Výukový model vědecké kalkulačky

Mikuláš Kozubík

Bakalářská práce
2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav automatizace a řídicí techniky

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Mikuláš Kozubík**
Osobní číslo:	**A17573**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**Prezenční**
Téma práce:	**Výukový model vědecké kalkulačky**
Téma práce anglicky:	**An Educational Model of a Scientific Calculator**

Zásady pro vypracování

1. Popište existující vědecké kalkulačky a jejich možnosti.
2. Proveďte návrh kalkulačky řízené mikropočítačem s ohledem na její využití jako výukové pomůcky pro výuku programování mikropočítačů.
3. Návrh hardwarově realizujte.
4. Implementujte ukázkové programové vybavení pro řídicí mikropočítač.
5. Vytvořte programovou knihovnu pro usnadnění tvorby vlastního software pro vytvořenou kalkulačku.

Rozsah bakalářské práce:
Rozsah příloh:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. BARR, Michael a Anthony J. MASSA. Programming embedded systems: with C and GNU development tools. 2nd ed. Sebastopol: O'Reilly, 2006. ISBN 0596009836.
2. CATSOULIS, John. Designing embedded hardware. 2nd ed. Sebastopol, CA: O'Reilly, 2005, xvi, 377 p. ISBN 0596007558.
3. LADMAN, Josef. Elektronické konstrukce pro začátečníky. Praha: BEN – technická literatura, 2001. ISBN 80-730-0015-6.
4. PINKER, Jiří. Mikroprocesory a mikropočítače. Praha: BEN – technická literatura, 2004. ISBN 80-7300-110-1.
5. SMITH, Warwick A. C programming for embedded microcontrollers. 2nd ed. Susteren: Elektor International Media BV, 2008. ISBN 978-090-5705-804.

Vedoucí bakalářské práce: **Ing. Jan Dolinay, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **20. prosince 2019**
Termín odevzdání bakalářské práce: **15. května 2020**

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Ve Zlíně dne 20. prosince 2019

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor;
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 12. 08. 2020 …….

podpis diplomanta

ABSTRAKT

Cílem této práce je stručný úvod do historie vědeckých kalkulaček a jejich vnitřního řešení. Dále pak návrh studijního modelu kalkulačky včetně programového vybavení a knihoven, které obsluhují zkompletovaný hardware. Hardware byl sestaven z vývojové desky FRDM KL25Z obsahující mikroprocesor pro řízení kalkulačky, dvou maticových klávesnic pro uživatelský vstup a tří čtyřmístných 7segmentových displejů pro zobrazování. Sestavená kalkulačka s modelovým programem pro obsluhu používá pro zadávání reverzní polskou notaci, což usnadňuje programování a zároveň ukazuje alternativní metodu vstupu dat. Tento model může sloužit pro seznámení studentů s principy funkcí moderních vědeckých kalkulaček a výuce implementací výpočetních algoritmů.

Klíčová slova: Vědecká kalkulačka, KL25Z, CORDIC, Mikropočítač, Programová knihovna

ABSTRACT

The goal of this thesis is a brief introduction to the history and inner workings of scientific calculators and a working educational model of one. This model includes a demonstration code and libraries, which control the hardware. The hardware consists of a FRDM-KL25Z control board, which includes a microprocessor controlling the calculator, two matrix switches for user input, and three four-digit seven-segment LED displays. The calculator uses the reverse Polish notation, which is easier to implement and also shows an alternative method of input. This model can introduce students to the principles of inner workings of modern scientific calculators and help with teaching of different computational algorithms used in calculators.

Keywords: Scientific calculator, KL25Z, CORDIC, Microcontroller, Code library

Děkuji vedoucímu mé práce, panu Ing. Janu Dolinay, Ph.D., za trpělivost a nedocenitelné rady při návrhu modelu a vypracování mé práce. Děkuji také panu prof. Ing. Vladimíru Vaškovi, CSc., za pomoc a vstřícný přístup při řešení administrativních problémů. V neposlední řadě děkuji mým rodičům a příteli za trpělivost a podporu při práci.

OBSAH

ÚVOD

Pomůcky pro usnadnění matematických výpočtů jsou nedílnou součástí moderní civilizace. Potřeba rychle, přesně a jednoduše provádět složité matematické operace je přítomna v široké škále lidských činností.

Ačkoliv tato potřeba v dávné historii nebyla tak silná, jako v posledních staletích, lidé neustále toužili po jisté míře usnadnění počítání, a tak se po tisíciletí snažili vynalézat různá počítadla. Velké přelomy ale vždy přicházely až po hlubším porozumění matematickým principům.

Umění práce s kalkulačkou je důležitou zkušeností, bez které je život v moderní době relativně obtížný, na rozdíl od vědomostí o vnitřním fungování a limitech těchto přístrojů. Málokdo dnes bohužel tuší, jaké algoritmy pohánějí elektronické kalkulačky a velké množství dostupných informací je buďto velmi zjednodušených, nebo přímo nepravdivých. Zvídavý člověk se tak k přesným informacím dostává jen obtížně.

Jedním z vedlejších cílů této práce je seznámit čtenáře s problematikou funkce vědeckých kalkulaček a nasměrovat jeho pátrání, pokud by se chtěl dozvědět více. Bez vědomostí alespoň základních pojmů je vyhledávání další odborné literatury náročné.

Hlavním cílem této práce je pak návrh, sestavení a naprogramování modelu vědecké kalkulačky tak, aby mohl student jednoduše zkoušet implementaci různých algoritmů, případně metod zadávání dat do kalkulačky.

Výstupem praktické části je pak programová knihovna obsluhující sestavený hardware, od čtení vstupů z klávesnice, přes odesílání čísel k zobrazení na displeji přes I2C sběrnici až po práci s registry virtuální kalkulačky. Jako ukázky jsou do programové knihovny zahrnuty i programy pro základní výpočetní operace.

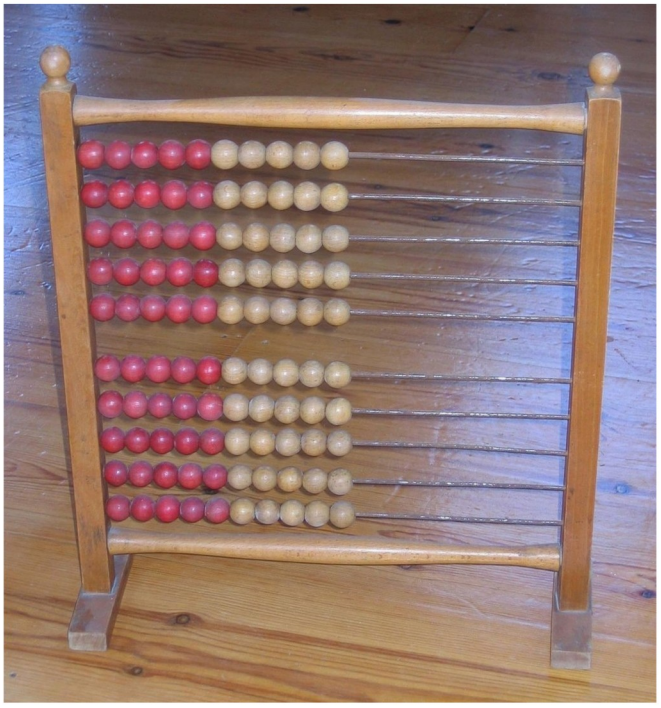
I. TEORETICKÁ ČÁST

1 HISTORIE KALKULAČNÍCH POMŮCEK

Lidé se snažili si náročné počítání usnadnit již v dávných dobách. Obchodníci, architekti a samozřejmě i vědci a učenci bezpochyby nechtěli trávit svůj čas repetitivním počítáním a hledali tak způsoby usnadnění své práce.

1.1 Abakus

První důkazy o abaku se objevily ve třetím tisíciletí před našim letopočtem, v Mezopotámii. Původní Sumerský abakus měl podobu nakreslené či vyryté tabulky, která oddělovala jednotlivé řády jejich šedesátkové číselné soustavy. Počítání bylo prováděno pomocí malých kuliček či válečků. [1]



Obr. 1: Moderní abakus

1.2 Logaritmické pravítko

Větší přelom přišel až v 17. století s objevem konceptu logaritmů a vynálezem logaritmického pravítka. S jeho pomocí se dá snadno násobit, dělit, počítat mocniny a odmocniny nebo exponenciální funkce.

Ačkoliv je logaritmické pravítko mocnějším nástrojem než abakus, je spíše jeho doplněním než jeho náhradou, jelikož na logaritmickém pravítku nelze sčítat a odečítat.

Principem výpočtů na logaritmickém pravítku je sčítání logaritmů operandů, což je ve výsledku součin či podíl původních operandů.[2]

Obr. 2: Logaritmické pravítko

1.3 Mechanické kalkulačky

Dalším krokem v automaticaci matematických výpočtů byl příchod mechanických kalkulaček. První skutečně mechanickou kalkulačku sestrojil už v roce 1642 Blaise Pascal, avšak

komerční výroba mechanických kalkulaček se datuje až do druhé poloviny 19. století. Za zmínku stojí kalkulačka Curta, která se začala vyrábět v roce 1948. Jednalo se o velice kompaktní kalkulačku, která se vešla do dlaně.[3]

Obr. 3: Mechanická kalkulačka Curta (Zdroj: Valentine, N., 2019 [3])

1.4 Elektronické kalkulačky

Elektronické kalkulačky mají svou historii svázánu s příchodem prvních elektronkových počítačů ve 40. letech 19. století. Jak se v průběhu let elektronika miniaturizovala, výrobci byli schopni vyrábět menší a menší počítače, které ve své době byly vlastně objemnými kalkulačkami.

První vědeckou kapesní kalkulačku začala vyrábět společnost Hewlett-Packard v roce 1972. Jednalo se o model HP-35, který dokázal počítat trigonometrické a algebraické funkce. V této době začala vývojová válka mezi společnostmi Hewlett-Packard a Texas Instruments, které každých několik měsíců na trh vydaly nový model vědecké, nebo programovatelné, kapesní kalkulačky, který byl levnější, výkonnější, nebo obojí.

1.4.1 Kalkulačka HP-35

Kalkulačka HP-35, představená v roce 1972, byla první kapesní vědeckou kalkulačkou uvedenou na trh.

Obr. 4: Hewlett-Packard HP-35 (Zdroj: Valentine, N., 2019 [3])

Tato kalkulačka používala postfixní notaci, jako její následovníci z dílen HP, a kromě čtyř základních funkcí umožňovala i výpočty goniometrických funkcí, logaritmů, odmocniny, umocnění a obsahovala základní paměť pro mezivýpočty.

2 VNITŘNÍ FUNGOVÁNÍ ELEKTRONICKÝCH KALKULAČEK

Zatímco sčítání, potažmo odečítání a do jisté míry i násobení a dělení jsou jednoduché operace, snadno implementovatelné na TTL či CMOS logice, trigonometrické, logaritmické a jiné výpočty již mnohdy nelze na primitivním hradlovém poli jednoduše řešit. Výrobci a návrháři procesorů vědeckých kalkulaček tak musejí řešit problém, jak tyto operace implementovat s co možná nejmenší hardwarovou náročností

2.1 Taylorův rozvoj

Jedním z intuitivních řešení by mohlo být použití Taylorova rozvoje pro aproximaci složitějších funkcí:

$$f(x)=f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f'''(a)}{3!}(x-a)^3+\ldots=\sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}(x-a)^n$$

Pomocí předem vyřešených sérií pro každou funkci lze na primitivním hardware implementovat výpočty mnoha funkcí. Přesnost takové aproximace záleží na délce předem vyřešené série.

Jedním z problémů tohoto řešení je fakt, že Taylorova řada pro mnoho funkcí konverguje jen velice pomalu, což by způsobovalo neúnosnou časovou náročnost výpočtů na jednodušších (a pro některé funkci i na složitých a výkonných) procesorech. Tento problém je pak dále umocněn tím, že operace násobení, dělení a umocňování jsou pro implementaci na hradlových polích velice náročné na počet hradel, čímž roste i cenová náročnost takového řešení.

Větším problémem je ovšem fakt, že Taylorův rozvoj pro některé funkce konverguje pouze na omezených intervalech (například pro funkci logaritmu je Taylorův rozvoj použitelný pouze na intervalu (-1; 1>) a pro takové funkce by se implementace musela lišit například i podle zadaného čísla. Netřeba říkat, že takovéto řešení není pro účely výroby kalkulaček vhodné.

2.2 Algoritmy CORDIC

Z důvodu nevhodnosti použití Taylorových sérií a jiných aproximací pro výpočty pokročilejších funkcí (zpočátku hlavně goniometrických funkcí) na primitivním hardware své doby, byl v 50. a 60. letech vyvinut algoritmus CORDIC (**Co**ordinate **R**otation **Di**gital **C**omputer). Matematické techniky, na kterých je CORDIC založen, se datují již do první poloviny 17. století, avšak jejich implementace na jednoduchých CPU byla započata až s potřebou digitalizace analogových počítačů v navigačních systémech a radarech v 60. letech 20. století. [4]

Během let byly vyvinuty algoritmy, které dokáží na podobném principu řešit i jiné funkce, jako jsou inverzní goniometrické funkce, hyperbolické, logaritmické, exponenciální funkce, odmocniny, ale i násobení a dělení, které se tak na hradlových polích zjednodušilo. Všechny tyto algoritmy byly zahrnuty pod název CORDIC.

Algoritmy CORDIC pracují pouze se sčítáním a bitovými posuny a k výsledku konvergují velice rychle. Přesnost výsledku se zvyšuje zhruba o jeden řádový bit pro každou iteraci algoritmu. [5]

Původní algoritmus počítal současně funkce sin(x) a cos(x). Jeho princip spočívá v rotaci jednotkového vektoru

$$v_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

o zadaný úhel, souřadnice výsledného vektoru jsou pak přímo výsledkem funkcí sinus a kosinus pro zadaný úhel.

2.2.1 Původní rotační algoritmus

Matice rotace vektoru je:

$$R_i = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

Můžeme použít goniometrické identity:

$$\cos(\alpha)=\frac{1}{\sqrt{1+\tan^2(\alpha)}}$$

a

$$\sin(\alpha)=\frac{\tan(\alpha)}{\sqrt{1+\tan^2(\alpha)}}$$

Matici rotace pak můžeme upravit na:

$$R_i=\frac{1}{\sqrt{1+\tan^2(\alpha)}}\begin{pmatrix}1 & -\tan(\alpha)\\ \tan(\alpha) & 1\end{pmatrix}$$

Pokud vstupní úhel omezíme na interval <0; $\frac{\pi}{2}$ > a krok úhlu **α** v každé iteraci tak, že

$$\tan(\alpha)=\pm 2^{-i}$$

kde *i* je číslo kroku iterace, můžeme matici rotace vektoru zredukovat na:

$$R_i=K_i\begin{pmatrix}1 & -\sigma_i 2^{-i}\\ \sigma_i 2^{-i} & 1\end{pmatrix}$$

kde

$$K_i=\frac{1}{\sqrt{1+2^{-2i}}}$$

a σ_i nabývá hodnot [-1; 1], v závislosti, zda má vektor v_i rotovat v kladném, nebo záporném směru, tak, aby se jeho úhel přibližoval ke vstupnímu úhlu.

Výpočet σ_i můžeme provést jednoduchým rozdílem požadovaného a dosaženého úhlu. Dosažený úhel vypočítáme jako:

$$\beta_i=\beta_{i-1}-\sigma_i\gamma_i$$

kde:

$$\gamma_i=\arctan(2^{-i})$$

Hodnoty pro γ_i musejí však být předpočítány a uloženy v paměti procesoru. Pro úsporu paměti lze pro malé úhly aproximovat

$$\arctan(\gamma_i)=\gamma_i$$

Násobení číslem 2^{-i} pak lze v hardware provést velice jednoduše, a to bitovým posunem vpravo o *i* bitů.

V každé iteraci počítáme nový vektor:

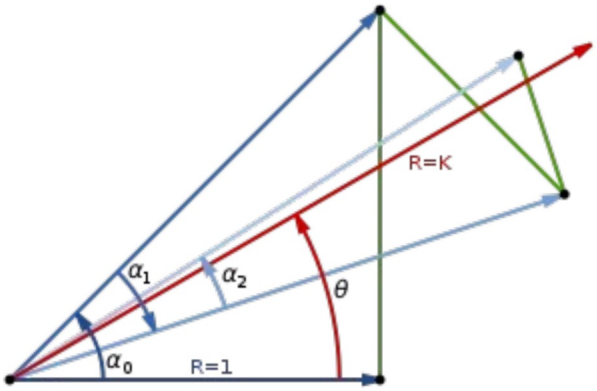
$$v_i=R_i\cdot v_{i-1}$$

K_i můžeme v průběhu výpočtu ignorovat a aplikovat jej až po poslední, n-té iteraci, kde:

$$K(n)=\prod_{i=0}^{n-1}\frac{1}{\sqrt{1+2^{-2i}}}$$

Pokud provádíme při každém výpočtu stejný počet iterací, můžeme K(n) považovat za konstantu, kterou můžeme uložit do hardware kalkulačky a násobit jí po každém výpočtu, nebo jí můžeme vynásobit původní jednotkový vektor, čímž se sníží paměťová náročnost výpočtu.

Pro běžné kalkulačky může být počet iterací zvolen fixně, jelikož například 40 iterací je plně dostačujících pro dosažení přesnosti na 10 desetinných míst při většině výpočtů.



Obr. 5: Postupná pseudorotace jednotkového vektoru. Délka výsledného vektoru je 1/K(n), je proto nutné jej vynásobit K(n)

Při použití tohoto algoritmu je však třeba dbát na zpracování vstupu. Jelikož algoritmus pracuje pouze na intervalu <0; $\frac{\pi}{2}$ >, musíme vstupní úhel napřed přepočíst na tento interval a výsledek následně vhodně upravit. Toto rozhodování je však relativně jednoduché a levné i na velmi primitivních procesorech.

Algoritmy CORDIC byly v minulosti využívány i v knihovnách pro operace s čísly s plovoucí řádovou čárkou na procesorech bez matematického koprocesoru. Dnes jsou opět vyvíjeny a zjednodušovány pro implementaci na moderních FPGA, kde se tak uvolňuje kapacita pro ostatní potřeby projektu, využívající hradlové pole. [4][5][6]

2.2.2 Vstupní metody kalkulaček

Vstup uživatelských dat může být kalkulačkou zpracováván mnoha různými způsoby, ovšem nejpoužívanější a nejznámější je metoda postupného zadávání čísel od nejvyššího řádu spolu s inflixní notací. Tím se moderní elektronické kalkulačky liší od mechanických, kde se číslo zadávalo najednou, pomocí obsáhlé klávesnice, obsahující všechny číslice pro všechny řády.

Inflixní notace je intuitivním zadáváním členů výpočtu, plynoucí z přirozeného zápisu rovnic:

$$operand - operátor - operand - rovnítko$$

Tento způsob notace je velice rozšířen a lze s jistotou říci, že téměř každý člověk, který umí pracovat s kalkulačkou je s touto notací dobře seznámen.

Další známou metodou vstupu je **posfixní**, neboli **reverzní polská notace**. Při této metodě se nejprve zadávají operandy a až posléze operátor. Není potřeba znaménko rovná se, jelikož zadání operátoru je už indikací k provedení výpočtu. Naopak je potřeba klávesa *enter*, která posune zadaný operand do stacku. Nejsou vůbec potřeba závorky, priorita výpočtů je zajištěna posloupností příkazů.

Postfixní notace je výhodná pro použití v jednoduchých kalkulačkách, jelikož je úspornější na implementaci. Z tohoto důvodu byla také velmi rozšířená mezi prvními kalkulačkami v 60. a 70. letech 20. století. Kalkulačky od firmy Hewlett-Packard jsou historicky známé

používáním postfixní notace i v moderních přístrojích a na jejím používání si v historii firma zakládala. Stejně tak je reverzní polská notace hojně používaná v historických kalkulačkách značky Elektronika z bývalého Sovětského svazu.

Obr. 6: Kalkulačka Elektronika MK 52 - povšimněte si absence klávesy '='

Kromě těchto metod existuje i tzv. **Prefixní notace**, kde se operátor zadává před oběma operandy. Tento způsob vstupu se však v kalkulačkách nepoužívá, jelikož nepřináší žádné výhody (naopak lehce komplikuje práci se stackem) oproti postfixní notaci.

II. PRAKTICKÁ ČÁST

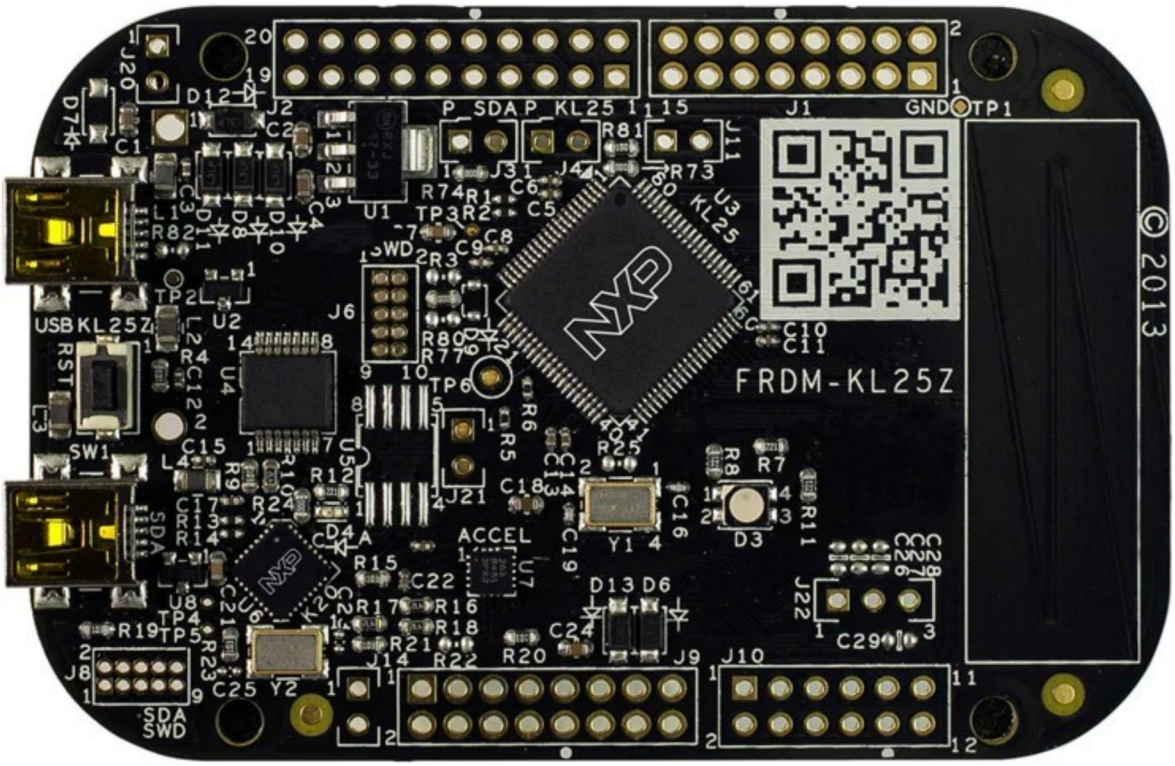
3 NÁVRH VÝUKOVÉHO MODELU

Cílem této práce je navrhnout a vyrobit model kalkulačky, který bude plně konfigurovatelný uživatelem. Myšlenou je, že si uživatel může vyzkoušet implementaci výpočetních algoritmů, vstupní metody (RPN nebo inflixní notace, případně prefixní či zcela nestandardní notace) i zobrazování na displeji.

Jádrem modelu je vývojová deska FRDM-KL25Z s CPU na ARM architektuře. Vstupním zařízením jsou dvě klávesnice s tlačítky v matici 4x4, poskytující dostatek vstupních kláves i pro komplexnější návrhy funkcí.

Všechny součásti modelu jsou propojeny propojovacími dráty s konektory DuPont.

3.1 NXP FRDM-KL25Z



Obr. 7: Deska FRDM-KL25Z [7]

Jádrem kalkulačky je vývojová deska od firmy NXP, která obsahuje:

- 32bitový MCU s jádrem Cortex-M0
- 48 MHz frekvence CPU
- 128 kB Flash paměti a 16 kB paměti SRAM
- 16bitový A/D převodník a 12bitový D/A převodník
- Generátor hodinového signálu FLL a PLL
- Šestkrát časovač TMP 1 a dvakrát časovač TMP 2
- Systémový časovač, časovač s nízkou spotřebou
- Dvě 8bitové sériové periferní rozhraní
- Jeden nízkopříkonový modul UART a dva standardní UART moduly
- Dva I2C moduly
- Akcelelometr MMA8451Q
- RGB LED
- Teplotní senzor LM75AD

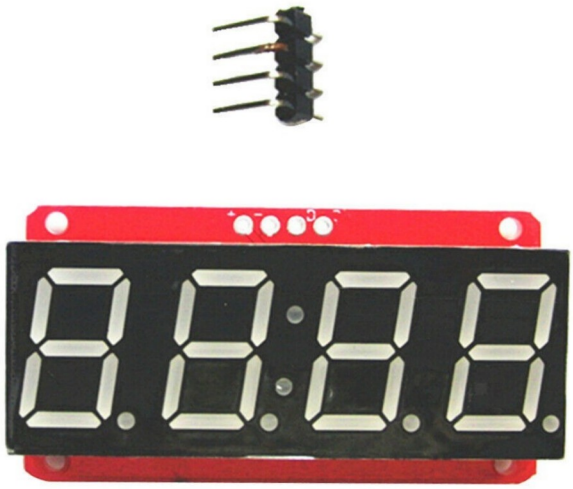
[7]

Procesor je dostatečně výkonný pro účely modelu, ačkoliv model potřebuje v zásadě kalkulačku softwarově emulovat.

3.2 Displeje

Jako výstupní zařízení byly použity tři displeje od nejmenovaného výrobce. Každý z displejů obsahuje čtyři číslice a je řízen budičem HT16K33.

Tyto budiče jsou řízeny po I2C sběrnici a adresují se pomocí tří párů pájecích bodů na zadní straně displeje.

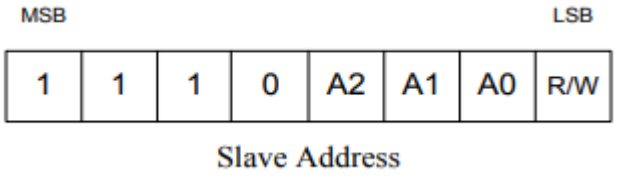


Obr. 8: Displej použitý v modelu

3.2.1 Budiče HT16K33

Budiče displejů jsou určeny k buzení šestnácti sedmisegmentových displejů, případně matice LED 8x16 nebo ke čtení propojovacího pole stejné velikosti. K účelu sestavení modelu by tak stačil jen jeden. Displeje jsou ovšem hotovým výrobkem, musíme proto řešit komunikaci se třemi budiči.

Komunikace po I2C sběrnici je naštěstí jednoduchá a displeje lze připojit fyzicky paralelně k sobě.



Obr. 9: Adresace budičů HT16K33 [8]

Adresace budičů na I2C sběrnici je znázorněna na obr. 9. Bity A0-A2 jsou nastavitelné pomocí propájení plošek na zadní straně displejů. Bit R/W určuje, zda bude následovat operace čtení či zápisu. Pro použití jako zobrazovacích prvků budeme vždy jen zapisovat.

Po adresovém bytu se na sběrnici odesílá příkazový byte.

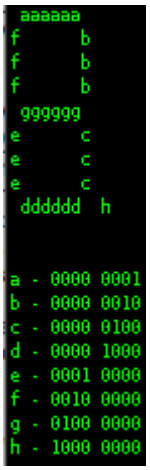
Name	Command / Address								Option	Description	Def.
	D15	D14	D13	D12	D11	D10	D9	D8			
Display data Address pointer	0	0	0	0	A3	A2	A1	A0	{A0~A3} R/W	• Five bits of immediate data, bits A0 to A3, are transferred to the data pointer to define one of sixteen display RAM addresses. • If the Display data register address (An) is 0X00h ~ 0X0Fh, after reaching the memory location 0X0Fh, the pointer will reset to 0X00h	00H
System setup	0	0	1	0	X	X	X	S	{S} Write only	Defines internal system oscillator on/off • {0}:Turn off System oscillator (standby mode) • {1}:Turn on System oscillator (normal operation mode)	20H

Obr. 10: Ukázka formátu příkazů

Tento příkazový byte určuje, zda a kolik bude následovat datových bytů. Například příkaz pro zápis do paměti výstupů je následován libovolným množstvím datových bytů, které se

zapisují do adres začínajících adresou uvedenou v příkazovém bytu. Tato paměť má velikost 16 bytů a po dosažení konce se následující data zapisují opět od adresy 0x00.

Každý byte v paměti odpovídá osmi výstupům budiče, který tyto výstupy multiplexuje na 16 společných elektrod. Jednotlivé bitové masky pro konkrétní segmenty a pozice číslic v paměti byly určeny experimentálně, jelikož k displejům není k dipozici žádný datasheet.



Obr. 11: Tradiční značení segmentů a odpovídající bitové masky

Pořadí číslic ve výstupní paměti budičů je:

Tab. 1: Pozice číslic v paměti budiče displejů

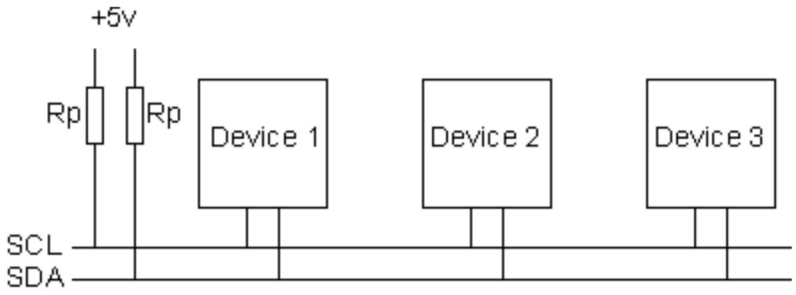
Adresa	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa		0xb	0xc	0xd	0xe	0xf
Číslice	1.	NC	2.	NC	dvojtečka	NC	3.	NC	4.	NC	NC	NC	NC	NC	NC	NC	

Na displeji se dá rovněž tlumit jas, a to pomocí příkazového bytu 0x1110$P_3P_2P_1P_0$ kde

P0-P3 jsou bity čísla, které udává dobu svícení displeje pro danou číslici (1/16 – 16/16). Jelikož displej multiplexuje, nebude nikdy střída plných 100 %, nicméně displej je jasný dostatečně. [8]

3.2.2 I2C sběrnice

I2C sběrnice je datové rozhraní hojně používané v průmyslu. Je velice jednoduché a ačkoliv není příliš odolné proti rušení, na krátké vzdálenosti je mnohdy díky své nenáročnosti ideálním řešením.



Obr. 12: Schéma připojení zařízení na I2C sběrnici

Vodiče SCL (clock) a SDA (data) jsou pull-up rezistory připojeny na kladný potenciál, zařízení je pak při vysílání stahují k zemnímu potenciálu.

I2C protokol má master-slave strukturu, kdy master vždy generuje clock pro slave, na který se dotazuje.

I2C rozhraní nemá jedno univerzální napětí, zařízení mohou komunikovat při napěťových hladinách 3,3 V nebo 5 V, případně jakýmkoliv jiným. Tento fakt byl mírnou překážkou při sestavování modelu kalkulačky, jelikož byl potřeba převodník napěťových úrovní mezi KL25Z, které komunikuje při 3,3 V a displeji, které komunikují při 5 V. Tyto převodníky jsou však naštěstí levné a díky nenáročnosti I2C komunikace lze v tomto případě použít takřka jakýkoliv.[9]

3.3 Klávesnice

Jako klávesnice byly použity dvě jednoduché maticové klávesnice 4x4. Sloupce obou klávesnic jsou zapojeny paralelně, takže se klávesnice chovají jako jedno pole 4x8. Při stisku klávesy má spojení relativně velký odpor, 200 až 1000 ohmů, což naštěstí nečinilo problém se čtením stisků.

Obr. 13: Klávesnice (Zdroj: eshop Laskarduino.cz)

4 POPIS PROGRAMOVÉHO VYBAVENÍ MODELU

Programové vybavení a knihovny byly napsány v jazyce C. Pro vývoj bylo použito vývojového prostředí Kinetis Design Studio od firmy NXP. Podpůrné knihovny obsahují dva soubory: hlavičkový soubor obsahující prototypy funkcí a zdrojový soubor obsahující definice funkcí pro obsluhu hardware kalkulačky. Dále programové vybavení obsahuje hlavní program pro ukázku obsluhy modelu.

Knihovna obsahuje funkce:

- **void init_disp**(uint8_t addr[3]);
- **void draw_disp**(uint8_t addr[3], **char** char_arr[12]);
- **void set_brightness**(uint8_t addr[3], **int** dir);
- **void init_gpio**();
- **void make_array**(**double** num, **char** *out);
- **void move_regs**(**char** dir, **double** *registers);
- **void push_disp**(uint8_t addr[3], **double** number);
- **void push_to_reg**(**bool** point, **bool** exp, **int** reset, **bool** sgn, **int new**, **double** *registers, **char** *out);
- **void add_xy**(**double** *registers);

- **void sub_xy**(**double** *registers);
- **void mul_xy**(**double** *registers);
- **void div_xy**(**double** *registers);
- **void pow_xy**(**double** *registers);
- **void delay**();
- **int read_key**();
- uint8_t **char_to_disp**(**char** input);

4.1 Funkce init_disp

Jako parametr přijímá tato funkce pole tří I2C adres pro jednotlivé displeje.

Tato funkce aktivuje budiče displejů povolením jejich hodinového signálu. Dále vynuluje obsah paměti pro výstup, nastaví jas na polovinu a sepne buzení displejů.

Prototyp funkce:

```
void init_disp(uint8_t addr[3]);
```

4.2 Funkce draw_disp

Jako parametr přijímá funkce pole tří I2C adres pro jednotlivé displeje a pole znaků určených pro zapsání na displej.

Tato funkce pošle do pamětí výstupů budičů displejů data, předávaná v proměnné char_arr. Data správně rozdělí mezi jednotlivé byty pamětí jednotlivých displejů.

Prototyp funkce:

```
void draw_disp(uint8_t addr[3], char *char_arr);
```

4.3 Funkce set_brightness

Jako parametr přijímá tato funkce pole tří I2C adres pro jednotlivé displeje a číslo značící směr změny jasu.

Tato funkce inkrementuje (v případě dir = 1) nebo dekrementuje (v případě dir = -1) jas displejů. Drží si statickou proměnnou o aktuálním jasu.

Prototyp funkce:

```
void set_brightness(uint8_t addr[3], int dir);
```

4.4 Funkce init_gpio

Žádný parametr tato funkce nepřijímá.

Tato funkce inicializuje používané vstupně-výstupní piny desky FRDM-KL25Z.

Prototyp funkce:

```
void init_gpio();
```

4.5 Funkce make_array

Jako parametr přijímá tato funkce přesné číslo typu double a ukazatel na pole, do kterého bude zapisovat.

Tato funkce konvertuje číslo typu double na jednotlivé číslice, které se mají na displejích zobrazovat. Rozpoznává, jakým zápisem má číslo zobrazit (decimální nebo vědecký).

Prototyp funkce:

```
void make_array(double num, char *out);
```

4.6 Funkce move_regs

Vstupy této funkce jsou směr a ukazatel na registry virtuální kalkulačky.

Tato funkce posune registry kalkulačky buďto nahoru (dir = 1) nebo dolů (dir = 0).

Prototyp funkce:

```
void move_regs(char dir, double *registers);
```

4.7 Funkce push_disp

Vstupy funkce jsou adresy displejů a číslo ve formátu double.

Tato funkce slouží k usnadnění vypsání čísla na displej, jelikož sjednocuje funkce make_array a draw_disp, kdy první konvertuje číslo na pole znaků a druhá pak vykreslí pole znaků na displeje. Touto funkcí lze na displeje přímo vykreslit double.

Prototyp funkce:

```
void push_disp(uint8_t addr[3], double number);
```

4.8 Funkce push_to_reg

Vstupy funkce jsou příznaky volání desetinné čárky, zadávání exponentu, reset vstupu do registru a příznaku znaménka. Dále pak celé číslo, ukazatel na registry kalkulačky a ukazatel na pole pro vykreslení na displeje.

Tato funkce slouží k zadávání čísla z klávesnice do registrů a ke korektnímu zobrazení zadávaného čísla. Přijme buď některý z příznaků pro změnu stavu zadávání (kdy ignoruje případné zadávané číslo, situace, která by v normálním provozu neměla nastat), nebo celé číslo, které právě uživatel stisknul, pro zadání do registru a vykreslení na displeje.

Prototyp funkce:

```
void push_to_reg(bool point, bool exp, int reset, bool sgn, int new,
double *registers, char *out);
```

4.9 Funkce add_xy, sub_xy, mul_xy, div_xy, pow_xy

Tyto funkce přijímají jako parametr ukazatel na registry kalkulačky.

Tyto funkce slouží pro ukázku funkce modelu jako kalkulačky. Vzhledem k relativně robustnímu základu lze takovéto výpočetní funkce definovat velice snadno a student si tak může například vyzkoušet implementaci CORDIC algoritmů, aniž by se musel starat o nízkoúrovňové funkce mikropočítače či korektní zpracování a zobrazování vstupů.

```
void pow_xy(double *registers){
      *(registers + 1) = pow(*(registers + 1), *registers);
      move_regs(DOWN, registers);
}

void div_xy(double *registers){
      *(registers + 1) /= *registers;
      move_regs(DOWN, registers);
}

void mul_xy(double *registers){
      *(registers + 1) *= *registers;
      move_regs(DOWN, registers);
}

void sub_xy(double *registers){
      *registers *= -1;
      add_xy(registers);
}

void add_xy(double *registers){
      *(registers + 1) += *registers;
      move_regs(DOWN, registers);
}
```

4.10 Funkce delay

Tato funkce žádný parametr nepřijímá.

Tato funkce slouží ke krátkému zastavení výkonu programu. Slouží pro korektní zpracování vstupů z klávesnice, kde zabraňuje opakovanému vyhodnocení jednoho stisku při zákmitech během průběhu stisku či puštění klávesy.

Prototyp funkce:

```
void delay();
```

4.11 Funkce read_key

Tato funkce žádný parametr nepřijímá.

Tato funkce se stará o multiplexování klávesnice a vyhodnocování, která klávesa byla stisknuta. Návratovou hodnotou je celé číslo jako souřadnice stisknuté klávesy ve formátu 10*sloupec + řádek. V případě, že žádná klávesa stisknuta není, vrací hodnotu -1.

Funkce vždy sepne jeden sloupec klávesnice proti zemi (logická 0), ostatní ponechá na logické 1 a porovnává, zda je na některém z řádků přítomna log. 0.

Prototyp funkce:

```
int read_key();
```

4.12 Funkce char_to_disp

Vstupem této funkce je znak ve formátu char (ASCII)

Tato funkce konvertuje znak k vypsání na displej na jednotlivé segmenty (hodnota, která se zapíše do paměti budiče displeje). V případě nedefinovaného stavu vrací hodnotu odpovídající rozsvícení spodního segmentu displeje – toto je užitečné při ladění programu.

```
uint8_t char_to_disp(char input);
```

4.13 Ukázkový program obsluhy kalkulačky `int main(void)`

Tento ukázkový program obsluhuje kalkulačku – ve smyčce čte klávesnici, na základě stisknuté klávesy pak volá příslušné funkce.

```
int main(void)
{
      ARM_I2C_STATUS status;

      // Inicializace GPIO pinu
      init_gpio();

    // Inicializace a konfigurace ovladace I2C
      Driver_I2C1.Initialize(0);
      Driver_I2C1.PowerControl(ARM_POWER_FULL);
      Driver_I2C1.Control(ARM_I2C_BUS_SPEED, ARM_I2C_BUS_SPEED_FAST);
```

```
	init_disp(LED_ADDR);

	char display[DISP_LENGTH];
	for(int i = 0; i < DISP_LENGTH; i++)
		display[i] = 0x0;
	double reg[NUMBER_OF_REGISTERS];
	for(int i = 0; i < NUMBER_OF_REGISTERS; i++)
		reg[i] = 0;

	int state = 0;
	//stav po stisku tlacitka

	bool is_pressed = 0;
	//pomocna promenna pro detekci nabezne hrany a debounce

	bool draw_arr = 0;
	//pomocna promenna pro vykresleni displeje

	bool draw_enter = 0;
	//pomocna promenna pro pushnuti stacku a vykresleni displeje

	char error[] = {0x79, 0x50, 0x50, 0x5c, 0x50, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00};
	char tag[] = {0x71, 0x3e, 0x50, 0x50, 0x6e, 0x00, 0x00, 0x3d, 0x77,
0x54, 0x3d, 0x00};
	push_disp(LED_ADDR, reg[0]);

	for (;;){
		state = read_key();

		if(is_pressed){
			if(state == -1){
				is_pressed = 0;
			}
		}

		if(!is_pressed){
			switch(state){
			case 0:
				push_to_reg(0, 0, 0, 0, 1, reg, display);
//vloz 1 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
				draw_arr = 1;
				break;

			case 1:
				push_to_reg(0, 0, 0, 0, 4, reg, display);
//vloz 4 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
				draw_arr = 1;
				break;

			case 2:
				push_to_reg(0, 0, 0, 0, 7, reg, display);
//vloz 7 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
				draw_arr = 1;
				break;
```

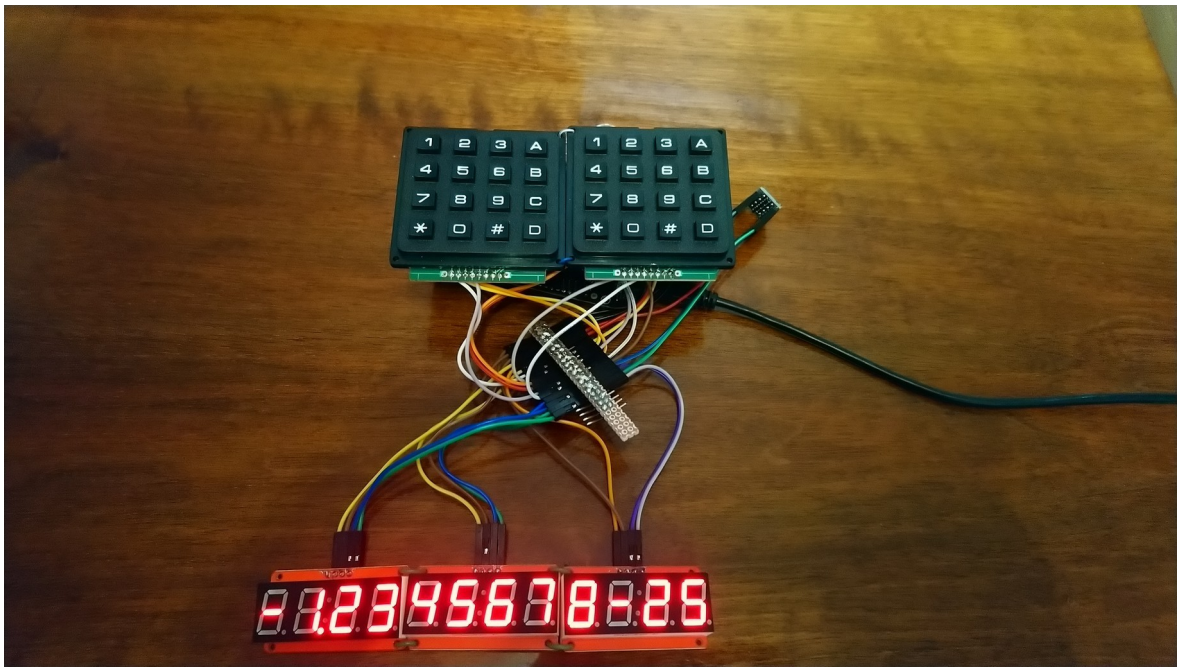
```
                    case 3:
                          push_to_reg(1, 0, 0, 0, 0, reg, display);
//vloz desetinou tecku - flagy exponentu, resetu a znamenka jsou 0, cislo
neposilame
                          draw_arr = 1;
                          break;

                    case 4:
                    //vynasob x * y
                          mul_xy(reg);
                          draw_enter = 1;
                          break;

                    case 5:
                    //vydel y / x
                          div_xy(reg);
                          draw_enter = 1;
                          break;

                    case 6:
                    //umocni y ^ x
                          pow_xy(reg);
                          draw_enter = 1;
                          break;

                    case 7:
                          push_to_reg(0, 1, 0, 0, 0, reg, display);
//zacni vkladat exponent - flagy des. tecky, resetu a znamenka jsou 0,
cislo neposilame
                          draw_arr = 1;
                          break;

                    case 10:
                          push_to_reg(0, 0, 0, 0, 2, reg, display);
//vloz 2 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
                          draw_arr = 1;
                          break;

                    case 11:
                          push_to_reg(0, 0, 0, 0, 5, reg, display);
//vloz 5 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
                          draw_arr = 1;
                          break;

                    case 12:
                          push_to_reg(0, 0, 0, 0, 8, reg, display);
//vloz 8 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
                          draw_arr = 1;
                          break;

                    case 13:
                          push_to_reg(0, 0, 0, 0, 0, reg, display);
//vloz 0 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
                          draw_arr = 1;
                          break;
```

```
                    case 20:
                          push_to_reg(0, 0, 0, 0, 3, reg, display);
//vloz 3 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
                          draw_arr = 1;
                          break;

                    case 21:
                          push_to_reg(0, 0, 0, 0, 6, reg, display);
//vloz 6 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
                          draw_arr = 1;
                          break;

                    case 22:
                          push_to_reg(0, 0, 0, 0, 9, reg, display);
//vloz 9 - flagy desetinne tecky, exponentu, resetu a znamenka jsou 0
                          draw_arr = 1;
                          break;

                    case 23:
                          push_to_reg(0, 0, 0, 1, 0, reg, display);
//zmen znamenko (sgn = 1) - flagy desetinne tecky, exponentu a resetu
jsou 0, cislo neposilame
                          draw_arr = 1;
                          break;

                    case 30:
                          add_xy(reg);
              //secti x + y
                          draw_enter = 1;
                          break;

                    case 31:
              //odecti y - x
                          sub_xy(reg);
                          draw_enter = 1;
                          break;

                    case 32:
                          draw_enter = 1;
//enter (push stack)(reset = 1) - flagy desetinne tecky, exponentu,
resetu a znamenka jsou 0, cislo neposilame
                          break;

                    case 33:
                          push_to_reg(0, 0, 1, 0, 0, reg, display);
//smaz registr x (reset = 1) - flagy desetinne tecky, exponentu a
znamenka jsou nula
                          reg[0] = 0;
                          make_array(reg[0], display);
                          push_disp(LED_ADDR, reg[0]);
                          break;

                    case 34:
                          set_brightness(LED_ADDR, UP);
      //zvys jas o jeden krok
                          break;

                    case 35:
                          set_brightness(LED_ADDR, DOWN);
      //sniz jas o jeden krok
```

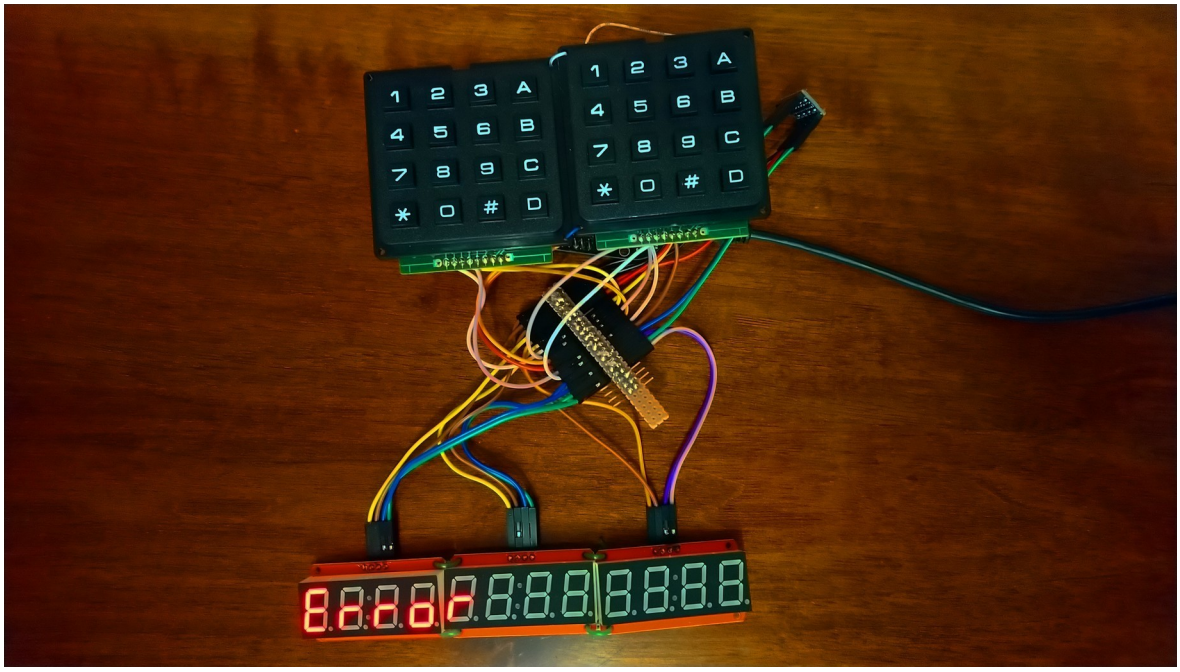
```
                        break;

                    case 37:
                        draw_disp(LED_ADDR, tag);
                        break;

                    default:
                        break;
                    }

                    if(draw_arr){
                        draw_disp(LED_ADDR, display);
                        draw_arr = 0;
                    }
                    if(draw_enter){
                        push_to_reg(0, 0, 1, 0, 0, reg, display);
                        if(((fabs(reg[0]) >= MAX_NO) || (fabs(reg[0]) <=
MIN_NO) || (isnan(reg[0]))) && (reg[0] != 0)){
                            draw_disp(LED_ADDR, error);
                        }else{
                            if(state == 32){
//tento blok pouzivame i pro zakonceni vypoctu, pak neni potreba pushnout
registry
                                move_regs(UP, reg);
                                push_to_reg(0, 0, 2, 0, 0, reg,
display);
                            }
                            make_array(reg[0], display);
                            push_disp(LED_ADDR, reg[0]);
                        }
                        for(int i = 0; i < DISP_LENGTH; i++)
                            display[i] = 0x0;
                        draw_enter = 0;
                    }
                }

                if(state != -1){
                    is_pressed = 1;
                    delay();
                }
            }

            /* Never leave main */
            return 0;
}
```

5 UKÁZKA FUNKČNÍHO MODELU

Ačkoliv model není fyzicky zhotoven do finální podoby, je funkční ukázkou, jak programové vybavení funguje. Zabudování do krabičky a dokončení je už jen otázka výroby krabičky pro model a lepší urovnání, případně použití delších propojů.



Obr. 14: Model zobrazující číslo



Obr. 15: Model zobrazující chybový stav

Model je zcela funkční a logika zadávání čísel byla inspirována kalkulačkou Elektronika MK 52. Zadávání je tak pomocí RPN a po výpočtu lze okamžitě zadávat další operand bez

nutnosti enteru. Bylo ošetřeno několik bugů a vstupní chování modelu je tak poměrně robustní.

5.1 Rozložení klávesnice

Tab. 2: Rozložení klávesnice

1	2	3	add_xy	mul_xy			Jas++
4	5	6	sub_xy	div_xy			Jas--
7	8	9	Enter				
,	0	sgn	Clear	exp			

ZÁVĚR

Cílem práce bylo popsat vědecké kalkulačky a jejich historii a navrhnout, sestavit a napsat programové vybavení a knihovny pro výukový model vědecké kalkulačky.

Tato práce uvádí čtenáře do historie vývoje počítacích pomůcek a elektronických vědeckých kalkulaček. Ukazuje, jakým způsobem kalkulačky provádí výpočty na mnohdy velmi primitivních procesorech či hradlových polích a jakými způsoby mohou kalkulačky zpracovávat lidský vstup. Zároveň dává čtenáři základní znalosti terminologie a algoritmů použitých v oblasti vnitřní funkce kalkulaček, a umožňuje tak v případě zájmu další samostudium.

Výstupem práce je pak kompletní návrh a částečné řešení fyzického modelu vědecké kalkulačky, určené ke studijním účelům, a napsání programu a programových knihoven, které obsluhují hardware kalkulačky a usnadňují tvorbu vlastního programového vybavení a implementaci algoritmů.

Tento hardware byl sestaven pomocí vývojové desky FRDM-KL25Z od firmy NXP. Tato deska obsahuje mikroprocesor architektury ARM, dostatek vstupně-výstupních pinů i I2C sběrnici pro komunikaci s displeji. Displeje jsou tři, každý se čtyřmi 7segmentovými číslicemi, řízeny budiči HT16K33. Tyto budiče jsou ovládány po sběrnici I2C, což značně snižuje nároky na počet vstupně-výstupních pinů mikroprocesoru. Zároveň však používají napěťové úrovně 5 V, namísto 3,3 V, které používá mikroprocesor, proto bylo nutné použít konvertor napěťových úrovní. Klávesnice jsou dvě, obě sestávají ze spojovacího pole 4x4. Pro snížení počtu vstupů a zjednodušení programové obsluhy jsou sloupce obou klávesnic zapojeny paralelně, a klávesnice se tak chová jako spojovací pole 4x8.

Programové vybavení modelu bylo napsáno v jazyce C a obsahuje knihovny jak pro obsluhu přímo připojeného hardware (klávesnice), obsluhu I2C sběrnice a displejů, tak i funkce pro práci s registry virtuální kalkulačky. Do programového vybavení jsou zahrnuty i funkce pro základní matematické operace, model je tedy připraven pro okamžitou ukázku.

Tento model může být použit pro ukázku, co vše je potřeba pro vytvoření funkční vědecké kalkulačky a zároveň studentům umožňuje vlastní implementaci matematických algoritmů pro výpočty, jejich porovnání a díky možnosti krokování programu ve vývojovém prostředí i pozorování, jak se jejich algoritmy chovají v průběhu výpočtů.

SEZNAM POUŽITÉ LITERATURY

[1] IFRAH, Georges. *The Universal History of Computing: From the Abacus to the Quantum Computer.* New York, NY: John Wiley & Sons, Inc, 2001. ISBN 0-471-39671-0

[2] SMITH, David E. *History of Mathematics.* 2. New York: Dover Publications, 1958. ISBN 9780486204307

[3] VALENTINE, Nick. The History Of The Calculator. *The Calculator Site* [online]. 2019 [cit. 2020-08-12]. Dostupné z: https://www.thecalculatorsite.com/articles/units/history-of-the-calculator.php

[4] LAPORTE, Jacques. The Secret of the Algorithms. *Cordic (English)* [online]. 2005 [cit. 2020-08-12]. Dostupné z: http://www.jacques-laporte.org/TheSecretOfThe-Algorithms.htm

[5] ANDRAKA, Ray. *A survey of CORDIC algorithms for FPGA based computers*. **1998**, 2-3.

[6] VOLDER, Jack E. The CORDIC Trigonometric Computing Technique. *The Institute of Radio Enigeers*. 1959, **1959**, 226-230.

[7] NXP Semiconductors. *KL25 Sub-Family Reference Manual* [online]. 2014 [cit. 2020-08-12]. Dostupné z: http://www.nxp.com

[8] HOLTEK, *RAM Mapping 16*8 LED Controller Driver with keyscan HT16K33* [online]. 2011 [cit 2020-08-12]. Dostupné z: https://cdn-shop.adafruit.com/datasheets/ht16K33v110.pdf

[9] PIKNER, Jiří. *Mikroprocesory a mikropočítače*. Praha: BEN – technická literatura, 2004. ISBN 80-7300-110-1

[10] BARR, Michael a Anthony J. MASSA. *Progamming embedded systems: with C and GNU development tools*. 2Nd ed. Sebastopol: O'Reilly, 2006. ISBN 0596009836

[11] CATSOULIS, John. *Designing embedded hardware*. 2nd ed. Sebastopol, CA: O'Reilly, 2005, xvi, 377 p. ISBN 0596007558

[12] LADMAN, Josef. *Elektronické konstrukce pro začátečníky*. Praha: BEN – technická literatura, 2001. ISBN 80-730-0015-6

[13] SMITH, Warwick A. *C programming for embedded microcontrollers*. 2nd ed. Susteren: Elektor International Media BV, 2008. ISBN 978-090-5705-804

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

LED	Světlo emitující dioda.
RPN	Reverzní polská notace.
MCU	Mikroprocesor
CPU	Centrální procesorová jednotka

SEZNAM OBRÁZKŮ

SEZNAM TABULEK