>BIT 6</th><th>BIT 5</th><th>BIT 4</th><th>BIT 3</th><th>BIT 2</th><th>BIT 1</th><th>BIT 0</th></tr>
<tr><td>81h</td><td>80h</td><td>CH</td><td colspan="3">x10 sekund</td><td colspan="4">x1 sekund</td></tr>
<tr><td>83h</td><td>82h</td><td>0</td><td colspan="3">x10 minut</td><td colspan="4">x1 minut</td></tr>
<tr><td>85h</td><td>84h</td><td>12/24</td><td>0</td><td>x10
AM/PM</td><td>hodin</td><td colspan="4">x1 hodin</td></tr>
<tr><td>87h</td><td>86h</td><td>0</td><td>0</td><td colspan="2">x10 dní</td><td colspan="4">x1 den</td></tr>
<tr><td>89h</td><td>88h</td><td>0</td><td>0</td><td>0</td><td>x10
měsíců</td><td colspan="4">x1 měsíc</td></tr>
<tr><td>8Bh</td><td>8Ah</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="3">den v týdnu</td></tr>
<tr><td>8Dh</td><td>8Ch</td><td colspan="4">x10 let</td><td colspan="4">x1 rok</td></tr>
</table>

Po zapsání dat obvod automaticky začne počítat čas. Pozastavit hodiny lze nastavením bitu CH (Clock Halt) do log. 1. Pro výčet aktuálního data a času je výhodné použít tzv. „burst mode“. Po zapsání adresy 0xBF [7] obvod začne postupně odesílat bajty od 0x81, 0x83 atd., až po 0x8D. Burst komunikaci můžeme kdykoliv přerušit, pokud řídící obvod přestane posílat další hodinový signál a CE (CS) je nastaven do log. 0.

2.9 Převodník TTL na USB PL2303

Převodník TTL na USB převádí UART sériovou komunikaci na USB komunikaci. Tento převodník je potřeba kvůli odstranění sériových (COM) portů z moderních osobních počítačů. Čip PL2303 tento převod zajišťuje. Důležitými vývody na tomto čipu jsou TXD a RXD (vysílač a přijímač sériové komunikace) a D+ a D- (datové piny USB kabelu). Čip podporuje 3,3 V i 5 V logiku [8].

Obrázek 19: Převodník TTL na USB s čipem PL2303

2.10 Piezoměnič

Piezoměniče převádí elektrický signál na zvuk pomocí inverzního piezoelektrického jevu. Dělí se na aktivní a pasivní piezoměniče.

Aktivní piezoměnič obsahuje piezoelektrický disk se zesilovačem a oscilátorem, který generuje tón o předem daném kmitočtu po přivedení stejnosměrného napětí. Aktivní

piezoměniče se snadno ovládají, stačí přivést log. 1 pro vytvoření tónu a log. 0 pro vypnutí. Nevýhodou těchto měničů je možnost generovat pouze jeden tón daný výrobcem měniče.

Pasivní piezoměniče jsou konstruovány podobně jako reproduktory. Cívka vytváří magnetické pole, které působí na piezoelektrický disk, díky čemu lze generovat tóny. Oproti aktivním piezoměničům je potřeba na vstup přivádět střídavý signál. Výhodou je, že díky tomu můžeme generovat širší spektrum tónů než u aktivních měničů.

2.11 Programovací jazyk Python

Programovací jazyk Python je vysokoúrovňový objektově orientovaný programovací jazyk navržený Guidem Van Rossumem [9]. Python je dnes jeden z nejpopulárnějších programovacích jazyků, a to i díky velkému množství externích modulů a knihoven. Jednou z význačných vlastností jazyka je způsob ohraničení bloků, které se oddělují velikostí odsazení. Díky tomu je tak kód psaný v jazyce Python přehlednější a čitelnější něž v jiných jazycích.

3. Praktická část

Pro návrh bloků v jazyce VHDL byl využit program Quartus Prime Lite od firmy Intel. Práce je kompatibilní s FPGA poli řady MAX10.

3.1 Blokové schéma

Celá práce se skládá z několika dílčích komponent. V příloze A je uvedeno blokové schéma práce. Některé komponenty jsou v práci použity vícekrát. Pro větší přehlednost je toto v blokovém schématu sjednoceno do jednoho bloku a označeno *[A…N]*. Toto označení odpovídá samostatným blokům A, B, až N.

3.2 Blok main

Komponenta *main* zařizuje propojení všech ostatních komponent. Součástí této komponenty je také čítač s multiplexorem (pro přepínání hodnot zobrazovaných na displejích), komparátor (pro hlídání rozsahu nastavených hodnot) a RAM paměť (pro dočasné ukládání nastavených hodnot). Blok se dále stará o distribuci hodinového signálu.

3.2.1 Propojování komponent ve VHDL

Pro propojení komponent je nejprve potřeba komponentu vytvořit a definovat její porty. Toto lze vytvořit pomocí následujícího bloku kódu. (V příkladu je uveden převodník binárních dat na BCD kód)

```
component binary2bcd is
   port (binary   : in  unsigned(15 downto 0);
         bcd      : out unsigned(15 downto 0)
   );
end component;
```

Obrázek 20: Definice komponenty a jejích portů ve VHDL

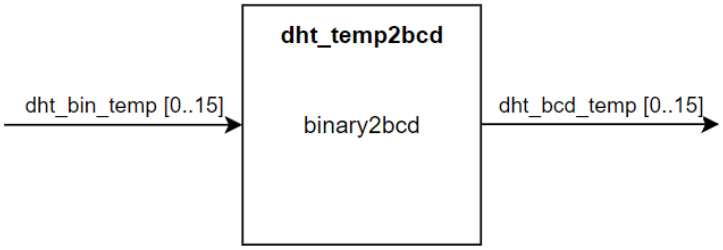
Ve výše uvedeném kódu jsme zadefinovali komponentu *binary2bcd* s porty označenými *binary* a *bcd*. Port *binary* je definován jako 16bitový vstup datového typu unsigned. Port *bcd* je definován jako 16bitový výstup datového typu unsigned. Při syntetizaci kódu bude syntetizátor hledat entitu pojmenovanou *binary2bcd*, která bude definovat chování komponenty.

Zadefinovanou komponentu nyní můžeme propojit. Propojit ji můžeme s fyzickými vstupy či výstupy kitu MAX10. Častěji se komponenty propojují pomocí kódové reprezentace vodiče označované jako signály. Propojení vytvoříme pomocí příkazu ***port map()***.

```
--definice signalu
signal dht_bin_temp  : unsigned(15 downto 0)
signal dht_bin_rhum  : unsigned(15 downto 0)

dht_temp2bcd: binary2bcd port map(dht_bin_temp(15 downto 0), dht_bcd_temp(15 downto 0));
```

Obrázek 21: Propojování komponent ve VHDL

Ve výše uvedeném kódu jsme si vytvořili dva 16bitové signály. Do instance pojmenované *dht_temp2bcd* komponenty *binary2bcd* jsme na vstup připojili signál *dht_bin_temp* a na výstup *dht_bcd_temp*. Blokově to můžeme znázornit např. takto:



Obrázek 22: Znázornění realizovaného propojení

Signály, které vedou z tohoto bloku pak pomocí ***port map()*** můžeme zapojovat do dalších komponent, nebo s nimi můžeme provádět logické operace.

3.2.2 Procesy v jazyce VHDL

Procesy definují sekvenční kód v jazyce VHDL. V následujícím bloku kódu je ukázána základní struktura procesu:

```
process (sensitivity_list) is
   -- deklarace promennych
   begin
   --sekvencni prostredi


end process;
```

Obrázek 23: Procesy v jazyce VHDL

Nejprve je proces definován pomocí ***process(****sensitivity_list****) is***. *Sensitivity_list* je soubor signálů, které způsobí spuštění daného sekvenčního kódu. Poté můžeme definovat proměnné, které mohou být užitečné k mezivýpočtům. Od klíčového slova ***begin*** lze psát sekvenční kód po klíčové slovo ***end process***.

Velmi často je potřeba vytvořit synchronní proces. To lze vytvořit zařazením hodinového signálu do *sensitivity_listu* a v sekvenčním kódu zařadit podmínku na ***rising_edge(****clk****)*** nebo ***falling_edge(****clk****)*** pro synchronizaci na náběžnou nebo sestupnou hranu.

3.2.3 Přepínání zobrazené hodnoty na 7 seg. displeji, nastavování hlídaných mezí

Při stisknutí tlačítka dochází k přepnutí zobrazené hodnoty na segmentovém displeji. To je ve VHDL realizováno procesem, který detekuje sestupnou hranu signálu z tlačítka (výstup tlačítka je u přípravku DE10-Lite invertován). Při detekci této hrany se zvýší hodnota čítače o jedna. Dle hodnoty čítače je se určuje, jaká data jsou multiplexována do bloků, které se starají o výpis hodnoty na displeji.

```
process (clk, btn0) is

   variable disp_select : integer := 0;    -- defaultni hodnota citace

   begin

      if rising_edge(clk)                  -- multiplexovani dat
         if disp_select = 0 then
            data_out <= dataA
         end if;
         if disp_select = 1 then
            data_out <= dataB
         end if;
      end if;


      if falling_edge(btn0) then           -- detekce sestupne hrany tlacitka
         disp_select := disp_select + 1;   -- citac

         if disp_select > 1 then           -- preteceni citace
            disp_select := 0;
         end if;
      end if;

end process;
```

Obrázek 24: VHDL kód čítače a multiplexoru

Kód výše ilustruje kód použitý v bloku *main*. Přestože je kód zjednodušený, principiálně je stejný.

Nastavování hlídaných mezí je realizováno velmi obdobně. Proces sleduje, jestli uživatel zobrazuje na displeji hodnotu mezí a pokud ano, tak na stisk tlačítka *btn1* pomocí čítače jedničku přičte (pokud je přepínač *sw8 v* log. 0) nebo odečte (pokud je *sw8* v log. 1).

3.2.4 Hlídání mezí

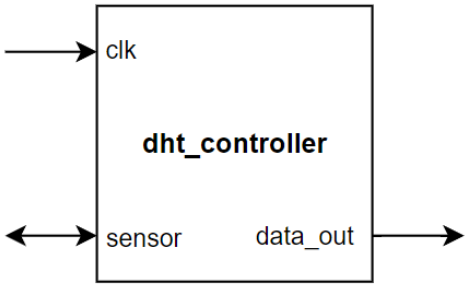
Realizace hlídání nastavených mezí ve VHDL je velmi jednoduchá. Pomocí komparátorů se porovnává, zda-li se hodnoty pohybují v nastavených mezí. Pokud alespoň jedna z hodnot je mimo nastavené meze, spustí se zvuková signalizace. Zjednodušený kód je zobrazen na obr. 25.

```
process (clk) is

   begin

      if rising_edge(clk)

         -- hlidani hodnoty teploty senzoru DHT11
         if ((dht_temp >= threshold_dhttmp) or (dht_temp <= threshold_dhttmp_low)) then
            alarm_dht_temp <= true;
         else
            alarm_dht_temp <= false;
         end if;

         -- hlidani hodnoty vlhkosti senzoru DHT11
         if ((dht_hum >= threshold_dhthum) or (dht_bin_hum <= threshold_dhthum_low)) then
            alarm_dht_humi <= true;
         else
            alarm_dht_humi <= false;
         end if;

         -- hlidani hodnoty teploty senzoru DS18B20
         if ((ds18b20_temp >= threshold_ds18) or (ds18b20_temp <= threshold_ds18_low)) then
            alarm_ds18 <= true;
         else
            alarm_ds18 <= false;
         end if;

         -- pokud je alespon jedna hodnota mimo, zapni zvukovou signalizaci
         if alarm_ds18 or alarm_dht_temp or alarm_dht_humi then
            buzzer <= '1';
         else
            buzzer <= '0';
         end if;
      end if;

end process;
```

Obrázek 25: Hlídání nastavených hodnot

3.3 Blok dht_controller

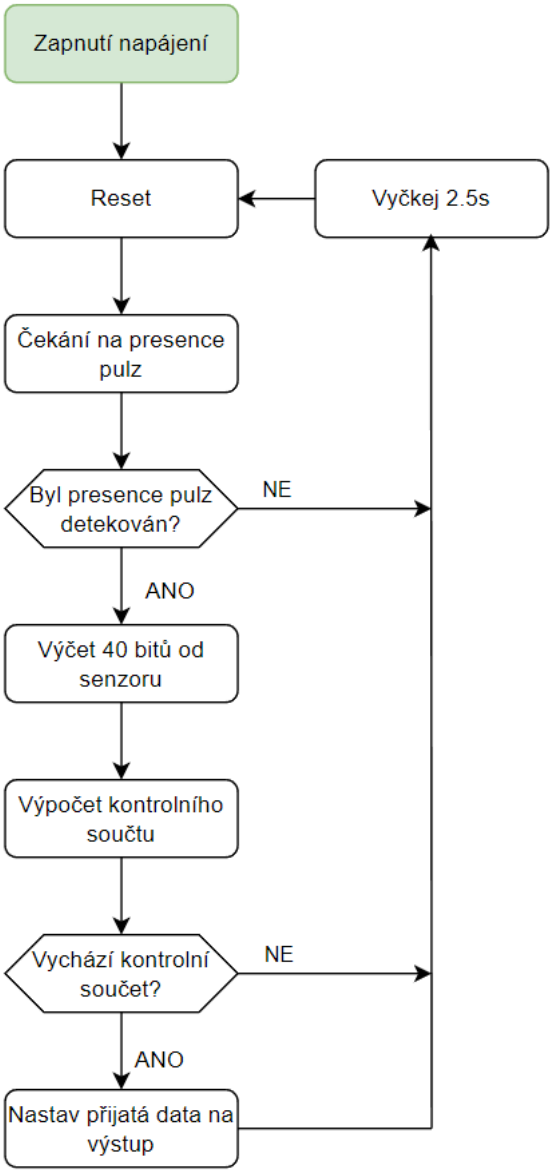
Komponenta *dht_controller* je blok, který se stará o komunikaci se senzorem DHT11 po sběrnici AOSONG Single Wire. Vyčtená data předává na výstup pro další zpracování.



Obrázek 26: Blok dht_controller

Do bloku je potřeba na vstup *clk* přivést hodinový signál o taktu 1 MHz. Na obousměrný vývod *sensor* se přímo připojí datový pin senzoru DHT11 a na výstup data_out blok přivádí vyčtená data ze senzoru.

Blok se chová jako stavový automat. Algoritmus bloku je popsán na obr. 27.



Obrázek 27: Algoritmus bloku dht_controller

Po zapnutí napájení blok resetuje sběrnici. Po resetu čeká nejdéle 200 µs na presence pulz od senzoru DHT11. Pokud žádný pulz nepřijde, blok vyčká 2,5 sekundy a znovu zkusí resetovat sběrnici. Pokud presence pulz přijde, přepne se do režimu čtení dat a začne vyčítat 40 bitů dat od senzoru. Po přijetí dat si blok ověří platnost dat kontrolním součtem: Pokud výpočet vychází, nastaví přijatá data na výstup. Pokud nevychází, data zahodí. Poté vyčká 2,5 s před dalším resetem sběrnice.

3.3.1 Implementace ve VHDL

Tento blok (jako většina v této práci) se chová jako stavový automat, to lze vytvořit ve VHDL následovně:

```
type sensor_state is (state0, state1, state2);
signal state : sensor_state := state0;

process (clk) is

   begin
      if rising_edge(clk) then

         case state is
            when state0 =>
               -- kod co se bude vykonavat, kdyz
               -- je blok ve stavu state0
            when state1 =>
               -- kod co se bude vykonavat, kdyz
               -- je blok ve stavu state1
            when state2 =>
               -- kod co se bude vykonavat, kdyz
               -- je blok ve stavu state1
         end case;

      end if;

end process;
```

Obrázek 28: Stavový automat ve VHDL

V uvedeném přikladu jsme si vytvořili vlastní datový typ *sensor_state*, který nabývá tří hodnot: *state0, state1* a *state2*. Poté jsme vytvořili signál, který má tento nový datový typ a ve výchozím stavu nabývá hodnoty *state0*. V procesu jsme pak použili ***case*** podmínku, která zjišťuje, v jakém stavu je náš signál a podle toho se vykonává část sekvenčního kódu.

Další z častých problémů, co je potřeba řešit, je odesílání pulzů v přesné časové intervaly, dle potřeby. Toho lze dosáhnout velmi jednoduchým rozšířením aktuálního kódu. Stačí zavést pomocnou proměnnou *i* typu integer, která zvýší o 1 na každý hodinový pulz. Při zvolené frekvenci 1 MHz, hodnota *i* odpovídá času od spuštění v mikrosekundách. Při správně zavedené logice lze pak jednoduše a přesně časovat.

```
type sensor_state is (state0, state1, state2);
signal state : sensor_state := state0;

process (clk) is

   variable i : integer := 0;

   begin
      if rising_edge(clk) then

         case state is
            when state0 =>
               -- kod co se bude vykonavat, kdyz
               -- je blok ve stavu state0
            when state1 =>
               -- kod co se bude vykonavat, kdyz
               -- je blok ve stavu state1
            when state2 =>
               -- kod co se bude vykonavat, kdyz
               -- je blok ve stavu state1
         end case;

         i := i + 1;

      end if;

end process;
```

Obrázek 29: Blok rozšířený o jednoduchou práci s časem

Tato práce s časem lze využít např. pro detekci odeslané hodnoty ze senzoru DHT11. V kapitole 2.4.4 bylo uvedeno, že log. hodnota odeslaných bitů je zakódována do délky pulzů logické jedničky.

```
when read_bit =>
   if sensor = '1' then
      j := j + 1;
   end if;

   if i > 60 then
      if sensor = '0' then
         if j > 45 then
            -- senzor odeslal log. 1
         else
            -- senzot odeslal log. 0
         end if;
         i := 0;
      end if;
   end if;
```

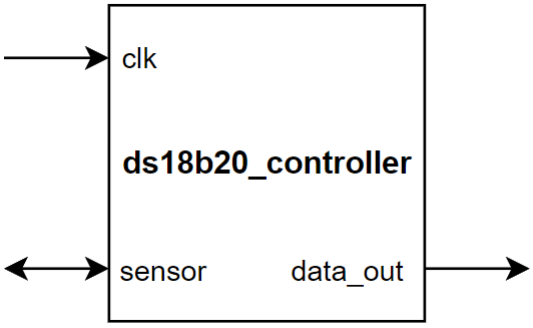
Obrázek 30: Detekce déky pulzu ve VHDL

Ve výše uvedeném kódu je výňatek ze stavu pro výčet přijatých bitů. V kódu je nová proměnná *j* typu integer. První podmínka zajišťuje, že při každém hodinovém pulzu, pokud senzor DHT11 drží sběrnici v log. 1 se hodnota proměnné *j* zvýší o jedna. To znamená, že je v proměnné *j* uložena délka pulzu log. 1 v mikrosekundách. Další podmínka kontroluje, jak dlouho už tento stav běží. Pokud uběhlo alespoň 60 µs (tímto se „filtruje“ 50 µs log. 0 na začátku přenosu bitu – viz. obr. 11 a 12), čeká se, kdy bude sběrnice stažena do log. 0. V momentě, co se tak stane, dojde k ověření poslední podmínky. Dle délky pulzu log. 1 uložené v proměnné *j* lze vyhodnotit odeslaný bit senzorem. Po vyhodnocení dojde k nastavení proměnné *i* do log. 0, aby tento stav *read_bit* mohl vyhodnotit další bit odeslaný senzorem.

Na těchto principech funguje většina bloků v práci.

3.4 Blok ds18b20_controller

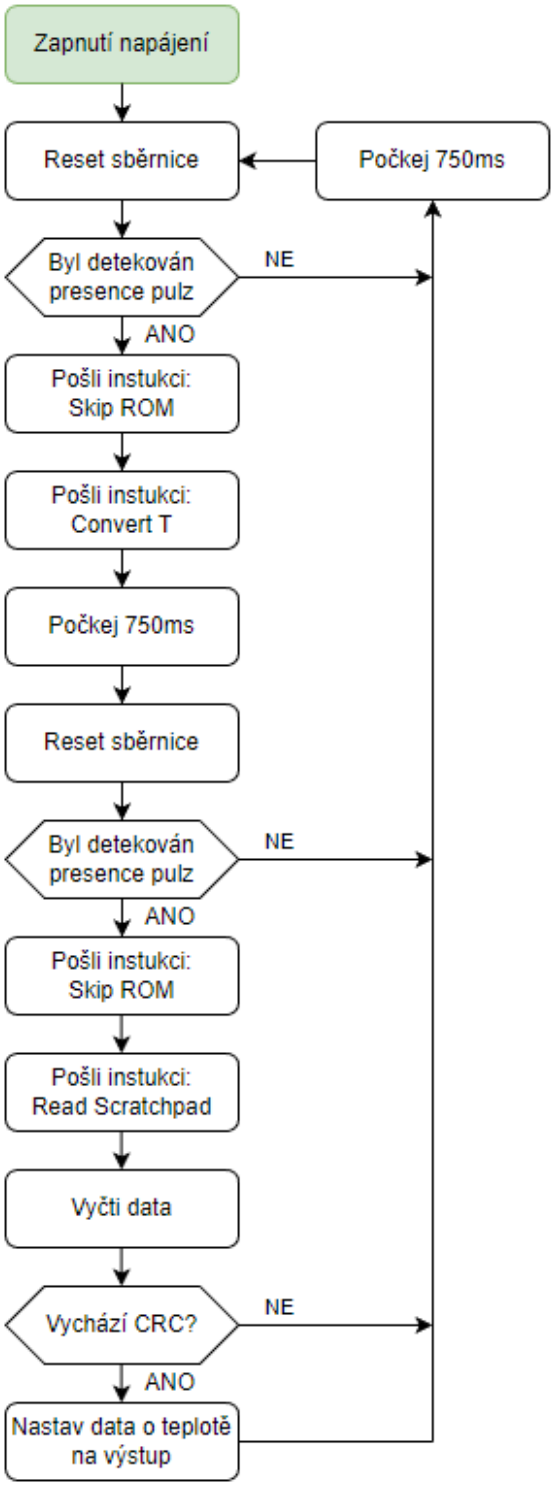
Blok *ds18b20_controller* se stará o komunikaci se senzorem DS18B20 po sběrnici 1-Wire. Vyčtená data o teplotě předává na výstup pro další zpracování.



Obrázek 31: Blok ds18b20_controller

Do bloku je potřeba na vstup *clk* přivést hodinový signál o taktu 1 MHz. Na obousměrný vývod *sensor* se přímo připojí datový pin senzoru DS18B20 a na výstup data_out blok přivádí vyčtená data ze senzoru. (Pozn.: Blok nepodporuje fukнci „parasite-power“ [4] pro napájení 1-Wire zařízení přímo z datové sběrnice. Připojené slave zařízení musí být napájeno externě.)

Blok se chová jako stavový automat, jeho algoritmus je popsán na obr. 32.

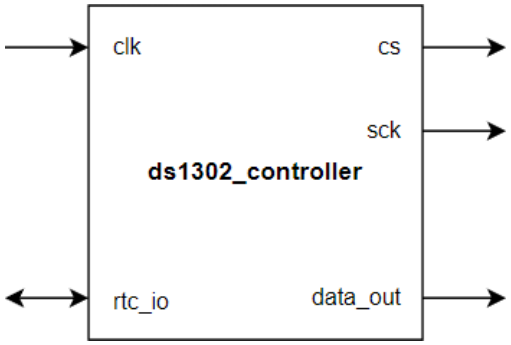


Obrázek 32: Algoritmus bloku ds18b20_controller

Po zapnutí napájení blok nejprve resetuje sběrnici. Po resetu čeká na presence pulz od senzoru DS18B20. Takto čeká nejdéle 1 s. Pokud žádný pulz nepřišel, počká dalších 750 ms a zkusí to znovu. Po přijetí presence pulzu nejprve pošle instrukci „Skip ROM“ a poté „Convert T“. Dle dokumentace měření tepoty s 12bitovým rozlišení trvá nejdéle 750 ms, tuto dobu blok čeká. Poté blok znovu resetuje sběrnici (dle dokumentace [4], je toto doporučený postup, jak ze senzoru vyčítat teplotu) a čeká na presnece pulz. Po přijetí presnece pulzu blok opět pošle příkaz „Skip ROM“ a poté instrukci na vyčtení dat „Read Scratchpad“. Po vyslání tohoto příkazu se blok přepne do stavu čtení dat. Po přijetí všech 9 bytů vypočte ověřovací CRC. Pokud souhlasí, data o teplotě nastaví na výstup, pokud nesouhlasí, data se zahodí. Poté blok vyčká 750 ms a proces se opakuje.

3.5 Blok ds1302_controller

Blok *ds1302_controller* se stará o komunikaci s integrovaným obvodem DS1302 po 3 vodičové SPI sběrnici. Vyčtená data o datu a času nastaví na výstup. Obvod byl do projektu zahrnut z důvodu pamatování si času i přes vypnuté napájení, díky zálohování z baterie.



Obrázek 33: Blok ds1302_controller

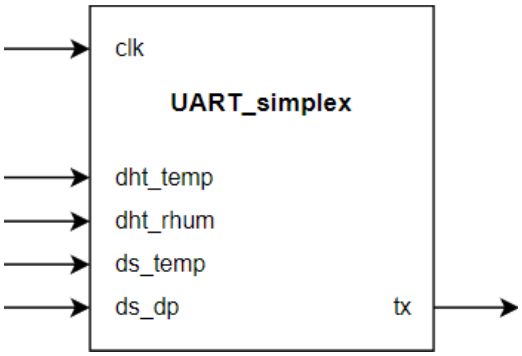
Na vstup *clk* je potřeba přivést hodinový signál o taktu 1 MHz. Na výstup *data_out* blok nastavuje přijatá data z integrovaného obvodu. Vývody *cs*, *sck* a *rtc_io* jsou vývody 3 vodičové sběrnice SPI.

Při návrhu bloku bylo potřeba dbát na nastavení sběrnice SPI pro obvod DS1302. Dle dokumentace [7] je pro zahájení přenosu potřeba nastavit *cs* do log. 1. Obvod přijímá data na náběžnou hranu hodinového signálu *sck* a odesílá data na sestupnou hranu. Data z bloku přichází zakódovaná ve formátu BCD, takže pro ně není potřeba provádět další konverze.

Algoritmus bloku je velmi jednoduchý, každých 100 ms odešle po sběrnici adresu 0xBF, která odpovídá příkazu „Burst Read“. Po odeslání tohoto příkazu blok ze senzoru vyčte 7 bytů dat, které nastaví na výstup.

3.6 Blok UART_simplex

Blok *UART_simplex* se stará o komunikaci se sériovým portem převodníku PL2302, který tato sériová data převádí na protokol USB. Blok posílá data o aktuální teplotě, která pak na PC mohou být zobrazena a zaznamenána pomocí vlastní Python aplikace.



Obrázek 34: Blok UART_simplex

Na vstup *clk* je potřeba zapojit hodinový signál o taktu 50 MHz. Vstup *dht_temp* obsahuje data o teplotě ze senzoru DHT11. Vstup *dht_rhum* obsahuje data o vlhkosti ze senzoru DHT11. Vstup *ds_temp* obsahuje data o teplotě bez desetinného místa ze senzoru DS18B20. Vstup *ds_dp* obsahuje data o desetinném hodnotě tepoty ze senzoru DS18B20. Všechna data ze senzorů jsou zakódována ve formátu BCD. Výstup *tx* odesílá data do sériového portu převodníku.

3.6.1 Algoritmus a impementace

Pro přenos pomocí protokolu UART je potřeba data správně časovat. Při návrhu byl zvolen rámec typu 8N1 o rychlosti 9600 baudů. Tato rychlost znamená, že každý bit v jednom rámci musí trvat 104,16 µs pro správný přenos dat. Každý rámec může odeslat pouze jeden ASCII znak.

V algoritmu je potřeba správně řadit odeslané znaky za sebou, aby data byla na straně PC čitelná. Posílání zpráv s neměnnými znaky je jednoduché, stačí pouze nastavit bity v rámci dle ASCII tabulky. Pro posílání proměnných znaků, v našem případě čísel odpovídajícím naměřeným hodnotám lze využít uspořádání ASCII tabulky. Ke znaku čísla 0 (v ASCII tabulce hodnota 48) stačí přičíst číslo, které chceme přenést, a tím dostaneme správné číslo znaku pro přenos.

Při návrhu bylo rozhodnuto vytvořit co nejjednodušší systém posílání dat, aby strana vysílače a zpracování na PC bylo co nejjednodušší. Z toho důvodu vysílač vysílá jen 11 znaků každou vteřinu. Posílané řádky vypadají následovně:

```
20 15 20.4
```

Obrázek 35: Vyčtená data na sériovém monitoru na PC

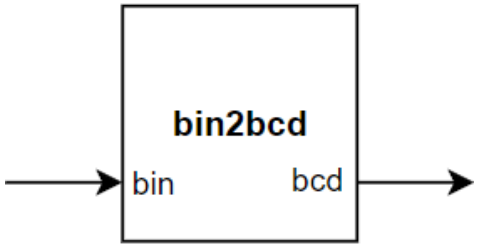
První číslo reprezentuje teplotu ze senzoru DHT11, druhé číslo vlhkost ze senzoru DHT11 a třetí číslo teplotu ze senzoru DS18B20. Za teplotou je znak nového řádku (v tabulce ASCII znak 10).

3.7 Blok binary2bcd

Blok *binary2bcd* se stará o převod binárních dat na data BCD (binary coded decimal). Data je potřeba převádět z důvodu výpisu hodnot na displej. Formát BCD kóduje desítkovou soustavu, kde každá číslice je reprezentována promocí 4 bitů. Příklad hodnot v různých soustavách je zachycen v tabulce 9.

Tabulka 9: Převod mezi dekadickou, binární a BCD soustavou

DEC	BIN	BCD
5	0000 0101	0000 0000 0101
10	0000 1010	0000 0001 0000
23	0001 1101	0000 0010 0011
128	1000 0000	0001 0010 1000



Obrázek 36: Blok převodníku binárních hodnot na BCD

Při realizaci práce bylo použito několik převodníků, protože bloky *UART_simplex* a *lcd1602_controller* potřebují přístup k BCD hodnotám ze senzorů najednou. Díky jednoduchosti vytváření nových instancí komponent ve VHDL byla tato varianta snadnější na implantaci, než používat jeden převodník a vstupní/výstupní data multiplexovat.

3.7.1 Double dabble algoritmus

Blok funguje na principu double dabble algoritmu. Algoritmus je znázorněn na následujícím obrázku:

BCD 2	BCD 1	BCD 0	Binární vstup	
· · · ·	· · · ·	· · · ·	1 0 0 1 0 1 1 1	Načtení vstupu do registru
· · · ·	· · · ·	· · · 1	0 0 1 0 1 1 1 ·	Posun vlevo
· · · ·	· · · ·	· · 1 0	0 1 0 1 1 1 · ·	Posun vlevo
· · · ·	· · · ·	· 1 0 0	1 0 1 1 1 · · ·	Posun vlevo
· · · ·	· · · ·	1 0 0 1	0 1 1 1 · · · ·	Posun vlevo
· · · ·	· · · ·	1 1 0 0	0 1 1 1 · · · ·	+3 (BCD0 > 4)
· · · ·	· · · 1	1 0 0 0	1 1 1 · · · · ·	Posun vlevo
· · · ·	· · · 1	1 0 1 1	1 1 1 · · · · ·	+3 (BCD0 > 4)
· · · ·	· · 1 1	0 1 1 1	1 1 · · · · · ·	Posun vlevo
· · · ·	· · 1 1	1 0 1 0	1 1 · · · · · ·	+3 (BCD0 > 4)
· · · ·	· 1 1 1	0 1 0 1	1 · · · · · · ·	Posun vlevo
· · · ·	1 0 1 0	1 0 0 0	1 · · · · · · ·	+3 (BCD1 > 4), +3 (BCD0 > 4)
· · · 1	0 1 0 1	0 0 0 1	· · · · · · · ·	Posun vlevo

Obrázek 37: Double dabble algoritmus

Na začátku převodu jsou binární data vložena do posuvného registru. Algoritmus probíhá několik cyklů. Na začátku každého cyklu je zkontrolováno, jestli některé BCD číslo je větší jak 4. Pokud ano, tak je k tomuto číslu přičteno číslo 3. Poté je celý posuvný registr posunut vlevo. Algoritmus probíhá tak dlouho, než jsou všechna data z binárního vstupu posunuta do části registru reprezentující BCD hodnotu.

3.7.2 Realizace double dabble algoritmu ve VHDL

Realizace algoritmu pro blok vypadá následovně:

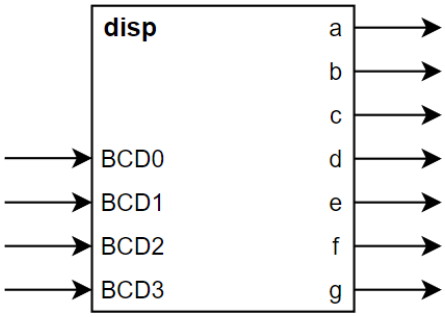
```
process (binary) is

-- VYTVORENI POSUVNEHO REGISTRU
variable temp_reg : unsigned(31 downto 0);

begin

   -- VYNULOVANI REGISTRU
   temp_reg := "00000000000000000000000000000000";

   -- NACTENI BINARNICH DAT DO REGISTRU
   temp_reg(15 downto 0) := binary;

   for i in 0 to 15 loop

      -- KONTROLA BCD3
      if temp_reg(31 downto 28) > "0100" then
         temp_reg(31 downto 28) := temp_reg(31 downto 28) + "0011";
      end if;

      -- KONTROLA BCD2
      if temp_reg(27 downto 24) > "0100" then
         temp_reg(27 downto 24) := temp_reg(27 downto 24) + "0011";
      end if;

      -- KONTROLA BCD1
      if temp_reg(23 downto 20) > "0100" then
         temp_reg(23 downto 20) := temp_reg(23 downto 20) + "0011";
      end if;

      -- KONTROLA BCD0
      if temp_reg(19 downto 16) > "0100" then
         temp_reg(19 downto 16) := temp_reg(19 downto 16) + "0011";
      end if;

      -- POSUN REGISTRU VLEVO
      temp_reg(31 downto 1) := temp_reg(30 downto 0);
      temp_reg(0) := '0';

   end loop;

   bcd <= temp_reg(31 downto 16);

end process;
```

Obrázek 38: Realizace double dabble algoritmu ve VHDL

Proces převodu se spustí na změnu vstupních binárních dat (označených *binary)*. Poté je vytvořen 32bitový posuvný registr označený jako *temp_reg*. Registr je před každým převodem vynulován a do posledních 16 bitů je vložena vstupní binární hodnota. Pomocí ***for loop*** cyklu se 16× zopakuje blok kódu, kde se nejprve zkontroluje, zda-li nějaká hodnota BCD číslice je větší jak 4. Pokud ano, přičte se k této číslici číslo 3. Na konci tohoto bloku dojde k posunutí registru vlevo. Po skončení ***for loop*** cyklu jsou data reprezentující BCD číslice přivedena na výstup označený *bcd*.

3.8 Blok disp (bcd2seg)

Blok *disp* zajišťuje komunikaci se 7 segmentovým displejem. Dle vstupních BCD dat rozsvěcí příslušné segmenty displeje a zobrazuje tak číslice. Pro každou číslici displeje je potřeba blok, který ji obsluhuje.



Vstupy *BCD 0* až *BCD 3* jsou jednotlivé bity příslušné BCD číslice. Výstupy *a* až *g* odpovídají segmentům na 7 segmentovém displeji.

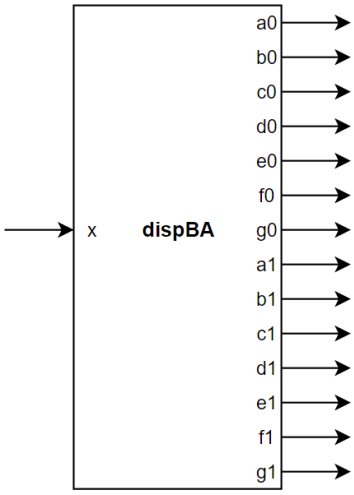
VHDL kód je realizovaný v tzv. paralelním prostředí, kde se všechny řádky realizují najednou. Jednotlivým výstupům jsou přiřazovány logické hodnoty dle výsledku Booleovské algebry. Kód je přezván z [1], upraven.

```
architecture Dataflow of bcd2seven is

   signal x1, x2, x3: std_logic;

begin

   x1 <= not(BCD0 and not BCD1 and not BCD2 and not BCD3);
   x2 <= not(not BCD0 and not BCD1 and BCD2);
   x3 <= not(BCD0 and BCD1 and BCD2);

   a <= not(x1 and x2);
   b <= not(not(BCD0 and not BCD1 and BCD2) and not(not BCD0 and BCD1 and BCD2));
   c <= not(not(not BCD0 and BCD1 and not BCD2));
   d <= not(x2 and x3 and x1);
   e <= not(not BCD0 and not(not BCD1 and BCD2));
   f <= not(not(BCD0 and BCD1) and not(BCD0 and not BCD2 and not BCD3)
        and not(BCD1 and not BCD2 and not BCD3));
   g <= not(not(not BCD1 and not BCD2 and not BCD3) and x3);

end Dataflow;
```

Obrázek 39: BCD dekodér na vstup 7 segmentového displeje ve VHDL

3.9 Blok dispBA (disp_special_character)

Blok *dispBA* zajišťuje komunikaci se dvěma 7 segmentovými displeji. Blok zajišťuje zobrazování zvláštních znaků (°C a rH) na displejích.

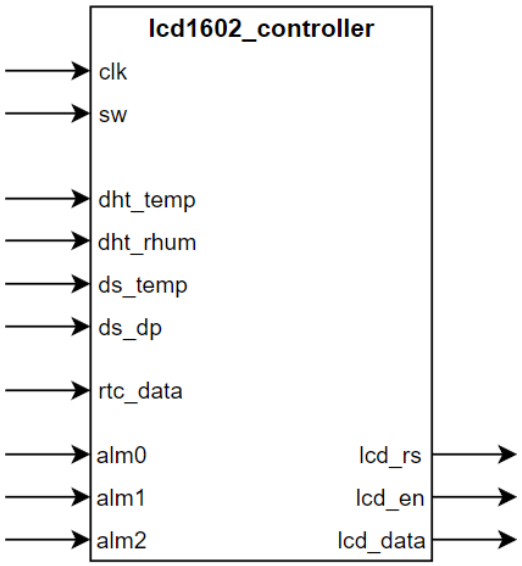


Obrázek 40: Blok zajišťující zobrazování speciálních znaků

Vstup *x* je binární vstup, dle kterého se rozhoduje, jaké znaky se na displeji zobrazí. Pokud *x* = 0, budou zobrazeny znaky „°C“. Při *x* = 1 se na displeji zobrazí „rH“. Blok funguje velmi podobně jako blok předchozí, také je navržen v paralelním prostředí.

3.10 Blok lcd1602_controller

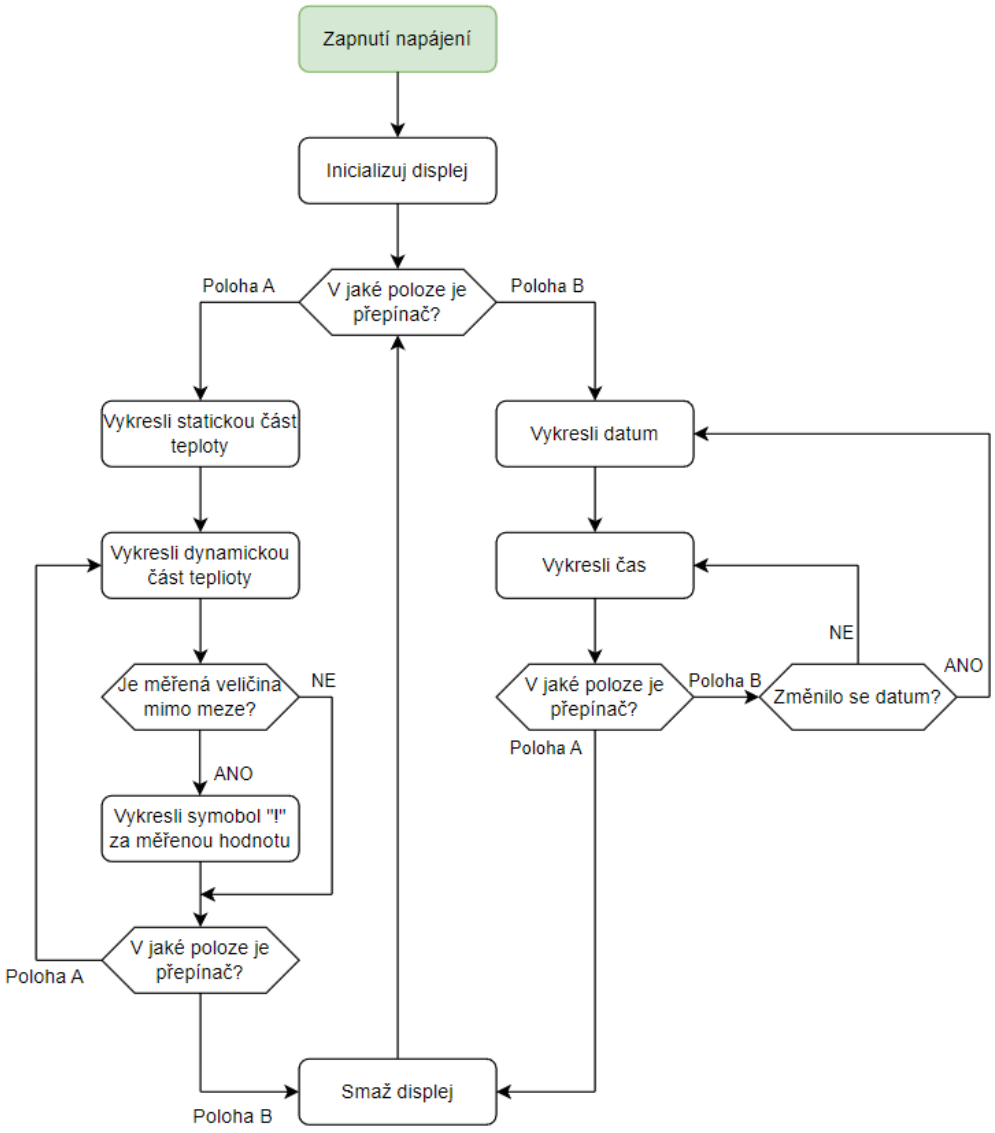
Blok *lcd1602_controller* zajišťuje komunikaci se znakovým displejem. Blok posílá instrukce pro výpis naměřených hodnot nebo aktuálního času na displej.



Obrázek 41: Blok lcd1602_controller

Na vstup *clk* je přiveden hodinový signál o taktu 1 MHz. Vstup *sw* slouží pro připojení přepínače, podle kterého si uživatel může vybrat, zda zobrazit na displeji teplotu, nebo čas. Vstupy *dht_temp* až *ds_dp* slouží pro připojení dat, které obsahují informace o teplotě v BCD formátu. Vstup *rtc_data* obsahuje informace o čase a datu z obvodu DS1302. Vstupy *alm0* až *alm2*, jsou binární vstupy, které indikují, zda je některá z měřených veličin mimo nastavenou hodnotu hlídání. Výstupy *lcd_rs*, *lcd_en* a *lcd_data* slouží pro komunikaci se znakovým displejem.

Algoritmus bloku je popsán na následujícím obrázku:



Obrázek 42: Algoritmus bloku lcd1602_controller

Po zapnutí napájení dojde k inicializaci displeje (dle kapitoly 2.7.4) v 8bitovém režimu. Dle polohy přepínače uživatel rozhoduje, zda-li chce vypsat na displej teplotu nebo čas. Pokud je přepínač v poloze A, dojde k vypsání teploty. Pro optimalizaci výpisu a redukci blikání displeje

je text rozdělen do „statické“ a „dynamické“ části. Statická část textu je ta, kterou stačí vypsat jen jednou, a není potřeba ji aktualizovat (popisky). Dynamická část je ta, kterou je potřeba aktualizovat často (měřené hodnoty). Na obr. 43 je zachyceno rozložení znaků, které se zobrazují při měření teploty. Černě jsou vyznačeny statické znaky, červeně dynamické znaky.

D	H	T	:		2	0	°	C	!	1	5	%	!		
D	S	1	8	:		2	0	.	5	°	C	!			

Obrázek 43: LCD displej v režimu zobrazování teploty

Pokud uživatel přepne přepínač, dojde k vymazání displeje a dle polohy přepínače se začne vykreslovat teplota nebo čas. U data a času nejsou žádné statické popisky, ale tam lze využít faktu, že datum se mění jednou za 24 hodin. Stačí tedy jen aktualizovat zobrazený čas, dokud se aktuální datum nezměnilo.

3.11 PC aplikace

Aplikace pro počítače komunikuje s FPGA po UART sériové lince. Pro návrh aplikace byl vybrán programovací jazyk Python, z důvodu velkého množství volně přístupných modulů ke stažení a možnosti kompilace aplikace pro platformy Windows, MacOS a Linux. Aplikace umožňuje ukládat naměřené hodnoty do textového souboru, a vykreslovat grafy přijatých dat v reálném čase.

3.11.1 Komunikace se sériovým portem a zpracování přijatých dat

Pro komunikaci se sériovým portem byla zvolena knihovna PySerial. Jednoduchý kód pro komunikaci po sériové lince je naznačen na obrázku 44.

```
import serial # vložení knihovny PySerial

# otevření portu COM1 s přenosovou rychlostí 9600 baudů
serialInstance = serial.Serial("COM1", 9600)

# nekonečná smyčka
while True:
    # pokud jsou data na portu
    if serialInstance.in_waiting:
        packet = serialInstance.readline()
```

Obrázek 44: Komunikace se sériovou linkou v jazyce Python

Výše uvedený kód pomocí knihovny PySerial otevře sériovou linku označenou *COM1* s přenosovou rychlostí 9600 baudů. Kód pak čeká na nová data a data vyčte.

Přijatá data je potřeba zpracovat. Existující kód o tuto funkcionalitu můžeme snadno rozšířit viz. obrázek 45.

```
import serial # vložení knihovny PySerial

# otevření portu COM1 s přenosovou rychlostí 9600 baudů
serialInstance = serial.Serial("COM1", 9600)

# listy přijatých dat
dht_temp = []
dht_rhum = []
ds18_temp = []

# nekonečná smyčka
while True:
    # pokud jsou data na portu
    if serialInstance.in_waiting:
        packet = serialInstance.readline()

        # dekodování přijatých dat, odstranění znaku nového řádku
        # oddělení jednotlivých hodnot mezerou
        data_parsed = packet.decode('ascii').rstrip('\n').split()

        # zařazení nových dat do listu, přetypování přijatých dat
        dht_temp.append(int(data_parsed[0]))
        dht_rhum.append(int(data_parsed[1]))
        ds18_temp.append(float(data_parsed[2]))
```

Obrázek 45: Zpracování přijatých dat

Kód je rozšířen o deklaraci listů, do kterých budeme přijatá data vkládat. Nově na přijaté řádky voláme funkce ***decode()***, ***rstrip()***, ***split()***. Funkce ***decode()*** vybírá podle kterého znakového setu jedlotlivé bity dekódovat (v našem případě ASCII). Funkce ***rstrip()*** odstraňuje z přijatých dat znak nového řádku. ***Split()*** rozděluje přijatý textový řetězec do listu podle mezer. Rozdělená data jsou poté přiřazena do příslušných listů pomocí funkce ***append()***. Data jsou přetypována na typ integer nebo float z důvodu že takto přijatá data by mohla být interpretována jako textový řetětzec typu string.

3.11.2 Ukládání přijatých dat

Pro funkci logování dat lze rozšířit stávající kód. Nejprve je vhodné importovat knihovnu datetime, která vyčítá čas operačního systému. Pomocí funkce ***open()*** se otevře textový soubor *log.txt* v režimu úprav. Pomocí ***write()*** se do souboru zapíše textový řetězec s aktuálním časem a naměřenou teplotou přijatou po sériové lince. Po zapsání soubor se soubor zavře funkcí ***close()***.

```
# import knihovny datetime
from datetime import datetime

# otevření souboru log.txt v režimu úprav
with open("log.txt", "a") as file:
    file.write(datetime.today().strftime('%H:%M:%S') + "DHT11: " + data_parsed[0] + " °C")
    file.close()
```

Obrázek 46: Rozšíření o zápis naměřené teploty do textového souboru

3.11.3 Tvorba uživatelského rozhraní

Pro zobrazování dat pomocí uživatelského rozhraní byla zvolena knihovna Tkinter. Pro tvorbu okna můžeme využít funkci ***tkinter.Tk()***. Do vytvořeného okna pak můžeme vkládat celou škálu prvků. Níže jsou vypsané prvky použité v práci:

- ***tkinter.Button()*** – tlačítko
- ***tkinter.Checkbutton()*** – zaškrtávací tlačítko
- ***tkinter.Label()*** – text
- ***tkinter.LabelFrame()*** – ohraničené textové pole s nadpisem
- ***tkinter.OptionMenu()*** – seznam

Na prvky lze aplikovat širokou škálu funkcí, které mění jejich vzhled, popis, velikost atd.

3.11.4 Kreslení grafů

Knihovna Tkinter neumí v základním balíčku vykreslovat grafy. Pro kresbu grafů je potřeba doinstalovat knihovnu Matplotlib. Do listů hodnot *x* a *y* jsme vložili body, které chceme vykreslit. Pomocí funkce ***plot()*** graf s námi zadanými body vykreslíme. Funkce ***xlabel()*** a ***ylabel()*** umožňují nastavení popisků os. Funkce ***show()*** zobrazí graf uživateli. Pomocí dalších funkcí lze měnit typ grafu, barvy vykreslených čar, tloušťku čar atd.

```
import matplotlib.pyplot as plt

x = [1,2,3,4]
y = [2,4,6,8]

plt.plot(x, y)

plt.xlabel("osa x")
plt.ylabel("osa y")

plt.show()
```

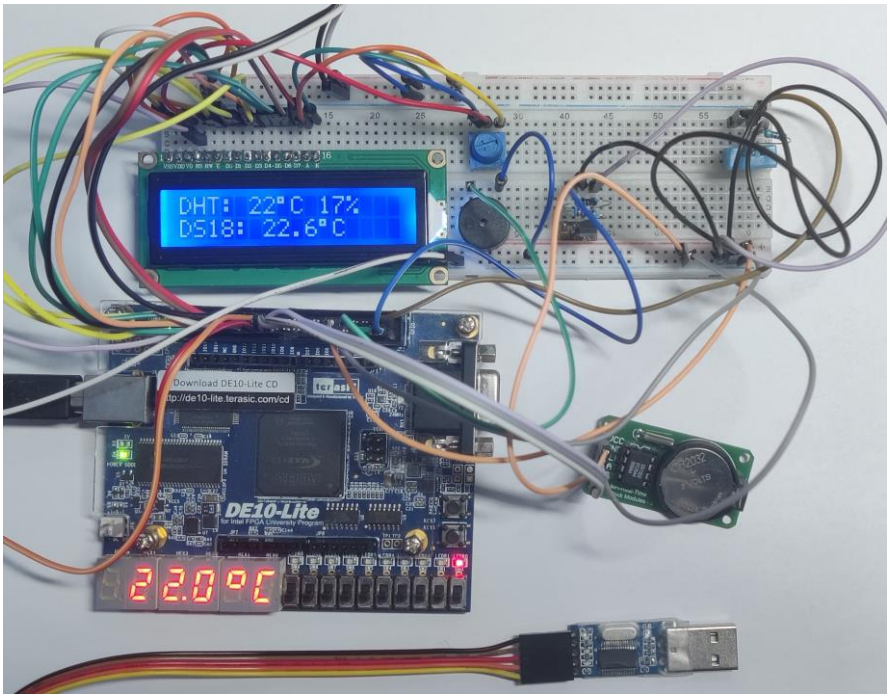
Obrázek 47: Kreslení grafů pomocí knihovny Matplotlib

V práci je potřeba kreslit v reálném čase podle příchozích dat. Bohužel knihovna Matplotlib toto neumožňuje. Jediným řešením, jak aktualizovat graf, je jeho smazáním a vykreslením nového na stejném místě.

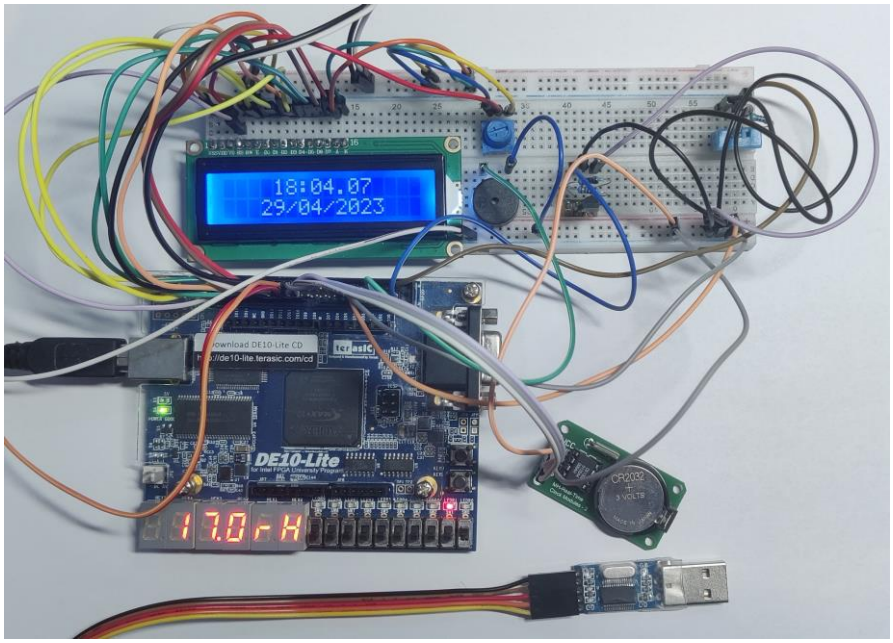
3.12 Realizovaná práce

Na obrázcích 48, 49 a 50 je výsledek realizované práce. Naprogramované hradlové pole získává data o teplotě a vlhkosti ze senzoru DHT11, a data o teplotě ze senzoru DS18B20. Data zobrazuje na 7 segmentovém displeji a znakovém LCD displeji. Pomocí přepínače lze na LCD displeji zobrazit čas, získaný z obvodu DS1302. Pomocí tlačítek a přepínačů lze měnit údaj

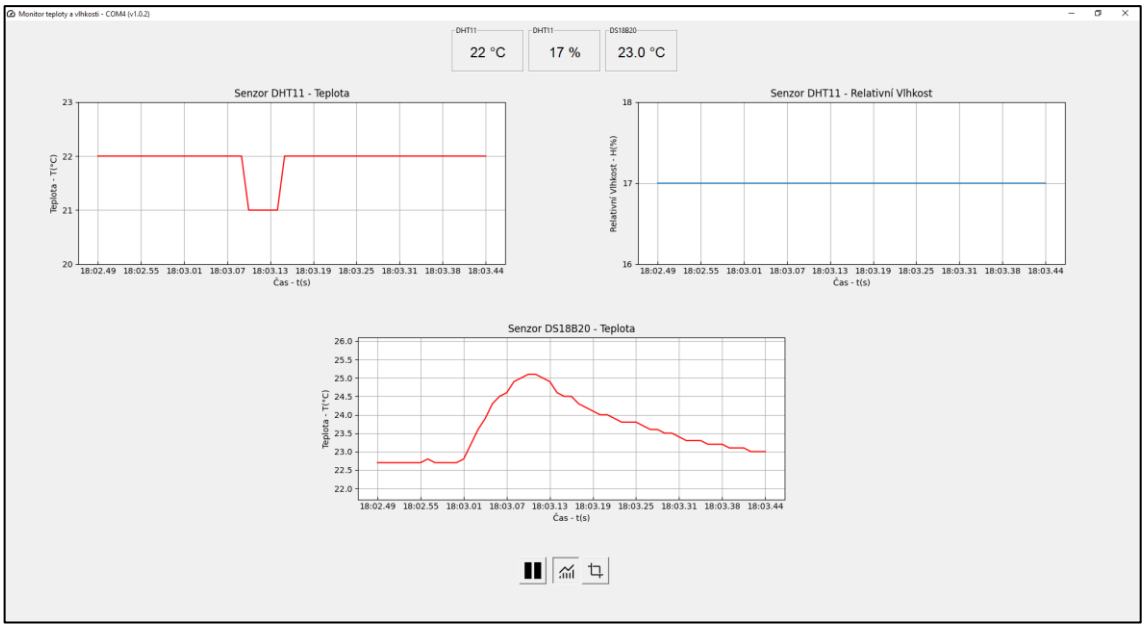
zobrazovaný na 7 segmentovém displeji a nastavovat hlídaný rozsah měřených hodnot. V případě vychýlení naměřených hodnot od hodnot požadovaných dojde ke spuštění zvukové signalizace. FPGA odesílá naměřená data po sériové sběrnici, která můžou být ukládána a graficky zobrazována pomocí aplikace na počítač.



Obrázek 48: Výsledná práce, měření teploty



Obrázek 49: Výsledná práce, zobrazení času, měření vlhkosti



Obrázek 50: Výsledná práce, PC aplikace

3.12.1 Ovládání práce

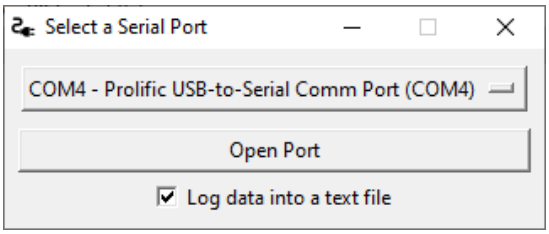
Uživatel může ovládat práci pomocí přepínačů. Pomocí přepínače *SW0* může uživatel přepínat mezi zobrazením teploty nebo času na LCD displeji. Pomocí tlačítka *KEY0* lze přepínat zobrazovanou hodnotu na 7 segmentovém displeji. Zobrazované hodnoty jsou v následujícím pořadí:

1. Teplota ze senzoru DHT11
2. Relativní vlhkost ze senzoru DHT11
3. Teplota ze senzoru DS18B20
4. Nastavení meze pro teploty ze senzoru DHT11
5. Nastavení meze pro vlhkost ze senzoru DHT11
6. Nastavení meze pro teplotu ze senzoru DS18B20

Vybraná hodnota je signalizována rozsvícením příslušné LED diody označené *LED0 – LED5*. Pokud uživatel zobrazuje hodnotu pro nastavení meze, může přepínačem *SW9* vybírat mezi spodní a horní mezí. Přepínáním přepínače *SW8* lze zvolit, zda-li se při nastavování meze bude přičítat nebo odečítat. Při stisknu tlačítka *KEY1* se zobrazená mez zvýší/sníží (dle polohy přepínače *SW8)* o jedna.

3.12.2 Ovládání PC aplikace

Po spuštění aplikace je uživatel vyzván k vybrání sériového portu, ke kterému je připojeno FPGA. Navíc si může vybrat, zda-li chce ukládat naměřené hodnoty do textového souboru.



Obrázek 51: Úvodní okno PC aplikace

V hlavním okně aplikace se nachází tři ovládací tlačítka. První tlačítko umožňuje pozastavit/obnovit vykreslování grafů v realném čase. Pokud je vykreslování grafů pozastavené, uživatel může grafy přibližovat/oddalovat, posouvat je, nebo je ukládat. Druhým tlačítkem uživatel vybere, že chce zobrazit v grafech všechna naměřená data. Třetím tlačítkem lze vybrat vykreslovaní posledních 100 hodnot v grafech.

4. Závěr

V této bakalářské práci byl zcela navržen a realizován monitor teploty a vlhkosti na přípravku DE10-Lite. Vytvořené VHDL kódy umožňují komunikaci přípravku se senzory teploty DHT11 a DS18B20. Naměřené hodnoty jsou zobrazovány na sedmi segmentovém displeji a znakovém LCD displeji. Uživatel si může nastavit rozsah povolených měřených hodnot. V případě, že se měřená hodnota nachází mimo nastavený rozsah, uživatel je upozorněn zvukovou signalizací. Nad rámec zadání práce se ve VHDL podařilo implementovat komunikaci s obvodem DS1302 pro udržování reálného času a data a jejich výpis na LCD displej. Dále se podařilo implementovat posílání naměřených hodnot do počítače, kde si uživatel může ukládat naměřené hodnoty a sledovat jejich průběh na grafech díky vytvořené aplikaci v programovacím jazyce Python.

Práce by se v budoucnu dala rozšířit o posílání dat na internet a o připojení dalších senzorů měřících fyzikální veličiny. Další rozšíření práce by se mohlo týkat lepšího ovládání pro výběr zobrazené veličiny a pro snazší nastavování rozsahů měřených veličin.

Vytvořené VHDL kódy by mohly být po mírných úpravách použity v jiných projektech pro komunikaci se zařízeními podporující sběrnice použité v práci. Seznam vytvořených kódů je v příloze B. Práce nad rámec splnila zadání.

Použitá literatura

[1] LAFATA, Pavel, Petr HAMPL a Michal PRAVDA. Digitální technika. V Praze: České vysoké učení technické, 2011. ISBN 978-80-01-04914-3.

[2] DE10-Lite User Manual [online]. Terasic, 2020 [cit. 2022-12-05]. Dostupné z: https://www.terasic.com.tw/cgibin/page/archive_download.pl?Language=English&No=1021&FID=a13a2782811152b4 77e60203d34b1baa

[3] LINKE, Bernhard. Guide to 1-Wire communication [online]. Maxim Integrated, 2008 [cit. 2022-12-05]. Dostupné z: https://pdfserv.maximintegrated.com/en/an/Overview_1wire_Technology_use.pdf

[4] DS18B20 – Programmable Resolution 1-Wire Digital Thermometer [online]. Maxim Integrated, 2019 [cit. 2022-12-06]. Dostupné z: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

[5] DHT11 Humidity & Temperature Sensor [online]. Aosong ElectronicsCo. Ltd. [cit. 2023-01-15]. Dostupné z: https://www.mouser.com/datasheet/2/758/DHT11- Technical-Data-Sheet-Translated-Version-1143054.pdf

[6] HD44780U [online]. Hitachi, 1999 [cit. 2023-04-29]. Dostupné z: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

[7] DS1302 Trickle-Charge Timekeeping Chip [online]. Maxim Integrated, 2015 [cit. 2023-04-29]. Dostupné z: https://www.analog.com/media/en/technical-documentation/data-sheets/DS1302.pdf

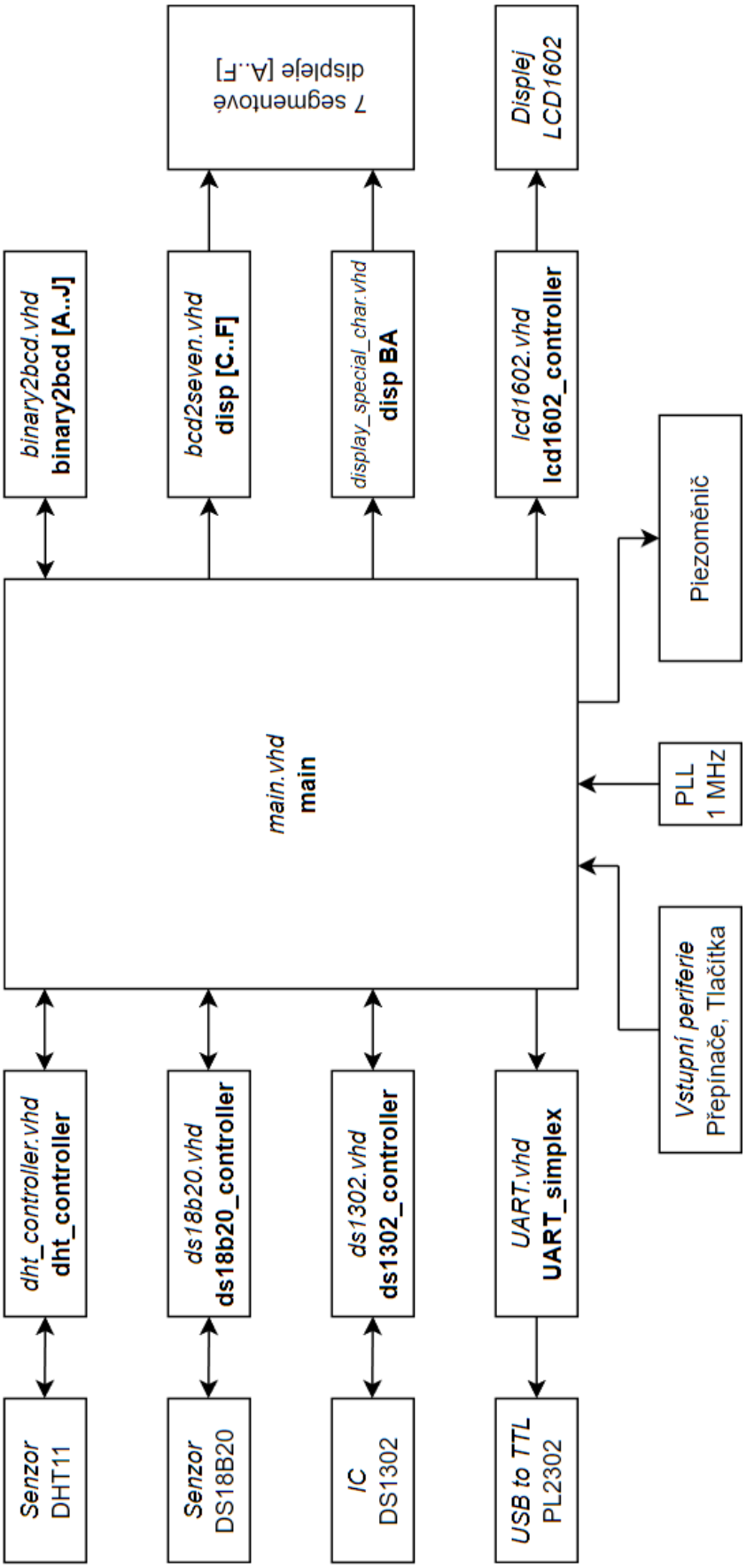
[8] PL-2303HX Edition USB to Serial Bridge Controller [online]. Prolific Technology, 2013 [cit. 2023-04-29]. Dostupné z: https://www.prolific.com.tw/UserFiles/files/ds_pl2303HXD_v1_4_4.pdf

[9] Python 3.11.3 documentation [online]. Python Software Foundation, 2023 [cit. 2023-04-29]. Dostupné z: https://docs.python.org/3/

Seznam použitých zkratek

- **ACK** (acknowledgement) – potvrzení
- **ASCII** (american standard code for information interchange) – typ znakové tabulky
- **BCD** (binary coded decimal) – binárně zakódována desítková soustava
- **BIN** (binary) – binární soustava/data
- **CS** (chip select) – typ vodiče v SPI komunikaci
- **DEC** (decimal) – dekadická soustava
- **FPGA** (field programmable gate array) – programovatelné hradlové pole
- **GND** (ground) – elektrická zem
- **HDL** (hardware description language) – popisovací jazyk
- **I^2C** (inter-integrated circuit) – typ sériové sběrnice
- **LCD** (liquid crystal display) – označení typu displeje
- **PC** (personal computer) – osobní počítač
- **PLL** (phase locked loop) – fázový závěs
- **RAM** (random access memory) – volatilní paměť
- **RX** (reciever) - příjmač
- **SCK, SCL** (serial clock) – seriový hodinový signál
- **SDA** (serial data) – sériová data
- **SPI** (seriál peripheral interface) – typ sériové sběrnice
- **LSB** (lowest significant bit) – bit s nejnižší váhou
- **Log.** – logický
- **MISO** (master in slave out) – vstup mastera, výstup slave
- **MSB** (most significant bit) – bit s největší váhou
- **MOSI** (master out slave in) – výstup mastera, vstup slave
- **TX** (transmitter) - vysílač
- **VCC** – elektrické napájení
- **VHDL** (VHSIC hardware description language) – název jazyka pro popis hardwaru
- **VHSIC** (very high speed integrated circuit) – velmi rychlý integrovaný obvod
- **UART** (universal asynchronous receiver-transmitter) – univerzální asynchronní příjmač-vysílač, název typu sběrnice
- **USB** (universal serial bus) – typ sériové sběrnice

Příloha A Blokové schéma práce



Příloha B Seznam přiložených VHDL kódů

- main.vhd
- bcd2seven.vhd
- binary2bcd.vhd
- dht_controller.vhd
- disp_controller.vhd
- display_special_character.vhd
- ds18b20.vhd
- ds1302.vhd
- UART_simplex.vhd
- pll_1MHz.vhd