ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ
KATEDRA APLIKOVANÉ ELEKTRONIKY A TELEKOMUNIKACÍ

DISERTAČNÍ PRÁCE

Metodika využití embedded systémů s nízkým výpočetním výkonem pro neuronové sítě s proměnlivými parametry a topologií

2014 Ing. Matouš Bartl

západočeská univerzita v Plzni

Fakulta elektrotechnická

DISERTAČNÍ PRÁCE

k získání akademického titulu doktor
v oboru

Elektronika

Ing. Matouš Bartl

Metodika využití embedded systémů s nízkým výpočetním výkonem pro neuronové sítě s proměnlivými parametry a topologií

Školitel: Doc. Ing. Jiří Skála, Ph.D.
Datum státní doktorské zkoušky: 6.2.2012
Datum odevzdání práce:

V Plzni, 2014

Prohlášení

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou v rámci doktorského studia na Katedře aplikované elektroniky a telekomunikací Fakulty elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto práci vytvořil samostatně, s použitím literatury a zdrojů uvedených v seznamu, který je její neoddělitelnou součástí a za pomoci legálních kopií řádně registrovaného, nebo volně šiřitelného softwarového vybavení.

V práci nejsou uvedeny žádné citlivé, či utajované skutečnosti podléhající obchodnímu tajemství, nebo vyžadující speciální režim přístupu. Jakékoli využití a uplatnění uvedených postupů a metod je nicméně možné pouze na základě autorské smlouvy a souhlasu Fakulty elektrotechnické Západočeské univerzity v Plzni.

V Plzni dne

Ing. Matouš Bartl

……

Poděkování

Rád bych tímto poděkoval všem, kteří se na vzniku této práce, ať přímo, nebo nepřímo podíleli. Jmenovitě svému dřívějšímu školiteli, Ing. Petru Weissarovi, Ph.D. za vedení a shovívavost při prvních krocích v pro mě neznámém a novém prostředí doktorského studia, stejně jako za poskytnutí odborných znalostí. Svému současnému školiteli, Doc. Ing. Jiřímu Skálovi, Ph.D., který v závěru studia převzal mé vedení, pomohl mi s orientací v administrativních náležitostech s ním spojených a svým příkladem mi v posledních, časově kritických, momentech dodal potřebnou motivaci a odhodlanost.

Chtěl bych poděkovat i svým blízkým, přátelům a rodině, kteří po dlouhá léta museli snášet důsledky mého časového vytížení a mnohdy i přetížení paralelním studiem při zaměstnání, za jejich trpělivost, a podporu. Jmenovitě pak svému kamarádovi, souputníkovi-spolustudentovi Ing. Karlu Čermákovi za to, že mi byl hodnotným partnerem a společníkem v průběhu celého studia.

Děkuji!

V Plzni dne

Ing. Matouš Bartl

……

Název

Metodika využití embedded systémů s nízkým výpočetním výkonem pro neuronové sítě s proměnlivými parametry a topologií

Anotace

Práce se zabývá problematikou aplikace neuronových sítí na embedded systémech s nízkým výpočetním výkonem a omezenými hardwarovými prostředky. Cílem je navrhnout a prověřit metodu zavádění v podobě umožňující adaptaci vah a změnu topologie v cílovém systému za běhu, bez nutnosti změny firmwaru.

První část je stručným obecným úvodem do problematiky neuronových sítí. Předkládá historické pozadí vzniku oboru, shrnuje základní teoretické principy a rozlišuje kategorie problémů vhodné pro řešení pomocí umělých neuronových sítí. Na základě literatury utváří přehled o současném stavu.

Druhá a třetí část práce teoreticky popisují navrhovanou metodu a její ukázkovou praktickou aplikaci na embedded systému osazeném procesorem PIC18F. Pro možnost srovnání je v textu předkládáno více variant řešení problému. Jsou prezentovány výsledky jednak v podobě vysvětlených a okomentovaných zdrojových kódů a jednak v podobě měření jejich relevantních parametrů. Nejčastěji přesnosti a náročnosti na výpočetní výkon.

V poslední části jsou shrnuty dosažené výsledky v podobě diskuse o splnění vytyčených cílů a nástinu možností dalšího vývoje.

Klíčová slova

Adaptabilní neuronové sítě, Aplikace, Embedded, Jednočipové mikropočítače, metodika zavádění ANN, PIC18F

Title

The methodology of use of embedded systems with low computing power for neural networks with changeable parameters and topology

Abstract

This paper deals with the application of neural networks to embedded systems with low computing power and limited hardware resources. The aim is to design and verify the method of implementation in a form that allows the adaptation of net scales and change of the net topology in the target system on-line, without having to change the firmware.

The first part is a brief introduction to artificial neural networks. It shows the historical background of the field, summarizes the basic theoretical principles and lists the categories of problems suitable for solution using artificial neural networks. This part also, based on literature, gives an overview of the current state.

The second and third parts of the paper describe the theory of the proposed method and a practical application to an embedded system fitted with PIC18F microprocessor. More possible solutions of the problem are presented for comparison. Results are presented both in the form of commented and explained source code examples and in the form of measurements of their relevant parameters, mostly the accuracy and computing power expenditure.

The final part summarizes the obtained results in the form of discussions about reaching the objectives and possible further development.

Keywords

Adaptable Neural Networks, Applications, Embedded, Microcontrollers, Implementation methodology for ANN, PIC18F

Заглавие

Методика использования встроенных систем с низкой вычислительной мощностью для нейронных сетей с переменными параметрами и топологией

Аннотация

Работа посвещена проблематике искусственных нейронных сетей, реализованных на встроенных системах с низкой вычислительной мощностью и ограниченнъми аппаратными ресурсами. Цель работы спроектировать и проверить методы реализации, которые позволяют адаптацию коеффициентов и смену топологии сети в целевой системе во время работы без необходимости смены микропрограммы.

Первая часть является кратким общим введением в проблематику нейронных сетей. В ней представлены исторические предпосылки возникновения научной области, резюмированы основные теоретицкие принципы и дифференциируется категории вопросов, подходящих для решения при помощи искусственных нейронных сетей. На основании литературы представлен обзор текущего состояния.

Вторая и третья главы работы теоретически описывают предлагаемый метод и ее демонстрационную практическое приложение встроенных систем, оснащенную процессорем PIC18F. Для сравнения в тексте представлено несколько вариант решения проблемы. Результаты представлены как ввиде описанных и документированных исходных кодов, так и в виде измерения их релятивных параметров. Чаще всего точность и требовательность к вычислительной мощности.

В последней части обобщены постигнутые результаты в форме дискуссии о выполнении намеченных целей, а так же предложен очерк возможностей последующего развития.

Ключевые слова

Адаптивные нейронные сети, аппликация, микроконтроллер, методика реализации ИНС, PIC18F

Obsah

1. Úvod

V současné době je drtivá většina elektronických zařízení, se kterými denně přicházíme do styku, konstruována s použitím řídicího embedded systému osazeného nějakou programovatelnou součástkou, nejčastěji jednočipovým mikropočítačem. Jejich masové nasazení a postupující vývoj způsobily rapidní pokles cen jich samotných, stejně jako technologií pro práci s nimi. Čím dál častěji se stává, že je i jednoduchý výrobek v rámci úspor na jiných prvcích nebo vývojových pracích osazen více výpočetními jednotkami vzájemně propojenými komunikační sběrnicí. Současně je nutné dodat, že výkony a možnosti mikrokontrolérů, včetně těch nízkonákladových, stoupají. Následkem pak mnohdy bývá pouze částečné využití jejich potenciálu.

Práce se zabývá možnostmi implementace umělých neuronových sítí právě na takových embedded systémech, jejichž výpočetní jednotka, nebo jednotky nižších výpočetních výkonů primárně určených pro jiné druhy činností nejsou zcela využity a použití tohoto zbytkového výkonu pro zavedení neuronové sítě by bylo přínosné. Hlavní myšlenkou je koncepčně navrhnout doplňkové firmwarové vybavení schopné, do jisté míry samostatně, využít zbytkovou programovou paměť, výpočetní výkon a volnou kapacitu sběrnice k vytvoření obecné neuronové sítě předem neznámé topologie. Jejím účelem by mělo být zvýšení celkové hodnoty sytému, například zavedením podpory vyspělejší komunikace s uživatelem, nebo zkvalitnění činnosti stávajícího firmwaru. V principu by ji však mělo být možné použít k libovolnému účelu.

2. Cíle disertační práce

Cílem disertační práce je navrhnout metodu pro zavedení obecné neuronové sítě s možností on-line učení a adaptace topologie do embedded systémů osazených procesory nižších tříd (výrobci obvykle označovanými „low-end). Podmínkou pro úspěšnou implementaci metody nechť jsou: Nezanedbatelné využitelné množství volné programové paměti a výpočetního výkonu zúčastněných jednotek a možnost přeprogramování programové paměti samotným procesorem za běhu programu.

Součástí řešení budou dva celky. Obecný postupný popis metody ve formě návodu pro implementaci do obecného systému splňujícího uvedené požadavky s návodnými náměty na konkrétní způsob realizace a ukázková aplikace po částech ověřující proveditelnost a praktické výsledky jednotlivých kroků na demonstračním systému, jehož návrh bude součástí práce. Tyto části budou dále obsahovat dílčí body.

Pro obecný popis metody:

1) Obecný popis neuronové sítě shrnující požadavky na jednotlivé funkční bloky
2) Rozbor jednotlivých bloků s popisem realizace
3) Metody pro pospojování jednotlivých funkčních bloků
4) Klasifikace cílových systémů s ohledem na téma práce
5) Metody navázání neuronové sítě do systému

Pro ukázkový systém:

1) Popis hardwaru a základního programového vybavení ukázkového systému
2) Implementace bodů v rozsahu obecného popisu metody
3) Zhodnocení funkčnosti a měření dosažených parametrů
4) Prakticky využitelné zdrojové kódy

3. Současný stav problematiky

3.1 Obecně o umělých neuronových sítích

Umělé neuronové sítě jsou inspirovány svými biologickými předlohami. Jsou to v podstatě jejich zjednodušené modely. Nejčastěji vystupují v roli konkurentů konvenčních metod matematické analýzy při řešení různých úloh. S jejich pomocí však lze řešit i problémy klasickými prostředky prakticky neřešitelné. Umělé neuronové sítě mají výhodu ve schopnosti učit se a zobecňovat, což je předurčuje zejména pro úlohy klasifikace dat. Například při realizaci funkce ***f:X => Y*** rozřazující na základě požadovaných vlastností prvky vstupní množiny ***X*** do kategorií obsažených v množině ***Y*** dokáže vhodně navržená a natrénovaná umělá neuronová síť (vlivem schopnosti zobecňovat) správně určit výstup i pro dříve nedefinované vstupy. Této robustnosti a tolerance vůči nepřesnosti se využívá například při rozpoznávání objektů v obrazovém záznamu, strojovém čtení textu, kompresi a dekompresi dat, predikci dat, řízení analyticky nepopsatelných nebo obtížně popsatelných procesů, či náhradě konvenčních regulátorů neuronovými tam, kde poskytují menší výpočetní náročnost, nebo vyšší flexibilitu.

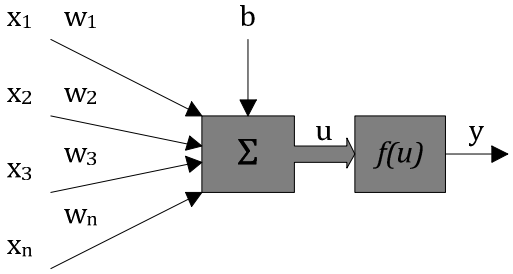
Největšími nevýhodami umělých neuronových sítí je jednak absence exaktních metod pro návrh jejich topologie vzhledem k řešené úloze a analytické stanovení váhových koeficientů sítě na základě požadované činnosti a jednak náročnost na hardwarové vybavení (u analyticky složitě reprezentovatelných vztahů tomu bývá i naopak, což je potom důvod k nasazení neuronové sítě). První ze jmenovaných nevýhod je řešena souborem empirických pravidel a učením nerunové sítě na trénovací množině dat. Druhá vyplývá z principu biologického vzoru – činnost jednotlivých elementů je paralelní. Umělé simulované neuronové sítě se tedy musí omezit na čistě sériové řešení - procesory, kde dochází k rapidnímu poklesu rychlosti odezvy v závislosti na komplexnosti, paralelní řešení – FPGA obvody, kde s narůstající složitostí proporcionálně roste i potřeba velikosti obvodu, zcela odlišný hardware speciálně navržený pro implementaci neuronových sítí, který na druhou stranu sám o sobě neumožňuje doplnění konvenčními algoritmy a není stále tolik rozšířen, anebo kompromisní kombinaci výše zmíněných vyžadující spojení více elektronických součástek, což zvyšuje komplexnost a cenu.

Historicky jsou za počátek umělých neurnových sítí považována 40. léta 20. století, konkrétně práce Warrena McCullocha a Waltera Pittse formulující jako první jednoduchý matematický model neuronu. Obor se z počátku prudce rozvíjí, od 60. do 80. let 20 století se však naopak dostává spíše do útlumu. Poté, hlavně díky novým technologiím a z nich vyplývajících možností, dochází k renesanci, která trvá do současnosti.

Další detailnější informace o vzniku, historickém vývoji a použití je možné najít například v (1) nebo (2).

3.2 Struktura umělých neuronových sítí

Základním prvkem umělých neuronových sítí je matematický model neuronu – formální neuron. Ač se objevují jisté odlišnosti ve struktuře, ať pouze formálního nebo i funkčního charakteru, základem je takřka bezvýhradně původní McCulloch-Pittsův model vyobrazený na *Obr. 3.2.*1. Jednotlivé vstupy $\mathbf{x_{1..n}}$ jsou násobeny příslušnými váhami $\mathbf{w_{1..n}}$ a výsledek je sečten. Speciálním vstupem je konstanta **b**, v některé literatuře uváděná jako neměnný vstup o hodnotě 1 s váhovým koeficientem $\mathbf{w_0}$. Určuje klidový potenciál neuronu. Vnitřní potenciál **u** je vstupem aktivační funkce **f(u)**. Výstupem neuronu je signál **y**. Matematicky lze činnost vyobrazeného modelu neuronu popsat *Vztah 3.2.1*. Zjednodušení lze dosáhnout formální úpravou $\mathbf{b=x_0.w_0}$, kde vždy $\mathbf{x_0=1}$. Počáteční hranice součtu je pak rozšířena na $\mathbf{n=0}$. V některé literatuře je zaveden pojem „**práh aktivační funkce**“ značený obvykle řeckým písmenem **θ**. Je zaveden v podobě součtu přímo na vstup bloku **f(u)**. Jeho funkce je po úpravě znamének z hlediska vnějšího chování formálního neuronu shodná s klidovým potenciálem **b**, respektive $\mathbf{x_0.w_0}$.



Obr. 3.2.1: Model neuronu

$$y = f\left[\left(\sum_{n=1}^{N} X_n . w_n\right) + b\right]$$ **Vztah 3.2.1**

Přehled aktivačních funkcí je v *Tabulka 3.2.1*. Výčet není vyčerpávající a omezuje se pouze na skupinu nejčastěji užívaných. Nezohledňuje také práh **b**, který je nutno vhodně doplnit dle typu funkce, aby splnil svůj účel. Vnitřní potenciál neuronu **u** je třeba dosadit do aktivační funkce ve tvaru *Vztah 3.2.2*, nebo *Vztah 3.2.3*, kde je třeba doplnit **b** jiným způsobem.

$$u = \left(\sum_{n=1}^{N} X_n . w_n\right) + b$$ **Vztah 3.2.2**

$$u = \sum_{n=1}^{N} X_n . w_n$$ **Vztah 3.2.3**

$f(u): y = \begin{cases} 1\ pro\ u \geq 0 \\ -1\ pro\ u < 0 \end{cases}$	Ostrá nelinearita
$f(u): y = k.u$	Čistě lineární funkce, kde **k** je konstanta směrnice
$f(u): y = \begin{cases} 1\ pro\ u > u_{max} \\ k.u\ pro\ u \in \langle u_{min}; u_{max} \rangle \\ -1\ pro\ u < u_{min} \end{cases}$	Lineární funkce se saturací. **k**, $\mathbf{u_{max}}$ a $\mathbf{u_{min}}$ jsou voleny tak, aby byla výsledná funkce spojitá
$f(u): y = \frac{e^{2u} - 1}{e^{2u} + 1}$	Hyperbolický tangens. Používá se častěji ve formě s přidanými koeficienty, kterými je možno ovlivňovat strmost průběhu a limitní hodnoty
$f(u): y = \frac{1}{1 + e^{-ku}}$	Sigmoida založená na logistické funkci. **k** je konstanta určující strmost. Funkce generuje výstupní hodnoty 0 až 1
$f(u): y = \frac{2}{1 + e^{-ku}} - 1$	Sigmoidální funkce založená na hyperbolickém tangentu. **k** ovlivňuje strmost. Tvarově podobná předchozí, generuje výstupní hodnoty od -1 do +1
$f(u): y = e^{-\left(\frac{u-m}{\sigma}\right)^2}$	Aktivační funkce založená na gaussově křivce. Používá se pro radiální neuronové sítě.

Tabulka 3.2.1: Běžně používané aktivační funkce

Model neuronu je možné použít samostatně, například ve formě n-vstupého perceptronu, nebo ve formě víceprvkové sítě. Obvykle uváděná základní kritéria pro jejich dělení jsou:

1) Organizační dynamika – popisuje strukturu sítě
2) Aktivní dynamika – popisuje princip činnosti sítě
3) Adaptivní dynamika – popisuje způsob změn vedoucí k požadované činnosti

Metodika další detailnější kategorizace je nejednotná a zdroje se v ní často rozcházejí. V důsledku se lze dostat do stavu, kdy každý konkrétní typ neuronové sítě tvoří samostatnou skupinu. Obecně jsou při rozřazování zohledňovány zejména následující otázky:

1) Přítomnost zpětných vazeb v síti – striktně dopředné, rekurentní
2) Způsob trénování sítě – s učitelem, učení posilováním, bez učitele
3) Hodnotový prostor sítě – binární uni nebo bipolární, reálný
4) Uspořádání – hierarchické, kompetitivní, rovnocenné
5) Určení – rozpoznávání, klasifikace, předpověď, paměť

Širší obecný přehled včetně popisů jednotlivých dynamik sítí lze nalézt v literatuře (1), (2), (3). Užití v praxi a specifičtější detailnější informace k jednotlivým typům sítí popisuje například (5), (6), (7) nebo (8).

Jak bylo uvedeno, aktivní dynamika biologické předlohy umělých neuronových sítí je charakterizována paralelním chodem všech uzlů a tedy jejich prakticky okamžitou odezvou na vstupní signály. Je-li pro simulaci neuronové sítě použita součástka se sekvenčním během programu, jsou i výstupní hodnoty **y** jednotlivých neuronů vypočítávány postupně. Problémem této implementace jsou požadavky na výpočetní výkon proporcionálně rostoucí s počtem neuronových jednotek a jejich vstupních vah. Na nejobecnější rovině lze náročnost výpočtu umělé neuronové sítě $\mathbf{V_{celk}}$ vyjádřit *Vztah 3.2.4*, kde $\mathbf{V_{vstup}}$ je výpočetní náročnost obsluhy jednoho vstupu neuronu, $\mathbf{V_{neuron}}$ je výpočetní náročnost samotného neuronu bez vstupů, tedy prakticky výpočet $\mathbf{f_{(u)}}$ a vlastní režie. Celkový počet neuronů sítě je označen písmenem **N** a indexován **n**, počet vstupů neuronu **n** je $\mathbf{I_{(n)}}$.

$$V_{celk} = \sum_{n=0}^{N-1} \left(I_n * V_{vstup} + V_{neuron} \right) \qquad \textbf{Vztah 3.2.4}$$

V tomto případě je naprosto logická snaha minimalizovat výpočetní náročnosti $\mathbf{V_{vstup}}$ a $\mathbf{V_{neuron}}$ a používat pro simulace neuronových sítí v aplikacích, kde je kladen důraz na rychlost, co nejvýkonnější součástky, tedy součástky zvládající objem výpočtů $\mathbf{V_{celk}}$ v co nejmenším čase. Důsledkem tohoto přístupu a bouřlivého rozvoje technologií je poněkud opomíjena problematika implementace neuronových sítí na systémy, které špičkovým výkonem současnosti nedisponují.

3.3 Současnost neuronových sítí na „low-end“ procesorech

Stěžejní literaturou a zdrojem motivace je disertační práce Nicolase Jay Cottona z roku 2010 (9). Autor se v ní zabývá implementací neuronových sítí s ohledem na nízkou výpočetní náročnost. V ukázkách pracuje na jednočipovém mikropočítači s jádrem PIC18F společnosti Microchip (10). Konkrétně ve výsledku navrhuje a předkládá kompletní metodiku, soubor obecnějších funkcí v jazyce „C“ a (vázaných na zvolené jádro, nicméně rychlejších) v jazyce Assembler a komfortní PC aplikace pro tvorbu, zavádění a učení dopředných neuronových sítí včetně hierarchických. Jako směr dalšího vývoje v závěru uvádí rozšíření metody v jazyce „C“ na další typy mikrokontrolérů, srovnávání výkonnosti, optimalizace a užití v praxi.

Autor v celém rozsahu své disertace předpokládá konstantní organizaci a nulovou adaptivní dynamiku sítě v okamžiku, kdy je jednou vygenerována a v podobě statického zdrojového kódu nahrána do mikrokontroléru. Změna vah nebo topologie sítě je tak možná pouze za pomoci přehrání firmwaru. Zde vzniká prostor pro nový směr vývoje. K němu se přidává i možnost rozšířit, alespoň v principiální rovině, základnu i na rekurentní sítě. Jak sám autor v závěru hodnotí, nejsou veškeré algoritmy sestaveny optimálně. Tím je otevřen prostor pro optimalizace nebo odlišné přístupy implementace.

Publikace Nicolase Jay Cottona jsou často zaměřeny právě na problematiku řešenou v jeho práci, přináší však spíše praktické výsledky a rozšířené diskuse nad prezentovanou oblastí zájmů, než nové poznatky (11). Autor spolupracuje se skupinou pohybující se okolo prof. Bogdana Wilamowskiho působícího na Alabama Microelectronics Science and Technology Center, Auburn University, Alabama (12) mimo jiné v oboru neuronových sítí včetně jejich implementací na systémech nízkého výkonu. Ani zde není v uvedených publikacích prof. Wiliamovskiho (13) řešena cílová problematika této disertační práce.

Mezi další autory zabývající se implementačními metodami umělých neuronových sítí a publikujícími na toto téma patří Thomas Behan působící v oboru na Ryerson University, Ontario, Kanada (14). Tématem neuronových sítí a „low-end“ embedded systémů se zabývá již ve své diplomové práci (15), stejně jako v dalších publikacích (16). Řešeny jsou opět neuronové sítě bez mechanismů podpory adaptace topologie a učení přímo v cílovém embedded systému. Autor se navíc postupem času (17) vydává spíše cestou navyšování výpočetního výkonu a prostředků použitého hardwaru. Diverguje tak od „low-end“ systémů spíše do oblasti „mid-range“. Používá například jednodušší DSP s jádrem dsPIC33E/F společnosti Microchip.

Spíše v minulosti (první dekáda 21. století) se cílovou problematikou zabýval Bonifacio Martin-del-Brio působící tou dobou na University of Zaragoza, Zaragoza, Španělsko (18). V jeho publikacích (19), (20), (21) jsou popsány aplikace zejména dopředných perceptronových sítí na

„low-end“ a „low-cost“ mikroprocesorech z období vzniku textů. Lze však získat například cenné informace o praktických aplikacích při použití neuronových sítí pro liearizaci. Opět se však, jakou u předchozích jmenovaných, jedná o statické aplikace neumožňující učení za běhu nebo změnu organizace bez zásahu do firmwaru.

4. Popis obecné metody

Následující kapitola se zabývá popisem obecné metody implementace automaticky expandujících neuronových sítí v rozsahu bodů uvedených v kapitole *2*. Těm v hrubých rysech odpovídá i dělení do podkapitol. Obsah je z většiny původním dílem autora a je založen zejména na zkušenostech z praxe programování real-time embedded systémů s vysokými požadavky na přesné časování prováděných činností v programovacích jazycích „C“ a Assembler.

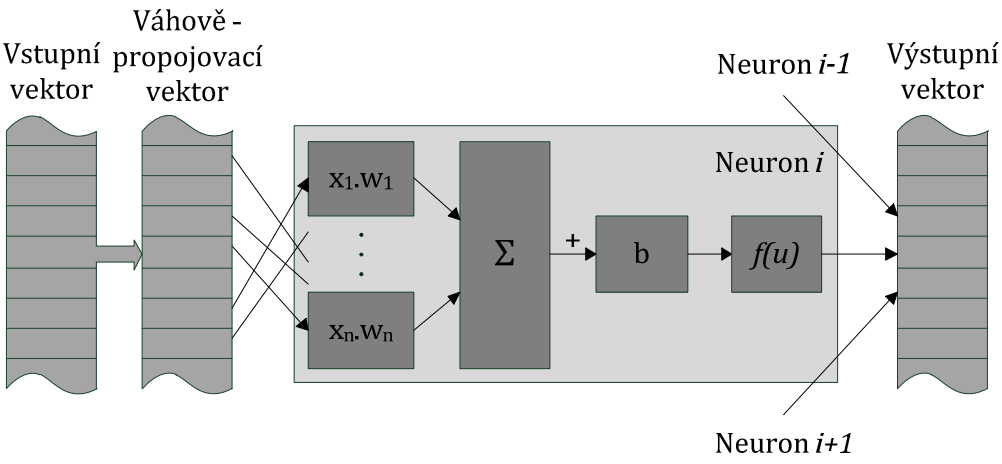
První podkapitoly budou věnovány shrnutí nutných základních poznatků o neuronových sítích v takovém rozsahu, aby bylo na jejich základu možné dále definovat stavební bloky proměnných neuronových sítí. Tato část kapitoly *4* je poněkud odlišná, poněvadž neobsahuje původní autorovy myšlenky, ale známou teorii. Do kapitoly *3* pojednávající o současném stavu problematiky se nicméně nehodí, protože svým obsahem nereflektuje stupeň poznání v oblasti využití „low-end“ embedded systémů pro adaptabilní umělé neuronové sítě, nýbrž jejich samotnou strukturu a podstatu, která je pouze jedním z prostředků, nikoli primárním předmětem této práce.

Druhá část sestavuje uvedené poznatky do vyšších funkčních celků. Budou v ní rozvíjeny teorií podložené návodné myšlenky směřující k finálnímu sestavení neuronové sítě konfigurovatelné za běhu cílového systému. Zejména souhrnu těchto podkapitol tvoří stěžejní část práce na v teoretické rovině. Sledována nebude pouze „přímá cesta“. Tam, kde to bude účelné a možné z hlediska rozsahu práce, budou předkládána i alternativní řešení problémů.

V závěru kapitoly budou popsány základní možnosti řešení časování programových smyček zájmových embedded systémů a zobecněny tak, aby bylo možno definovat způsoby začlenění algoritmu výpočtu sítě do stávajícího firmwaru. Důležitou součástí jsou i požadavky na hardware s následnou diverzifikací řešení a případná omezení či rozšíření plynoucí z absence, respektive prezence, různých možností použitých komponent. V této části dochází k prolínání s předchozím pomyslným celkem textu diskutujícím vhodné alternativy.

4.1 Softwarové komponenty sítě

Základním elementem pro tvorbu neuronové sítě je formální neuron (viz podkapitola *3.2*). Programová realizace vyžaduje jistá doplnění a upřesnění. Vzhledem k tomu, že je cílem implementovat síť s předem neznámou topologií, nikoli samostatně stojící jednotku, je nutné výpočetní jádro doplnit vhodným rozhraním. To představuje systém propojení jednotlivých neuronů, vektor vah a vstupní a výstupní buffery. Schematicky je celá situace stavebního bloku znázorněna na *Obr. 4.1.1*. Část označená **Neuron i** nechť je nadále označována jako *neuronová jednotka*, ostatní bloky jako *mezivrstvové rozhraní*. Jejich přesnější koncepce bude představena dále v textu.



Obr. 4.1.1: Základní stavební blok

Pro programovou implementaci výpočetního jádra je třeba realizovat násobení, sčítání a vyčíslení funkční hodnoty. Problémem cílových embedded systémů je nízký výpočetní výkon a slabá hardwarová podpora těchto operací, stejně jako mnohdy značně omezený rozsah paměti. Zásadním faktorem ovlivňujícím náročnost navrhované aplikace je typ použitých proměnných.

Nejlepší výsledky z hlediska přesnosti sítě jsou dosahovány při použití typů **float** či **double**. Pro simulace na zařízeních vyššího výkonu se používají takřka výhradně. Operace s nimi jsou na procesorech s nativní celočíselnou aritmetikou, které pro aplikaci předpokládáme, velmi komplexní a pro práci v reálném čase nevhodné Tato možnost proto nebude dále uvažována pro praktickou implementaci, nicméně bude základem pro návrh jednoho z vlastních řešení. Druhou variantou je použití celočíselných proměnných typu **char**, **integer**, nebo **long**. Potřebné operace jsou s nimi prováděny řádově rychleji. Sčítání a odečítání typicky v jednom instrukčním cyklu, násobení dle vybavenosti procesoru hardwarovou násobičkou také až s rychlostí odpovídající jednomu instrukčnímu cyklu. Omezen je dynamický rozsah, což má vliv zejména na přesnost

sítě. Třetím, kompromisním a poměrně zajímavým řešením je reprezentace čísel označená jako **pseudo floating point** (11) se všemi možnostmi a důsledky ve zdroji popsanými.

Výpočet *neuronové jednotky* bude dále v textu dělen na *jádro* (*kernel*) realizující výpočet vnitřního potenciálu neuronu **U**, tedy prakticky realizujícího instrukci MAC a výpočet aktivační funkce. Detailněji popisována a rozebírána bude zejména skladba algoritmů na nedostatečně vybavených jednočipových mikropočítačích. Uvedeny budou i příklady realizace od návrhů jednotlivých matematických operací – násobení, sčítání, návrh vlastního datového typu - přes celý *kernel* a aproximaci aktivační funkce až po sjednocení do *neuronové jednotky*.

Mezivrstvové rozhraní je úzce spjato s topologií neuronové sítě. Jeho účelem je navázat vnější vstupy na vstupní neurony, jednotlivé neuronové vrstvy na sebe a výstupy sítě uložit do požadovaného paměťového prostoru. Pro představovaný koncept neuronové sítě s možností „on-line“ učení a změny struktury je důležitá možnost snadné adaptace vah zvolenou výpočetní jednotkou, stejně jako vhodná reprezentace propojení. Konkrétním možnostem realizace bude věnována samostatná teoretická podkapitola, stejně jako praktickému prověření funkčností ukázkovou aplikací se zhodnocením dosažených výsledků.

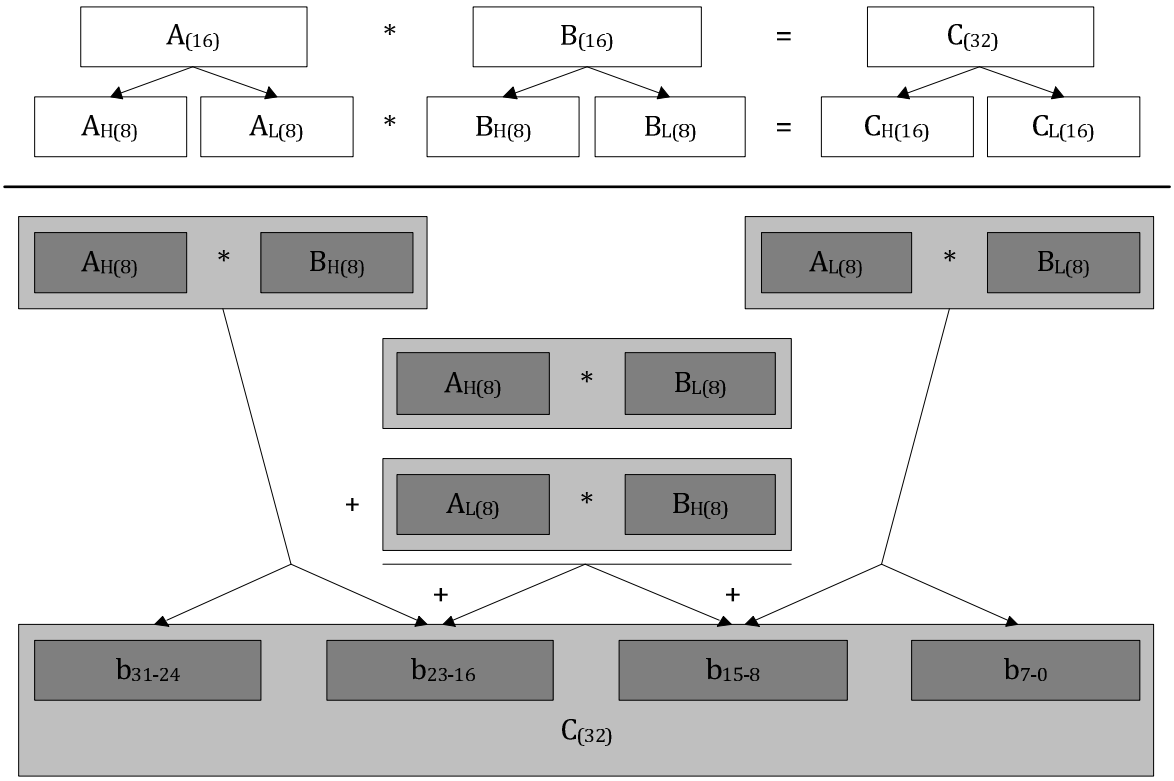
4.2 Operace celočíselného násobení

S rychlými celočíselnými operacemi sčítání nebývá problém. Horší situace je u násobení. Cílové mikrokontroléry můžeme v zásadě rozdělit do tří kategorií:

1) Má využitelný hardware pro násobení postačující dimenze
2) Má využitelný hardware pro násobení s nedostatečnou dimenzí
3) Nemá žádný využitelný hardware pro podporu násobení

První jmenovaný případ je nejpříznivější, pro operace celočíselného násobení lze použít zabudovanou násobičku. Časová náročnost operace je pak předem daná, typově závislá a prakticky, krom vlastní režie obsluhy násobičky, neměnná. Zabývat se dále teoreticky touto možností je neúčelné vzhledem k trivialitě řešení a bohužel nízké předpokládané četnosti výskytu.

Častější jsou situace, kdy mikrokontrolér obsahuje násobičku neumožňující provádět operace s čísly s požadovanou velikostí. Příkladem je hardwarová násobička 8bit x 8bit a násobení rozměru 16bit x 16bit. Řešení je možné provést například podle schématu na *Obr. 4.2.*1, kde **A** a **B** reprezentují 16bitové operandy a **C** 32bitový výsledek.



Obr. 4.2.1: Schéma násobení 16x16 s násobičkou 8x8

Uvedený systém násobení sám o sobě nezohledňuje znaménko. Postup lze rovněž napsat pomocí *Vztah 4.2.1*, kde **C** představuje 32bitový výsledek násobení a $\mathbf{A_H}$, $\mathbf{A_L}$, $\mathbf{B_H}$ a $\mathbf{B_L}$ horní a spodní bajty 16bitových operandů **A** a **B** tak, jak tomu je na předchozím schématu.

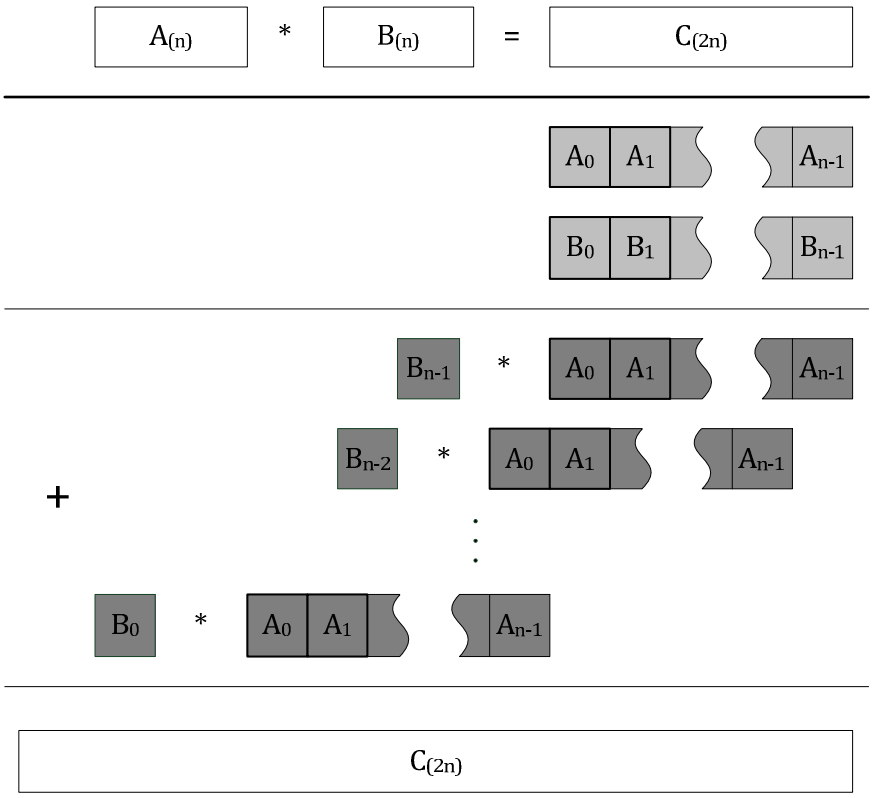
$$C = A_H * B_H * 2^{16} + 2^8 * (A_H * B_L + A_L * B_H) + A_L * B_L \qquad \textbf{Vztah 4.2.1}$$

Teoretická minimální náročnost celé operace bez režie závislé na konkrétní implementaci je shrnuta v *Tabulka 4.2.1*. Uvedené násobení mocninami dvou je nahrazeno bitovou rotací. Zdrojový kód využívající vhodné adresování výsledků na procesoru s 8bitovou architekturou datové sběrnice dokáže počet bitových posunů dále redukovat. Příklady použité pro účely této práce budou uvedeny a diskutovány dále včetně výsledků měření doby trvání.

Druh instrukce	Počet
Násobení 8x8	4
Sčítání 16bit	3
Bitový posun	2

Tabulka 4.2.1: Teoretická náročnost násobení 16x16 s násobičkou 8x8

Poslední možností jsou systémy neobsahující hardwarovou podporu násobení vůbec. Pro jednotlivé součástky zpravidla existují dostatečně kvalitní knihovny matematických operací distribuované výrobcem. Ve většině případů obsahují i ošetření znamének, velikosti a



Obr. 4.2.2: Princip algoritmu "shift and add"

speciálních případů operandů, což je nevýhodné jsou-li tyto ošetřeny jinak. Z hlediska rychlosti se pak může jevit vhodnější tvorba vlastního algoritmu, například „shift and add" založeného na principu „násobení na papíře". Schematicky je znázorněn na *Obr. 4.2.*2. Čísla $\mathbf{A_{(n)}}$ a $\mathbf{B_{(n)}}$ jsou **N**-bitové operandy, $\mathbf{C_{(2n)}}$ výsledek násobení dvojnásobného rozměru **2N** bitů. $\mathbf{A_{(n)}}$ je postupně násoben jednotlivými bity $\mathbf{B_{(n)}}$, což ve dvojkové soustavě znamená nulování nebo násobení jedničkou. V závislosti na řádu jsou dílčí mezivýsledky posouvány o 0 až n-1 pozic vlevo. Výsledkem násobení $\mathbf{C_{(2n)}}$ je součet dílčích mezivýsledků. Algoritmus je popsán *Vztah 4.2.2*.

$$C = \sum_{n=0}^{N-1} (A * B_n) \ll n$$

Vztah 4.2.2

Za předpokladu řešení operace násobení jedničkou a násobení nulou logickou podmínkou, mají-li oba vstupy shodný počet bitů nebo není-li dodatečně rozměr řešen a délka algoritmu odpovídajícím způsobem krácena a je-li zajištěna neznaménkovost, je teoretická náročnost procesu bez vlastní režie vyjádřena v *Tabulka 4.2.2*. Přítomnost posledního řádku „*bitový posun násobitele"* je závislá na instrukční sadě konkrétního procesoru, přesněji na pohodlné možnosti

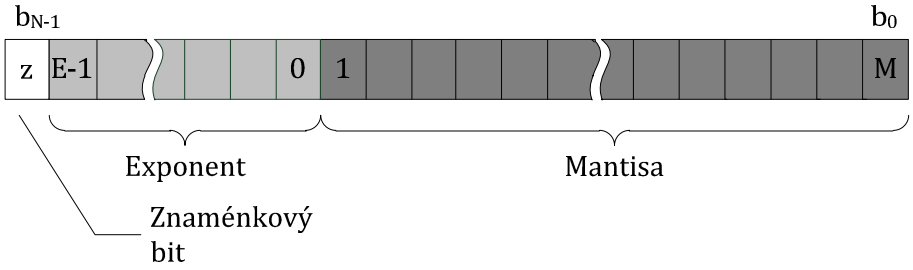
Druh instrukce	Počet
„Násobení“ podmínkou	N
Sčítání n-bit	N-1
Bitový posun mezivýsledku	N-1
Bitový posun nás bitele	*N-1*

Tabulka 4.2.2: Teoretická náročnost "shift and add" algoritmu násobení

testovat nulovost dílčích bitů násobitele. Algoritmus, včetně zdrojového kódu a konkrétních údajů o požadavcích na výpočetní výkon, bude dále rozebírán a na ukázkové aplikaci demonstrován v příslušných praktických kapitolách.

4.3 Čísla s plovoucí řádovou čárkou

Většina výrobců mikroprocesorů a jednočipových mikropočítačů používá pro reprezentaci čísel s plovoucí řádovou čárkou standard IEEE 754-1985, respektive jeho novější revizi IEEE 754-2008 (22). Proměnné s plovoucí řádovou čárkou mají oproti celočíselné reprezentaci výhodu ve větším dynamickém rozsahu. Nevýhodou je omezená přesnost patrná zejména u vyšších hodnot a hodnot s periodickým rozvojem za desetinou čárkou. Zápis čísla s plovoucí řádovou čárkou v paměti je nejběžněji v souladu s výše uvedeným standardem prováděn tak, jak je znázorněno na *Obr. 4.3.*1, kde **z** je znaménkový bit, **E** počet bitů exponentu, **M** počet bitů mantisy a **N** celkový počet bitů proměnné. V části zaznamenávající mantisu vzhledem ke tvaru, v jakém je číslo reprezentováno a vyplývajícím záporným mocninám základu binární soustavy – čísla **2**, opačné indexování s počátkem v **1**. Bit $\mathbf{b_0}$ má index **m**=**M**, bit $\mathbf{b_{M-1}}$ má index **m**=**1**.



Obr. 4.3.1: Záznam čísla s plovoucí řádovou čárkou v paměti

Velikosti jednotlivých úseků jsou pro níže uvažované 32bitové a 64bitové proměnné včetně použitelných hodnot spřažených parametrů uvedeny v *Tabulka 4.3.1*. Efektivní rozsah exponentu je z obou stran snížen o 1 díky speciálnímu významu čísel s exponenty složenými pouze z „1“, nebo pouze z „0“.

Celkový počet bitů	Počet bitů exponentu E	Efektivní rozsah exponentu	Počet bitů mantisy M	Rozsah mocninných indexů mantisy
32	8	-127 až 126	23	-23 až -1
64	11	-1023 až 1022	52	-52 až -1

Tabulka 4.3.1: Parametry čísel s plovoucí řádovou čárkou

Převod takto zaznamenaného čísla na dekadický desetinný zápis **C** je možný dle *Vztah 4.3.*1.

$$C = -1^{z}\left(1, \sum_{m=1}^{M} b_{M-m} * 2^{-m}\right) * 2^{Exp}$$

$$Exp = \left(\sum_{e=0}^{E} b_{M+e} * 2^{e}\right) - 128$$

Vztah 4.3.1

Srovnání nejběžněji používaných 32bitových (*single precision floating point*) a 64bitových (*double precision floating point*) proměnných s celočíselnými typy zabírajícími shodný paměťový prostor je uvedeno v *Tabulka 4.3.2*. Pro výpočet absolutního rozsahu celočíselných hodnot byly vzhledem k orientaci této práce na algoritmy s nízkou náročností na výpočetní výkon a z ní vyplívající nevhodnosti znaménkové reprezentace čísel zvoleny proměnné neznaménkové. Naproti tomu u typů s plovoucí řádovou čárkou je respektována výše zmíněná forma včetně znaménkového bitu. Do výpočtů v tabulce je tedy zahrnuta pouze kladná polovina rozsahu. Jako základ pro výpočet dynamického rozsahu byla vždy uvažována nejmenší nenulová hodnota vyjádřitelná daným typem.

Typ proměnné	Počet bitů	Absolutní rozsah	Dynamický rozsah
S plovoucí čárkou	32	$1*2^{-127}$ až $\sim(2-2^{-23})*2^{126}$	764, 2dB
S plovoucí čárkou	64	$1*2^{-1023}$ až $\sim(2-2^{-52})*2^{1022}$	6159,1dB
Celočíselná	32	0 až 4294967296	96,32dB
Celočíselná	64	0 až 18446744073709551616	192,66dB

Tabulka 4.3.2: Srovnání celočíselných proměnných a proměnných s plovoucí řádovou čárkou

Jak je patrné z výše uvedených údajů, dokáží proměnné s plovoucí řádovou čárkou na daném počtu bitů pokrýt znatelně vyšší rozsah hodnot, než celočíselné typy. Problém ovšem

nastává s matematickými operacemi na výpočetních jednotkách, které nejsou vybaveny příslušnou hardwarovou akcelerací. Triviální již zřejmě nejsou ani algoritmy sčítání a odečítání složené z celočíselných instrukcí. Operace vyžadují sjednocení exponentů a následnou úpravu výsledku do tvaru, kde platí: $1 \leq Mantisa < 2$. Obdobně je tomu i u násobení a dělení. Další konkrétnější informace o problematice spojené s čísly s plovoucí řádovou čárkou lze nalézt velmi podrobně v (23).

Výhodným řešením rozšiřujícím dynamický rozsah proměnné na úkor přesnosti vyšších hodnot a přitom zachovávajícím jednoduchost potřebných základních operací může být definice speciálního (ve smyslu „odlišného od standardu IEEE 754“) typu s plovoucí řádovou čárkou. Pro potřeby této práce byla čerpána inspirace z (9), kde je pod pojmem *„Pseudo floating-point“* zavedena 24bitová znaménková proměnná s 16bitovou mantisou a 8bitovým exponentem. Další text se bude na tento typ proměnné odkazovat zkratkou **PFP**. Nechť je dále rovněž zaveden pojem *„Vlastní datový typ“*, ve zkratce **VDT**, označující proměnnou s plovoucí řádovou čárkou níže popsaného formátu optimalizovaného pro představovanou metodu.

4.4 Vlastní datový typ

V souladu s poznatky uvedenými v podkapitole *4.3* mějme proměnnou **VDT** s plovoucí řádovou čárkou zapsanou ve formátu odpovídajícím *Vztah 4.3.1*. Vzhledem k předpokládanému použití výpočetních systémů s hardwarovou podporou pouze celočíselných operací násobení, sčítání a odečítání v 16bitové, nebo častěji 8bitové podobě, zřídka se speciálními instrukcemi typickými pro signálové procesory, jako je MAC, je třeba zavést následující požadavky jak na formát a umístění v paměti, tak na podobu a jednoduchost podpůrných algoritmů a taktéž na následné použití v aplikaci:

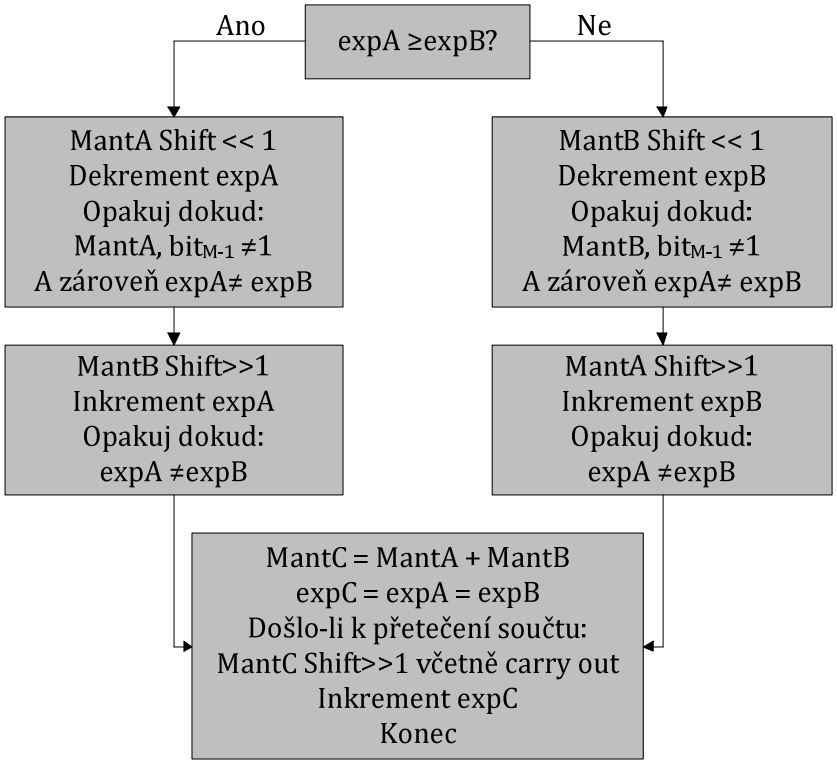
1) **VDT** musí obsadit v paměti takový počet bitů, který je násobkem 8mi, avšak není větší, než 32
2) Při stejné velikosti musí **VDT** disponovat znatelně větším dynamickým rozsahem, než celočíselná proměnná
3) Operace násobení a sčítání **VDT** nesmějí být výrazně výpočetně náročnější, než operace pro celočíselné typy se stejným počtem bitů
4) Musí existovat jednoduchý a rychlý algoritmus převodu mezi **VDT** a celočíselnými typy
5) Musí existovat jednoduchý a rychlý algoritmus převodu **VDT** s různými exponenty na **VDT** se shodnou hodnotou exponentů

Vytvářet, dle bodu *1)* 8bitový **VDT** nemá význam, neboť evidentně nedojde na 8 a více bitových procesorech k úspoře výpočetního výkonu u operací násobení, sčítání ani odečítání. Dynamický rozsah a přesnost takového čísla, kde mantisa zřejmě nemůže mít více, než 7bitů teoreticky a přibližně 5 až 6 bitů prakticky, aby mělo význam doplňovat exponentovou část, jsou nepřijatelné. 16bitové, 24 a 32bitové proměnné, dále značené jako **VDT16**, **VDT24** a **VDT32**, již smysl mají. Nechť je definován typ **VDT24**.

Operace násobení u čísel s plovoucí řádovou čárkou formálně nepředstavuje výraznější problém, než násobení celočíselné. Pro součin **C=A*B** je obecně platný předpis dle *Vztah 4.4.1*, kde **r** je základ číselné soustavy, **a** a **b** jsou mantisy a **expA** a **expB** jsou exponenty.

$$A = a * r^{expA};\ B = b * r^{expB}$$
$$C = A * B; C = a * b * r^{expA+expB}$$ **Vztah 4.4.1**

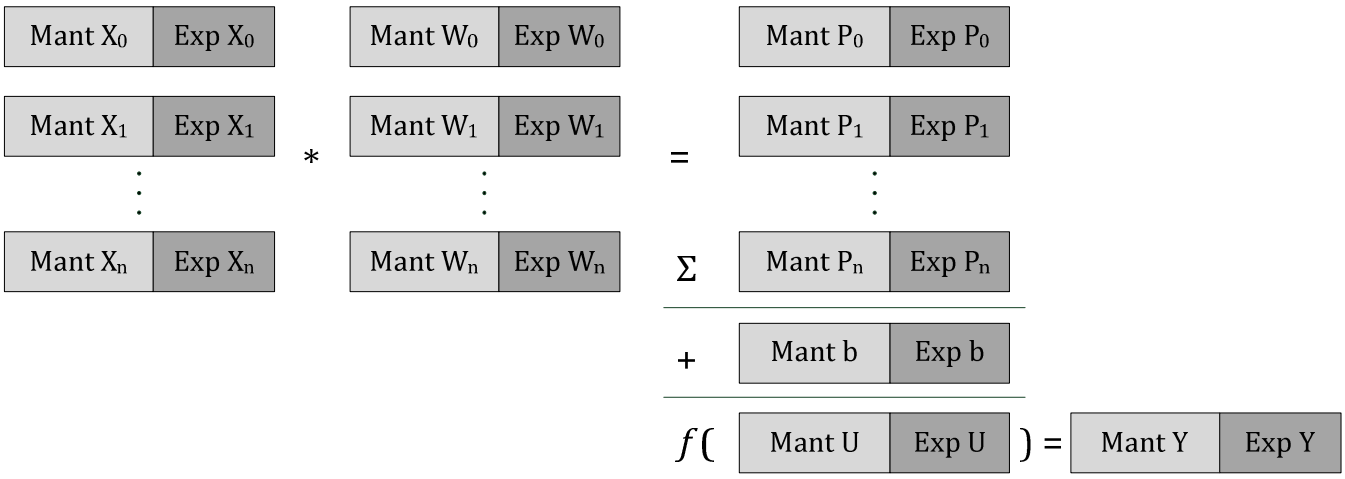
Naproti tomu operace sčítání a odečítání je na „celočíselné instrukční sadě“ značně komplikovanější. Mezitím co stejná hodnota základu soustavy **r** podmiňující násobení je v rámci systému takřka vždy samozřejmostí, shodu **expA** a **expB** nutnou pro sčítání již automaticky předpokládat nelze. První univerzálnější avšak náročnější možností je zavést operaci převodu exponentů algoritmicky zpracovatelnou například tak, jak je znázorněno na *Obr. 4.4.*1. Druhou, méně náročnou možností je zavést ve vhodných bodech daného systému normování na společný exponent tak, aby byla čísla pro sčítání předpřipravena. Referenční disertační práce (9) a relevantní prameny v ní citované používají normalizaci exponentu **PFP** čísla na jednotnou hodnotu již ve stádiu návrhu koeficientů neuronové sítě. Exponent, shodný pro všechny váhové koeficienty neuronu, je uložen v jednom paměťovém místě, čímž dojde k úspoře systémových prostředků. Tento přístup je ovšem značně nevhodný, neboť degraduje dynamický rozsah hodnot vah v rámci neuronu na úroveň 16bitové celočíselné hodnoty. Má-li být navíc opatření



Obr. 4.4.1: Algoritmus pro srovnání exponentů s maximálním zachováním přesnosti

pro zjednodušení sčítání účinné, je třeba značně omezit jednak rozsah mantisy a jednak rozsah vstupních hodnot tak, aby nebylo třeba další přizpůsobování exponentu produktu násobení a akumulační proměnné.

Aby mohl být navržen vhodný formát **VDT** s balíkem potřebných operací optimalizovaných na rychlost při zachování maximální možné přesnosti, je třeba zaměřit se na funkci v konkrétní aplikaci. Budeme-li pro praktické testy v rámci této práce volit 24bitovou proměnnou **VDT24** s 16bitovou mantisou a 8bitovým exponentem, můžeme – s ohledem na matematické operace - detailněji rozvést jádro z *Obr. 4.1.*1 do podoby *Obr. 4.*4.2.



Obr. 4.4.2: Matematické operace výpočtu výstupní hodnoty neuronu

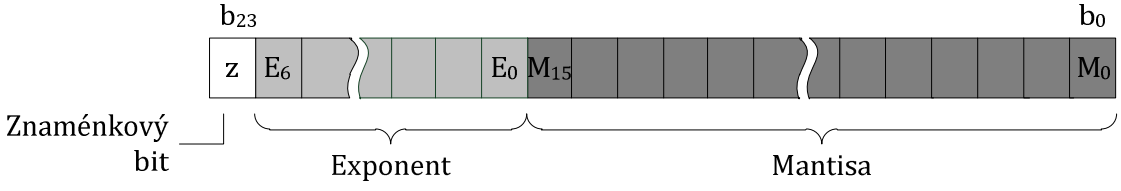
Vzhledem ke kaskádnímu řazení neuronů ve struktuře sítě je základním požadavkem srovnatelný rozměr vstupů **X** a výstupů **Y**. Zároveň je nutné definovat takový definiční obor a obor hodnot funkce **f(u)**, aby byla realizovatelná jak z hlediska výpočetní náročnosti, tak

z hlediska obsazení paměti mikrokontroléru. Disertační práce (9) připouští pouze celočíselné vstupy a výstupy s délkou 16bitů, což rovněž omezuje přesnost a potlačuje smysl použití čísel s plovoucí řádovou čárkou. Podrobnější popis dále navrhovaného přístupu spolu s praktickými omezeními bude následovat v dalším textu a kapitolách, stejně jako bližší zdůvodnění a konkretizace volených rozsahů. Nyní předpokládejme definiční obor **<-65535*2^63;65535*2^63>** a obor hodnot **<-65535;65535>**. Přesněji definovaná „pracovní oblast" nechť se pohybuje ve vstupním intervalu **<-65535;65535>**. Nelinearita funkce mimo tyto meze může být řešena například vhodným převedením exponentu **U** na mantisu **Y** a částečným zanedbáním mantisy **U**.

Z uvedeného plynou výrazná zjednodušení spojená s úsporami výpočetního výkonu při určování výstupního potenciálu neuronu **Y** a požadavky na konkrétní podobu **VDT24** a operací s ním:

1) Podmínkou nutnou pro následující zjednodušení je držet všechna čísla v podobě s nenulovým MSB mantisy.
2) V situaci, kdy kterýkoliv ze součtů exponentů **Exp X_n + Exp W_n > -16**, bude exponent **Exp P > 0** a při dodržení bodu 1) bude **Exp U > 0** i při maximálním možném zarovnání mantisy směrem k MSB a celočíselný ekvivalent výsledku bude zjevně větší, než **2^{16}**. Při vhodně řešené aktivační funkci již není třeba provádět kompletní sumu produktů **P**.
3) Násobení je třeba provádět co nejjednodušeji pomocí celočíselných prostředků. Znaménkový bit mantisy nechť je proto součástí exponentu.
4) Při dodržení bodu 1) a 3) lze při násobení očekávat hodnoty produktu v rozmezí **Mant P = <0x4000;0xFFFE>** a **Exp P = Exp X + Exp W + 16**, respektive **<0x8000;0xFFFE>** pro **Exp P = Exp X + Exp W + <15;16>**, což přispívá k jednoduchosti udržení podmínky 1)

Předložený princip vede ke struktuře typu **VDT24** uvedené na *Obr. 4.4.*3 formálně podobné typům diskutovaným v kapitole *4.3.* Znaménko mantisy je součástí exponentu. Hlavním pozitivním důsledkem je možnost užití algoritmu celočíselného neznaménkového násobení,



Obr. 4.4.3: Struktura VDT24 a rozložení v paměti

přičemž znaménkový bit výsledku je určen exkluzivním součtem znaménkových bitů operandů. U akumulace výsledku je podle něj rovněž možné řídit volání operace sčítání/odečítání, aniž by bylo nutné mantisy záporných čísel převádět na nějakou formu záporného čísla celého. Samotná hodnota exponentu je vyjádřena obdobným způsobem, jako ve standardu (22). Zejména sčítání a odečítání při bitové rotaci mantisy se v platném rozsahu **VDT24** obejde bez rizika přetečení, nebo podtečení do MSB (reprezentujícího znaménko mantisy). Na rozdíl od (z pohledu IEEE 754) standardního zápisu čísel s pohyblivou řádovou čárkou mantisa **VDT24** reprezentuje celou cifru, nikoli pouze její desetinou část s nevyjádřenou jednotkou. Dojde k částečnému zjednodušení bitové rotace při zarovnávání pro součet a rozdíl.

Pro převod na dekadické desetinné číslo je platný *Vztah 4.4.2*. Oproti dříve uvedenému *Vztah 4.3.1* pro datový typ **float** a **double** je patrná absence „jedničky“ před sumou mantisy a změna znaménka čísla bitu **m** v mocnině **2**. Zbytek zápisu je opět formálně stejný. Přehled parametrů **VDT24** je uveden v *Tabulka 4.4.1*. Dynamický rozsah udává poměr mezi nejmenším a největším reprezentovatelným číslem v kladném rozsahu. Rozdíl mezi teoretickým a reálným je způsoben výše zavedeným zarovnáním snižujícím rozsah mantisy (bez prvku „0“) na interval **<32768;65535>**, jehož zachovávání je v případě čísel blízkých nule omezující, nicméně další algoritmické ošetření by mělo za následek zvýšení nároků na výpočetní výkon a bylo by tak v přímém rozporu s požadavky na **VDT24**.

$$C = -1^{z}\left(\sum_{m=1}^{15} b_{15-m} * 2^{m}\right) * 2^{Exp}$$

$$Exp = \left(\sum_{e=0}^{6} b_{16+e} * 2^{e}\right) - 64$$

Vztah 4.4.2

Celkový počet bitů	Počet bitů exponentu E	Efektivní rozsah exponentu	Počet bitů mantisy M	Rozsah mocninných indexů mantisy
24	7	-64 až 63	16	0 až 15

Absolutní rozsah	Dynamický rozsah teoretický	Dynamický rozsah reálný
-65535*2^{63} až 65535*2^{63}	430,5dB	385,3dB

Tabulka 4.4.1: Parametry datového typu VDT24

Slabinou vytvořeného datového typu nadále zůstává sčítání a odečítání, kde je nevyhnutelné zarovnání na shodný exponent. Náročnost na výpočetní výkon je závislá na instrukční sadě jádra, konkrétně na možnostech bitového posunu vždy o 1 / o zvolený počet. Při práci s 8bitovými procesory ještě roste režie s nutností řešit „carry“ a „borrow“ při přechodech mezi jednotlivými bajty mantisy. Příklad realizace operací násobení a sčítání **VDT24** je uveden,

obdobně jako u celočíselného násobení, dále v textu a to včetně změřených parametrů a odkazu na zdrojový kód.

4.5 Násobení a sčítání VDT24

Operace násobení **VDT24** je prováděna podle *Vztah 4.4.1* uvedeného v podkapitole *4.4*. Úkonem předcházejícím násobení je detekce nulovosti jednoho z operandů, jejíž pozitivní vyhodnocení vede k okamžitému přerušení operace s výsledkem nulovým na úrovni mantisy i exponentu. Formálně správně, pokud není stav ošetřen jinak, je třeba vyhodnotit potenciální přetečení exponentu výsledku. Příklady užití násobení **VDT24** uvedené v této práci situaci přímo neřeší, neboť je jeho účinnou prevencí ošetření rozsahu vstupních hodnot, oboru funkčních hodnot aktivační funkce (vstupy následující vrstvy neuronové sítě) a vhodné řešení adaptace vah. Při zachování dříve uvedených pravidel, zejména o zarovnání na nenulový MSB, je možný následující návrh algoritmu:

1) Přímé násobení mantisy operandů **A** a **B**. Při použití algoritmu násobení celých čísel rozměru 16x16 za podpory násobičky 8x8 popsaného v podkapitole *4.2* je možné zcela vypustit násobení bajtů $\mathbf{A_{(L)}}$ * $\mathbf{B_{(L)}}$ a zredukovat adici produktů násobení $\mathbf{A_{(L)}}$ * $\mathbf{B_{(H)}}$ a $\mathbf{A_{(H)}}$ * $\mathbf{B_{(L)}}$ ve smyslu zanedbání L-bytu jejich produktu. Dojde tak k výrazné úspoře systémových prostředků za cenu omezení přesnosti na úrovni posledních dvou bitů celkového produktu, tedy hodnoty maximálně 3 z čísla o minimální velikosti 32768.
2) Stanovení znaménka výsledku násobení **C**, pro které lze použít exkluzivní bitový součet exponentů operandů **ExpA** a **ExpB** s maskou logického součinu o šestnáctkové hodnotě 0x80 ponechávající v bajtu exponentu pouze znaménkový bit mantisy.
3) Stanovení velikosti exponentu výsledku **ExpC** vzorcem **ExpC = ExpA + ExpB – 0x40 + 0x10**. Tento řeší jak odečtení offsetu velikosti 64, který je zahrnut v obou exponentech operandů, tak i přičtení konstanty 16 kompenzující zahození spodních dvou bajtů výsledku násobení mantis.
4) Případné zarovnání výsledku o 1bit vlevo s patřičnou úpravou exponentu (zachování formátu s nenulovým MSB) je-li to z hlediska přesnosti účelné pro další zpracování násobením. Tento krok, zvláště v případě, kdy nejsou produkty násobení dále násobeny, je možné na úkor přesnosti opomenout.

Teoretická výpočetní náročnost vyplývající z uvedených bodů je shrnuta v *Tabulka 4.5.1*. Uvažována je jak formálně kompletní, tak výkonnostně optimalizovaná varianta. Pro zvýraznění rozdílů je uvažován systém s hardwarovou podporou násobení do rozměru 8x8bitů. Taktéž je detailněji rozveden algoritmus celočíselného násobení. Je předpokládána eliminace násobení pro posun dílčích mezivýsledků násobení 8x8 čísly 256 a 65536 (respektive rotací o 8, nebo 16 bitů vlevo) prostřednictvím vhodného adresování tak, jak bylo naznačeno v podkapitole *4.2* a jak je následně provedeno v praktické ukázce.

Kompletní varianta		**Redukovaná varianta**	
Druh instrukce	**Počet**	**Druh instrukce**	**Počet**
Násobení 8x8	4	Násobení 8x8	3
Sčítání 16+16bit	3	Sčítání 16+8bit	2
Součty nad exponenty	3	Součty nad exponenty	3
XOR znaménka	1	XOR znaménka	1
Rotace výsledku	Max. 1		
Inkrement exponentu	Max. 1		

Tabulka 4.5.1: Výpočetní náročnost násobení VDT24 na 8bit procesoru

Součet **VDT24**, stejně jako všech standardních typů s plovoucí řádovou čárkou na jednotkách bez nativní hardwarové podpory, vyžaduje nejprve převod operandů na čísla shodných rozměrů, následné vyhodnocení znaménka a provedení příslušné celočíselné operace. Je třeba přitom zohlednit i maximální možnou délku mantisy výsledku **n+1**, kde **n=16** pro **VDT24**. Tím v rámci algoritmu potenciálně narůstá počet vykonávaných operací o další bitovou rotaci výsledku a inkrementaci exponentu. Na procesorech používajících zápis záporného čísla ve formátu dvojkového doplňku je také nutné řešit případný převod na absolutní hodnotu se znaménkovým bitem v bajtu exponentu. Samotný algoritmus lze shrnout do bodů:

1) Porovnání exponentů operandů. V případě shody je možné postupovat ihned k dalšímu bodu. V opačném případě zarovnání menšího na rozměr většího bitovou rotací mantisy vpravo a inkrementací exponentu. Je-li předpokládáno časté sčítání řádově velmi odlišných čísel, je možné potenciálně uspořit výpočetní výkon detekcí vstupů s exponenty vzájemně odlišnými o 16 a více, pro které je výsledek rotace mantisy roven nule a celý algoritmus je možné ihned ukončit.
2) Rozdělení následující akce dle exkluzivního součtu znamének na sčítání nebo odečítání.

3) Provedení operace sčítání nebo odečítání nad 16bitovým registrem s doplňkovým bitem carry/borrow a případné zarovnání mantisy výsledku a s ním spojená inkrementace/dekrementace exponentu na platný tvar **VDT24**. V případě, že je výsledek součtu záporný a z principu činnosti použitého hardwaru ve formátu dvojkového doplňku, je třeba jej převést na absolutní hodnotu a znaménkový bit.

Minimální požadavky na výpočetní výkon u systému obdobného, jako byl uvažován pro násobení **VDT24**, jsou shrnuty v *Tabulka 4.5.2*. K samostatné operaci sčítání a odečítání (respektive součtu se záporným číslem) není uvedena praktická ukázka, neboť je v rámci cílů vytyčených v práci používána pouze v podobě akumulace dílčích výsledků násobení do akumulačního registru, což s sebou nese jisté odlišnosti a zjednodušení, která budou zmíněna dále v textu. Parametry sčítání **VDT24** jsou proto k dispozici pouze jako součást výsledku pro soubornou funkci násobení a akumulace.

Druh instrukce	Počet
Bitová rotace vpravo	0 až 15
Inkrement exponentu	0 až 15
Součet/rozdíl 16bit s carry/borrow	1
Dvojkový doplněk 16bit	0 až 1
XOR znaménka výsledku	1
Bitová rotace vlevo	0 až 15
Dekrementace exponentu	0 až 15

Tabulka 4.5.2: Výpočetní náročnost sčítání VDT24 na 8bit procesoru

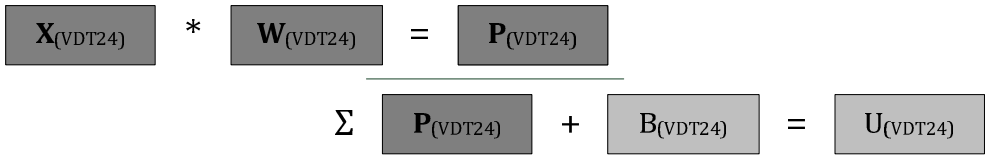
4.6 Výpočet vnitřního potenciálu neuronu

Při zachování názvosloví uvedeného v podkapitole *4.1* se jedná o tu část *výpočetní jednotky neuronu*, která realizuje násobení *vektoru vstupů* s *vektorem vah*, sumu jednotlivých prvků výsledného vektoru a přičtení klidového potenciálu. Tato funkce nechť je nadále označována jako *kernel,* nebo *jádro*.

Při zavádění umělých neuronových sítí na DSP nebo vyspělejší MCU s MAC instrukcí požadovaných parametrů je tato úloha snadná, výpočetně nenáročná a je neúčelné se jí dále zabývat. Cílem této práce je navíc popis metody použitelný právě pro jednotky bez takovéto podpory. Informace, včetně výsledků uvádějících rozdíly v náročnosti na výpočetní výkon, lze

nalézt například ve (24). V závislosti na požadovaném poměru přesnosti, rychlosti a dynamického rozsahu výpočtů v kombinaci se zvoleným hardwarem lze volit různé varianty od nejpomalejší při použití standardních čísel s plovoucí řádovou čárkou až po nejrychlejší, tedy celočíselnou s nízkým počtem bitů na proměnnou. V rámci této kapitoly budou blíže představeny dvě méně standardní varianty – *kernel VDT24* používající definovaný datový typ a *kernel HINT16* – hybridní výpočet využívající rychlosti celočíselných operací a rozsahu **VDT24**. V příslušné podkapitole bude opět uvedena praktická ukázka s odměřenými parametry obou metod.

Schéma řešení *kernelu VDT24* přináší *Obr. 4.6*.1. Absolutní rozsah platných prvků vstupních a váhových vektorů **X** a **W** je v celém pásmu hodnot **VDT24**. Výsledný prvek vektoru



Obr. 4.6.1: Operace prováděná kernelem VDT24

dílčích výsledků **P** (pomineme-li operaci násobení nulou) je saturován na hodnotu $\pm 32768*2^{-64}$, nebo $\pm 65535*2^{63}$ reprezentující kladnou a zápornou limitní nulu a kladné a záporné nekonečno. Součiny s nulovými prvky pak generují číslo s mantisou o hodnotě 0. Sumace výsledného vektoru **P** je prováděna až po dokončení všech operací násobení a to z důvodu stanovení exponentu výsledku **ExpU** - jakožto maxima z exponentů **ExpP** všech prvků vektoru, na jehož hodnotu je třeba všechny produkty dílčích násobení zarovnat.

Provedení vlastního součtu vyžaduje ve fázi návrhu algoritmu zohlednění zarovnání formátu **VDT24** s nenulovým MSB. Výsledek sčítání dvou prvků z vektoru **P** o **p** bitech může vést a v tomto případě vždy povede k maximálně **u**-bitovému výsledku, přičemž platí **u=p+1**. Bitová velikost mantisy akumulátoru **U** proto musí být minimálně **16+n+1**, kde **n** je počet vstupů, číslo **16** reprezentuje standardní délku mantisy a **1** je příspěvek od klidového potenciálu **B**. Řešení součtu kladných a záporných čísel s ohledem na rychlost výpočtu je vhodné řešit offsetem akumulátoru o hodnotě $\mathbf{2^u}$, přičemž musí být za tímto účelem k akumulátoru přidán ještě jeden doplňkový bit. Dle znaménka čísla **P** potom dojde k volání funkce sčítání nebo odečítání. Po ukončení algoritmu je třeba odečíst offset akumulátoru například funkcí exkluzivního součtu nejvyššího bitu a v případě záporného výsledku ještě provést operaci dvojkového doplňku a převedení znaménkového bitu do odpovídající části **VDT24**. Vzhledem k tomu, že takováto funkce je, kromě samotných matematických operací, značně zatížena vlastní režií v podobě pomocných úkonů, podmínek a větvení programu, jejichž trvání nelze bez znalosti specifikace konkrétního hardwaru předem stanovit, nemá smysl detailněji rozebírat výpočetní náročnost tak jako tomu bylo dříve v textu v případě dílčích funkcí násobení. Empiricky zjištěné hodnoty

pro zvolený systém, včetně popisu použitého algoritmu, jsou k dispozici v podkapitolách kapitoly *5*.

Stěžejním problémem funkcí výpočtu vnitřního potenciálu neuronu s plovoucí řádovou čárkou obecně (v tomto případě typu *kernel VDT24*)je komplikované řešení akumulace dílčích produktů **P** násobení vstupů **X** s váhami **W** bez hardwarové podpory operací sčítání a odečítání čísel s plovoucí řádovou čárkou z důvodů řečených v podkapitolách *4.2*, *4.3* a *4.4*. Celočíselné operace (stejně jako formálně shodné operace desetinných čísel s pevnou řádovou čárkou) jsou naproti tomu problematické z hlediska násobení, respektive nevýhodného poměru dynamického rozsahu a výpočetní náročnosti. Přitom zachovávají absolutní přesnost výsledku, což je pro účely výpočtu neuronových sítí na embedded systémech nízkého výkonu zbytečné. Disertační práce (9) problém řeší již zmiňovaným způsobem s diskutovanou nevýhodou – datovým typem **PFP**. Zde bude popsán vlastní algoritmus výpočtu vnitřního potenciálu neuronu *kernel HINT16* s ním srovnatelný.

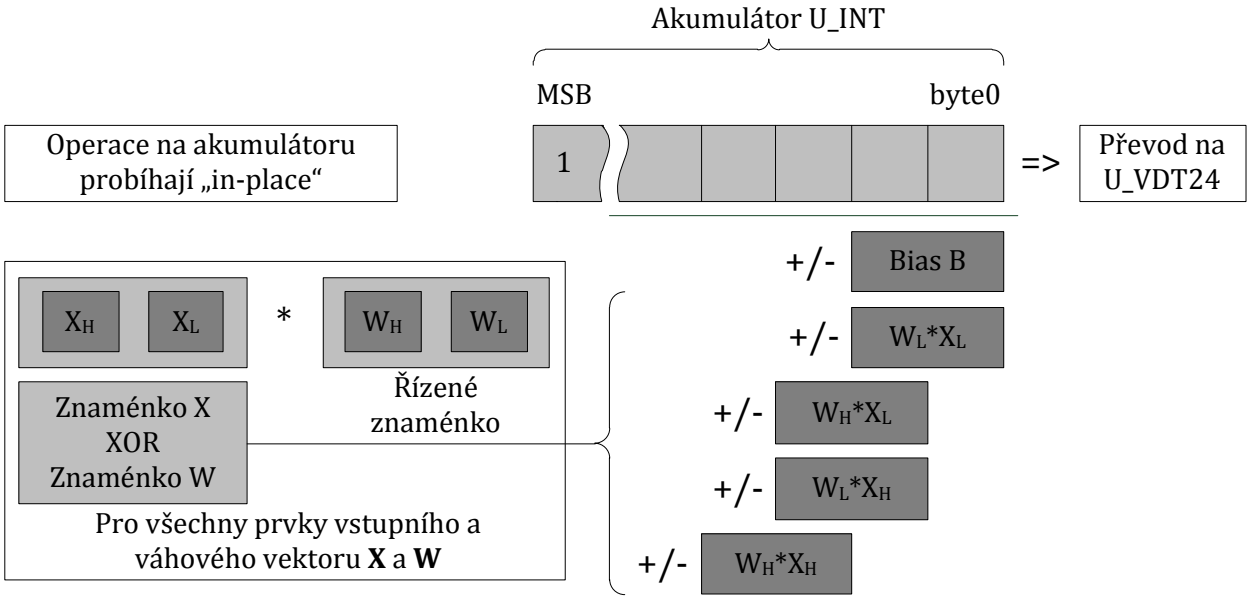
Principiální grafické schéma navrhovaného algoritmu je na *Obr. 4.6.*2. Jak napovídá název, jedná se o hybridní celočíselné jádro pracující se vstupním vektorem **X** a váhovým vektorem **W** složenými z celočíselných 16bitových proměnných. Operace násobení, na rozdíl od **VDT24** nebo **PFP** je prováděna kompletně s 32bitovým výsledkem. Úspora výpočetního výkonu

Obr. 4.6.2: Operace prováděná kernelem HINT16

probíhá díky absenci exponentu. Následná akumulace dílčích výsledků může probíhat bez mezivýsledků, přímo do akumulátoru bitové délky **u**, kde pro minimální velikost $\mathbf{u_{min}}$ poskytující takový prostor, který vylučuje přetečení výsledku, platí nerovnost $\mathbf{u_{min} \geq n+\log_2 poc}$, ve které je **n** bitová délka akumulovaných proměnných a **poc** označuje jejich celkový počet. Vzhledem k tomu, že počet bitů musí být celé číslo, je třeba za výsledek považovat nejbližší vyšší celočíselnou hodnotu.

Z hlediska *kernelu* je hodnota akumulačního registru výstupem. Součin vstupního a váhového vektoru generuje 32bitové hodnoty, což je zároveň i spodní hranice rozměru akumulátoru. Z pohledu celého výpočetního jádra neuronu se však jedná o vstup do aktivační funkce, která by v odpovídajícím tvaru buď spotřebovávala množství systémových prostředků neúměrné parametrům procesoru, nebo by nevyužívala přesnosti a rozsahu tak velké vstupní hodnoty, což rovněž vede k neopodstatněným požadavkům. V tomto okamžiku navrhovaný systém s výhodou používá převodu na formát **VDT24** a z čistě celočíselného se stává hybridní.

Při zavádění na určitý typ hardwaru je nutné rovněž zvážit možnosti znaménkových operací daného systému vzhledem k přirozené reprezentaci znaménkových celočíselných proměnných. Procesory z horní části spektra zájmu této práce vybavené 16bitovými instrukcemi MAC s dostatečně dimenzovaným akumulátorem budou *kernel HINT16* řešit bez větších problémů a s minimální režií při použití svého nativního formátu celých čísel. V případě jednotek bez MAC instrukce, podpory znaménkového násobení 16x16bitů a běžně používaného dvojkového doplňku je ovšem výhodnější převést vstupní operandy násobení na formát absolutní hodnoty se znaménkovým bitem na pozici MSB. Naroste vlastní režie *kernelu HINT16* o exkluzivní součet znamének operandů a následnou podmínku větvení na operaci přičtení k akumulátoru, nebo odečtení od akumulátoru. Rovněž stoupne obsazení programové paměti vlivem rozdělení na větev násobení se sčítáním a násobení s odečítáním. Vznikají ovšem dvě stěžení výhody v podobě oproštění od znaménkového násobení 16bitových čísel na zařízení k tomu nevybaveném a nutnost separátního ukládání produktu násobení za účelem doplnění záporných čísel ve dvojkovém doplňku na rozměr akumulátoru. *Kernel HINT16* v navrhované podobě může mít strukturu dle *Obr. 4.6*.3.



Obr. 4.6.3: Schéma výpočtů pro kernel HINT16

Vstupní a výstupní rozsah obou prezentovaných metod lze odvodit z parametrů používaných datových typů a vlastností akumulačního registru. Pro *kernel VDT24* s akumulačním registrem typu **VDT24** je stěžejní právě jeho rozsah, jinými slovy – vektor vstupů a vektor vah musí být limitován takovým způsobem, aby nedošlo k vybočení z rozsahu ani u jednotlivých součinů (což se týká nejen čísel s vysokým, ale také s nízkým exponentem), ani u akumulátoru. Je-li předpokládán navrhovaný rozsah vstupů v ekvivalentech celočíselných hodnot od 0 do 65535, respektive -32768 až 32767, je možné rozsah vah a akumulátoru

dopočítat. V případě užití celého rozsahu **VDT24** pro vektor **X** již situace není triviální a je třeba doplnit další algoritmy vyhodnocení platnosti dat za cenu zvýšení výpočetních nároků. V případě *kernelu HINT16* jsou vektory **X** i **W** omezeny na hodnoty 0 až 65535, respektive -32768 až 32767, nebo -32767 až 32767 při navrhovaném použití znaménkového bitu. Akumulátor je celočíselný a má dostatečnou kapacitu pro obsažení součtů všech dílčích součinů. Limitujícím faktorem pro rozsah funkce proto není. Finální převod na formát **VDT24** teoreticky limituje maximální velikost celočíselného akumulátoru na 79+1bitů (počet bitů pro celočíselný ekvivalent maxima **VDT24**), jehož přetečení by mohlo být dosaženo až při nereálném počtu vstupů neuronu. Přehled hranic zpracovatelných hodnot je uveden v *Tabulka 4.6.1*. Pro zvýšení vypovídací hodnoty jsou v ní funkce navíc rozděleny na bipolární a unipolární.

Výpočetní funkce	*Kernel VDT24*		
Typ	Unipolární	Bipolární	
Vstupní rozsah	$0 \cup \langle 32768*2^{-15};65535*2^{0}\rangle \in \mathbf{N^0}$	$\langle -32768*2^{0};-32768*2^{-15}\rangle \cup 0 \cup \langle 32768*2^{-15};32768*2^{0}\rangle \in \mathbf{Z}$	
Rozsah vah	$0 \cup \langle 32768*2^{-64};32768*2^{48}\rangle \in \mathbf{R}$	$\langle -32768*2^{48};-32768*2^{-64}\rangle \cup 0 \cup \langle 32768*2^{-64};32768*2^{48}\rangle \in \mathbf{R}$	
Výstupní rozsah	$0 \cup \langle 32768*2^{-64};32768*2^{63}\rangle \in \mathbf{R}$	$\langle -32768*2^{63};-32768*2^{-64}\rangle \cup 0 \cup \langle 32768*2^{-64};32768*2^{48}\rangle \in \mathbf{R}$	
Výpočetní funkce	*Kernel HINT16*		
Typ	Unipolární	Bipolární se znaménkovým bitem	Bipolární ve dvojkovém doplňku
Vstupní rozsah	$\langle 0;65535\rangle \in \mathbf{N^0}$	$\langle -32767;32767\rangle \in \mathbf{Z}$	$\langle -32768;32767\rangle \in \mathbf{Z}$
Rozsah vah	$\langle 0;65535\rangle \in \mathbf{N^0}$	$\langle -32767;32767\rangle \in \mathbf{Z}$	$\langle -32768;32767\rangle \in \mathbf{Z}$
Výstupní rozsah	$0 \cup \langle 32768*2^{-15};65534*2^{16}\rangle \in \mathbf{R}$	$\langle -65532*2^{13};32768*2^{-15}\rangle \cup 0 \cup \langle 32768*2^{-15};65532*2^{13}\rangle \in \mathbf{R}$	$\langle -32768*2^{15};32768*2^{-15}\rangle \cup 0 \cup \langle 32768*2^{-15};65534*2^{16}\rangle \in \mathbf{R}$

Tabulka 4.6.1: Platné rozsahy funkcí kernel VDT24 a kernel HINT16

Z postupů navržených v této podkapitole je možné vyvodit několik závěrů v podobě doporučení pro zavádění do systémů s různými parametry:

1) Použití *kernelu VDT24* je opodstatněné pouze v případě, kdy aplikace umělé neuronové sítě vyžaduje vysokou dynamiku vstupů a/nebo jednotlivých vah. Zároveň je nutné přiřadit aktivační funkci v takovém tvaru, který je schopný

rozsahu proměnných **VDT24** efektivně využít. *Kernel VDT24* nelze doporučit pro sítě zpracovávající hardwarem generované signály, jako jsou například výstupy A/D převodníku, či CAPTURE jednotek a to z důvodu nevyužití potenciálu metody při současném zachování její relativní výpočetní náročnosti.

2) Funkci *kernel HINT16* lze realizovat s různými modifikacemi, v základu dvěma způsoby závislými na použitém hardwaru. Buď jako velmi rychlou za použití 16bitové MAC instrukce za použití neznaménkových, případně znaménkových proměnných v nativní formě procesoru (nejčastěji ve dvojkovém doplňku) nebo jako relativně pomalejší (avšak stále výrazně rychlejší, než *kernel VDT24*) bez podpory MAC instrukcí a to v unipolární podobě, nebo bipolární s převodem celočíselných hodnot na formát absolutní hodnoty se znaménkovým bitem a klidovou hodnotu akumulačního registru s MSB = 1.
3) Použití funkce *kernel HINT16* ve všech možných variantách je ve většině případů výhodnější, než použití *kernel VDT24*, neboť při stále značně širokém vstupním a výstupním rozsahu dostačujícím pro zpracování většiny signálů z převodníků cílové skupiny procesorů nárokuje menší množství systémových prostředků.

4.7 Řešení aktivační funkce neuronu

Posledním prvkem *jádra neuronu* tak, jak je vymezeno v podkapitole *4.1*, je aktivační funkce neuronu. Běžně užívané typy jsou shrnuty v podkapitole *3.2*. Problémem její implementace na nízkovýkonových embedded systémech jsou buď vysoké požadavky na výpočetní výkon (je-li aproximována analytickým výpočtem), nebo vysoké požadavky na obsazení paměti konstant (je-li použita look-up tabulka), respektive vhodná volba poměru k přesnosti. Pro učící algoritmy typu „back propagation“ je také důležitá diferencovatelnost funkce.

Řešení nastíněné v disertaci (9) používá kombinaci look-up tabulky a aproximaci analytickým výpočtem. Relativně nízký počet bodů v LUT je adresován horními bity vstupu. Ze zbylých bitů je vypočítána lineární aproximace v prostoru mezi dvěma tabulkovými body a nadále korigována aproximací kvadratickou. Autor uvádí celkovou výpočetní náročnost bez zohlednění vlastní režie: 3x 8bitové násobení, 1x 8bitové odečítání, bitová rotace vlevo o 7bitů, bitová rotace vpravo o 14bitů. Pro výpočty používá vlastní datový typ **PFP** (stručně popsán v citované práci). Aproximuje aktivační funkci *tanh*. Zároveň však zmiňuje (ovšem více nerozvádí) další potřebnou režii pro úpravu měřítka. Vzhledem k tomu, že v uvedené práci

nepředpokládá online trénování neuronové sítě, není zohledněna potřeba spojité a snadno vypočitatelné první derivace funkce pro učící algoritmy typu back propagation.

V diplomové práci (15) zaměřené na studium aplikací celočíselných umělých neuronových sítích v „low-cost“ embedded systémech je odkazováno pouze na použití look-up tabulek. Výhody v podobě velmi rychlého získání požadované hodnoty pouhým přístupem do paměti jsou zřejmé. Sám autor ovšem hovoří o velmi nepříznivém poměru obsazené paměti konstant a přesnosti, stejně jako o vlivu na vlastnosti sítě, konkrétně rozsah výstupních hodnot, které jsou aktivační funkcí ve výstupních neuronech přímo generovány. Text je poněkud obecnější, než výše citovaná disertační práce. Nepředkládá dosažené výsledky pro nějakou konkrétní funkci, zato ale řeší i otázku derivací. Navrhuje používání takových aktivačních funkcí, které samy vystupují ve své derivaci a lze proto pro její vyčíslení použít stejné LUT a minima doplňujících výpočtů. Obdobný přístup je propagován i ve článku (24).

Pro potřeby metody představované v této práci je možné formulovat následující parametry a vlastnosti, které by měla implementace aktivační funkce neuronu splňovat:

1) Vyšší přesnost s použitím menší velikosti paměti, než umožňuje look-up tabulka
2) Aproximace funkce se spojitou první derivací umožňující případné použití učících algoritmů typu back propagation
3) Definiční obor funkce umožňující využít benefitu dynamického rozsahu datového typu **VDT24**
4) Minimální náročnost na výpočetní výkon, tedy maximální možné využití hardwarových prostředků procesoru, vyhýbání se nebo zjednodušování nativně nepodporovaných operací, nepoužívání vyšších řádů polynomů apod.
5) Univerzalita použitého principu ve smyslu použitelnosti pro různé aktivační funkce

Požadované parametry vylučují oba extrémní přístupy, tedy jak použití kompletní look-up tabulky, tak použití čistě analytického předpisu funkce (to je možné a zároveň účelné při použití některých jednoduchých aktivační funkcí, jako jsou ostrá nelinearita, lineární funkce, nebo lineární funkce se saturací). Vhodným přístupem se jeví kombinace relativně řídké LUT s aproximací analytickou funkcí mezi body. Přístup volený v (9) je nevhodný díky složitému výpočtu derivace. Dobré výsledky v tomto ohledu je možné očekávat od metody interpolace splajnem (25). Konkrétně kvadratickým, neboť nejčastěji používaný kubický již obsahuje polynom příliš vysokého řádu. Kvadratický spajn zaručuje jak jednoduchost předpisu tvořeného

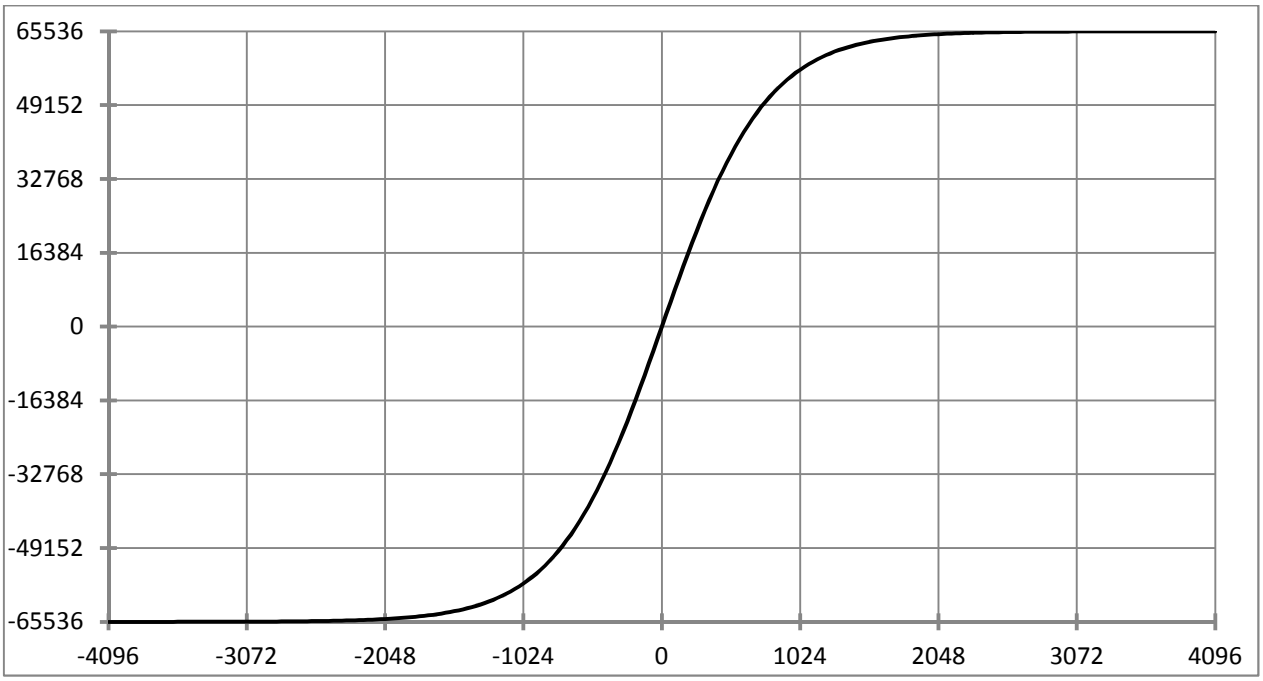
kvadratickou funkcí, tak „hladkost“ průběhu, tedy spojitost první derivace. Pro sigmoidální funkce navíc umožňuje použití malého množství bodů LUT.

Základním principem splajnu 2. řádu je nahrazení funkčního průběhu kvadratickým polynomem a to vždy v oblasti mezi dvěma po sobě jdoucími známými uzlovými body $\mathbf{x_i}$, $\mathbf{y_i}$ a $\mathbf{x_{i+1}}$, $\mathbf{y_{i+1}}$, které jsou zároveň koncovými body interpolační funkce v daném úseku. Index **i** nabývá hodnot **0** až **N**, kde **N+1** je celkový počet bodů a **N** je zároveň výsledný počet dílčích funkcí pro aproximaci průběhu v intervalu definičního oboru funkce $\langle \mathbf{x_0}; \mathbf{x_N} \rangle$. Úloha určení jednoznačných koeficientů všech **N** funkcí (indexovaných dále písmenem **n** v hodnotách od **0** včetně do **N-1** včetně) spočívá ve vytvoření a řešení soustavy rovnic, kde jsou neznámými koeficienty dané kvadratické funkce. Aby hledané jednoznačné řešení soustavy existovalo, je třeba znát nejen uzlové body, ale také formulovat podmínky spojitosti derivací ve vnitřních bodech a okrajové podmínky v bodech koncových. Pro obecný kvadratický splajn $\mathbf{S_{i(x)}}$ platí *Vztah 4.7.1*. Podmínky platné pro tvorbu soustavy rovnic vedoucí k nalezení hledaného řešení aproximace průběhu původní funkce $\mathbf{f_{(x)}}$ jsou shrnuty *Vztahy 4.7.2*. Obecně ho nelze formulovat pouze pro okrajové body $\mathbf{x_0}$ a $\mathbf{x_N}$, kde je třeba definovat podmínky jiným způsobem (například přirozený splajn má v těchto bodech derivaci rovnou nule).

$$S_{i(x)} = a_i + b_i(x - x_i) + c_i(x - x_i)^2 \qquad \textbf{Vztah 4.7.1}$$

$$S_{i(x_i)} = f_{(x_i)}; i = 0 \dots N - 1$$
$$S_{i(x_{i+1})} = f_{(x_{i+1})}; i = 0 \dots N - 1$$
$$\frac{dS_{i(x_{i+1})}}{dx} = \frac{dS_{i+1(x_{i+1})}}{dx}; i = 0 \dots N - 2 \qquad \textbf{Vztahy 4.7.2}$$

Jako příklad řešení aproximace kvadratickým splajnem bude dále uvedena náhrada funkce hyperbolické tangenty, která bude použita v kapitole *5* popisující ukázkovou aplikaci pro změření konkrétních výsledků. Předpis je oproti uvedenému v *Tabulka 3.2.1* upraven v měřítku 768:1 pro vstupní hodnoty a 65536:1 pro hodnoty výstupní. Tento krok je obzvláště důležitý kvůli využití rozsahu umožňovaného v neuronové síti použitým typem proměnných a závisí na něm její celková přesnost. Vstupní i výstupní měřítko značené konstantami $\mathbf{M_{in}}$ a $\mathbf{M_{out}}$ by mělo být shodné, aby byla zachována proporcionalita s původní funkcí. Volba rozdílných konstant bude vysvětlena níže. Graf funkce je na *Obr. 4.7.*1. Její předpis pak udává *Vztah 4.7.3*.



Obr. 4.7.1: Graf aproximované hyperbolické tangenty v příslušeném měřítku

$$f_{(u)} = M_{out} * \frac{e^{\frac{2u}{M_{in}}} - 1}{e^{\frac{2u}{M_{in}}} + 1} = 65536 * \frac{e^{\frac{2u}{768}} - 1}{e^{\frac{2u}{768}} + 1}$$ **Vztah 4.7.3**

Hyperbolický tangens je zároveň funkce použitá v práci (9) (což umožňuje případné srovnání dosažených výsledků). V uvedeném vztahu je již vstupní proměnná označena písmenem **u**, což koresponduje s využitím v umělých neuronových sítích více, než výše (u obecné teorie splajnů) užívané **x** podléhající matematickým konvencím.

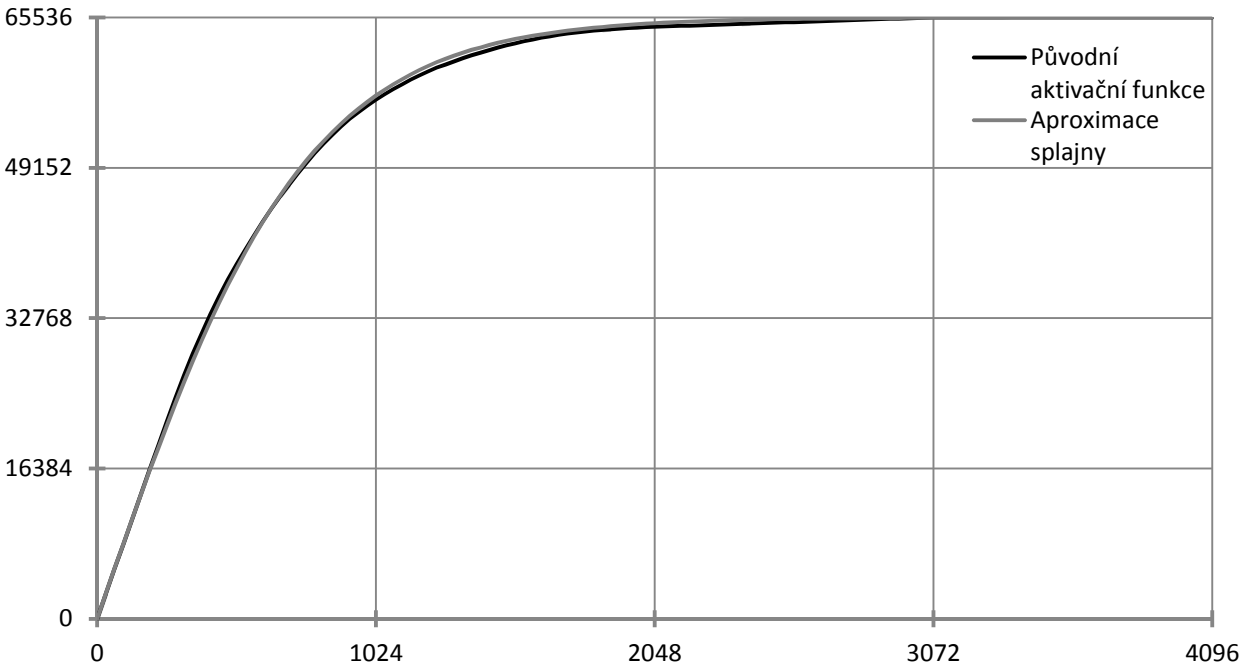
Použitá aktivační funkce je středově souměrná dle počátku souřadného systému, což umožňuje při hledání aproximace vypustit 3. kvadrant. Vzhledem k tomu, že hledaná aproximace bude konkrétně implementována do 8bitového procesoru s hardwarovou podporou násobení rozměru 8x8 bitů, je v tomto případě vhodné omezit přesnost mantisy při použití datového typu **VDT24** a to konkrétně vypuštěním pěti LSB. První tři MSB budou použity pro adresování subintervalu, tedy jednoho z celkem osmi kvadratických splajnů. Zbývajících 8 bitů bude použito jako vstup příslušní funkce $\mathbf{S_{i(u)}}$. Tím bude umožněna hardwarová akcelerace operací. Podmínkou nutnou je ovšem ještě volba kvadratických koeficientů $\mathbf{C_i}$ v mocninách nebo podílech mocnin dvou, aby mohlo být násobení či dělení nahrazeno bitovou rotací a to, v tomto případě, i na úkor přesnosti. Volba rozdílných koeficientů $\mathbf{M_{in}}$ a $\mathbf{M_{out}}$ vyplývá z uvedeného zahození posledních bitů mantisy a oblasti začínající vstupní hodnotou přibližně 2 až 3 násobku $\mathbf{M_{in}}$, kde se funkce začíná dostávat do saturace. V této oblasti již pro většinu aplikací postačuje lineární

aproximace přímkou se směrnicí odpovídající derivaci posledního splajnu v koncovém bodě až k hodnotě asymptoty, kde dojde k ostrému omezení nebo omezení oboru hodnot aproximované funkce na hodnotě koncového bodu.

i	**x_i**	**$f_{(x)}$**	**$S_{i-1(x)}$**	**a_i**	**b_i**	**c_i**	**$\Delta S_{i(xi+1)}$**
0	0	0	-	**0**	**85**	**-1/256**	83
1	256	21070,66	21504	**21504**	**83**	**-1/16**	51
2	512	38193,26	38656	**38656**	**51**	**-1/32**	35
3	768	49911,83	49664	**49664**	**35**	**-1/32**	19
4	1024	57020,36	56576	**56576**	**19**	**-1/64**	11
5	1280	61021,2	60416	**60416**	**11**	**-1/128**	7
6	1536	63178,51	62720	**62720**	**7**	**-1/128**	3
7	1792	64314,94	64000	**64000**	**3**	**-1/256**	1
8	2048	64906,23	64512	**64512**	**1**	**0**	1

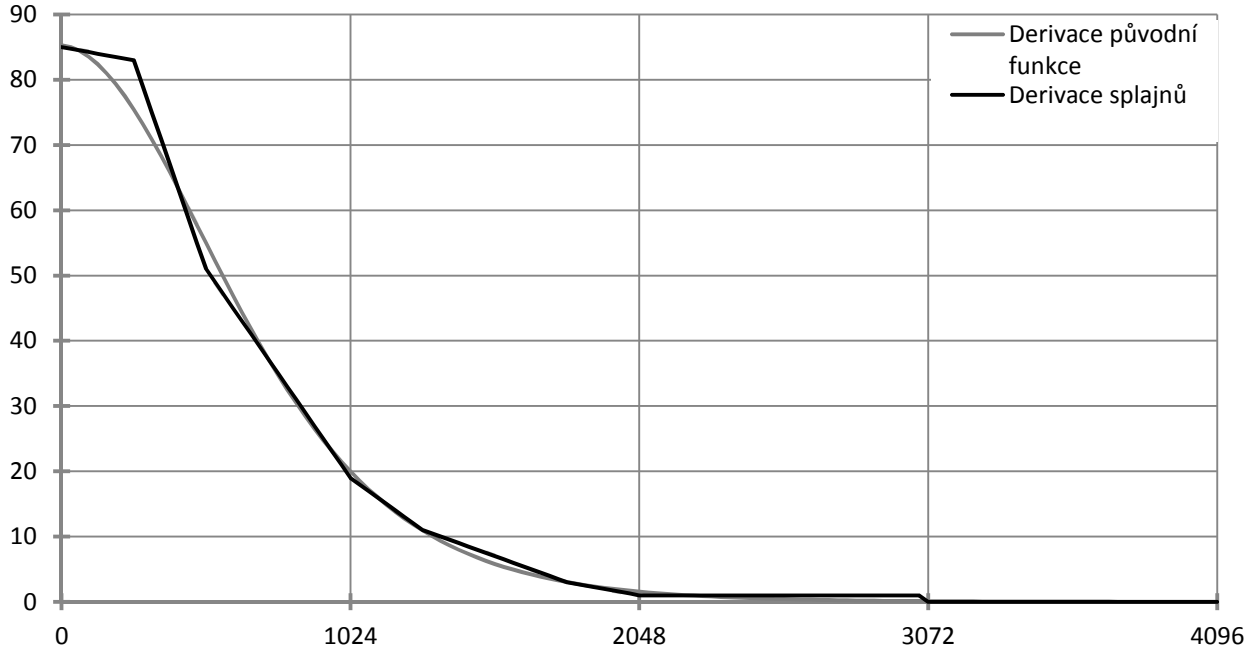
Tabulka 4.7.1: Ukázkové řešení aproximace hyperbolické tangenty splajny

Na základě těchto úvah, počáteční podmínky $\mathbf{\Delta u_0=85}$ stanovené z průběhu derivace funkce ze *Vztah 4.7.3* a požadavku tvarové podobnosti průběhů derivace vzoru a aproximace (funkce s jedním inflexním bodem) je možné sestavit *Tabulka 4.7.1* jednoho z řešení upravených pro dělení bitovou rotací. Tučně vyznačené a silně orámované koeficienty jsou hledaným výsledkem. Kvadratických aproximačních funkcí je celkem 8, poslední řádek reprezentuje zmiňovaný lineární úsek v oblasti blízké saturaci. Graficky, spolu s původní funkcí, je výsledek vyjádřen na *Obr. 4.7.*2, odkud jsou patrné pouze zanedbatelné odchylky. Jako správná se zde projevuje i volba konstanty $\mathbf{M_{in}=768}$. Při její vyšší hodnotě by poslední splajn končil na příliš nízké výstupní hodnotě a lineární pokračování funkce by se znatelně odchylovalo od původního průběhu. Naproti tomu při hodnotě nižší by (na úkor konkávní části křivky) byla zbytečně kvadratickým polynomem aproximována i lineární část.



Obr. 4.7.2: Grafické srovnání prvního kvadrantu původní tangenciální aktivační funkce a její aproximace

Pro účely učících algoritmů pracujících s derivacemi aktivační funkce lze použít stejných koeficientů, jako pro výpočet kvadratického splajnu a analyticky stanovit derivaci, neboť se prakticky jedná o triviální úkol výpočtu lineární funkce, kde je pouze třeba vynásobit dvěma (lze použít bitovou rotaci) původně kvadratický koeficient. Takovýto postup zaručí přesné derivování aproximace, nicméně průběhu derivace původní hyperbolické tangenty se značně liší. Nahrazuje ji spojitě navazujícími přímkovými úseky. Pokud by toto bylo na závadu v zamýšlené aplikaci, existují (při zachování aproximační metody) dvě základní možnosti: 1) i derivaci aproximovat pomocí kvadratických splajnů, což vede k většímu obsazení paměti dalšími konstantami, 2) náhrada kvadratických splajnů kubickými, které zaručují i spojitost druhé derivace, tedy hladkost derivace první. Ty ovšem již pracují s polynomem 3. řádu, což buď výrazně omezuje vstupní rozsah, nebo znemožňuje použití hardwarové akcelerace násobení 8x8. Výhodou je pouze úspora (v řádu nízkých desítek bajtů) paměti. Graficky jsou derivace původní funkce a analytická derivace splajnů znázorněny na *Obr. 4.7.*3.



Obr. 4.7.3: Grafické srovnání derivace původní hyperbolické tangenty a aproximačních splajnů

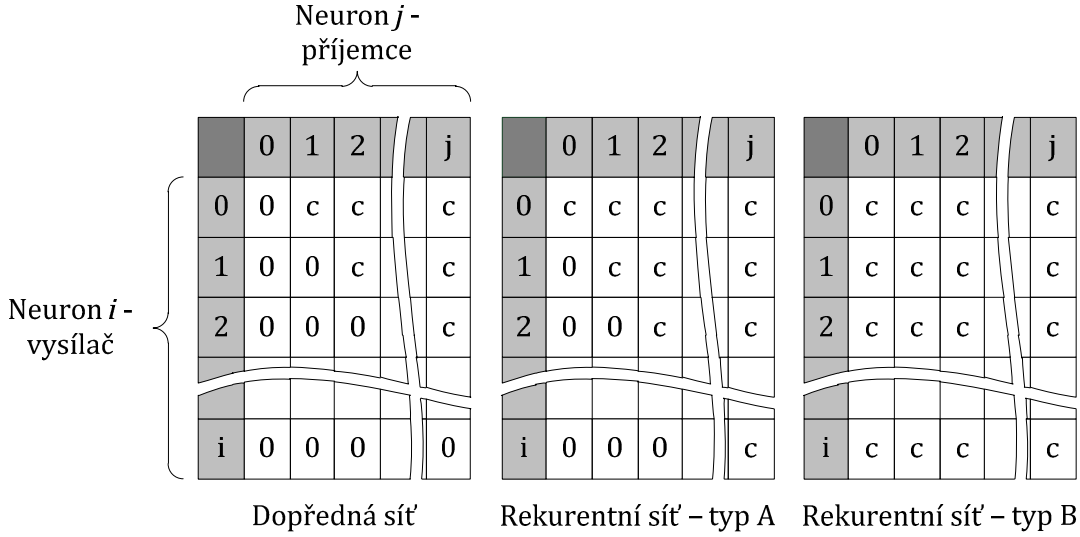
Programové řešení takto připravené funkce v celočíselném intervalu <-65535;65535> je jednoduché. Vyžaduje bitovou rotaci 16bitového vstupu o 5bitů vpravo, vyčtení tří koeficientů z LUT, dvě celočíselná násobení 8x8bitů, bitovou rotaci o 4 až 8bitů vpravo, jeden součet 16x16bitů a jeden rozdíl 16x16bitů. To je ve výsledku méně, než výpočet v práci (9) referencovaný na začátku této podkapitoly. Tak jak je funkce postavena nevyužívá možností rozsahu **VDT24**. Zarovnané hodnoty s exponentem větším, než 0 vždy vracejí hodnotu +/-65535. Funkce ovšem s výhodou využívá znaménkového bitu, který do výpočtu nevstupuje a je pouze kopírován ze vstupu na výstup. Funkce dokáže pracovat efektivněji v případě, že u typu **VDT24** není prováděno zarovnání na nenulový **MSB** pro exponenty menší, než 0. Pro zavedení do *kernelu HINT16* je nutné přepočítat koeficienty pro $\mathbf{M_{out}=32768}$, nebo výstup funkce rotovat o jeden bit vpravo. Doplnění znaménkového bitu na **MSB** je pro zpracování v další vrstvě neuronů nutné vždy. Praktická ukázka implementace takovéto aktivační funkce je v příslušné podkapitole kapitoly *5*.

4.8 Vzájemné propojení neuronových jednotek

Teoreticky je tato otázka velmi jednoduchá, intuitivně zřejmá a v literatuře zpravidla řešena pouze okrajově jako přidružená část statě zabývající se topologií nebo strukturou *neuronové jednotky*. Například v (3), (5), nebo (26). Konkrétní řešení jsou pak navrhována na

míru pro danou aplikaci. Tím je na druhou stranu vyvolána i špatná dostupnost obecnější ustálené teorie.

Jako základ návrhu pro účely této práce poslouží přístup prezentovaný v kapitole 4 publikace (27). Zde je pro znázornění použita propojovací matice ve tvaru zobrazeném na *Obr. 4.8.*1. Písmeno **c** (connection) může nabývat logických hodnot 0 nebo 1, kde číslice „1“ znační přítomnost vazby mezi výstupem neuronu **i** a vstupem neuronu **j**. Propojovací matice dopředných sítí je vždy horní trojúhelníková, s nulovými prvky na diagonále. Je-li diagonála nenulová, jedná se již o rekurentní síť, zde označenou jako „**typ A**“, kde jsou povoleny zpětné vazby pouze v rámci jednoho neuronu, tedy výstup **i**-tého na vstup **i**-tého. Matice obecné rekurentní sítě (zde značeno jako „**typ B**“) má všechny své prvky nastavitelné a existuje v ní možnost propojení libovolného **i**-tého neuronu s libovolným **j**-tým neuronem.



Obr. 4.8.1: Propojovací matice pro různé typy sítí z citovaného zdroje

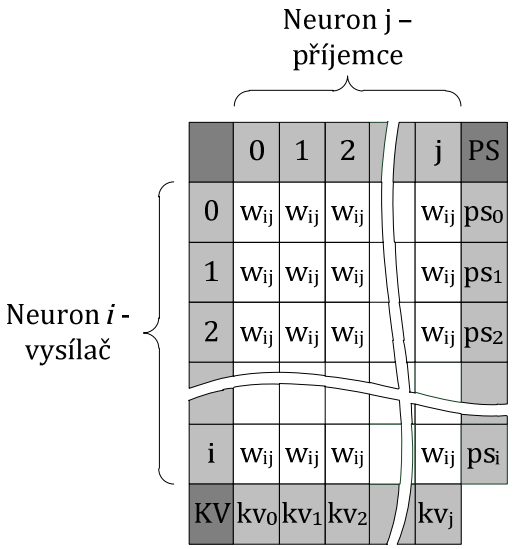
Takovýto princip reprezentace topologie je pro účely implementace vhodný, nicméně oproti verzi uvedené v literatuře se nabízí několik modifikací:

1) Náhrada prvků **c** matice za váhy $\mathbf{w_{ij}}$. Ty jsou ve (27) uváděny mimo propojovací matici blíže nespecifikovaným způsobem. Dojde tím k úspoře paměti o velikosti $\mathbf{N^2}$ bitů, kde **N** je počet neuronů.
2) Rozšíření matice o řádek **N+1**. Nechť je nazván „*řádek koeficientu výkonu*“ a dále v textu označován **KV**. Pro jeho prvky $\mathbf{KV_j}$ platí *Vztah 4.8.1*, kde $\mathbf{f_{(c)}}$ je funkce detekující nulovost operandu. Předpis určuje počet nenulových prvků v daném sloupci, tedy počet vstupů neuronu s indexem **j**.

$$KV_j = \sum_{j=0}^{N-1} f_{(c_j)}; f_{(c)} = \begin{cases} 1\ pro\ c \neq 0 \\ 0\ pro\ c = 0 \end{cases}$$ **Vztah 4.8.1**

3) Rozšíření matice o sloupec **N+1** obsahující na pozici s indexem **i** informaci o příslušnosti **i**-tého neuronu k určité skupině, například k dané vrstvě sítě a dále nazývaný „*sloupec příslušnosti ke skupině*“, ve zkratce **PS**. Tuto informaci může využít dopředná topologie pro úsporu paměti určením další potřebnosti mezivýsledků. Využije ji taktéž algoritmus pro úpravu topologie jako informaci stanovující omezení pro tvorbu nežádoucích vazeb.

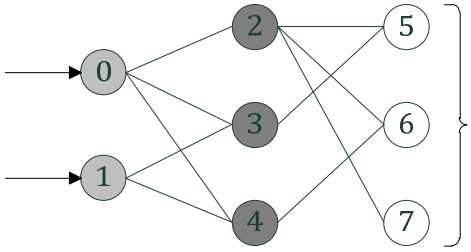
Rozšířená modifikovaná propojovací matice sestavená dle uvedených bodů je na *Obr. 4.8*.2.



Obr. 4.8.2: Rozšířená modifikovaná propojovací matice

Jednotlivé prvky přidaného řádku **KV**, jak je uvedeno výše, reprezentují počet vstupů daného neuronu, který je určující pro jeho výpočetní náročnost. Pro praktické použití je účelné – například prostřednictvím look-up tabulky – nahradit jednotlivé $\mathbf{kv_j}$ přímo počtem časových jednotek potřebných pro výpočet. Ty je nejpřesněji možné stanovit empiricky na základě měření výpočetní náročnosti konkrétní použité *neuronové jednotky* jako maximální počet instrukčních cyklů (nebo jiných zvolených časových jednotek dobře měřitelných využívaným embedded systémem) nutných pro její výpočet. Suma řádku **KV** potom udává celkovou náročnost výpočtu sítě bez některých podpůrných operací sestávajících již výhradně z přesunů v paměťovém prostoru a skoků programového čítače při volání funkcí, nikoli z matematických operací. Nepřesnost je dána použitou metodou stanovení prvků řádku **KV**, neměla by však přesahovat jednotky procent. Suma takto upravených prvků $\mathbf{kv_j}$ nechť je nadále označována jako *výpočetní náročnost sítě*. Je nutná pro algoritmy adaptace topologie jako informace o využitelné výkonové rezervě systému.

Podoba prvků *sloupce příslušnosti ke skupině* **PS** nemusí být dána tak striktně, jako podoba **KV**. Jeho přítomnost pro popisovanou metodu zavádění neuronových sítí do embedded systémů s nízkým výpočetním výkonem je nepovinná. Slouží zejména pro řízení adaptačních algoritmů sítě ve smyslu úpravy nebo zákazu některých vazeb mezi nebo v rámci jednotlivých skupin reprezentujících například vrstvy sítě (perceptronová síť). Je-li v implementaci umělé neuronové sítě dodrženo indexování neuronů systémem zleva doprava - shora dolů tak, jak je uvedeno v příkladu na *Obr. 4.8.*3, je dobře využitelným způsobem stanovení prvků $\mathbf{ps_i}$ zápis indexů prvního prvku následující vrstvy. Tedy pro vstupní vrstvu 2, pro skrytou vrstvu 5 a pro výstupní vrstvu 8 (první index, který již nepatří síti). Při požadavku na tvorbu dopředné sítě bez zpětných vazeb je hodnotou $\mathbf{ps_i}$ určen počet po sobě jdoucích nulových prvků od začátku řádku neuronu s indexem **i**. Při přidávání neuronů je pak ovšem třeba všechny koeficienty $\mathbf{ps_i}$, kde **i** je index první jednotky ve vrstvě, kam je přidáván neuron, inkrementovat. To je ovšem stejně jako změna podoby propojovací tabulky už úkolem jiných algoritmů.



Obr. 4.8.3: Ilustrační příklad topologie sítě

Nevýhodou tohoto řešení je značná paměťová náročnost **PN** vyjádřená v bitech *Vztah 4.8.*2, kde **N** je celkový počet neuronů, $\mathbf{bit_w}$ počet bitů proměnné váhy, $\mathbf{bit_{PS}}$ počet bitů prvku vektoru **PS** a $\mathbf{bit_{KV}}$ počet bitů prvku vektoru **KV**.

$$PN = bit_w * N^2 + (bit_{PS} + bit_{KV}) * N \qquad \textbf{Vztah 4.8.2}$$

Je-li požadována implementace sítě s možností kompletního propojení, tedy taková, jaká je v textu výše označená jako „rekurentní typu B“, nelze uvedený vztah při zachované obecnosti optimalizovat a je nutné paměťový prostor velikost **PN** vyhradit. V rámci vlastností konkrétní aplikace lze pouze vyměnit obsazení paměti RAM, které bývá v cílových embedded systémech výrazně méně, než paměti programové, za sníženou rychlost algoritmu. Koeficienty *rozšířené modifikované propojovací matice* musejí být v každém případě uchovány i po vypnutí napájení. Je tedy jasné, že odpovídající prostor v paměti programu bude obsazen. Při běhu ovšem není třeba vytvářet kompletní pracovní kopii do paměti datové, váhy **W** lze (ty jsou, na rozdíl od **PS** a **KV** využívány během *aktivní dynamiky sítě* a proto je práce s nimi časově kritická) na úkor rychlosti při současném výrazném zmenšení požadovaného prostoru v RAM přesouvat do pracovního bufferu velikosti potřebné pro pokrytí neuronu s maximálním počtem vstupů, po částech potřebných pro aktuální výpočet.

Pokud je požadován systém se zpětnými vazbami v podobě označené jako „typ A“ nebo systém pouze s dopřednými vazbami, lze nejen použít oba výše zmíněné přístupy, ale i redukovat velikost části propojovací matice obsahující váhové koeficienty **w** ve smyslu vyškrtnutí těch prvků $\mathbf{w_{ij}}$, které jsou z principu nulové. Požadovaný paměťový prostor **PN** pak přechází do tvaru *Vztah 4.8.3*.

$$PN = bit_w * P + (bit_{PS} + bit_{KV}) * N; P \begin{cases} \frac{N^2 + N}{2} \; pro \; "typ \; A" \\ \frac{N^2 - N}{2} \; pro \; dopředné \end{cases} \qquad \textbf{Vztah 4.8.3}$$

Podobné redukce **PN** lze dosáhnout i u rekurentní sítě „typu B“, kde jsou vynechány ty koeficienty $\mathbf{w_{ij}}$, které mají vektorem **PS** nebo jiným mechanismem vnucenou nulovou velikost, jinými slovy propojení neuronů **i-j** je zakázáno. Dosažená paměťová úspora je silně závislá na topologii dané aplikace a již nelze vyčíslit obecným vzorcem.

Obdobným způsobem je možné řešit i vektor klidových potenciálů **Bias**, který rovněž může být buď uložen v celé své délce pouze v programové paměti a načítán vždy pro konkrétní výpočet, nebo může být po startu aplikace celý převeden do datové paměti, kde sice zabírá větší prostor – konkrétně $\mathbf{bit_B * N}$, kde $\mathbf{bit_B}$ je počet bitů jedné hodnoty klidového potenciálu – nicméně může být při vhodné metodice adresování rychleji k dispozici.

Jiná situace nastává u výstupních hodnot **y** jednotlivých neuronů. Uchovány jsou vždy a pouze v paměti RAM, kde zabírají maximální prostor velikosti $\mathbf{N * bit_y}$, kde $\mathbf{bit_y}$ je bitová velikost proměnné **y**. Redukce lze dosáhnout uchováváním pouze těch hodnot, které jsou ještě potřebné pro další výpočty. Tato úprava je tedy proveditelná pouze u striktně dopředných sítí. Je opět vykoupena zvýšením vlastní režie o algoritmy ukončující platnost jednotlivých výstupů **y**. Dosažený efekt úspory RAM navíc nemusí být výrazný vzhledem k tomu, že celkový obsazený prostor roste s **N** pouze lineárně (na rozdíl od kvadratické závislosti velikosti pole koeficientů **w**).

Příklad realizace výše popsané rozšířené modifikované propojovací matice včetně vhodného způsobu redukce a uložení vektoru výstupů **y** budou uvedeny v příslušné podkapitole kapitoly *5*. Pro stanovení koeficientů řádky **KV** budou můžou být použity například informace získané měřením algoritmu *_unitHINT16* představeného v podkapitole *5.6*.

4.9 Rozdělení cílových embedded systémů

Tato práce se soustřeďuje na embedded systémy osazené mikrokontroléry nižších řad označovaných výrobci převážně jako „low-end“, případně „low-cost“ (28), (29). Jejich výkony a

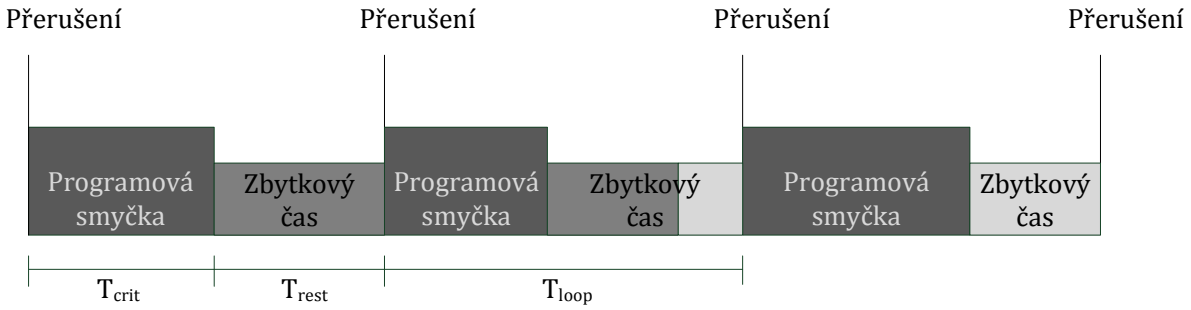
kapacity zpravidla neumožňují efektivní nasazení běžných RTOS. Ty jsou navíc pro aplikace, ve kterých je použití předpokládáno, zbytečné a neefektivní. Vývojář firmwaru pak zpravidla používá jazyk „C“, nebo „C++“, zřídka Assembler, s cílem vytvořit zdrojový kód přímo ovládající hardwarové prostředky dané součástky. Přenositelnost kódu na jiné typy bývá zajištěna buď přepsáním odpovídajících částí programu, nebo tvorbou nových knihovních funkcí, normovaných dle vnitřních předpisů výrobce zařízení, suplujících činnost ovladačů v operačních systémech a opětovnou kompilací jinak nezměněného kódu firmwaru (tento systém v době vzniku práce úspěšně používá i pro složitější aplikace například ZF Engineering Plzeň s.r.o. ,pozn. autora).

Pro softwarovou implementaci navrhovaných umělých neuronových sítí je nejdůležitější možnost zakomponování příslušných bloků do stávajícího kódu. Za předpokladu, že velikost programové paměti a výpočetní výkon a případné komunikační linky mají dostatečné rezervy, je základním kritériem kategorizace struktura programu. Dělení lze provést na následující tři druhy, které jsou předmětem zájmu:

1) Systémy s využitelnou programovou smyčkou konstantní doby trvání
2) Systémy s nevyužitelnou programovou smyčkou konstantní doby trvání
3) Systémy bez programové smyčky konstantní doby trvání

Za systémy s využitelnou programovou smyčkou konstantní doby trvání (dále jen systémy s využitelnou programovou smyčkou) lze považovat ty, které jsou určeny k řešení úloh s požadavkem na přesné časování. Z autorových praktických zkušeností jsou to například embedded systémy určené pro řízení statických elektrických měničů. Jako příklad pro vysvětlení poslouží ovládací mikropočítač můstkového DC-DC měniče s transformátorem.

Spínání výkonových prvků – tranzistorů je realizováno prostřednictvím PWM modulátoru. Šířku otevření počítá softwarový regulátor na základě hodnot měřených A/D převodníkem. Implementovány jsou rovněž logické funkce a stavové automaty ošetřující běh měniče v různých režimech (startování, chod na jmenovitých parametrech, vypínání, reakce na signály obsluhy a další). Pro snadnou diagnostiku a údržbu je zaveden systém detekce poruch se záznamem do paměti a možnost připojení PC prostřednictvím komunikačního portu za účelem výpisu aktuálních i zaznamenaných dat a zadávání přímých povelů řízení v servisním módu. Za časově kritické lze považovat funkce výpočtu regulátoru, měření, detekcí a záznamů chyb a stavových automatů. Naproti tomu výpis na obrazovku PC časově kritický není. Princip takovéto struktury je znázorněn na *Obr. 4.9.*1.



Obr. 4.9.1: Využitelná programová smyčka s konstantní dobou trvání

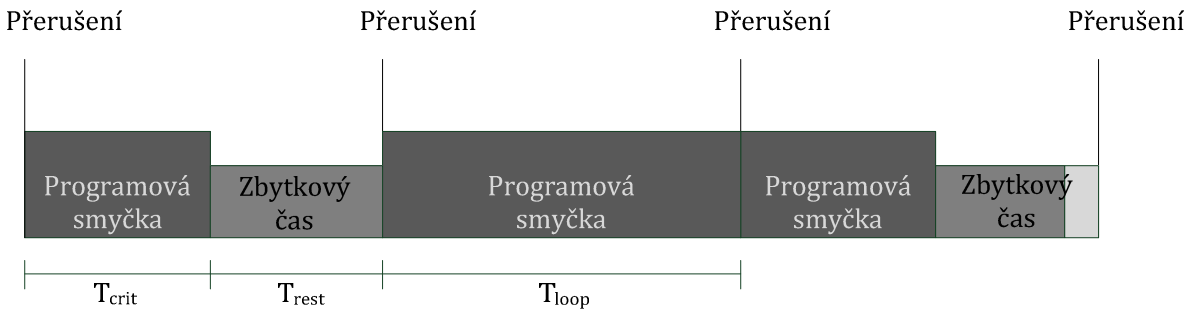
Jak je vidět z ilustrace, zmiňované časově kritické části programu značené jako **Programová smyčka** jsou prováděny v cyklicky se opakujících blocích taktovaných přerušením typicky vyvolávaným čítačem, A/D převodníkem, DMA kanálem spřaženým s A/D převodníkem, či jinou periferií procesoru s vhodnou, konstantní opakovací frekvencí. Ostatní části programu jsou prováděny ve **Zbytkovém čase**. Konstantní opakovací perioda pro ně není zaručena. Je značně proměnlivá v závislosti na aktuální délce programu a v závislosti na větvení podmínek. Nutností pro zařazení do kategorie systémů s využitelnou programovou smyčkou konstantní doby trvání je splnění podmínky *Vztah 4.9.1*.

$$\forall n \in N : T_{loop} \geq T_{crit_{max}} + T_{rest_{min}} + \Delta T_{min} \quad [s; s, s, s] \qquad \textbf{Vztah 4.9.1}$$

kde $\mathbf{T_{loop}}$ je perioda smyčkového přerušení, $\mathbf{T_{critmax}}$ maximální doba trvání programové smyčky a $\mathbf{T_{restmin}}$ minimální přípustný čas pro zbytkové operace. Písmeno **n** označuje iteraci programového cyklu taktovaného přerušením z množiny všech opakování **N** v rámci konečného, libovolně dlouhého běhu systému. Tato část nemá ve vztahu žádný formální význam, protože všechny použité symboly mají absolutní charakter. Uvedena je pouze pro zdůraznění platnosti podmínky. Symbol $\mathbf{\Delta T_{min}}$ značí minimální přípustný úbytek výpočetního času v důsledku implementace metody. Jeho velikost volí uživatel. Vymezuje prostřednictvím něho výpočetní čas rezervovaný navrhovaným úpravám. U systémů tohoto typu lze zaručit jistou minimální úroveň rychlosti výpočtu modelu neuronové sítě, její velikost – právě z důvodu garantované, nicméně konečné doby $\mathbf{\Delta T_{min}}$ – je ovšem omezena. Ve smyslu této práce je takový systém vhodný.

Rozložení výpočetního výkonu druhé kategorie embedded systémů dle výše naznačeného dělení lze vyjádřit schématem na *Obr. 4.9.2*. Rozdíl oproti předchozímu případu je v maximální době řešení časově kritických událostí $\mathbf{T_{critmax}}$. Svým trváním zabírají celou výpočetní smyčku $\mathbf{T_{loop}}$, nebo alespoň takovou její část, že nelze splnit *Vztah 4.9.1*. pro reálnou hodnotu $\mathbf{\Delta T_{min}}$. Minimální rychlost výpočtu zaváděné metody není zaručena. Nelze-li takový systém převést na systém prvního druhu, tedy s využitelnou programovou smyčkou například

optimalizací rozložení výpočetního výkonu zkracujícím $\mathbf{T_{critmax}}$ prodloužením doby $\mathbf{T_{loop}}$, nebo hardwarovou úpravou, kteréžto kroky jsou obvykle nereálné a nekorespondují s cíli této práce, je zařízení zcela nevhodné pro tvorbu sítě se zaručeným časem odezvy; může však dobře posloužit pro neuronovou síť výrazně rozsáhlejší, než v předchozím popisovaném případě, avšak pouze pro aplikace, kde nejsou stálost doby odezvy a doba odezvy samotná sledovanými parametry. Síť je vypočítávána nepravidelně v nekonečné smyčce **Zbytkového času** programu.



Obr. 4.9.2: Nevyužitelná programová smyčka s konstantní dobou trvání

Dalším jistým východiskem je modifikace požadavků na úsek **ΔT**. V takovémto případě již také nelze garantovat minimální rezervovaný čas v každém úseku $\mathbf{T_{loop}}$. V extrémních případech, jako je prostřední blok schématu běhu programu na *Obr. 4.9.*2, nemusí vůbec dojít k výpočtu přidávaných algoritmů. Můžou ovšem za jistých podmínek zůstat ve smyčce přerušení a mít tak vyšší prioritu, než smyčka nekonečná. Nutnou podmínku pro klasifikaci systému tohoto typu formuluje *Vztah 4.9.2*.

$$\forall n \in N : T_{loop} \geq T_{crit_{min}} + T_{rest_{min}} + \Delta T_{min} \quad [s; s, s, s] \qquad \textbf{Vztah 4.9.2}$$

Oproti předchozímu je zde nahrazena maximální doba výpočtu časově kritických událostí $\mathbf{T_{critmax}}$ dobou minimální $\mathbf{T_{critmin}}$. Použité hodnoty jsou opět svou povahou absolutní vzhledem k celému libovolně dlouhému běhu programu a všem možným scénářům jeho větvení.

Byl-li *Vztah 4.9.1* zároveň podmínkou nutnou i postačující k bezpečnému využití systému pro automaticky expandující neuronové sítě se zajištěným minimálním výkonem, je *Vztah 4.9.2* a typ systému, který determinuje, pouze podmínkou nutnou. Otázka skutečné využitelnosti pak záleží na pravděpodobnosti výskytu takových $\mathbf{a \in N}$ iterací, pro které není nerovnost splněna z titulu hodnoty $\mathbf{T_{crit(a)}}$. Posouzení vhodnosti takového systému pro zavedení metody bude dále diskutováno v příslušných pasážích.

Poslední kategorií dle naznačeného dělení je systém bez programové smyčky konstantní doby trvání. Díky zaměření této práce na real-time embedded je třeba zúžit výběr právě na ně.

Lze tedy předpokládat, že se bude jednat o zařízení řešící sice úkoly časově kritické, nicméně takového charakteru, že lze dosáhnout požadovaných výsledků i jinými způsoby, než u předchozích dvou typů. Obsluha zájmových událostí probíhá například na základě neperiodického přerušení, nebo je maximální doba trvání úloh zbytkového času taková, že nekoliduje s požadavky na latenci vyhodnocení prioritních. V tomto režimu také může fungovat firmware systému vhodně hardwarově vybaveného. Příkladem je jednočipový mikropočítač v úloze digitizéru analogového signálu s převodem na sériovou linku disponující DMA řadičem, který požadované akce vykonává automaticky a v ustáleném bezporuchovém stavu bez intervence jádra. Lze prohlásit, že systémy tohoto typu se ve většině případů používají pro nejjednodušší aplikace. Schéma využití výpočetního výkonu je na *Obr. 4.9*.3.

Obr. 4.9.3: Bez programové smyčky konstantní doby trvání

Kritériem pro zařazení do této kategorie je právě absence systému periodicky se opakujících přerušení zaváděných za účelem taktování výpočtu vybraných bloků programu. Výše používané proměnné $\mathbf{T_{loop}}$ a $\mathbf{T_{crit}}$ ztrácejí význam. Díky nepřítomnosti smyčky s konstantním časem také kromě speciálních případů není problém rezervovat rozumně volený, tedy v řádech srovnatelných s $\mathbf{T_{rest}}$, úsek $\mathbf{\Delta T}$. Volán může být po skončení každé iterace programu. Problémem je, že v tomto případě nemusí korespondovat s rychlostí výpočtu metody, neboť ani jeho periodicita není zaručena, což systém tohoto typu neumí z principu. Je-li to požadováno, musí být převeden na jeden z předchozích druhů.

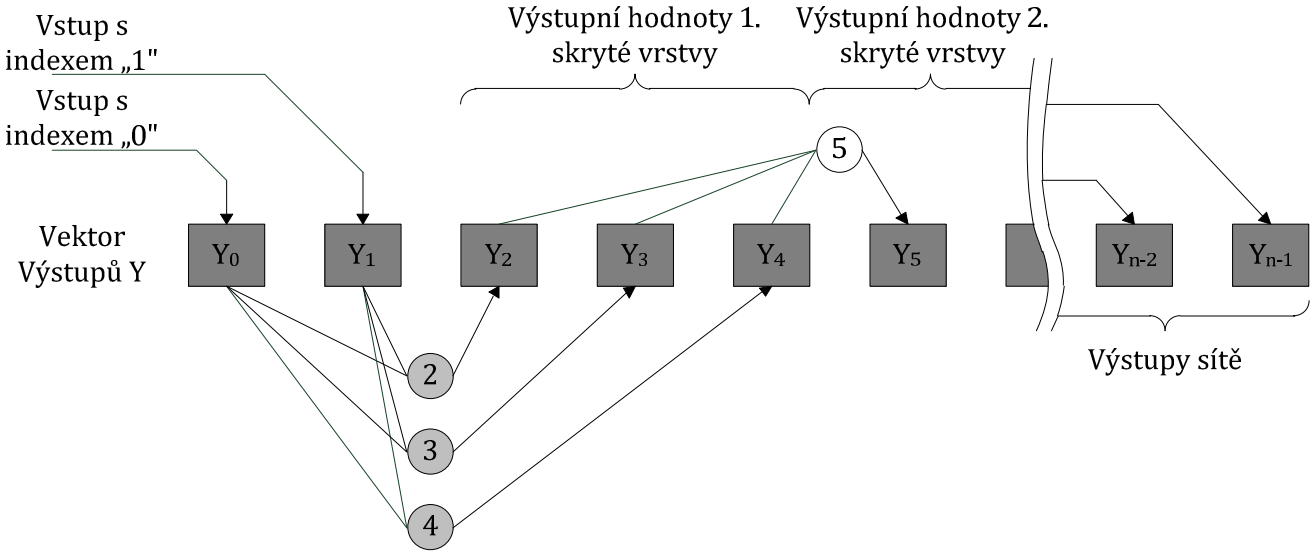
4.10 Navázání neurnové sítě do systému

Posledním krokem po klasifikaci cílového embedded systému a formulování všech pravidel a algoritmů řízení a provozu sítě je třeba tuto navázat do systému. Prakticky se jedná o definici a následnou praktickou realizaci tří bodů:

1) Datový typ, počet a adresa vstupních proměnných
2) Datový typ, počet a adresa výstupních proměnných
3) Okamžik volání výpočetní funkce

Pro převod na zvolený datový typ je samozřejmě nutné použít příslušnou funkci přetypování. V případě zde prezentované neuronové sítě postavené na jednotkách *_HINT16* (při celočíselném 16bitovém standardním vstupu) se jedná o pouhý přesun 1:1 v případě kladných hodnot a výpočet bitového doplňku, případně při nekritickém snížení přesnosti o hodnotu 1, o bitovou negaci – oba postupy doplněné nahozením MSB jako znaménkového bitu. O zápornosti čísla ve formátu dvojkového doplňku lze rozhodnout rovněž dle MSB. Vlastní navázání vstupu je vhodné provést bezprostředně před voláním funkce výpočtu sítě prostřednictvím instrukcí přesunu na požadovanou pozici do RAM. Koncept popisovaný v předchozích kapitolách očekává zápis na tolik prvních pozic vektoru **Y** výstupních hodnot, kolik je neuronů ve vstupní vrstvě.

Obdobným způsobem je možno provést i navázání výstupů, případně libovolné použití mezivýsledků uložených ve vektoru **Y**. Postup převodu na požadovaný datový typ je přirozeně v opačném směru. Navrhovaný způsob navázání včetně ilustrace způsobu propojení jednotlivých neuronů ukazuje *Obr. 4.10.*1.



Obr. 4.10.1: Grafické znázornění možné realizace napojení vstupů a výstupů na neuronovou síť

Okamžik volání funkce výpočtu neuronové sítě je závislý na požadavcích uživatele předkládané metody. Určující je pro něj přitom typ systému dle dělení použitého v podkapitole *4.9*.

Systémy s využitelným volným výpočetním výkonem v periodické hlavní programové smyčce umožňují implementaci relativně menších sítí s garantovanou maximální dobou odezvy. Jejich volání je třeba realizovat na konci programové smyčky (před návratem z obsluhy přerušení zajišťujícího její vykonávání). Za účelem řízení algoritmů adaptace topologie je vhodné také vložit měření výpočetního výkonu spotřebovaného sítí a výkonovou rezervu. Tyto algoritmy samotné je pak vhodnější volat ve zbytkovém čase. Kdykoli je pak do propojovací

matice sítě zasahováno, je třeba z důvodu prevence geneze neplatných výsledků volání výpočtu sítě zabránit. Lze toho dosáhnout například definicí a použitím registru stavového slova s bitem zákazu volání funkce.

Systém tohoto druhu samozřejmě také umožňuje implementovat sítě rozsáhlejší, bez garantované maximální doby odezvy. Pro její volání platí všechna předchozí pravidla s tím rozdílem, že je provedeno z nekonečné smyčky zbytkového času. Tento způsob lze také jako jediný použít u embedded systémů zbývajících typů, tedy bez cyklicky opakované výpočetní smyčky a s cyklicky opakovanou výpočetní smyčkou bez využitelného volného výkonu.

5. Ukázková aplikace

Pro otestování prezentované metody byl vyvinut speciální hardware v podobě jednoduchých modulů s jednočipovými mikropočítači. Zejména z důvodu urychlení práce bylo na základě bohatých praktických zkušeností vybíráno z portfolia společnosti Microchip (10). Vznikly dva výkonově odstupňované druhy samoúčelných zařízení, pro které byly v programovacích jazycích „C“ a Assembler napsány ukázkové aplikace postihující svou strukturou principy popsané v podkapitole *4.9*. Následně proběhlo doplnění o algoritmy adaptabilních neuronových sítí ve smyslu popsané metody.

5.1 Hardware

Navržena a realizována byla výpočetní jednotka s pracovním názvem „**Modul A**“. Pod tímto označením bude nadále odkazována. Schéma, deska s plošnými spoji i podklady pro výrobu jsou vyhotoveny v návrhovém programu Eagle 6.4 popsaném a v použité freewarové verzi dostupném z (30). Výrobu realizovala společnost ITEAD (31). Jednotka je v principu jednoúčelová a při tvorbě byl kladen důraz zejména na rychlost a jednoduchost návrhu. I přesto ale zvolený koncept umožňuje k jednotce dodatečně připojit další externí obvody.

„**Modul A**“ je základním zapojením osmibitového jednočipového mikropočítače PIC18K46F80 (32) doplněným o stabilizátory napájecího napětí, zdroj referenčního napětí, budič sběrnice CAN, jednolinkové sériové asynchronní sběrnice a externí sériovou paměť FLASH. Jednotka je dále osazena napájecím konektorem, konektorem programovacího rozhraní ICSP, kolíkovými lištami nevyužitých pinů mikropočítače, primitivním resetovacím obvodem s možností manuálního spuštění tlačítkem a LED indikátory přítomnosti napájecích napětí. Jádro procesoru je taktováno vestavěným oscilátorem s násobičkou čtyřmi a externím krystalem 12MHz. Za těchto podmínek je dosahováno výpočetního výkonu 12MIPS. Vlastní zapojení je triviální, zcela v souladu s obecnými doporučeními a katalogovými listy použitých součástek. Jeho detailnější rozbor není předmětem této práce.

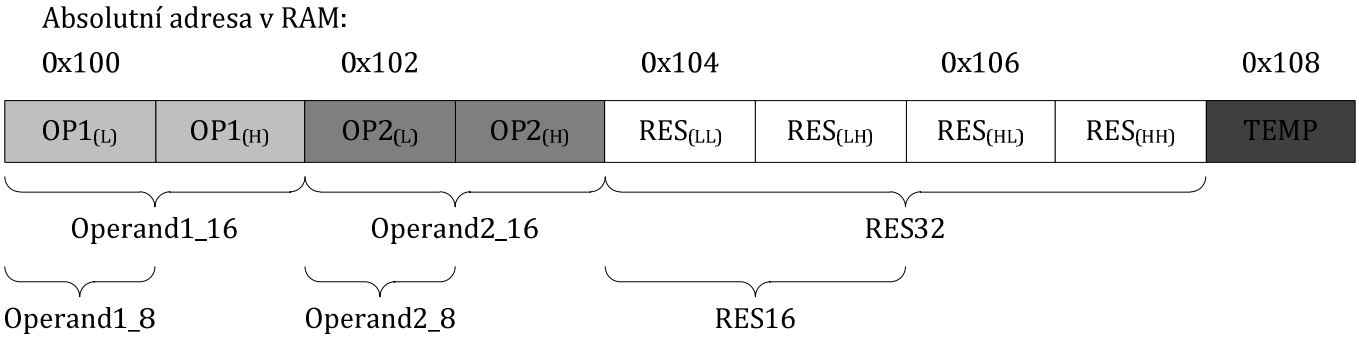
Obvodové schéma, motiv desky plošných spojů a osazovací plánek jsou k dispozici v tištěných přílohách **(A)** a **(B)**. Příslušné soubory pro návrhový program Eagle 6.4 jsou také k dispozici na přiloženém CD.

5.2 Implementace celočíselného násobení

Jádro PIC18 obsahuje pouze neznaménkovou hardwarovou násobičku rozměru 8x8. Jednak jako srovnávací příklad a z důvodu potřeby funkce rychlého celočíselného neznaménkového násobení rozměru 16x16 vznikl v jazyce Assembler zdrojový soubor *Mul.asm*, který je včetně komentářů k dispozici na přiloženém CD. Soubor obsahuje následující funkce:

1) *_umul8a* – neznaménkové násobení 8x8 bez použití HW násobičky
2) *_umul8af* – neznaménkové násobení 8x8 s použitím HW násobičky
3) *_umul16a* – neznaménkové násobení 16x16 s použitím HW násobičky

Uvedené assemblerovské funkce je po deklaraci v hlavičkovém souboru možné volat jako funkce jazyka C. Předávání vstupů a výstupu probíhá prostřednictvím předem známé v paměti staticky alokované tabulky. Tento přístup je zvolen z důvodu úspory výpočetního výkonu vyplývajícího z nepoužití instrukcí nepřímého adresování svázaných na daném typu jádra s jistými omezeními zvyšujícími vlastní režii funkce. Mapu relevantní části paměťového prostoru



Obr. 5.2.1: Organizace paměťového prostoru funkcí ze souboru *umul.asm*

použitého v *Mul.asm* znázorňuje *Obr. 5.2.1*. Příslušné zápisy direktiv vývojového prostředí MPLAB X IDE v2.1 a překladače jazyka „C“ XC8 v1.31 v Assembleru a „C“ tak, aby byly části kódu ve vzájemné shodě, je možné nalézt v *Tabulka 5.2.1*, přičemž definice 8bitových operandů a 16bitového výsledku v „C“ je uvedena kvůli použití násobení rozměru 8x8 a to pouze z formálního důvodu. Rovněž funkce *_umul8a* nevyužívající hardwarové podpory je vytvořena čistě z komparativních důvodů. Společnost Microchip používá při práci s registry procesorů organizaci „Little Endian“, což je zohledněno i při mapování proměnných v diskutovaných

Assemblerovských funkcích násobení a proto proměnné RES32, Operand1_16 a Operand2_16 můžou být použity pro obě dimenze výpočtu.

Mapování v jazyce Assembler	Mapování v jazyce C
OP1_L equ 0x100	unsigned int **Operand1_16** @ 0x100;
OP1_H equ 0x101	unsigned int **Operand2_16** @ 0x102;
OP2_L equ 0x102	unsigned char **Operand1_8** @ 0x100;
OP2_H equ 0x103	unsigned char **Operand2_8** @ 0x102;
RES_LL equ 0x104	unsigned int **RES16** @ 0x104;
RES_LH equ 0x105	unsigned long **RES32** @ 0x104;
RES_HL equ 0x106	unsigned char **TEMP_MUL** @ 0x108;
RES_HH equ 0x107	
TEMP equ 0x108	

Tabulka 5.2.1: Organizace paměti pro předávání proměnných mezi jazyky „C“ a Assembler

Měření náročnosti jednotlivých verzí násobení bylo provedeno na procesorové jednotce **„Modul A“** stručně popsané v podkapitole *5.1*. Použita byla následující metodika: Čítači označovanému jako „TIMER1“ (32) byl nastaven zdroj hodinového signálu s frekvencí instrukčního cyklu jádra, což odpovídá ¼ frekvence oscilátoru mikrokontroléru. Výchozí hodnota čítače před každým násobením je 0. Čítač je zastaven, spouští se bezprostředně před danou operací násobení, vypíná se ihned po návratu z funkce. Měření algoritmů násobení 8x8 proběhlo ve dvou kaskádních for-cyklech s hodnotami 2x 0 až 255, inkrement 1, tedy pro všechny možné kombinace operandů 1 a 2. V případě násobení 16x16 byly z časových důvodů cykly upraveny na rozsah 2x 0 až 65525 s inkrementem 100. Vyhodnocena byla nejmenší a největší doba trvání každého algoritmu. Zaznamenány byly operandy, pro které daný případ nastal jako první. Ověřování správnosti algoritmů probíhalo srovnáváním všech výsledků se stejnými vstupy. V případě neshody byla navýšena hodnota chybového čítače. Výsledek měření ukazuje *Tabulka 5.2.2*. Jednotka [**ic**] reprezentuje dobu trvání jednoho instrukčního cyklu.

Funkce	Doba trvání	Operandy
_umul8a	**14** až **176** ic	Min. při **0,0**;Max při **0,255**
_umul8af	**10** ic	V celém rozsahu vstupů
Standardní funkce „*“ pro 8bit operandy	**10** ic	V celém rozsahu vstupů
_umul16a	**37** až **39** ic	Min. při **0,0**;Max při **300,500**
Standardní funkce „*“ pro 16bit operandy	**68** až **548** ic	Min při **0,0**;Max. při **47100,0**

Tabulka 5.2.2: Výsledky měření doby trvání operací násobení

Z uvedených výsledků měření je možné učinit následující závěry:

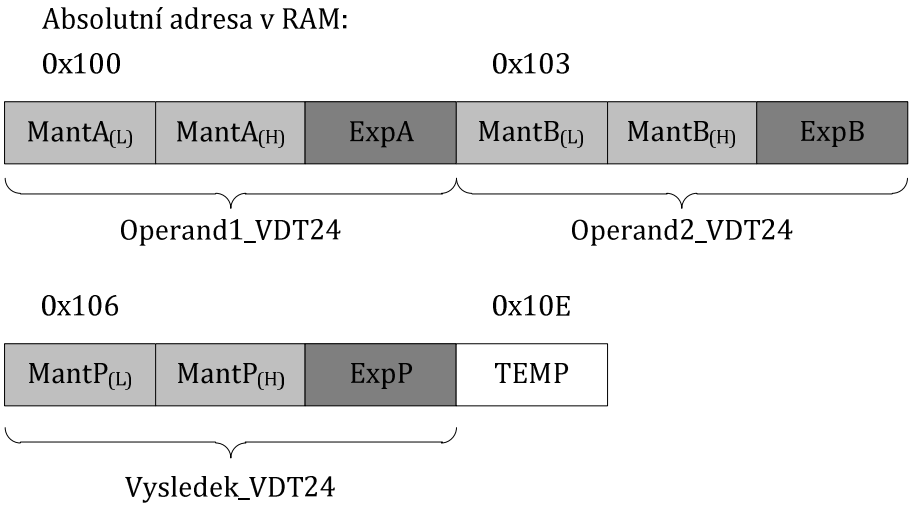
1) Násobení *_umul8a*, jak se dalo předpokládat, je výrazně náročnější na výpočetní výkon, než ostatní 8bitová násobení používající HW násobičku. Použití této funkce má smysl pouze u procesorů, které potřebnou periferií nedisponují.
2) Násobení *_umul8af* a standardní funkce násobení generovaná překladačem pro násobení dvou proměnných typu „unsigned char“ se v délce trvání neliší a je proto dále možné používat funkci v dané situaci vhodnější z hlediska adresování operandů a výsledku.
3) Při násobení neznaménkových 16bitových proměnných typu „unsigned int“ je výrazně výhodnější použití vytvořené Assemblerovské funkce, než standardního násobení generovaného překladačem, který, dle dalších měření, nedisponuje optimalizovaným algoritmem pro neznaménkové násobení 16x16. Pro násobení větší než 8x8 používá MPLAB univerzální funkci použitelnou i pro znaménkové operace rozměru M x N, kde M a N je libovolný počet bitů operandů s limitem $M + N \leq 32$.

5.3 Implementace násobení VDT24

Obdobně (jako v podkapitole *5.2*) byl pro praktickou zkoušku zvolen „**Modul A**“ osazený 8bitovým procesorem PIC18F46K80 (32) s hardwarovou podporou celočíselného neznaménkového násobení rozměru 8x8bitů. Metodika měření výpočetní náročnosti za pomoci „TIMERU 1“ zůstává stejná, jako v předchozím případě. Funkce násobení **VDT24** je provedena ve dvou variantách a to kompletní, která v případě mantisy provádí plný součin rozměru 16x16 se 32bitovým výsledkem a redukovanou zanedbávající spodní 2 bajty výsledku, tedy s redukovanou přesností na úrovni dvou LSB mantisy. Pro zachování přijatelné doby výpočtu testu není prováděno násobení pro všechny možné kombinace mantis a exponentů. Hlavní zdrojový soubor testu, ze kterého je patrné s jakým omezením vyčíslení výpočetní náročnosti probíhalo, je nazván *Main_mulVDT.c*. Zdrojový kód násobení je psán v jazyce Assembler a v plné okomentované verzi je rovněž k dispozici na přiloženém CD pod názvem *VDT24_18F.asm*. Obsahuje následující funkce:

1) *_mulVDT24* – kompletní násobení dvou čísel typu **VDT24** se zohledněním příznaku přetečení všech bytů výsledku násobení mantisy
2) *_mulVDT24f* – zrychlená verze násobení dvou čísel typu **VDT24** s vynechaným vyčíslením spodních dvou bytů a příznaků přetečení produktu součinu mantis

U funkcí je předpokládáno externí ošetření nenulovosti vstupu. Očekávány jsou operandy, jejichž exponenty jsou takové velikosti, která u výsledku nezapříčiní přetečení rozsahu nebo dosažení hranice minimálního rozlišení. Vstupy a výstupy funkcí násobení jsou opět provedeny za pomoci statického mapování. Mapa relevantní části RAM paměti je na *Obr. 5.3.*1. Styčná rozhraní mezi assemblerovským a C prostředním jsou opět realizovány vhodnými



Obr. 5.3.1: Organizace paměťového prostoru funkcí ze souboru VDT24_18F.asm

direktivami. Jejich přehled je v *Tabulka 5.3.1*. Jako v případě celočíselného násobení, i zde je nutné v nastavení projektu vyjmout příslušné adresy z volně využitelného prostoru.

Mapování v jazyce Assembler	Mapování v jazyce C
MantA_L equ 0x100	unsigned int **MantisaA** @ 0x100;
MantA_H equ 0x101	unsigned char **ExponentA** @ 0x102;
ExpA equ 0x102	unsigned int **MantisaB** @ 0x103;
MantB_L equ 0x103	unsigned char **ExponentB** @ 0x105;
MantB_H equ 0x104	unsigned int **MantisaP** @ 0x106;
ExpB equ 0x105	unsigned char **ExponentP** @ 0x108;
MantP_L equ 0x106	unsigned char **TEMP1** @ 0x10E;
MantP_H equ 0x107	
ExpP equ 0x108	

Tabulka 5.3.1: Organizace paměti pro předávání proměnných mezi jazyka Assembler a C

Výsledkem měření výpočetní náročnosti jsou pro obě funkce opět rozsahy minimálních a maximálních dob trvání pro různé velikosti a znaménka operandů. Jak již bylo řečeno, měření z časových důvodů neproběhlo pro všechny možné kombinace vstupů, a proto nejsou (na rozdíl od dříve prezentovaných výsledků celočíselných funkcí) uvedeny hodnoty, ve kterých bylo mezních parametrů dosaženo. Hodnocena rovněž není přesnost výpočtů založená na komparaci s výsledkem celočíselného násobení, které má v rámci možných vstupních hodnot absolutní přesnost. K takovému testu by bylo zapotřebí definovat operace násobení, dělení a sčítání pro 80bitové proměnné umožňující svým rozsahem pokrýt veškeré možné hodnoty **VDT24**. Takovéto zkoušky již přesahují rámec práce, ovšem jsou zajímavým námětem pro získání dalších parametrů spojených s **VDT24**. Přehled výsledků provedených měření je v *Tabulka 5.3.2*. Jednotkou je počet instrukčních cyklů [**ic**].

Funkce	Doba trvání
_mulVDT24	**53** až **59** ic
_mulVDT24f	**42** až **46** ic

Tabulka 5.3.2: Výpočetní náročnost násobení VDT24

Ze získaných údajů vyplývají následující poznatky:

1) Rychlost výpočtu *_mulVDT24f* je v průměru o 27,3% vyšší, než rychlost výpočtu *_mulVDT24* a to za cenu relativně zanedbatelného zhoršení přesnosti (viz podkapitola *4.5*.) a kompletní verzi *_mulVDT24* tedy nemá smysl používat.
2) V porovnání s celočíselným násobením je poměr zvětšení dynamického rozsahu nesrovnatelný s poměrem zvýšení náročnosti na výpočetní výkon a to i bez zohlednění faktu, že celočíselné násobení, právě z důvodu udržení co nejmenších požadavků na hardware, bylo prezentováno pouze jako neznaménkové.

5.4 Řešení funkcí výpočtu kernelu *_kernelVDT24* a *_kernelHINT16*

Dle informací sepsaných v podkapitole *4.6* byly pro zařízení „**Modul A**“ vytvořeny funkce *kernel VDT24* a *kernel HINT16* ve verzi bez použití MAC instrukce s bipolárními vstupy ve tvaru absolutní hodnoty se znaménkovým bitem na pozici MSB. Z důvodu požadavků na maximální výpočetní rychlost byl použit jazyk Assembler. Vstupní a výstupní rozhraní obou funkcí je přizpůsobeno dalšímu použití. V souladu s teoretickými podkapitolami je navržena struktura předávacího zásobníku vstupů, vah a výstupu. V této fázi ještě není připojena aktivační funkce. Předpokládá se využití stejného výstupního prostoru avšak s odpovídajícím způsobem změněným datovým typem.

Funkce *kernelu VDT24* nazvaná *_kernelVDT24* využívá výhradně proměnných typu **VDT24**. Vstupem jsou vektory **X**, **W** a proměnná klidového potenciálu **B**. Výstupem je akumulační proměnná **U**. Z důvodu nutnosti zarovnání produktů násobení před akumulací do výstupních registrů jsou tyto uchovávány v podobě vektoru mezivýsledků **P**. Akumulátor samotný má pro účely sumace prvků **P** rozšířenou mantisu na 4 bajty. Mapovány jsou i pomocné celočíselné 8bitové registry **TEMP0** až **TEMP6** a 16bitový stavový registr **STATUS**. Z těchto jsou prozatím využit pouze **TEMP0** až **TEMP2**. Přístup z jazyka „C“ je možný prostřednictvím ukazatelů. Předpokládá se práce s rozhraním v podobě **X**, **W**, **B** a **U**. Smysluplnost hodnot uložených v ostatních registrech po skončení funkce není zaručena. *_kernelVDT24*, na rozdíl od dříve prezentovaných násobení, již používá pro **X** a **W** relativní adresování pomocí registrů a instrukčních operandů „FSRx“, „INDFx“, „PREINCx“, „POSTINCx“ a „POSTDECx“ (32). Parametry nastavovanými při překladu funkce jsou konstanty **COUNT** vyjadřující počet prvků vstupního a váhového vektoru, **BANK_point** jako konstanta označující použitou paměťovou banku RAM a **FSR_point** jako 12bitová konstanta označující absolutní adresu počátku prostoru RAM obsazeného zásobníkem. Odsud vyplývá i technické omezení implementace *kernelu VDT24* na procesoru s jádrem PIC18 – jedním neuronem nelze adresovat více paměťového prostoru, než 255bajtů. Počet vstupů neuronu je proto fakticky omezen na 26. O přípravu - vyčištění a přednastavení proměnných – se stará funkce *_feedkVDT24*. Mapování zásobníku tvořícího rozhraní s hodnotami pro testovací vzorek je v *Tabulka 5.4.1.*

Konstanta		**Hodnota**	
COUNT		0x08	
Bank_point		0x02	
FSR_point		0x200	
Offset v RAM oproti FSR_point	**Proměnná**	**Offset v RAM oproti FSR_point**	**Proměnná**
0x00	TEMP0	0x0D	MantisaB_L
	TEMP1		MantisaB_H
	TEMP2		ExponentB
	TEMP3	0x10+(n*9)	MantisaXn_L
	TEMP4		MantisaXn_H
	TEMP5		ExponentXn
0x06	STATUS_L	0x13+(n*9)	MantisaWn_L
	STATUS_H		MantisaWn_H
0x08	Akum_LL (LSByte)		ExponentWn
	Akum_LH	0x16+(n*9)	MantisaPn_L
0x0A	Akum_L = MantisaU_L		MantisaPn_H
	Akum_H = MantisaU_H		ExponentPn
	ExponentU		

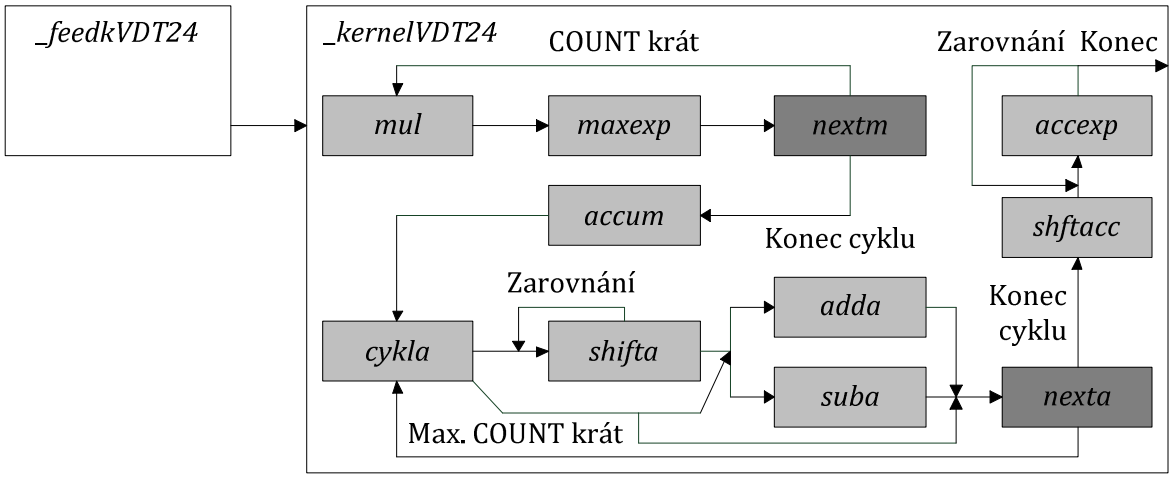
Tabulka 5.4.1:Mapa zásobníku funkce _kernelVDT24 v paměti RAM včetně konstant

V části začínající adresovým offsetem **0x10+(9*n)** začíná periodicky se opakující oddíl korespondujících prvků vektorů **X**, **W** a **P** s indexy **n** v intervalu <**0**;**COUNT-1**> ∈ **N**+. Práce s nimi, jak bylo řečeno, probíhá mechanismy nepřímého adresování. Začátek zásobníku, tedy adresové offsety 0x00 až 0x0F, jsou adresovány absolutně na základě zadaných konstant.

Tělo funkce *_kernelVDT24* je rozděleno na několik částí. Jejich výčet se stručným popisem je uveden v *Tabulka 5.4.2*. Diagram činnosti pak na *Obr. 5.4.*1. Největším problémem z hlediska nároků na výpočetní výkon je u této implementace zjevně akumulace do registru **U** začínající přípravným oddílem *accum*. V plném rozsahu se projevuje nevýhoda procesorového jádra PIC18, jehož instrukční sada má k dispozici pouze instrukci rotace registrů o jeden bit. To vede ke vzniku smyček spojených s další vlastní režií v podobě skoků v programu (cyklus zarovnání *shfta*), které je navíc třeba v rámci zarovnání všech relevantních prvků vektoru **P** provádět až **COUNT** násobně.

Část funkce	Popis
_feedkVDT24	Funkce je deklarována jako globální, volá se separátně, přednastavuje pracovní proměnné TEMPx na hodnoty **COUNT** a **COUNT**+1 a přednastavuje akumulátor na počáteční hodnotu 0x8000 0000
_kernelVDT24	Funkce je deklarována globálně, dále se dělí na níže popsané části. V základu nastavuje registry nepřímého adresování a volí banku RAM paměti dle BANK_point a FSR_point
mul	Provádí samotné násobení vstupu **X** a váhy **W** do výsledku **P**
maxexp	Porovnává současnou hodnotu exponentu akumulačního registru s exponentem aktuálního výsledku **P**. Exponent **U** je nahrazen, pokud **ExpP>ExpU**
nextm	Skok zpět na funkci *mul*, dokud není dosaženo počtu opakování **COUNT**. Poté přechod na *accum*
accum	Přenastavení registrů nepřímého adresování na **Bias** a **U** registry
cykla	Výpočet vzdálenosti **ExpU** a současného **ExpP**. **ExpU** je již maximum ze všech **ExpP**. Je-li vzdálenost větší, než 15, přeskakuje na *nexta*, je-li shodná, jde na *adda*/*suba*, není-li, jde na *shifta*
shifta	Smyčka cyklického posouvání mantisy aktuálního **P**, dokud není srovnán **ExpP** s **ExpU**. Pak jde na *adda*/*suba*
adda	Při shodě znamének akumulátoru **U** a současného produktu násobení **P** provede součet
suba	Při neshodě znamének akumulátoru **U** a současného produktu násobení **P** provede rozdíl
nexta	Pokud neproběhla smyčka sčítání *cykla* pro všechny prvky vektoru **P**, nastaví registry nepřímého adresování na další bod a volá *cykla*. Pokud vše proběhlo, přechází na *shftacc*
shftacc	Příprava na zarovnání **U** do platného tvaru **VDT24** s nenulovým MSB mantisy. Zbavuje akumulátor offsetu 0x8000 0000 a v případě záporného výsledku převádí mantisu dvojkovým doplňkem na absolutní hodnotu
accexp	Rotuje **MantU** do platného tvaru **VDT24**, nakonec upravuje **ExpU** přičtením konstanty 16 kompenzující zkrácení 32bitového akumulačního registru na 16bitovou mantisu **MantU**. Končí funkci

Tabulka 5.4.2: Popis částí funkce výpočtu kernelu VDT24



Obr. 5.4.1: Schéma činnosti funkce výpočtu kernelu VDT24

Obdobným způsobem jako je popsáno výše je řešeno i rozhraní funkce *_kernelHINT16* realizující hybridní výpočet vnitřního potenciálu neuronu se 16bitovými celočíselnými znaménkovými vstupy, váhami a klidovým potenciálem ve formátu s MSB ve formě znaménkového bitu a výstupem typu **VDT24**. Funkce je nazvána *_kernelHINT16*. Její oddělenou součástí je, obdobně jako v předchozím případě, globálně definovaná funkce *_feedkHINT16* určená pro přípravu pracovních proměnných.

Podstatnou principiální změnou oproti předchozí je odlišná filosofie zpracování dílčích produktů násobení vektorů **X** a **W**. Mezitím co *_kernelVDT24* je z principu uchovával v podobě **P** vektoru za účelem pozdějšího posouzení relevantnosti vzhledem k výslednému součtu **U** a úpravě na předem neznámý společný exponent, probíhá u *_kernelHINT16* přímá akumulace parciálních produktů algoritmu násobení 16x16 s hardwarovou násobičkou 8x8 do celočíselného registru **U**. Tím je zároveň dosažena značná úspora paměti RAM. Pro práci se vstupy a váhami je opět využito nepřímého adresování, přístup k pracovním proměnným a výstupu je statický skrze konstantní ukazatel řešený při překladu. Mapa předávacího zásobníku hodnot včetně konstant pro ukázkovou aplikaci je uvedena v *Tabulka 5.4.3*.

Konstanta		Hodnota	
COUNT		0x08	
Bank_point		0x02	
FSR_point		0x200	
Offset v RAM oproti FSR_point	**Proměnná**	**Offset v RAM oproti FSR_point**	**Proměnná**
0x00	TEMP0	0x0A	Akum_LHL
	TEMP1		Akum_LHH
	TEMP2		Akum_HLL
	TEMP3		Akum_HLL
	TEMP4	0x0E	Bias_L
0x05	ExponentU		Bias_H
0x06	STATUS_L	0x10+(n*4)	Xn_L
	STATUS_H		Xn_H
0x08	Akum_LLL = MantisaU_L	0x12+(n*4)	Wn_L
	Akum_LLH = MantisaU_H		Wn_H

Tabulka 5.4.3: Mapa zásobníku a konstant funkce _kernelHINT16

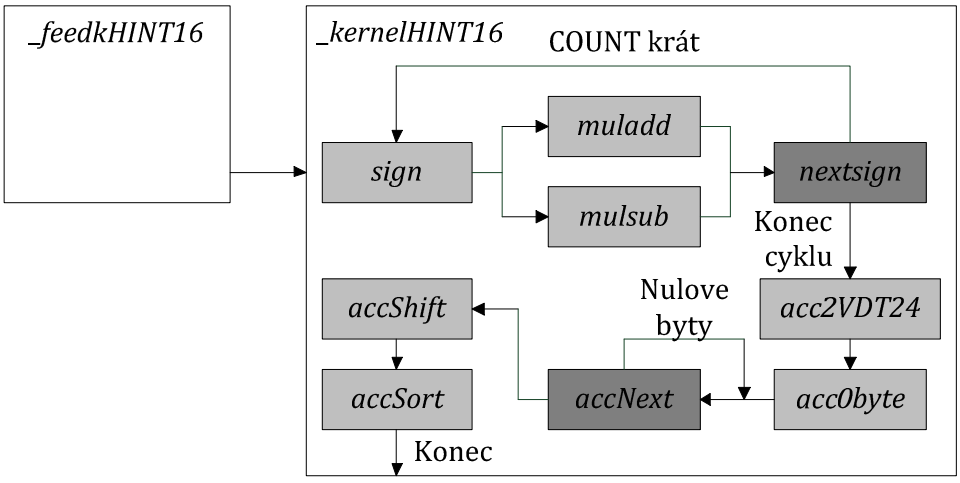
Dělení funkce na menší celky je znázorněno v *Tabulka 5.4.4*. Grafický přehled pak na *Obr. 5.4.*2. Samotnému výpočtu předchází funkce *_feedkHINT16*. Jejím úkolem je vyčistit pracovní proměnné **TEMPx**, přednastavit 48bitový celočíselný akumulační registr na hodnotu 0x8000 0000 0000, přičíst/odečíst klidový potenciál a vynulovat exponent akumulačního registru **VDT24**. Zároveň přednastavuje registry nepřímého adresování. Následný počáteční oddíl funkce *_kernelHINT16* má alias *sign* a pomocí exkluzivního součtu MSB odpovídajících si prvků vektorů **X** a **W** dělí další postup na násobení s přičítáním k akumulačnímu registru *muladd*, nebo násobení s odečítáním od akumulačního registru *mulsub*. Zároveň zálohuje „high byte“ vstupu **X** do **TEMP3** registru a v původním **X** maže znaménkový bit. (K rušení znaménkového bitu **W** dochází pouze v pracovním registru procesoru **Wreg**, není jej proto třeba zálohovat). Tím je násobení převedeno na striktně neznaménkové. Současně s akumulací do registru **U** jej řeší oddíly *muladd* a *mulsub*. Následuje uzavření smyčky *nextsign* zajišťující **COUNT** násobné opakování průběhu funkce od návěstí *sign*. Následuje převod celočíselného akumulátoru s offsetem na hodnotu typu **VDT24**. Výmaz offsetu probíhá v části *Acc2VDT24* prostřednictvím negace MSB **Akum_HLH** a jeho převedením do **ExpU**. Je-li akumulátor menší, než 0x8000 0000 0000, je negace MSB = 1 tj. záporné číslo a doplnění nevyužitých bitů akumulátoru na „samé 1“, je-li akumulátor větší, nebo roven 0x8000 0000 0000, je negace MSB = 0 tj. kladné číslo a

doplnění nevyužitých bitů akumulátoru na „samé 0“. V případě záporného výsledku následuje bitová negace celočíselného **U**, čímž (s nepřesností na LSB) dojde k vytvoření absolutní hodnoty. Část *acc0byte* předpřipraví konstanty a ukazatele nepřímého adresování pro činnost oddílu *accNext* řešícího smyčku, v níž se detekuje nulovost maximálně 4 nejvyšších bytů celočíselného akumulátoru. Zde je nalezeno optimální zarovnání budoucí mantisy **U** ve tvaru **VDT24** tak, aby byl potřebný co nejmenší počet problematických bitových rotací pro úpravu na nenulový MSB **MantU**. Zároveň je odpovídajícím způsobem upravován i **ExpU**. Úpravu dokončí smyčka *accShift*. Z funkce *_kernelHINT16* se odchází přes úsek nazvaný *accSort*, jehož úkolem je přemístit mantisu **U** na správnou pozici v zásobníku.

Část funkce	Popis
_feedkHINT16	Připravuje pracovní proměnné **TEMPx**, přednastavuje 48bitový celočíselný akumulátor **U** na hodnotu 0x8000 0000 0000 a přičítá/odečítá klidový potenciál neuronu **B**. Přednastavuje registry nepřímého adresování na požadované hodnoty
_kernelHINT16	Je globálním aliasem pro následující část – *sign*
sign	Dle exkluzivního součtu MSB operandů násobení z vektorů **X** a **W** stanoví znaménko budoucího produktu násobení, zálohuje prvek vektoru **X** a přechází na operaci *muladd* v případě kladného, nebo *mulsub* v případě záporného výsledku
muladd	Provede algoritmus shodný s celočíselným neznaménkovým násobením 16x16 na systému s hardwarovou podporou násobení 8x8. Jednotlivé meziprodukty neskládá do separátního produktu násobení, ale přímo přičítá k akumulačnímu registru **U**
mulsub	Má podobnou funkci, jako *muladd* s tím rozdílem, že meziprodukty násobení od akumulačního registru **U** odečítá
nextsign	Posouvá registry nepřímého adresování na další prvky vektorů **X** a **W** a zajišťuje opakování od značky *sign*, dokud neproběhne konstantou **COUNT** stanovený počet násobení a akumulací. Funkce zároveň navrací ze zálohy původní hodnotu operandu **X** zbaveného v části *sign* znaménka
acc2VDT24	48bitový celočíselný registr **U** zbavuje offsetu negací MSB. Tento zároveň přesouvá do registru **ExpU** na pozici znaménkového bitu mantisy. Negovaný bit v akumulátoru v případě záporných čísel doplňuje nepoužité bity na řadu jedniček, v případě kladného čísla

	na řadu nul. Záporná hodnota je bitovým doplňkem převedena na absolutní
acc0byte	Detekuje nejvyšší nenulový byte celočíselného akumulátoru **U**. Není-li to žádný ze 4 nejvyšších, budou v následujících krocích do mantisy **U** ve formátu **VDT24** přesunuty spodní 2 bajty celočíselného **U**. V jiném případě jsou nejvyšší nenulový bajt a dva za ním následující postoupeny k dalšímu zpracování
accNext	Smyčka uzavírající činnost *acc0byte* a zajišťující její správný průběh. Zároveň upravuje **ExpU** tak, aby odpovídal budoucímu počtu případně zahozených LSB
accShift	Smyčka zarovnávající bitovou rotací vybrané dva nejvyšší nenulové bajty na formát mantisy **VDT24** s nenulovým MSB
accSort	Překopíruje zarovnanou mantisu do odpovídajícího paměťového prostoru a ukončí funkci *_kernelHINT16*

Tabulka 5.4.4: Popis částí funkce výpočtu kernelu HINT16



Obr. 5.4.2: Grafické schéma činnosti funkce _kernelHINT16

Pro srovnávací měření výpočetních nároků obou funkcí byl zvolen test založený na podobném principu, jako v podkapitolách *5.2* a *5.3*. Oba typy výpočtu kernelu byly nastaveny jako osmivstupé, s předem náhodně zvolenými bipolárními váhami a vstupy. Ty jsou z důvodu porovnatelnosti výsledku shodné (z oboru 16bitových celých čísel). Při měření doby trvání výpočtu „TIMEREM 1“ byly zahrnuty i funkce *_feedkVDT24* a *_feedKHINT16*. Výsledky jsou shrnuty v *Tabulka 5.4.5*.

Seznam operandů		
X	**W**	
3120	456	
-8327	27789	
13	-637	
4576	562	
-158	-9235	
236	4357	
305	3409	
7640	-15437	
Klidový potenciál neuronu	0	
Reálný výsledek	-341824405	
Funkce	*_kernelVDT24 + feed*	*_kernelHINT16 + feed*
Doba výpočtu	**1260** ic	**594** ic
Výsledek	**-41669*2^{13}**	**-41726*2^{13}**
Odchylka od reálné hodnoty	**-0,1381**%	**-0,0025**%

Tabulka 5.4.5: Výsledky měření náročnosti funkcí _kernelVDT24 a _kernelHINT16

Výše uvedené výsledky nemají kvalitní vypovídací hodnotu o rozpětí odchylek výstupních hodnot spočítaných funkcemi oproti reálné, neboť jsou vyčísleny pouze v jednom bodě. Z principu výpočtů, které v *kernelech* probíhají, jistě nelze očekávat odchylky ve výrazně vyšších řádech, než jsou nižší desetiny procent, což pro neuronové sítě postačuje. Tento test domněnku nevyvrací. Mnohem zajímavějším a z hlediska aplikace relevantnějším výsledkem je doba trvání výpočtu, která, včetně vlastní režie, vychází v průměru na **74,25ic/vstup** pro *_kernelHINT16* a **157,5ic/vstup** pro *_kernelVDT24* při 8mivstupém neuronu, což potvrzuje teoretické závěry podkapitoly *4.6* a hovoří ve prospěch používání *_kernelHINT16*. Pro upřesnění představy o vztahu náročnosti na výpočetní výkon a počtu vstupů neuronu bylo provedeno další měření, kde při shodných hodnotách vektorů **X** a **W**, jako v *Tabulka 5.4.5,* proběhl výpočet pouze pro prvních 2, 4 a 6 prvků. Dosažené hodnoty včetně původního testu jsou v *Tabulka 5.4.6*. V porovnání s náročností samotného násobení příslušného datového typu (výsledky jsou uvedeny v pokapitolách *5.2* a *5.3*) je zřejmé, že dominantní roli v náročnosti kernelů hraje vlastní režie a práce s akumulátorem. To je dle teoretických předpovědí patrné zejména u funkce *_kernelVDT24*, kde i při osmivstupém neuronu dosahuje průměrný výkonový požadavek více než trojnásobku požadavku samotného násobení. Pro *_kernelHINT16* dosahuje stejný parametr pouze dvojnásobku. Tato fakta ovšem pouze potvrzují nutnost tvorby speciálního *kernelu* pro

výpočet neuronů vstupní vrstvy, kde se předpokládá vždy pouze jeden váhovaný vstup a klidový potenciál. Možné je i řešení přímého zadávání hodnot bez přepočtu neurnovou jednotkou.

Počet vstupů	Funkce _kernelVDT24		Funkce _kernelHINT16	
	Absolutní náročnost	Relativní náročnost	Absolutní náročnost	Relativní náročnost
2	**453**ic	**226,5**ic/vstup	**230**ic	**115**ic/vstup
4	**756**ic	**189**ic/vstup	**354**ic	**88,5**ic/vstup
6	**1034**ic	**172,3**ic/vstup	**480**ic	**80**ic/vstup
8	**1260**ic	**157,5**ic/vstup	**594**ic	**74,25**ic/vstup

Tabulka 5.4.6: Srovnání absolutní a relativní náročnosti kernelů při různém počtu vstupů

Testované funkce jsou k dispozici na přiloženém CD v souboru *kernel_18F.asm* s podporou hlavičkového souboru *Kernel.h*. Samotný test je zapsán v hlavním souboru *main_kernel.c*.

5.5 Praktický výpočet aktivační funkce

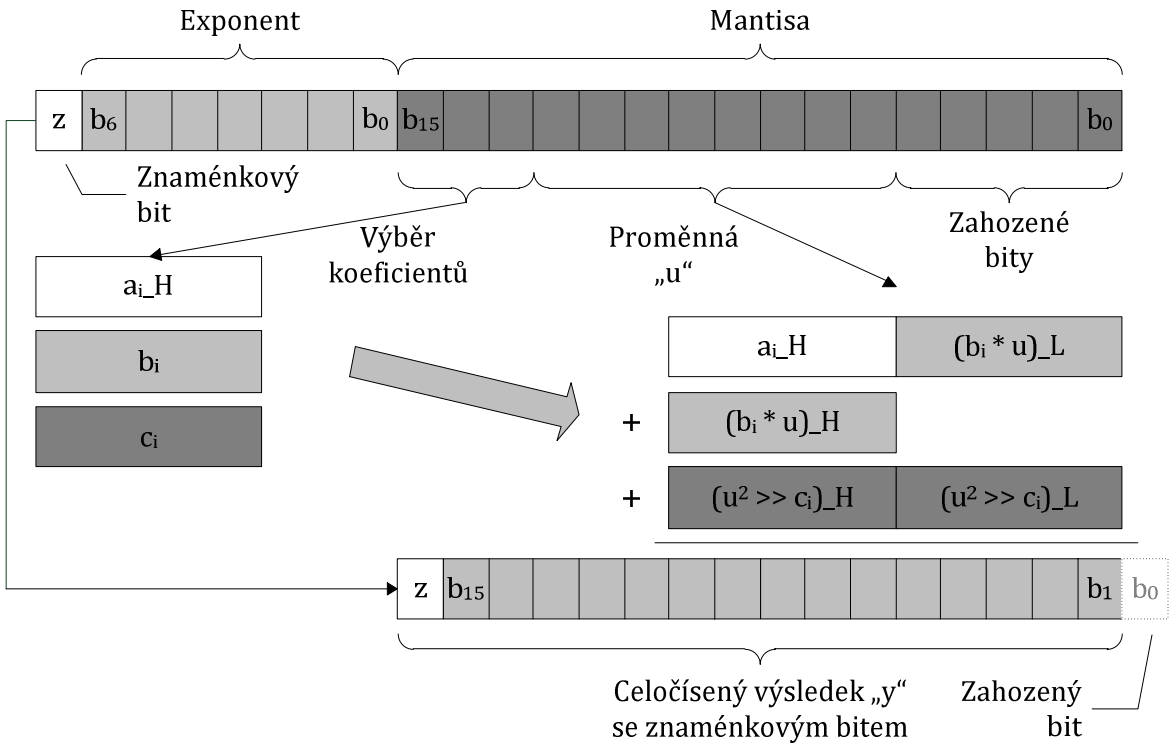
Řešení výpočtu aktivační funkce neuronu bylo provedeno dle poznatků uvedených v podkapitole *4.7*. Pro **„Modul A“** byla v jazyce Assembler vytvořena funkce *_aktfunc* aproximující pomocí kvadratických splajnů průběh hyperbolické tangenty. Jako vstup je očekáván datový typ **VDT24**. Jelikož je uvažováno použití v kombinaci s HINT16 kernelem, je v rámci úspor výpočetního výkonu u **VDT24** zrušeno zarovnání na nenulový MSB v případě, že jím reprezentovaná hodnota nevybočuje z intervalu <**-65535**;**65535**>. Výstupem je 16bitová celočíselná hodnota s rozsahem <**-32256**;**32256**> (samotná funkce generuje dvojnásobné hodnoty, které jsou následně bitovou rotací zmenšeny na polovinu) ve formátu se znaménkovým bitem na pozici MSB tak, jak ji vyžadují vstupy funkce *_kernelHINT16*. Konstanty jednotlivých aproximačních úseků jsou zavedeny do RAM paměti funkcí *_aktfuncTAB* spouštěné jednou na začátku programu.

Diskutovaná funkce realizuje pomocí kvadratických splajnů aproximaci části kladné větve hyperbolické tangenty popsané *Vztah 4.7.3* a rozdělené do osmi úseků. Je-li absolutní hodnota vstupu větší, než **65535**, tedy jedná-li se, u datového typu **VDT24**, o číslo s exponentem větším, než **0x40**, je automaticky z funkce navrácena hodnota **±32256**. Vstupy s exponentem menším, než **0x40** jsou nepřípustné. Zadávání hodnot menších, než 32768 je řešeno zmiňovaným porušením pravidla zarovnání mantisy **VDT24** na nenulový MSB. Tím je redukována nadbytečná režie jak v podobě zarovnávání vnitřního potenciálu neuronu ve funkci

_kernelHINT16, tak v podobě vyhodnocování čísel s nízkým exponentem v *_aktfunc*. Rozlišení je z principu použitého kernelu omezeno na celočíselnou jednotku. Takto ošetřený vstup je dále redukován bitovou rotací na 11bitovou hodnotu, kde horní tři bity slouží jako adresa úseku, tedy pro načtení příslušných konstant $\mathbf{a_i}$, $\mathbf{b_i}$ a $\mathbf{c_i}$ (viz konvence v podkapitole *4.7*) a spodních osm bitů je vstup. Tím je na použitém procesoru zároveň zajištěna možnost použít hardwarovou podporu výpočtů 8x8. Další velkou výhodou je také fakt, že pak pro po sobě jdoucí splajny platí: $\mathbf{S_{i+1(0)} = S_{i(256)}}$. Tím jsou zaručeny následující úspory paměti RAM a výpočetního výkonu:

1) V případě, kdy žádný z kvadratických koeficientů $\mathbf{c_i}$ není menší, než $\mathbf{1/256}$ – což pro tabulku vypočtenou v podkapitole *4.7* zajištěno je – budou funkční hodnoty všech splajnů v bodě **x=0** násobky 256, stejně jako jejich koeficienty $\mathbf{a_i}$, u kterých proto nebude třeba ukládat vždy nulový nižší bajt.
2) Koeficienty $\mathbf{a_i}$ s nulovým nižším bytem zredukují operaci $\mathbf{a_i + b_i*x}$ z jednoho násobení 8x8 a jednoho 16bitového sčítání (tedy dvě 8bitová sčítání s řešením carry out bitu součtu nižších bytů) na jedno 8bitové sčítání (vyšší bajty) a jeden přesun do registru (nižší bajt výsledku násobení do nižšího bajtu výsledné funkční hodnoty)

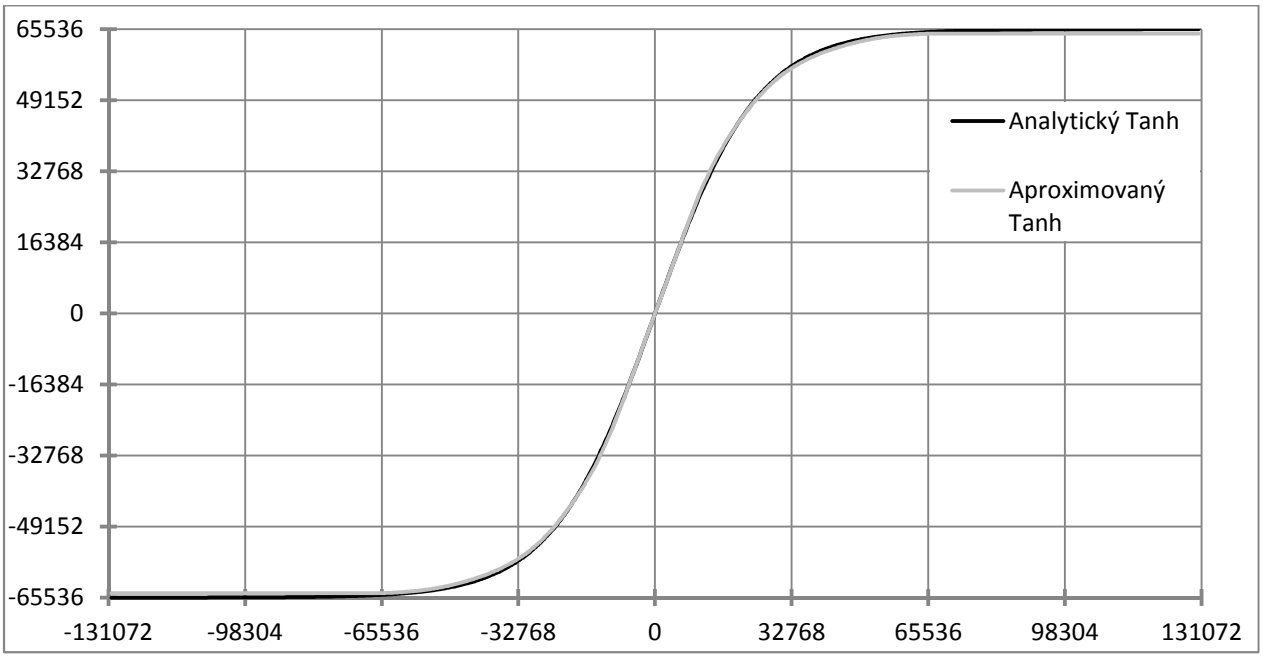
Činnost *_aktfunc* pro vstupní intervaly, kde pracuje aproximace splajny, tedy pro **Exponent = 0x40**,**0xC0**, je schematicky znázorněna na *Obr. 5.5.*1.



Obr. 5.5.1: Struktura algoritmu _aktfunc

Výše jmenované výhody spojené se segmentací jsou zobecnitelné i na ostatní polynomatické aproximace. V referenční disertační práci (9), kde je vysvětlen vlastní přístup pro stanovení hodnoty hyperbolické tangenty rovněž založený na nahrazování subintervalů kvadratickou funkcí, jsou nevhodně použity úseky délky 128, což nesplňuje výše uvedené body a zbytečně zvyšuje náročnost algoritmu.

Funkce byla testována pro vstupní celočíselný interval <**-130336**;**130336**> s inkrementačním krokem **600** pro hodnoty **|65536|** a větší, **300** pro ostatní. Pro porovnatelnost s původní křivkou z podkapitoly *4.7* byla vyblokována rotace výsledku o jeden bit vpravo, je tedy ponechán rozměr výstupu <**-64512**;**64512**>. Graf výsledku je na *Obr. 5.5.*2. Z hlediska výpočetního výkonu algoritmus zabere **14** instrukcí v případě, že je vstup mimo hranice výpočtu a je vracena saturační hodnota, **77** až **101** instrukcí v případě, že je počítána splajnová aproximace.



Obr. 5.5.2: Graf výsledku aproximace hyperbolické tangenty "Modulem A"

Zdrojový kód aktivační funkce a funkce zavádějící napočítané konstanty do paměti RAM jsou k dispozici na přiloženém CD v soboru s názvem *aktfunc_18F.asm*. Potřebné deklarace pro jazyk „C“ pak v hlavičkovém souboru *Aktfunc.h*. Test, jehož výsledky byly prezentovány v této kapitole, je sestaven v souboru *main_tanh.c*. Stejně jako v předchozích případech je i zde zdrojový kód okomentován.

5.6 Kompletní výpočet neuronové jednotky

Neuronová jednotka v rozsahu specifikovaném v podkapitole *4.1* je složením *kernelu* a aktivační funkce. Použít lze, v závislosti na požadovaných parametrech, prakticky libovolné kombinace představené v teoretických podkapitolách. Pro účely této práce je vytvořena *neuronová jednotka* z funkcí *_kernelHINT16* a *_aktfunc*. Jsou částečně upraveny tak, aby na sebe vhodně navazovaly a nebyly nadměrně výpočetně náročné. Jako realizační hardware je opět zvolen „**Modul A**“. Vzhledem k tomu, že tomuto subtématu není věnována samostatná stať v části popisující obecnou metodu, budou případné teoretické pasáže začleněny do této podkapitoly.

Aby byly výše uvedené funkce vzájemně kompatibilní, je třeba u obou sladit vstupy a výstupy. Zároveň je nutné přijmout fakt, že neuronové sítě používající celočíselnou aritmetiku nejsou z principu schopné na svých jednotlivých vstupech $\mathbf{X_n}$ dosahovat směrem k vnitřnímu potenciálu neuronu **U** přenosů v intervalu (**-1;1**). Úskalí těchto vlastností jsou detailněji

diskutovány například v (33), (34) a částečně i v (15). Vlastnosti vyžadující pozornost při přizpůsobování lze shrnout do následujících bodů:

1) Výstupní rozsah aktivační funkce musí z důvodu návaznosti na další neurony odpovídat vstupnímu rozsahu neuronu: $\mathbf{H_{f(\_aktfunc)} = D_{f(\_kernelHINT16)}}$
2) Dílčí produkty násobení vektorů **X*W** by měly být dodatečně upraveny dělením nebo bitovou rotací tak, aby bylo možno s rozumným krokem dosáhnout přenosu vstupů **X** v intervalu (**-1;1**)
3) Vstup aktivační funkce by měl být nastaven tak, aby byl každý prvek součinu **X*W** sám o sobě – při vhodně nastavené váze - schopen vyvolat její saturaci

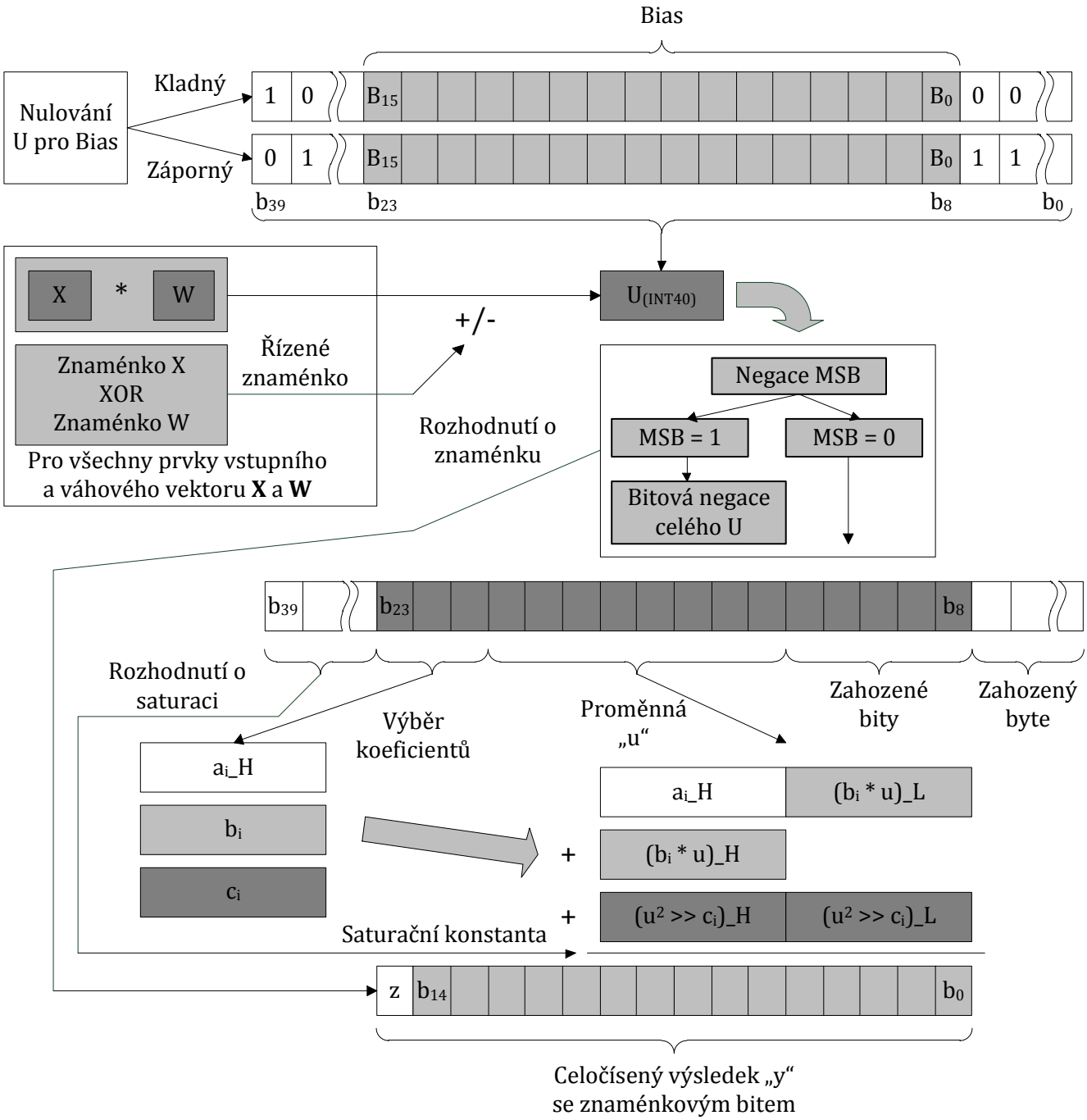
V souladu s uvedenými body byly v ukázkové aplikaci nahrazeny původní koeficienty splajnové aproximace hyperbolické tangenty uvedené v podkapitole *4.7* a použité v příkladu *5.5* nahrazeny takovými, které jsou vypsány v *Tabulka 5.6.1*. Tento postup je nutný pro omezení výstupních hodnot na 16bitové znaménkové. Konkurenční možností je výše uvedená bitová rotace/dělení dvěma výstupu dříve představené verze aktivační funkce. Vzhledem k celočíselnému charakteru koeficientů, zejména omezené volbě $\mathbf{c_i}$ lze očekávat, že pro některé volby parametrů $\mathbf{M_{out}}$, $\mathbf{M_{in}}$ (podkapitola *4.7*) bude omezena přesnost a nelze proto obecně jednoznačně určit, který z přístupů je lepší (konkrétně v tomto případě je věrnější původní aproximace, výsledky je možné vyvodit ze souborů pro MS Excel 2010 na přiloženém CD).

Z hlediska vstupu zůstala filosofie práce se vstupní celočíselnou hodnotou shodná, jako v příkladu *5.5*. Pro adresování koeficientů $\mathbf{a_i}$,$\mathbf{b_i}$ a $\mathbf{c_i}$ jsou použity horní 3bity, následujících 8bitů slouží pro výpočet konkrétního bodu ve zvoleném subintervalu. Znaménko vstupu je převedeno na výstup bez jakékoli účasti na výpočtu. Podstatným rozdílem je použití celočíselné –konkrétně 40bitové, znaménkové – proměnné namísto **VDT24**. Její paměťový prostor slouží zároveň jako akumulátor vnitřního potenciálu neuronu **U**. Aby byl splněn bod 2) výčtu uvedeného na začátku této podkapitoly, je nejnižší byte zahozen. Pro stanovení výsledku aktivační funkce jsou použity pouze zbylé 4bajty, přičemž na základě nulovosti horních dvou je pouze rozhodováno o saturaci neuronu a případném přeskočení aproximace. Tím prakticky dochází k dělení produktů násobení **X*W** číslem 256 a snížení hodnoty přenosu vstupu na 1/256 pro **W=1**. Zároveň je splněn i bod 3). Každý vstup $\mathbf{X_n}$ je, bez příspěvku ostatních a klidového potenciálu neuronu, schopen navodit saturaci neuronu, je-li jeho váha $\mathbf{W_n \geq 513}$.

i	**x_i**	**$f_{(x)}$**	**$S_{i-1(x)}$**	**a_i**	**b_i**	**c_i**	**$\Delta S_{i(xi+1)}$**
0	0	0	-	**0**	**42**	**-1/256**	40
1	256	10535,33	10496	**10496**	**40**	**-1/32**	24
2	512	19096,63	18688	**18688**	**24**	**-1/64**	16
3	768	24955,92	23808	**23808**	**16**	**-1/128**	12
4	1024	28510,18	27392	**27392**	**12**	**-1/128**	8
5	1280	30510,6	29952	**29952**	**8**	**-1/128**	4
6	1536	31589,26	31488	**31488**	**4**	**-1/256**	2
7	1792	32157,47	32256	**32256**	**2**	**-1/256**	0
8	2048	32453,12	32512	**32512**	**0**	**0**	0

Tabulka 5.6.1: Konstanty a aproximační body pro upravenou aktivační funkci výpočtu jádra neuronu

Kernel neuronu založený na funkci *_kernelHINT16* má zjednodušený závěr výpočtu, tedy převod akumulačního registru **U** na typ **VDT24**, jak je uvedeno výše. Klidový potenciál **Bias** je, v souladu se zahozením nejnižšího bytu **U** před výpočtem aktivační funkce, násoben konstantou 256, což je realizováno jeho vhodným umístěním do akumulátoru v přípravné části funkce *_unitHINT16* pro kompletní výpočet neuronové jednotky. Hodnota **COUNT** dříve vystupující jako konstanta určující počet prvků vstupního a váhového vektoru nyní vystupuje jako proměnná. Vstupy **X** a váhy **W** jsou očekávány v celočíselném rozsahu <**-32767;32767**>, v podobě 15bitové absolutní hodnoty se znaménkem na pozici šestnáctého bitu (MSB). 1 pro záporné hodnoty, 0 pro kladné. Tato pro jádro **PIC18F** nestandardní reprezentace je volena z důvodu urychlení operace násobení, jak je popsáno v podkapitolách *4.6* a *5.4*. **Bias** je naproti tomu ve standardní podobě dvojkového doplňku. Je třeba počítat s tím, že záporné hodnoty **Biasu** jsou v průběhu výpočtu zvýšeny o +1. Z důvodu rychlosti není tento rozdíl kompenzován. Akumulační registr **U** je zkrácen z původních 48bitů na 40. Takto je schopen bezpečně pojmout 512 vstupů, což bohatě převyšuje potřeby budoucích aplikací na zvoleném procesoru (další zrychlování funkce je možné pouze zkracováním o celé bajty, přičemž velikost 32bitů je již nedostatečná, nepojme bezpečně více, než jeden vstup a **Bias**). Kompletní schematické znázornění popsané funkce pro výpočet neuronové jednotky je na *Obr. 5.6.*1.



Obr. 5.6.1: Schéma uspořádání funkce výpočtu neuronové jednotky _unitHINT16

Za účelem prověření správnosti a kvalitativního posouzení představené funkce je v souboru *main_unit.c* (k dispozici na přiloženém CD) sestaven test. Ten, na základě dat složených z počtu vstupů neuronu, klidového potenciálu a vlastních hodnot vstupů a vah, vypočte výstupní hodnotu neuronové jednotky a spolu s dobou trvání výpočtu měřenou shodnou metodikou, jako v předchozích praktických podkapitolách, ji opět po sériovém rozhraní vrátí. Vstupní data jsou získána pomocí generátoru náhodných čísel v tabulkovém procesoru

MS Excel 2010, doplněna o řídicí znaky a ve formátu CSV (volba „se znakovou sadou pro MS-DOS“) odeslána z PC prostřednictvím komunikačního terminálu *Realterm* (35), který zároveň zachytává výsledky generované jednotkou „**Modul A**“. Přijatý soubor je rovněž ve formátu snadno zpracovatelném tabulkovým procesorem. Datové rámce jsou popsány v *Tabulka 5.6.2*.

Znak	**Význam**
„X“	Návěstí hodnoty vstupu, bezprostředně po něm následuje volitelné znaménko „-„ a 5 číslic hodnoty vstupu
„W“	Návěstí hodnoty váhy, bezprostředně po něm následuje volitelné znaménko „-„ a 5 číslic hodnoty váhy
„B“	Návěstí klidové hodnoty **Bias**, bezprostředně po něm následuje znaménko „-„ a 5 číslic hodnoty klidového potenciálu
„C“	Nepovinné návěstí počtu prvků vektoru vah a vstupů, bezprostředně po něm následují až dvě číslice hodnoty počtu vstupů
„N“	Návěstí označující začátek dvojice prvků **X** a **W**
„S“	Příkaz k vynulování **Biasu**, aktuálně napočítaného počtu **COUNT** a ukazatele aktuálně nastavovaného páru **X**, **W**
„O“	Příkaz k odeslání aktuální hodnot maximálního a minimálního počtu instrukčních cyklů spotřebovaných pro výpočty od startu systému
„R“	Příkaz k provedení výpočtu neuronového jádra s následným kompletním výpisem použitých vstupů, váhových koeficientů, klidového potenciálu, výsledného vnitřního potenciálu neuronu, výstupní hodnoty neuronů a času výpočtu
„F“	Návěstí zrychleného výpočtu. Bezprostředně po něm následuje hodnota určující počet prvků vektorů **X** a **W**, poté klidový potenciál a odpovídající počet dvojic čísel $\mathbf{X_n}$ a $\mathbf{W_n}$, přičemž návěstí „B“, „C“, „X“ a „W“ se již nepíší. Po přečtení poslední hodnoty automaticky následuje výpočet neuronové jednotky a výpis výsledku a doby trvání výpočtu oddělených středníkem
“|”, “;“, cr, space	Znaky “|” a ”;” a klávesy *Enter* a *Space* ukončují aktuálně probíhající proces čtení číslic hodnoty
Příklad zadání výpočtu s automatickým čítáním vstupů a kompletním výsledkem	
X23845|W5873|NX16237|W24532|NX-18724|W-23421|NX-28754|W30974|B25890|R	
Příklad zrychleného zadání výpočtu s pevným počtem vstupů a výsledkem pro Excel	
SF6;4572;-1340;536;-1527;-3419;-1836;717;686;3315;-3894;28;-3510;1480	

Tabulka 5.6.2: Tabulka popisu instrukčních znaků a datových rámců pro test main_unit.c

Konkrétní data použitá pro získání níže uvedených výsledků, stejně jako nezpracované i zpracované výsledky, jsou k dispozici na přiloženém CD pod názvy *feed2.txt* až *feed8.txt* pro vstupní hodnoty, *feed2res.txt* až *feed8res.txt* pro surová výstupní data a *resulttable2.xlsx* až *resulttable8.xlsx* pro zpracovaná vstupní i výstupní data včetně grafů. Grafy jsou rovněž součástí tištěného souboru příloh této práce.

Pro testy funkce *_unitHINT16* byly použity uvedené datové soubory obsahující vždy 1000 náhodně generovaných hodnot **Bias** a vektorů **X** a **W** po 2, 4, 6 a 8mi prvcích. Interval pro **Bias**:<**-32767;32767**>, interval pro **X** a **W**:<**-4096;4096**>. Vyhodnocena následně byla rychlost výpočtu a odchylka výstupní hodnoty od ideální stanovené v tabulkovém procesoru MS Excel 2010 včetně analytického výpočtu aktivační funkce. Výsledek přehledově shrnuje *Tabulka 5.6.3*. Histogramy zobrazující procentuální rozložení četností výskytu jednotlivých pásem přesností a výpočetních časů jsou uvedený v přílohách **(C)** až **(F)**.

Dvojvstupý neuron					
Procentní odchylka od ideálního výstupu			**Doba trvání výpočtu v instrukčních cyklech**		
Maximu	**Minimum**	**Vážený průměr**	**Maximum**	**Minimum**	**Vážený průměr**
0,38%	-4,78%	2,12%	264ic	159ic	245,3ic
Čtyřvstupý neuron					
0,38%	-4,73%	1,78%	382ic	277ic	356,4ic
Šestivstupý neuron					
0,37%	-4,73%	1,67%	502ic	393ic	466,4ic
Osmivstupý neuron					
0,38%	4,78%	1,51%	620ic	509ic	577,9ic

Tabulka 5.6.3: Dosažené hodnoty přesnosti a výpočetních časů jednotky _unitHINT16 pro různý počet vstupů

Z výsledků je patrná značně vysoká záporná odchylka přesnosti, která je způsobena nepřesnou aproximací aktivační funkce. Její příčiny a možnosti odstranění jsou objasněny teorií v počátku této podkapitoly a v podkapitolách *5.5* a *4.7*. Grafické znázornění analyticky vypočtené a aproximované aktivační funkce použité pro *_unitHINT16* je k dispozici v tištěné příloze **(G)**.

5.7 Umělá neuronová síť

Doplněním algoritmu *_unitHINT16* schopného samostatně vypočíst výstup neuronu s obecným počtem vstupů o propojovací matici teoreticky popsanou v podkapitole *4.8* vznikne kompletní neuronová síť připravená na navázání do systému. Pro představovanou aplikaci bude samozřejmě využito dříve popsaných komponent, cílovým hardwarem je tedy opět **„Modul A**“.

Jelikož se stále částečně jedná o funkci s kritickou dobou výpočtu, je využito jazyka Assembler. Propojovací matice však musí být přístupná i pro - relativně pomalu probíhající – další funkce realizující učení sítě, adaptaci topologie, načítání a zálohování parametrů z/do nonvolatilní paměti a případné další úkony. Dojde tedy i na části psané v jazyce „C“.

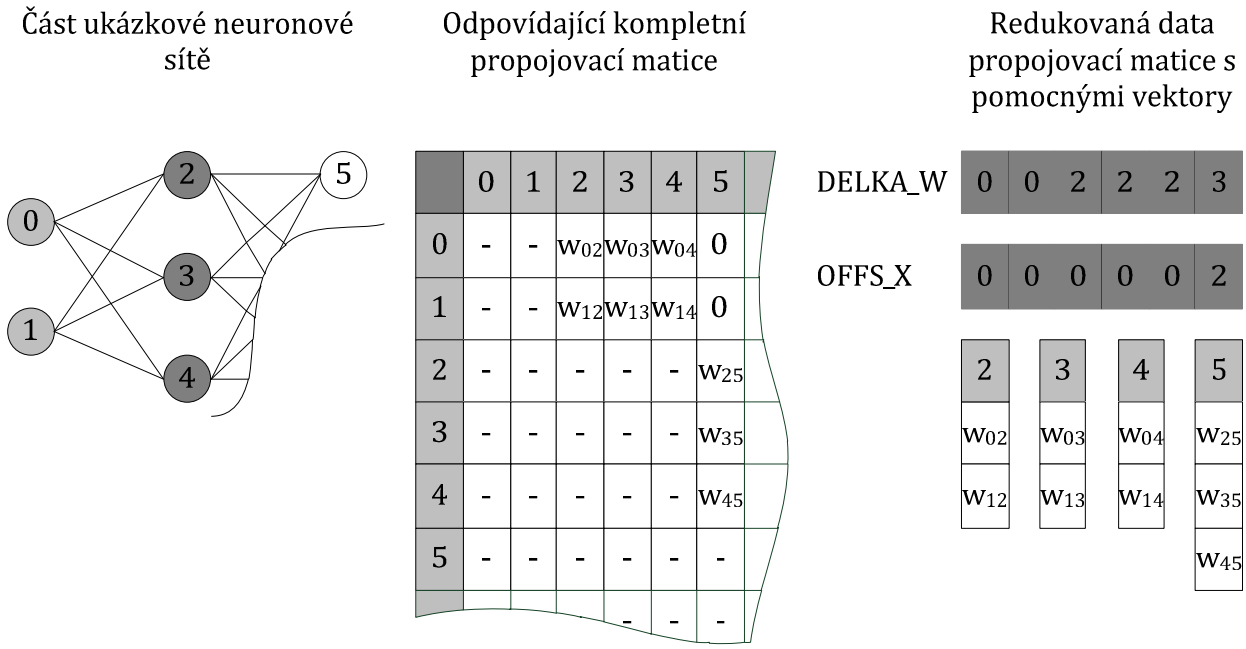
Assemblerovská funkce *_unitHINT16* s podpůrným zavaděčem konstant aktivační funkce do paměti RAM *_aktfuncTAB* jsou doplněny systémem propojení vytvořeným na základě poznatků uvedených v podkapitole *4.8*. Základem je tabulka váhových koeficientů uložená v programové paměti FLASH procesoru. Zejména z exemplárních důvodů je v této ukázkové aplikaci zvolena topologie s dopřednými vazbami hodící se k demonstraci redukce velikosti pole hodnot $\mathbf{w_{ij}}$. Stejně tak je volen komplikovanější přístup k načítání prvků $\mathbf{w_{ij}}$ do datové paměti, tedy způsob „po neuronu“.

O sesazení vzájemně si odpovídajících vah a vstupních hodnot se stará funkce *_feedNet*. Propojovací matice je redukována ve smyslu vypuštění těch prvků, které realizují zpětné vazby a vazby mezi neurony stejné vrstvy. Každý sloupec **j** obsahující prvky vah s indexem **i** informující o propojení j-tého neuronu s i-tým tak může, na rozdíl od neredukované matice, kde je délka konstantní odpovídající počtu neuronů **N**, nabývat různých délek. Situace je navíc komplikována tím, že prvky $\mathbf{w_{ij}}$ nemusejí být povinně nulové pouze pro $\mathbf{i \geq j}$ – což je splnění podmínky nepřítomnosti zpětných vazeb – ale i pro další indexy stanovené podmínkami propojení mezi jednotlivými vrstvami, tedy na základě uživatelsky definovaného a proto obecně předem neznámého vektoru **PS**. Z tohoto důvodu je přidán N-prvkový vektor **DELKA_W** určující počet prvků každého ze zkrácených sloupců. Zaveden je i vektor **OFFS_X** určující počet prvků ubraných na začátku sloupce, zároveň odpovídající offsetu (odtud název) mezi odpovídajícím vstupem **x** a váhou **w**. **OFFS_X** je praktickým důsledkem zákazu spojení neuronů „ob vrstvy“, které jsou jinak přítomny například v hierarchických strukturách dopředných neuronových sítí. Jelikož je v ukázkové aplikaci nepožadujeme, jsou vynechány hlavně pro úsporu výpočetního výkonu u funkce *_unitHINT16*, která by jinak musela vyhodnocovat značné množství vstupů s nulovými váhami.

Speciální význam mají první prvky vektorů **DELKA_W** a **OFFS_X** odpovídající vstupním neuronům. Jejich hodnota v **DELKA_W** je povinně nulová, **OFFS_X** pro ně pak ztrácí smysl. Výpočet těchto jednotek je algoritmem automaticky přeskakován. Očekává se, že ve vektoru výstupů **Y** budou na jejich pozicích před voláním funkce *_feedNet* již zapsány příslušné hodnoty, které jsou vstupem celé sítě a tvoří rozhraní mezi stávající aplikací běžící na embedded systému a zaváděnou neuronovou sítí. V tomto příkladu je přímo nahrávána číselná hodnota z přijímacího bufferu UART řadiče procesoru převedená do formátu absolutní hodnoty se znaménkovým bitem. Přenos vstupních neuronů je tak lineární, bez zesílení. Alternativně je však

možné a zároveň (ač náročnější na výpočetní výkon) technicky správnější volat pro přepočet například v podkapitole *5.5* představovanou funkci *_aktfunc*.

Vztahy mezi vektory **DELKA_W**, **OFFS_X**, propojovací maticí a grafickým znázorněním topologie neuronové sítě tak, jak je výše popsáno, na smyšleném příkladu ilustruje *Obr. 5.7*.1.



Obr. 5.7.1: Ilustrace významu a tvorby hodnot vektorů DELKA_W a OFFS_X v ukázkové aplikaci

Informace v nich obsažené, stejně jako za sebou řazené sloupcové vektory vah a konstanta s celkovým počtem neuronů **CPN**, jsou uloženy v programové paměti FLASH, odkud jsou do registrů RAM načítány vždy pouze hodnoty relevantní pro nadcházející výpočet konkrétní neuronové jednotky. Fyzicky jsou z důvodu maximalizace využití instrukcí TBLRD*, TBLRD*+, TBLRD*- a TBLRD+* realizujících čtení z programové paměti s nebo bez operaci předcházejícího, nebo následujícího inkrementu nebo dekrementu adresy (32), prvky vektorů uloženy v posloupnosti vyjádřené *Tabulka 5.7.1*. Tím dojde k eliminaci potřeby měnit ukazatel do programové paměti o jinou hodnotu, než „1“ a tím pádem i k odstranění značné části pointerové aritmetiky a následné úspoře výpočetního výkonu. Ze stejného důvodu je před redukované sloupcové vektory vah zařazen klidový potenciál daného neuronu **Bias**. Jeho přítomnost není předpokládána u vstupních neuronů (z výše uvedených důvodů úspory výpočetního výkonu v podobě praktického odstranění těchto neuronů je zároveň předpokládáno, že je klidový potenciál do příslušné hodnoty vektoru **Y** již zohledněn).

Název prvku	Zkratka	Počet bitů	Offset adresy FLASH
Celkový počet neuronů	**CPN**	8	0x00
Offset mezi **W** a **X** neuronu **0**	**OFFS_X$_0$**	8	0x01
Délka sloupce **0** prop. matice	**DELKA_W$_0$**	8	0x02
Klidový potenciál neuronu **0**	**Bias$_0$**	16	0x03
Prvky sloupce **0** prop. matice	**W$_{i0}$**	16*DELKA_W$_0$	0x05
Offset mezi **W** a **X** neuronu **j**	**OFFS_X$_j$**	8	$0x01+2\sum_{n=0}^{j-1} DELKA\_W_n$ +**j***4
Délka sloupce **j** prop.matice	**DELKA_W$_j$**	8	$0x02+2\sum_{n=0}^{j-1} DELKA\_W_n$ +**j***4
Klidový potenciál neuronu **j**	**Bias$_j$**	16	$0x03+2\sum_{n=0}^{j-1} DELKA\_W_n$ +**j***4
Prvky sloupce **j** prop.matice	**W$_{ij}$**	16*DELKA_W$_j$	$0x05+2\sum_{n=0}^{j-1} DELKA\_W_n$ +**j***4

Tabulka 5.7.1: Mapa rozmístění prvků vektorů OFFS_X, DELKA_W a sloupců propojovací matice ve FLASH

Volání výpočtu neuronové sítě se děje prostřednictvím assemblerovské funkce *_Net*. Jedinými jejími parametry zadávanými při překladu jsou ukazatel na počátek výše popsané propojovací tabulky **W_TAB**, ukazatel na počátek vyhrazeného prostoru v RAM paměti **RAM_point** a od něho odvozený ukazatel banky RAM paměti **BANK_point**. Jedinou podmínkou v ukázkové aplikaci, pomineme-li samozřejmost, že zvolený prostor RAM ani FLASH nesmí být používán jinými funkcemi, je soustředění veškeré paměti proměnných do jedné banky. Konkrétní nastavení parametrů v popisovaném příkladu je v *Tabulka 5.7.2*. Fyzicky je lze nalézt zapsané v hlavičce zdrojového souboru *net_18F.asm* na přiloženém CD.

Význam	Zkratka	Hodnota
Nejvyšší bajt adresy počátku propojovací tabulky	**W_TAB_U**	0x00
Střední bajt adresy počátku propojovací tabulky	**W_TAB_H**	0x10
Nejnižší bajt adresy počátku propojovací tabulky	**W_TAB_L**	0x00
Ukazatel na vyhrazený prostor v datové paměti	**RAM_point**	0x200
Ukazatel na banku vyhrazené datové paměti	**BANK_point**	0x02

Tabulka 5.7.2: Nastavení adresních prostorů ukázkové aplikace

Pro podporu implementované neuronové sítě, konkrétně pro její adaptaci za běhu, vznikly podpůrné funkce v jazyce „C“ dostupné ve zdrojovém souboru *Net.c* na přiloženém CD. Spolu s *net_18F.asm* sdílejí hlavičkový soubor s deklaracemi *Net.h*. Jejich výčet se stručným popisem činnosti je uveden v *Tabulka 5.7.3*. Vznikly na základě informací získaných z literatury (32) a (36). **FLASH_wr_set** musí být volána na začátku programu v inicializačních procedurách procesoru a případně obnovována, je-li v programu pracováno s uživatelskou pamětí EEPROM.

Dále jsou důležitými zejména funkce **FLASH_vec_save** a **FLASH_byte_change**, které zajišťují prvotní zápis struktury a parametrů sítě do programové paměti a následné adaptace parametrů. Další níže vyjmenované slouží spíše jako podpora pro ladění, případně další uživatelská rozšíření například o vyčítání parametrů pro ladicí adaptační a učící algoritmy. Čtení potřebných parametrů během výpočtu sítě si z důvodu rychlosti zajišťuje sama assemblerovská funkce *_Net*.

Prototyp funkce	Stručný popis činnosti
void FLASH_wr_set(*void*)	Základní inicializace periferie obsluhující zápis do uživatelské EEPROM a programové FLASH mikrokontroléru. Volá se na začátku programu
void FLASH_row_erase(*ulong* **Adr**)	Funkce vymazání 64bajtového bloku programové FLASH. Posledních 6bitů adresy **Adr** je ignorováno, zbytek indexuje požadovaný blok
void FLASH_byte_save(*uchar* **Data**, *ulong* **Adr**)	Funkce ukládá jeden bajt hodnoty **Data** na absolutní adresu **Adr**. Očekává se, že je daná buňka vymazána, tedy má hodnotu 0xFF
void FLASH_vec_save(*uchar* **Count**, *uchar** **Data_point**, *ulong* **Adr**)	Funkce uloží do programové paměti na adresu **Adr** vektor délky **Count**, jehož první prvek je uložen v datové paměti na adrese **Data_point**. Funkce očekává, že je cílová FLASH paměť prázdná
void FLASH_byte_change(*uchar* **Data**, *ulong* **Adr**)	Funkce uloží bajt o hodnotě **Data** do programové paměti na adresu **Adr**, přičemž cílová buňka nemusí být předem smazána. Vzhledem k nutnosti předem formátovat celý blok paměti je nutné zajistit nepoužívání daného bloku během provádění funkce
uchar FLASH_byte_load(*ulong* **Adr**)	Funkce vrací hodnotu bajt vyčteného z programové paměti na adrese **Adr**
void FLASH_vec_load(*uchar* **Count**, *uchar** **Vystup_point**, *ulong* **Adr**)	Funkce do paměti RAM začínající adresou **Vystup_point** přenese **Count** prvkový vektor umístěný v programové paměti na adrese **Adr**

Tabulka 5.7.3: Seznam a stručný popis funkcí v souboru Net.c

Do stávajícího programu je celá síť začleněna pomocí volání funkce *_Net* ve smyčce přerušení. Pro možnosti správné činnosti je nutné zahrnout měření výpočetního času sítě jako korekci odhadu stanoveného metodou popsanou v podkapitole *4.8*. V ukázkové aplikaci je to provedeno již dříve popisovaným způsobem pomocí čítače **TIMER1**. Činnost sítě je řízena

registrem **STATUS_L** mapovaným do uživatelské RAM na adresu **RAM_point+0x23**. Síť je v činnosti pouze v případě, že je jeho LSB = 1. V opačném případě dochází ihned na začátku assemblerovské funkce *_Net* k návratu do původního programu.

Ukázková aplikace je navázána na rozhraní UART, jehož prostřednictvím je možné zapsat do vyhrazeného prostoru FLASH paměti informace o topologii a váhách sítě. Děje se tak prostřednictvím dávkového souboru začínajícího znakem „**S**“ následovaným celkovým počtem bajtů dávky. Jednotlivé bajty sestavené do řetězce systémem uvedeným v *Tabulka 5.7.1* jsou odděleny středníkem nebo mezerou. Ihned po zápisu lze pro kontrolu řetězec zpětně vyčíst příkazem „**R**“. Zadávání vstupních hodnot je řešeno obdobným způsobem. Dávka začíná písmenem „**Y**“ následovaným celkovým počtem nahrávaných vstupů. Jednotlivé hodnoty nejsou ve formátu oddělených bajtů jako v případě dávky „**S**“, ale v podobě celého čísla v intervalu <**-32767**;**32767**>. Klávesa „**C**“ vydává povel k výpisu hodnot jednotlivých uzlů sítě a časové statistiky minimálního a maximálního požadovaného času, který zároveň resetuje, na obrazovku. Příkaz „**T**“ vypisuje pouze maximální a minimální čas výpočtu oddělený středníkem vždy na nový řádek. Resetuje i paměti těchto časů. Je určen pro generování dat souborů s příponou „csv“ pro tabulkové editory. Spouštění a zastavování sítě je pak ovládáno klávesou „**N**“. Kompletní seznam popsaných příkazů je v *Tabulka 5.7.4*. Jako druhá strana komunikační linky je opět doporučeno PC s terminálem *Realterm* (35).

Znak	Význam
„S“	Návěstí bloku propojovacích a organizačních dat neuronové sítě ve formátu dle *Tabulka 5.7.1*. Znak „S“ je následován celkovým počtem bajtů v dekadickém tvaru vzájemně oddělených středníkem, které budou přímo zapsány do FLASH paměti na adresu **W_TAB**. Probíhá-li výpočet neuronové sítě, instrukce jej pozastaví
„R“	Instrukce pro výpis hodnot z programové paměti na adrese **W_TAB**. Slouží pro kontrolu bezchybného průběhu instrukce „S“ a musí být použita bezprostředně po jejím volání. Vypíše na terminál takový počet bajtů, který byl deklarován za „S“
„C“	Instrukce výpisu aktuálních hodnot vektoru výstupů jednotlivých neuronů na obrazovku
„Y“	Návěstí pro zaslání vektoru vstupních hodnot sítě prostřednictvím UART řadiče. Po „Y“ následuje cifra určující počet přenášených hodnot. Ty jsou ve formátu kompatibilním s HINT16 a vzájemně oddělené středníkem Nahrávány jsou do vektoru výstupů neuronů. Všechny hodnoty, které jsou

	nahrávány na pozice neuronů s nenulovým počtem vstupů, jsou ignorovány
„T“	Příkaz pro výpis aktuálního maximálního a minimálního času výpočtu sítě v instrukčních cyklech. Výstup je optimalizován pro zachytávání datového souboru „csv“ pro tabulkový editor. Instrukce zároveň resetuje paměti maximálního a minimálního výpočetního času
„N“	Ovládací klávesa/příkaz pro zastavení a spuštění výpočtu sítě. Vstupních hodnot, ani aktuálních hodnot pamětí výpočetních časů se nedotýká
Příklad zadání dávkového souboru se strukturou sítě – příkaz „S“	
*S61;8;***0;0;**0;0;**0;2;241;256;135;0;220;128;**0;2;17;256;97;0;47;128;**0;2;125;0;185;0;36;0;**0;2;63;256;69;0;211;128;**2;4;22;0;218;128;87;0;159;0;78;128;**2;4;213;0;173;128;202;128;255;0;225;0;	
Jednotlivé neurony jsou v příkladu rozlišeny tučným/standardním písmem. Kurzívou je psána hlavička. Příkaz obsahuje 61 bajtů informací o neuronové síti s 8mi prvky. První dva mají offsety vstupů a vah a počty vstupů rovny nule, další informace nesmí být poskytnuty. Jedná se o vstupní neurony. Následující (v pořadí třetí) neuron náležící skryté vrstvě má offset X<->W = 0 (výstup neuronu s indexem 0 $\mathbf{Y_0}$ vede na jeho váhu $\mathbf{W_0}$) a celkem dva vstupy. Jeho **Bias** je 256*256+241 = FFF1 = -15 (znaménkový *int16* ve dvojkovém doplňku, little endian) první váha $\mathbf{W_0}$ = 0*256+135 = 0087 = 135 (znaménkový formát *HINT16* – absolutní hodnota a znaménkový bit na pozici $\mathbf{b_{15}}$, little endian), druhá váha $\mathbf{W_1}$=128*256+220 = 80DC = -220 (*HINT16*, tentokrát záporná hodnota). Obdobně je tomu i u následujících tří neuronů. Neurony s indexy 6 a 7 mají offset X<->W = 2, jejich váhy $\mathbf{W_0}$ jsou proto navázány na výstup neuronu s indexem 2 $\mathbf{Y_2}$. Jedná se tedy o neurony následující, zde výstupní vrstvy. Neurony mají 4 vstupy.	
Příklad zadání vstupních hodnot sítě – příkaz „Y“	
Y2;-3560;**120;**	
Příkaz obsahuje dvě hodnoty zapsané v běžném dekadickém tvaru a zakončené středníkem. Po načtení do mikrokontroléru budou převedeny na formát *HINT16* a nahrány do vektoru výstupních hodnot neuronů na pozice $\mathbf{Y_0}$ a $\mathbf{Y_1}$, kde suplují výstupní hodnoty vstupních neuronů.	

Tabulka 5.7.4: Tabulka příkazů ukázkové aplikace výpočtu neuronové sítě s příkladem a vysvětlením

Pro generování „S-příkazů“ na základě vah a počtu neuronů jednotlivých vrstev dopředné sítě je na přiloženém CD k dispozici soubor *Net.xlsm* s popisem a komentovanými makry pro MS Excel 2010, kde makro *DataGen* generuje řetězec „S-příkazu“ a makro *DataBack* jej dokáže převést zpět, vygenerovat z něj kompletní propojovací matici rozměru **NxN**, kde **N** je počet neuronů sítě a pro zadané vstupní hodnoty zároveň vypočítá výstup sítě, přičemž

aktivační funkce je i v MS Excelu aproximována za použití celočíselné aritmetiky a splajnů s koeficienty používanými v jednotce „**Modul A**“.

Pro ověření správné činnosti funkce *_Net*, přidružených algoritmů a celé popsané metody pro zavádění umělé neuronové sítě do embedded systému s nízkým výpočetním výkonem byl na cílovém zařízení sestaven firmware s programovou smyčkou konstantní doby trvání **1ms** s dostatečným využitelným zbytkovým časem **ΔT**, generovanou periodickým přerušením čítače **TIMER2** (připomenutí informací v podkapitole *4.9* a v katalogovém listu (32)). Takt procesorového jádra je **12MIPS**. Pomocí zmiňovaného souboru s makry pro MS Excel 2010 byly vygenerovány čtyři topologie dopředných neuronových sítí s náhodnými klidovými potenciály a váhami v intervalu <**-1000**;**1000**> a převedeny do dávkového souboru s podobou vyžadovanou popsanou instrukcí „**S**“ čtenou v nekonečné smyčce funkce *main*, tedy ve zbytkovém čase. Jako vstupní hodnoty sítí posloužily dávkové soubory *FeedNet2Y.csv* a *FeedNet3Y.csv* s příkazy „**Y**“ po 500 řádcích s odpovídajícím počtem prvků – rovněž v rozmezí <**-1000**;**1000**> – odesílané *Realterm* terminálem, který zároveň zachytával do souboru výpočetní výsledky pro kontrolu matematické správnosti (prokázána pro 100% výpočtů na základě srovnání s hodnotami generovanými algoritmem v MS Excelu) a dobu trvání. Výsledkové listy s grafickým znázorněním sítě, výpisem dávkového „S-řetězce“ a histogramem výpočetních časů v instrukčních cyklech jsou v přílohách **(H)** až **(K)**. Veškerá data jsou rovněž k dispozici na přiloženém CD včetně zdrojových dávkových souborů i zdrojových souborů programu.

6. Závěr

6.1 Shrnutí dosažených výsledků

V souladu s cíli vytyčenými v kapitole *2* a s ohledem na stav problematiky implementace neurnových sítí na nízkovýkonových embedded systémech v době vzniku této práce byly v kapitole *4* představeny teoretické poznatky popisující metodiku tvorby a zavádění algoritmů výpočtu umělých neuronových sítí do „low-end" mikrokontrolérů. Mezi základní požadavky, které byly sledovány a splněny, patří jednak podmínka adaptability topologie a vah zavedené sítě bez nutnosti externího zásahu do programové paměti mikrokontroléru a jednak podmínka koexistence stávajícího firmwaru (splňujícího jisté požadavky v textu uvedené) s přidanou neuronovou sítí. Hlavními výsledky, kterých bylo v teoretické kapitole dosaženo, jsou:

1) Detailní rozbor jednotlivých základních komponent umělé neuronové sítě s ohledem na teoretické vysvětlení těch aspektů, které mají podstatný dopad na praktickou realizaci
2) Definování vlastního datového typu **VDT24** s plovoucí řádovou čárkou včetně operací sčítání, násobení a normalizace, optimalizovaného pro výpočty s vysokým dynamickým rozsahem na celočíselných systémech se slabou hardwarovou podporou
3) Odvození hybridního celočíselného 16bitového algoritmu **HINT16** výpočtu vnitřního potenciálu neuronu s dvojí reprezentací záporných hodnot s ohledem na optimalizaci výkonových požadavků včetně popisu a odvození řešení aproximace tangenciální aktivační funkce
4) Návodný popis možností řešení za běhu konfigurovatelné neuronové sítě včetně kategorizace cílových embedded systémů a definice požadavků, které musejí pro úspěšnou implementaci splňovat

Kapitola *5* fakticky kopíruje teoretický popis metody a prostřednictvím ukázkové aplikace na jednotce „**Modul A**" s jednočipovým mikropočítačem PIC18F46K80 společnosti Microchip názorně demonstruje proveditelnost prezentované metody. Nesleduje navíc pouze přímou linii směřující k jednomu konkrétnímu zdrojovému kódu, ale v podobě dále nerozvíjených odklonů nabízí srovnání s alternativami, například výpočet jádra neuronu metodou **VDT24** v porovnání s **HINT16**. Tím plní druhou – praktickou část vytyčených cílů disertační práce. Mezi hlavní přínosy této kapitoly patří:

1) Představení praktických řešení některých problémů způsobených absencí speciálních podpůrných instrukcí zvoleného procesoru v podobě aplikačních poznámek a zdrojových kódů v jazyce Assembler. Konkrétně mezi ně patří: Vícebitové celočíselné znaménkové a neznaménkové násobení, násobení čísel typu **VDT24**, rychlá analytická aproximace funkce pomocí splajnů…
2) Kompletní soubor odladěných okomentovaných zdrojových kódů navrhované metody v jazyce Assembler a „C“ pro zvolený typ mikrokontroléru od dílčích funkcí přes nižší celky až po celou neuronovou síť zavedenou do programové smyčky embedded systému a schopnou adaptace vah i topologie (včetně počtu neuronů a jejich přesunu mezi vrstvami) „za běhu“ bez změny firmwaru
3) Kompletní výsledky v podobě měření časových náročností jednotlivých operací použitelných pro srovnání jednotlivých přístupů mezi sebou včetně použitých zdrojových dat pro statistické výpočty
4) Podpůrné soubory generování dat a zpracování výsledků pro MS Excel

6.2 Směřování dalšího vývoje

Další vývoj je proveditelný ve dvou rovinách. První je rozšiřování základny představené metody tak, jak je popsána. V práci není věnován prostor jednotlivým kombinacím diskutovaných přístupů a širší škále použitého hardwaru. Nabízí se tak možnost implementovat například jádro **HINT16** na jednoduchý a dostupný 16bitový signálový procesor dsPIC33 (37) vybavený MAC instrukcí, vytvořit neuronová jádra **VDT24** a celočíselné znaménkové ve formátu dvojkového doplňku a srovnávat jejich výkonnost, nebo optimalizovat stávající funkce. Rovněž by bylo zajímavé podrobit algoritmy výkonnostním testům a testům přesnosti na reálné neuronové síti se vstupy například z A/D převodníku, nikoli z umělého dávkového souboru.

Druhým směrem následujících kroků je pokračování ve vývoji metodiky. Práce končí předložením algoritmu výpočtu neuronové sítě s širokými možnostmi adaptace. Nejsou však řešeny algoritmy těchto adaptací, ať se již jedná o změnu topologie, nebo adaptaci vah. Není fyzicky řešena implementace automatických ochran proti přečerpání přidělených prostředků (například na základě v podkapitole *4.8* popsaných vektorů **KV** a **PS**, nebo jim podobným). Zajímavou možností je také automatická distribuce či expanze neuronové sítě do okolních embedded systémů stejného typu propojených komunikační sběrnicí. Metoda řízení všech dynamik sítě prostřednictvím RAM a FLASH paměťových bufferů s relativním adresováním tak, jak je v práci představena a s dodanými funkcemi čtení a zápisu do programové paměti je vhodná pro takovéto akce (po provedení příslušných úvah a doplnění o vhodné algoritmy).

7. Seznam zkratek

A/D	Analog to Digital	Používáno ve spojení A/D převodník - Analogově digitální převodník
ANN	Artificial Neural Network	Umělá neuronová síť
CSV	Comma Separated Variable	Formát datového souboru pro PC s prvky oddělenými zvoleným znakem
DMA	Direct Memory Access	Přímý přístup do paměti
EEPROM	Electrically Erasable Programmable Read-Only Memory	Elektricky smazatelná a programovatelná paměť
FLASH	Fast Low-Latency Access with Seamless Handoff	Druh elektronické nonvolatilní paměti
FPGA	Field Programmable Gate Array	Programovatelný logický obvod na bázi vyhledávacích tabulek
HW	Hardware	Označení pro fyzickou část zařízení
ICSP	In-circuit Serial Programming	Firemní standard společnosti Microchip pro programování součástek již obvodově zapojených
IEEE	Institute of Electrical and Electronics Engineers	Světová elektrotechnická organizace
LSB	Least significant bit	Nejméně význačný bit. Označuje nejnižší pozici v registru
LUT	Look Up Table	Paměťová tabulka s výsledky kombinační funkce adresových vstupů, používá se u obvodů FPGA
MAC	Multiply And Accumulate	Instrukce násobení s následnou akumulací do výstupního registru typicky přítomná na signálových procesorech
MCU	Microcontroller Unit	Elektronická součástka – mikrokontrolér, jednočipový mikropočítač, mikroprocesor

MIPS	Mega Instructions Per Second	Jednotka výpočetní rychlosti - milióny provedených instrukcí za sekundu
MSB	Most Significant Bit	Nejvýznačnější bit. Označuje nejvyšší pozici v registru
PC	Personal Computer	Běžný osobní počítač
PFP	Pseudo Floating Point	Speciální typ proměnné s plovoucí řádovou čárkou používaný v práci (9)
PWM	Pulse-Width Modulation	Pulzně-šířková modulace
RAM	Random Access Memory	Paměť s libovolným přístupem
RTOS	Real-Time Operating System	Operační systém s možnostmi časování v reálném čase
SW	Software	Programové vybavení
VDT24	Vlastní Datový Typ 24	Typ 24bitové proměnné s plovoucí řádovou čárkou definovaný a použitý v této práci

Double	Datový typ programovacího jazyka „C“ s plovoucí řádovou čárkou a dvojitou přesností o délce 64bitů definovaný standardem IEEE
Exp	V práci používaná zkratka pro odkaz na exponent čísla s plovoucí řádovou čárkou
Float	Datový typ programovacího jazyka „C“ s plovoucí řádovou čárkou a základní přesností o délce 32bitů definovaný standardem IEEE
H-byte	Označení používané pro odkaz na vyšší, nebo nejvyšší byte proměnné složené z více než jednoho bytu
Char	Typ celočíselné datové proměnné jazyka „C“ s délkou 8bitů
Int	Zkratka pro *integer*

Integer	Typ celočíselné datové proměnné jazyka „C“ o velikosti 16, nebo 32bitů, přičemž v této práci je vždy považován za 16bitový
L-byte	Označení používané pro odkaz na nižší, nebo nejnižší byte proměnné složené z více než jednoho bytu
Long	Typ celočíselné proměnné jazyka „C“ o velikosti 32bitů
Low-cost	Označení pro součástky a systémy spadající u daného výrobce do skupiny produktů s nejnižšími pořizovacími náklady
Low-end	Produktů spadající svými parametry do nejnižší kategorie
Mant	V práci používaná zkratka pro odkaz na mantisu čísla s plovoucí řádovou čárkou
Mid-range	Označení pro produkty spadající svými parametry do skupiny ležící mezi *Low-end* a špičkovými
On-line	V této práci označení pro standardně neprováděné činnosti, jako jsou adaptace, probíhající na systému bez pozastavení výkonu jeho běžné funkce
Signed	V této práci označení pro znaménkovou verzi datového typu
Unsigned	V této práci označení pro neznaménkovou verzi datového typu

8. Seznam obrázků

9. Seznam tabulek

10. Seznam použité literatury

(1). **RNDr. PaedDr. Eva Volná, PhD.** *Neuronové sítě 1.* Ostrava : Ostravská univerzita v Ostravě, 2008.

(2). **Ing. Václav Jirsík, CSc., Ing. Petr Hráček.** *Neuronové sítě, expertní systémy a rozpoznávání řeči.* Brno : Fakulta elektrotechniky a komunikačních technologií VUT v Brně, 2003.

(3). **Ben Krose, Patrick van der Smagt.** *An Introduction to Neural Networks*. Amsterdam : The University of Amsterdam, 1996.

(4). **Yu Hen Hu, Jenq-Neng Hwang.** *Handbook of neural network signal processing*. : CRC Press LLC, 2002.

(5). **Tebelskis, Joe.** *Speech Recognition using Neural Networks*. Disertační práce. Pittsburgh, Pennsylvania, 1995.

(6). **McNelis, Paul D.** *Neural networks in finance*. Elsevier Academic Press, 2005.

(7). **Rezaul Begg a. kol.** *Neural Networks in Healthcare*. Idea Group Publishing, 2006.

(8). **Cotton, Nicolas Jay.** *A Neural Network Implementation on Embedded Systems*. Disertační práce. Auburn, Alabama : Auburn University, 2010.

(9). **Microchip Technology Inc.** *Microchip Technology Inc.* Microchip Technology Inc. website. [Online] 2011. [Citace: 18. září 2011.] http://www.microchip.com/.

(10). **Nicholas J. Cotton, Bogdan M. Wilamowski.** *Compensation of Nonlinearities Using Neural Networks Implemented on Inexpensive Microcontrollers*. IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS. Březen 2011, Sv. 58, 3, stránky 733-740.

(11). **Auburn University, Alabama.** *Bogdan M. Wilamowski*. Univerzitní webová stránka. [Online] [Citace: 23. říjen 2013.] http://www.eng.auburn.edu/~wilambm/.

(12). **Wilamowski, Bogdan M.** *List of Publications*. Univerzitní webová stránka osobnosti. [Online] [Citace: 23. říjen 2013.] http://www.eng.auburn.edu/users/wilambm/pap/index.htm.

(13). **Ryerson University.** *Ryerson University website*. Univerzitní webová stránka. [Online] [Citace: 8. prosinec 2013.] http://www.ryerson.ca.

(14). **Behan, Thomas.** *Investigation of integer neural networks for low cost embeddes systems*. Diplomová práce. Ontario, Canada : Ryerson University, 2007.

(15). **Behan, Thomas a kol.** *Accelerating Integer Neural Networks on low cost DSPs*. Stať ve sborníku. World Academy of Science, Engineering and Technology, 2008. Vol. 2.

(16). **Thomas Behan, Ziayi Liao and Lian Zhao.** *Integer Neural Networks On Embedded Systems*, . část knižní publikace Recent Advances in Technologies. InTech, 2009. ISBN: 978-953-307-017-9.

(17). **University of Zaragoza.** *University of Zaragoza*. Univerzitní webová stránka. [Online] [Citace: 19. leden 2014.] http://www.unizar.es/EN.

(18). **Nicolás J. Medrano-Marqués, B. Martín-del-Brío a kol.** *A Thermocouple Model Base on Neural Networks*. stať ve sborníku. International Work-Conference on Artificial and Natural Neural Networks IWANN, 2001.

(19). —. Implementing Neural Networks onto Standard Low-Cost Microcontrollers for Sensor Signal Processing. *stať ve sborníku.* 2001.

(20). —. Sensor linearization with neural networks. *stať ve sborníku .* místo neznámé : IEEE Transaction on Industrial Electronics, 2001. Sv. Vol. 48.

(21). **IEEE.** *IEEE 754: Standard for Binary Floating-Point Arithmetic*. webová stránka IEEE. [Online] 2008. http://grouper.ieee.org/groups/754/.

(22). **Jean-Michel Muller a kol.** *Handbook of Floating-Point Arithmetic.* 2010. ISBN 978-0817647049.

(23). **Thomas Behan a kol.** *Accelerating Integer Neural Networks On Low Cost DSPs*. World Academy of Science, Engineering and Technology. 2008, Sv. Vol. 2, 27.12.

(24). **Fajmon, B., Ružicková, I.** *Matematika 3*. Skriptum FEKT VUT v elektronické forme. Brno 2003. identifikační číslo v informacním systému VUT: MAT103.

(25). **Jiří Šíma, Roman Neruda.** *Teoretické otázky neuronových sítí*. Matfyzpress, 1996. ISBN:8085863189.

(26). **Juan R. Rabunal, Julian Dorado.** *Artificial neural networks in real-life applications*. Idea Group Publishing, 2006.

(27). **Analog Devices, Inc.** *Analog Devices, Inc. website.* [Online] 2011. [Citace: 20. září 2011.] http://www.analog.com/en/index.html.

(28). **Texas Instruments Inc.** *Texas Instruments Inc. website.* [Online] 2011. [Citace: 21. Září 2011.] www.ti.com.

(29). **CadSoft.** *CadSoft website.* [Online] 2011. [Citace: 2. únor 2014.] http://www.cadsoftusa.com/.

(30). **ITEAD Intelligent Systems Co.Ltd.** *ITEAD Intelligent Systems Co.Ltd website.* [Online] 2012. [Citace: 7. prosinec 2013.] http://blog.iteadstudio.com/.

(31). **Microchip Technology Inc.** *PIC18F46K80 family datasheet.* Microchip Technology Inc. website. [Online] 2010. [Citace: 12. srpen 2013.] http://ww1.microchip.com/downloads/en/DeviceDoc/39977f.pdf.

(32). **V. P. Plagianakos, M. N. Vrahatis.** *Neural network training with constrained integer weights*. Statě ve sborníku. Washington, USA : IEEE, 1999.

(33). **Draghici, S.** *Some new results on the capabilities of integer weights neural networks in classification problems*. Stať ve sborníku. Washington : IEEE, 1999. ISSN 1098-7576.

(34). **Realterm.** *Realterm: Serial Terminal.* Realterm website. [Online] http://realterm.sourceforge.net/.

(35). **Microchip Technology Inc.** *Implementing Table Read and Table Write*. Aplikační poznámka. 2002. AN548.

(36). —. *dsPIC33EP64MC506 datasheet.* Microchip Technology Inc. website. [Online] 2013. [Citace: 16. leden 2014.] http://www.microchip.com/.

(37). **Atmel Corporation.** *Atmel Corporation*. Atmel Corporation website. [Online] 2011. [Citace: 20. září 2011.] http://www.atmel.com/default.aspx.

(38). **Freescale Semiconductor, Inc.** *Freescale Semiconductor, Inc.website.* [Online] 2011. [Citace: 19. září 2011.] http://www.freescale.com/.

(39). **Microchip Technology Inc.** *dsPIC30F/33F Programmer's Reference Manual.* Microchip Technology Inc. website. [Online] 1. Únor 2008.

(40). **Goldberg, David.** *What Every Computer Scientist Should Know About Floating-Point Arithmetic*. Computing Surveys. 1991.

11. Seznam autorových publikovaných prací

(A1). **Bartl, Matouš.** *Konstrukce aktivního PFC filtru s obvodem UC2854A.* Stať ve sborníku. Elektrotechnika a informatika 2008. Část 2., Elektronika, s. 1-4: 9. ročník konference doktorských prací, zámek Nečtiny, 6.-7. listopadu 2008. Západočeská univerzita v Plzni. ISBN 978-80-7043-701-8.

(A2). **Bartl, Matouš.** *Diskrétní Fourierova transformace v praxi digitálních signálových procesorů*. Stať ve sborníku. Elektrotechnika a informatika 2009. Část 2., Elektronika: 10. ročník konference doktorských prací, zámek Nečtiny, 4.-5. listopadu 2009. Západočeská univerzita v Plzni. ISBN 978-80-7043-809-1.

(A3). **Bartl, Matouš.** *Odvození rychlé fourierovy transformace vhodné pro praktické použití v procesorech*. Stať ve sborníku. Elektrotechnika a informatika 2010. Část.2, Elektronika, s. 1-4: 11. ročník konference doktorských prací, zámek Nečtiny, listopad 2010. Západočeská univerzita v Plzni. ISBN 978-80-7043-914-2.

(A4). **Bartl, Matouš.** *Řešení samostatné jednotky pro implementaci FFT*. Stať ve sborníku. Elektrotechnika a informatika 2011. Část 2., Elektronika, s. 1-4: 12. ročník konference doktorských prací, zámek Nečtiny, 3.-2. listopadu 2011. Západočeská univerzita v Plzni. ISBN 978-80-261-0015-7.

(A5). **Kysela, Adam. Bartl, Matouš.** *Elektromagnetické emise koróny*. Stať ve sborníku. Electric Power Engineering 2012. Proceedings of the 13th International Scientific Conference Electric Power Engineering 2012: Brno 23.-25. května 2012. VUT Brno. ISBN 978-80-214-4514-7.

(A6). **Bartl, Matouš. Kysela, Adam.** *Rezonanční měnič s širokým rozsahem vstupního napětí*. Stať ve sborníku. Electric Power Engineering 2012. Proceedings of the 13th International Scientific Conference Electric Power Engineering 2012: Brno 23.-25. května 2012. VUT Brno. ISBN 978-80-214-4514-7.

(A7). **Bartl, Matouš.** *Číslicový regulátor s automatickou optimalizací konstant*. Stať ve sborníku. Elektrotechnika a informatika 2012. Část 2., Elektronika s. 1-4: 13. ročník konference doktorských prací, zámek Nečtiny, 7.-11. listopadu 2012. Západočeská univerzita v Plzni. ISBN 978-80-261-0119-2.

(A8). **Bartl, Matouš.** *Procesorová jednotka pro zpracování audiosignálů Audio 1v1*. Funkční vzorek. Plzeň: Západočeská univerzita v Plzni. 2014.

(A9). **Kysela, Adam. Bartl, Matouš.** *Rezonanční měnič se vstupním stabilizátorem*. Stať ve sborníku. Electric Power Engineering 2013. Proceedings of the 14th International Scientific Conference Electric Power Engineering 2013: Kouty nad Desnou 28.-30. května 2013. VŠB Ostrava. ISBN 978-80-248-2988-3.

(A10). **Švejda, Martin. Bartl, Matouš.** *Vliv regulačního algoritmu na dynamické parametry procesorem řízeného spínaného zdroje a porovnání výpočetních rychlostí regulační smyčky*. Stať ve sborníku. Electric Power Engineering 2013. Proceedings of the 14th International Scientific Conference Electric Power Engineering 2013: Kouty nad Desnou 28.-30. května 2013. VŠB Ostrava. ISBN 978-80-248-2988-3.

(A11). **Bartl, Matouš. Kysela, Adam. Štětka, Petr.** *Řídící algoritmus rezonančního měniče s kompenzací tolerancí rezonančního tanku*. Stať ve sborníku. Electric Power Engineering 2013. Proceedings of the 14th International Scientific Conference Electric Power Engineering 2013: Kouty nad Desnou 28.-30. května 2013. VŠB Ostrava. ISBN 978-80-248-2988-3.

(A12). **Štětka, Petr. Bartl, Matouš. Kysela, Adam.** *Konstrukce vstupního výkonového polovodičového spínacího prvku trakčního měniče*. Stať ve sborníku. Electric Power Engineering 2013. Proceedings of the 14th International Scientific Conference Electric Power Engineering 2013: Kouty nad Desnou 28.-30. května 2013. VŠB Ostrava. ISBN 978-80-248-2988-3.

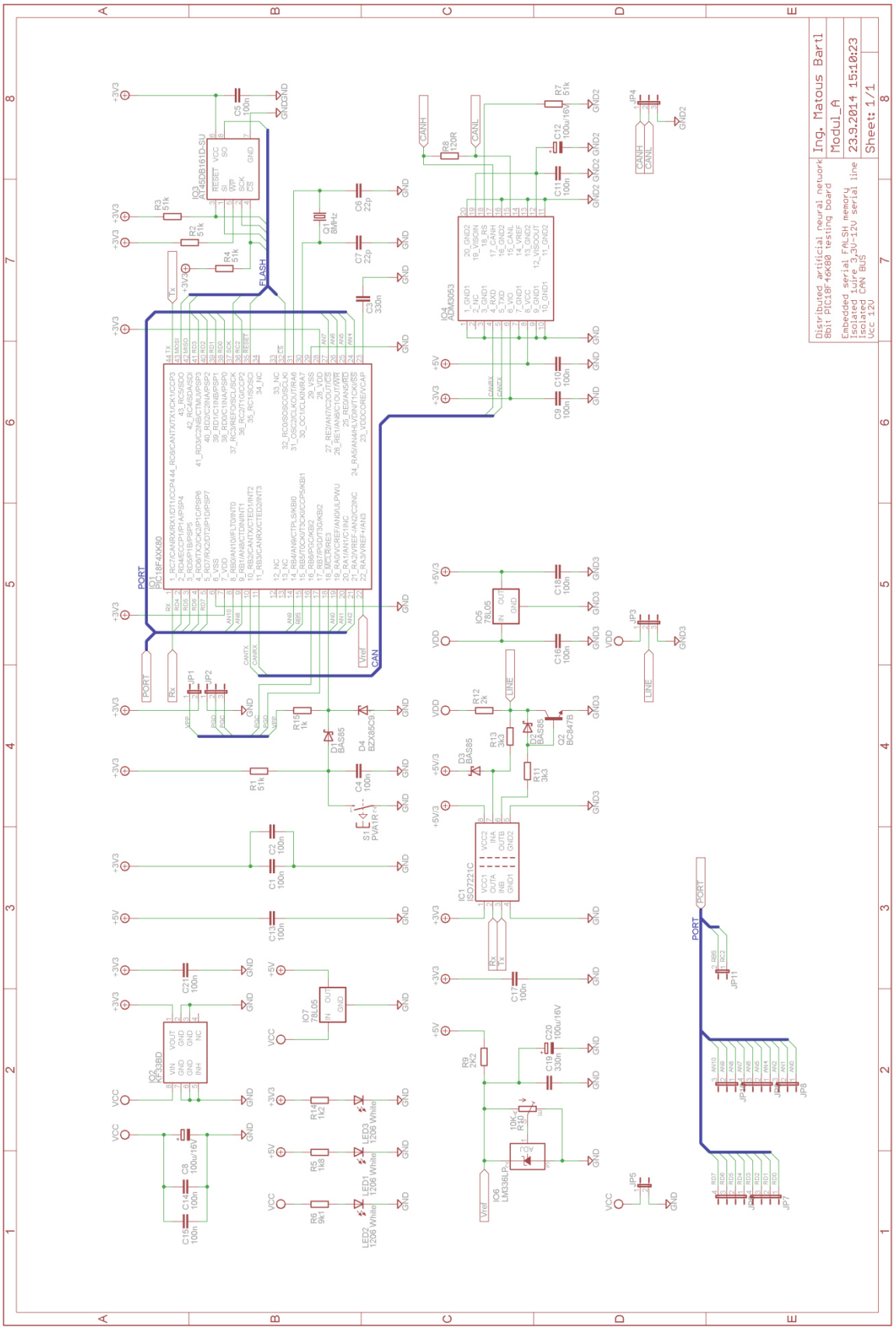
(A13). **Bartl, Matouš.** *Kompenzace výrobních tolerancí kritických součástí rezonančního měniče řídícím algoritmem*. Stať ve sborníku. Elektrotechnika a informatika 2013. Část 2., Elektronika s. 1-4: 14. ročník konference doktorských prací, zámek Nečtiny, 6.-7. listopadu 2013. Západočeská univerzita v Plzni. ISBN 978-80-261-0232-8.

(A14). **Čermák, Karel. Bartl, Matouš.** *Decentralized Battery Management Systém with the communication over power line*. Stať ve sborníku. Electric Power Engineering 2014. Proceedings of the 15th International Scientific Conference Electric Power Engineering 2014. S. 599-603: Brno 12.-14. května 2014. VUT Brno. ISBN 978-1-4799-3806-3.

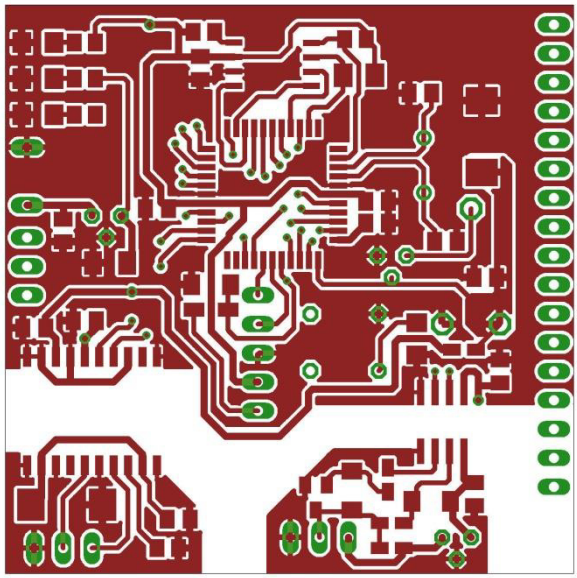
(A15). **Bartl, Matouš. Čermák, Karel.** *Robust digital regulator for railway battery charging station*. Stať ve sborníku. Electric Power Engineering 2014. Proceedings of the 15th International Scientific Conference Electric Power Engineering 2014. S. 605-608: Brno 12.-14. května 2014. VUT Brno. ISBN 978-1-4799-3806-3.

12. Seznam příloh

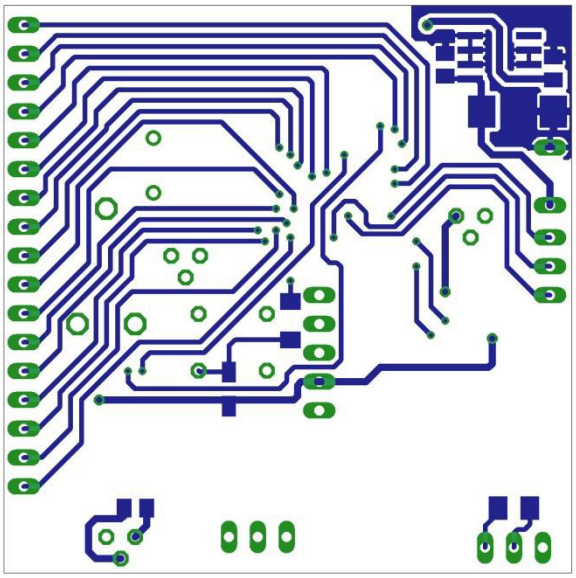
Příloha A


Distributed artificial neural network
8bit PIC18F46K80 testing board
Embedded serial FALSH memory
Isolated LVCMOS 3.3V-12V serial line
Isolated CAN BUS
Vcc 12V
Ing. Matous Bartl
Modul_A
23.9.2014 15:10:23
Sheet: 1/1

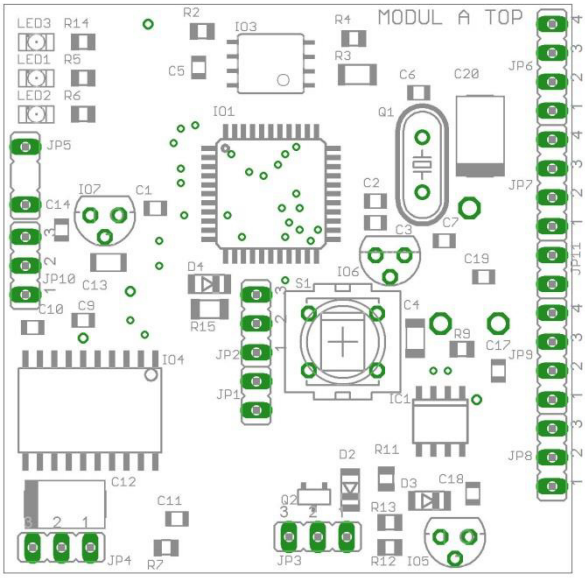
Příloha B



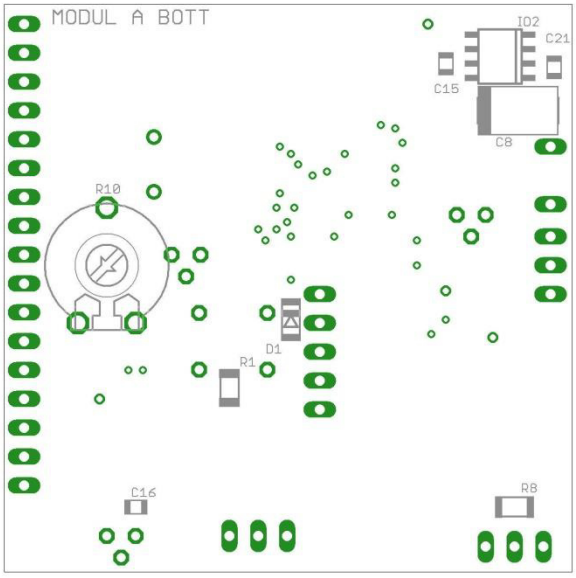
Motiv plošných spojů - horní vrstva v celkovém zvětšení 150%



Motiv plošných spojů - spodní vrstva v celkovém zvětšení 150%, zrcadlená

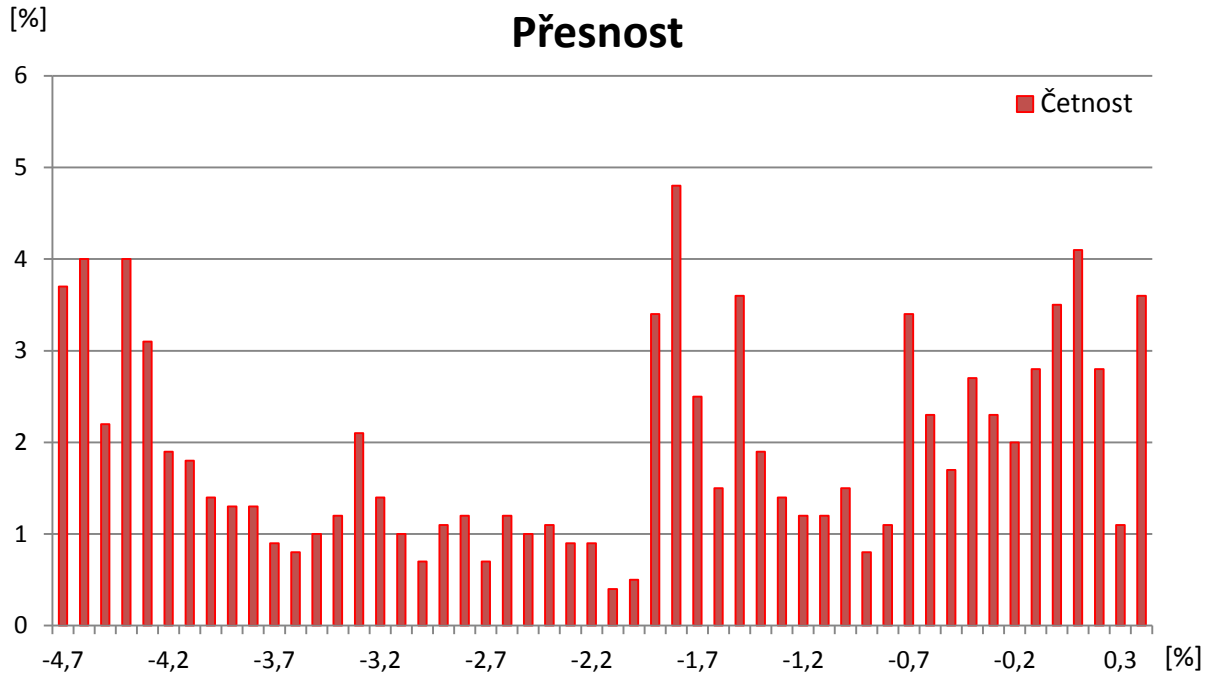


Osazovací plánek - horní vrstva v celkovém zvětšení 150%

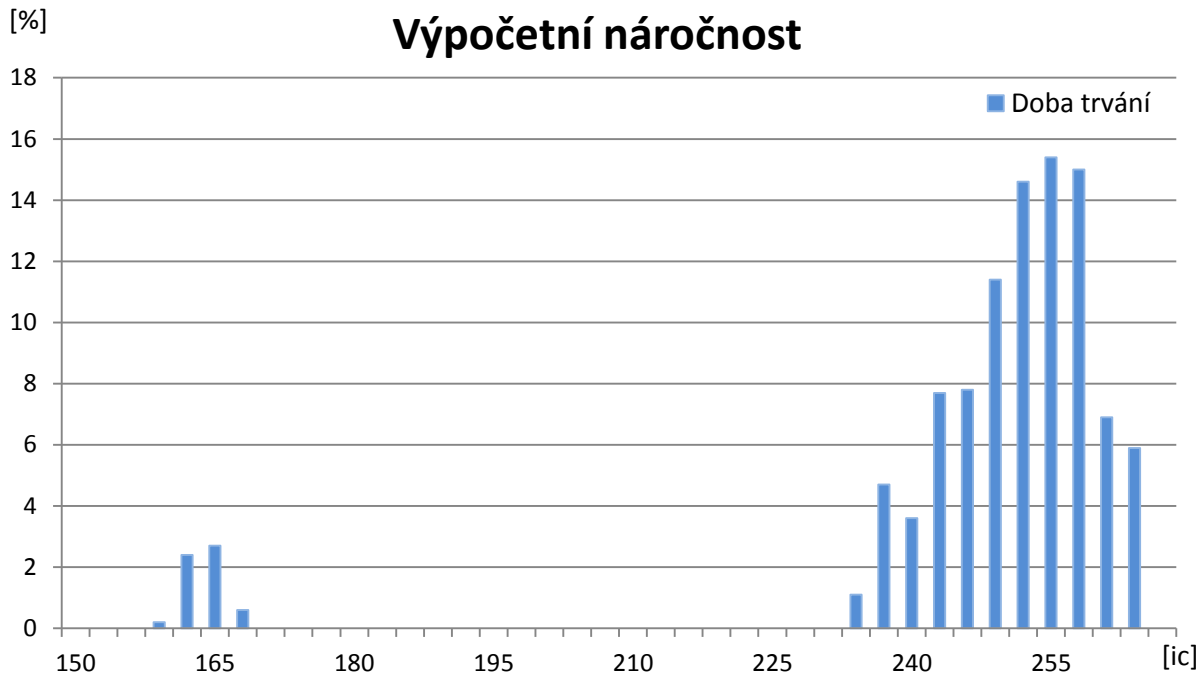


Osazovací plánek – spodní vrstva v celkovém zvětšení 150%

Příloha C

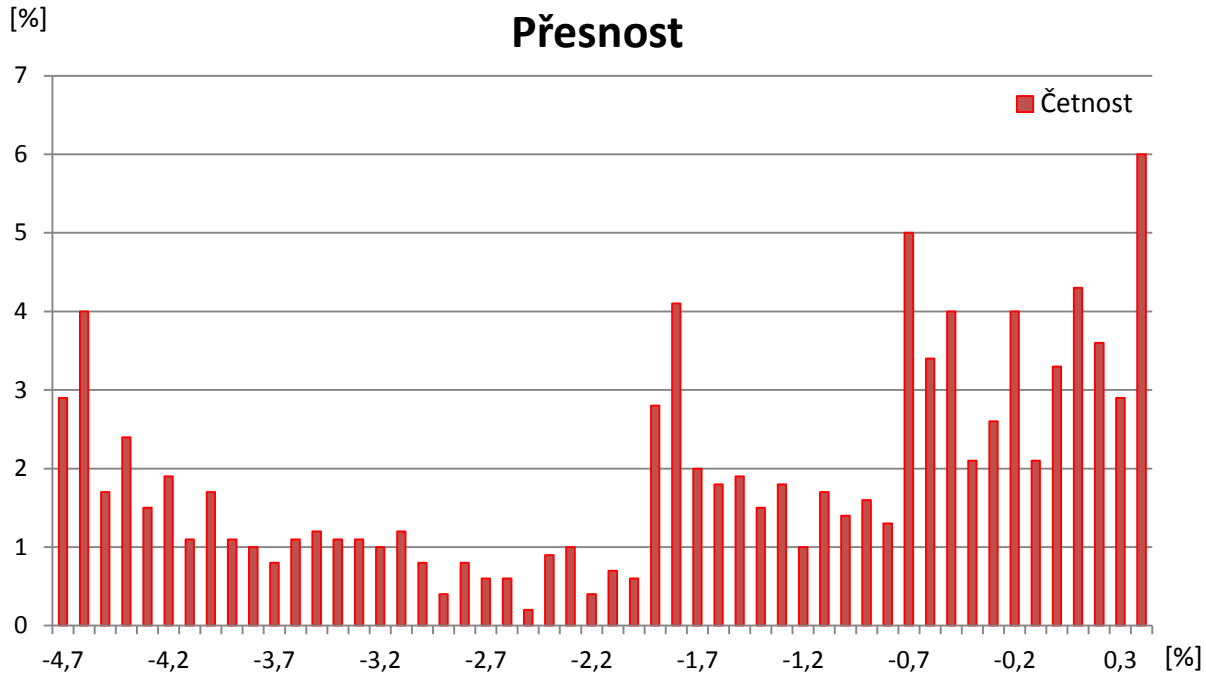


Procentuální četnost výpočtů s danou přesností pro dvojvstupý neuron

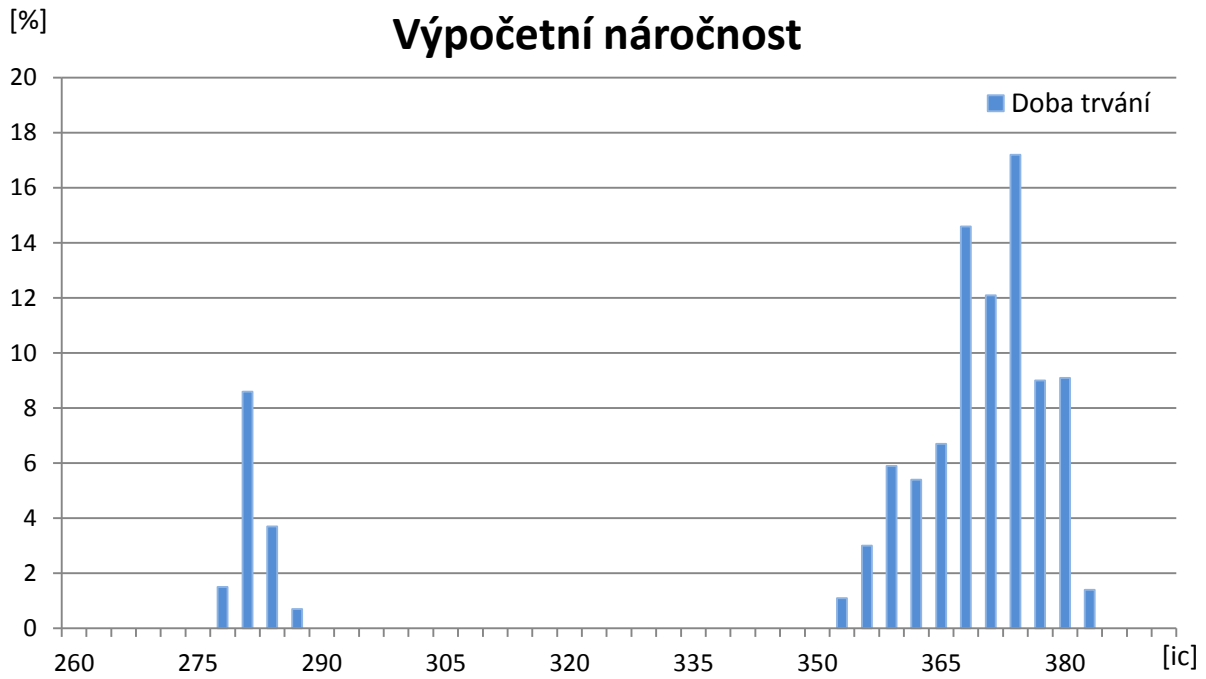


Procentuální četnost výpočtů s danou náročností pro dvojvstupý neuron

Příloha D

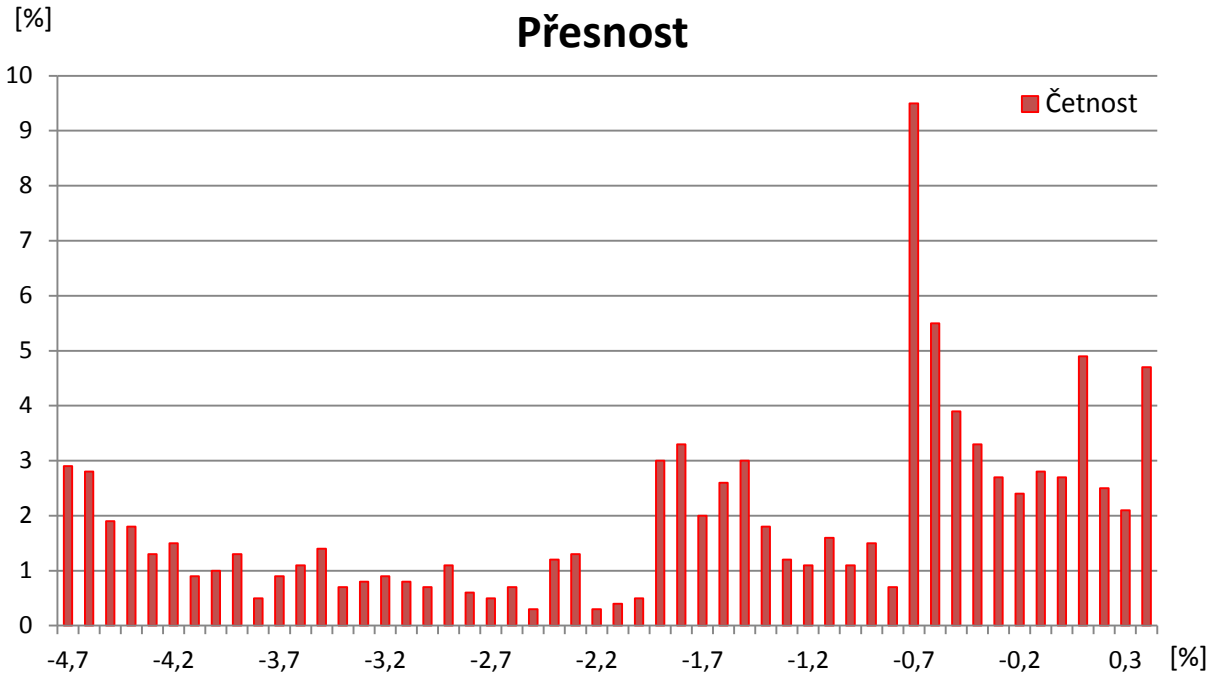


Procentuální četnost výpočtů s danou přesností pro čtyřvstupý neuron

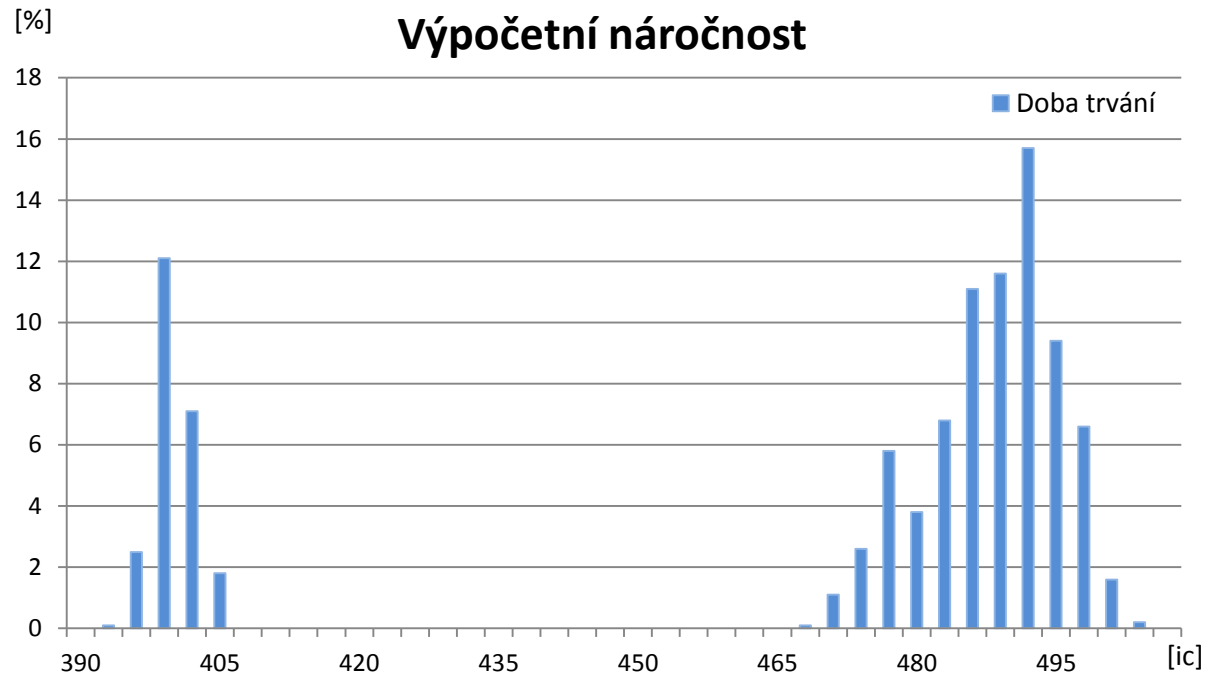


Procentuální četnost výpočtů s danou náročností pro čtyřvstupý neuron

Příloha E

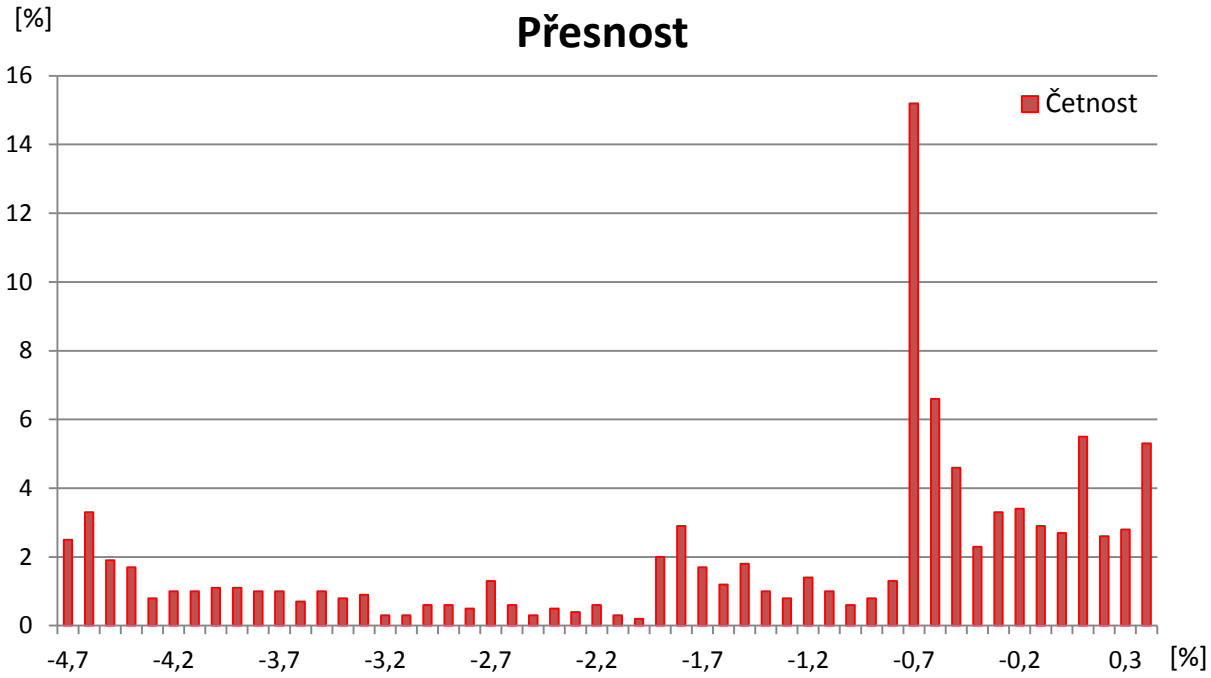


Procentuální četnost výpočtů s danou přesností pro šestivstupý neuron

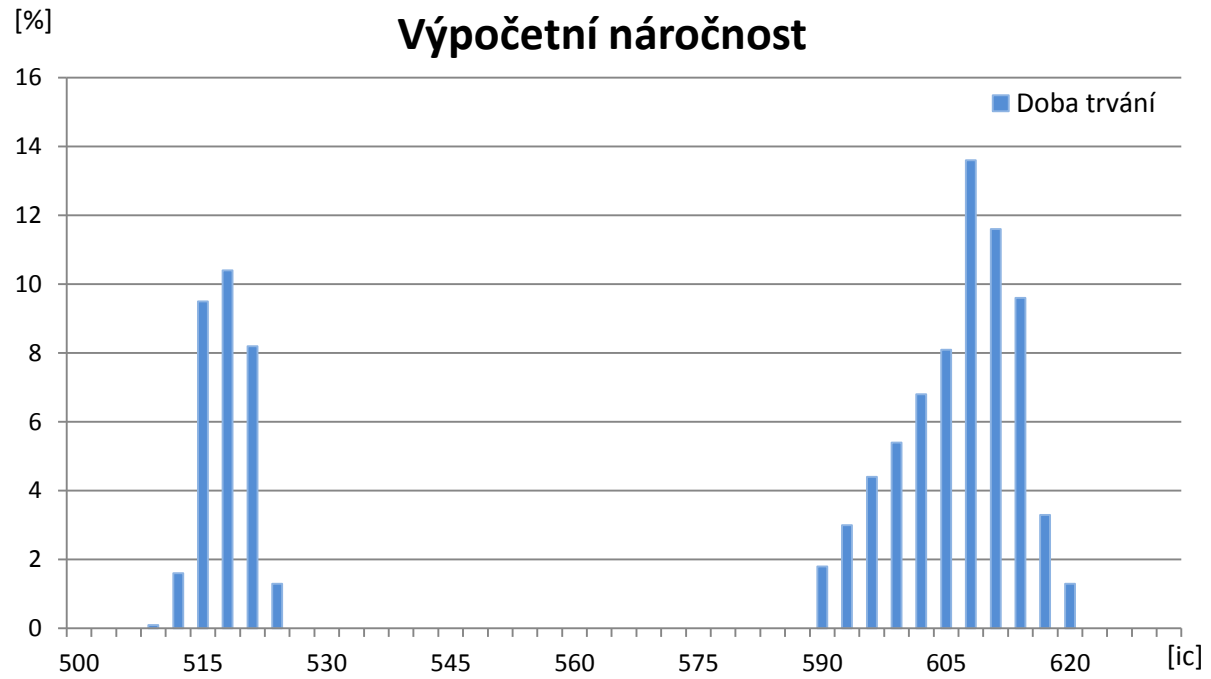


Procentuální četnost výpočtů s danou náročností pro šestivstupý neuron

Příloha F

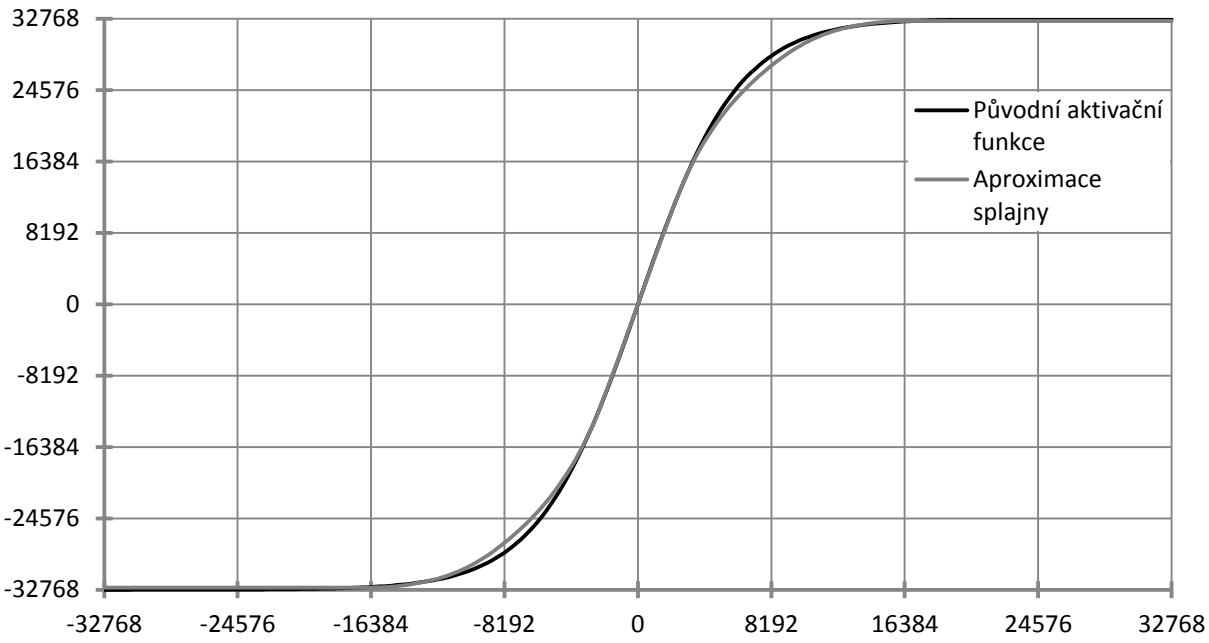


Procentuální četnost výpočtů s danou přesností pro osmivstupý neuron

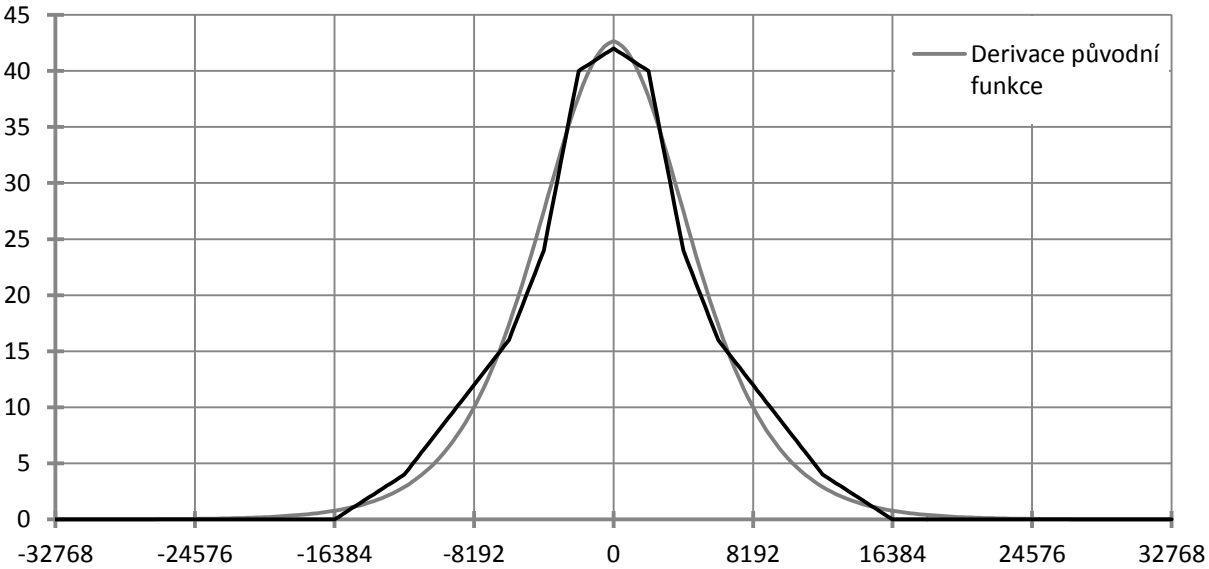


Procentuální četnost výpočtů s danou náročností pro osmivstupý neuron

Příloha G

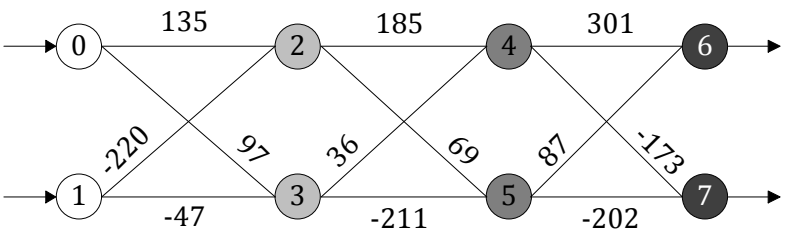


Průběh původní a aproximované hyperbolické tangenty s koeficienty pro *_unitHINT16*



Průběh derivace původní a aproximované hyperbolické tangenty s koeficienty pro *_unitHINT16*

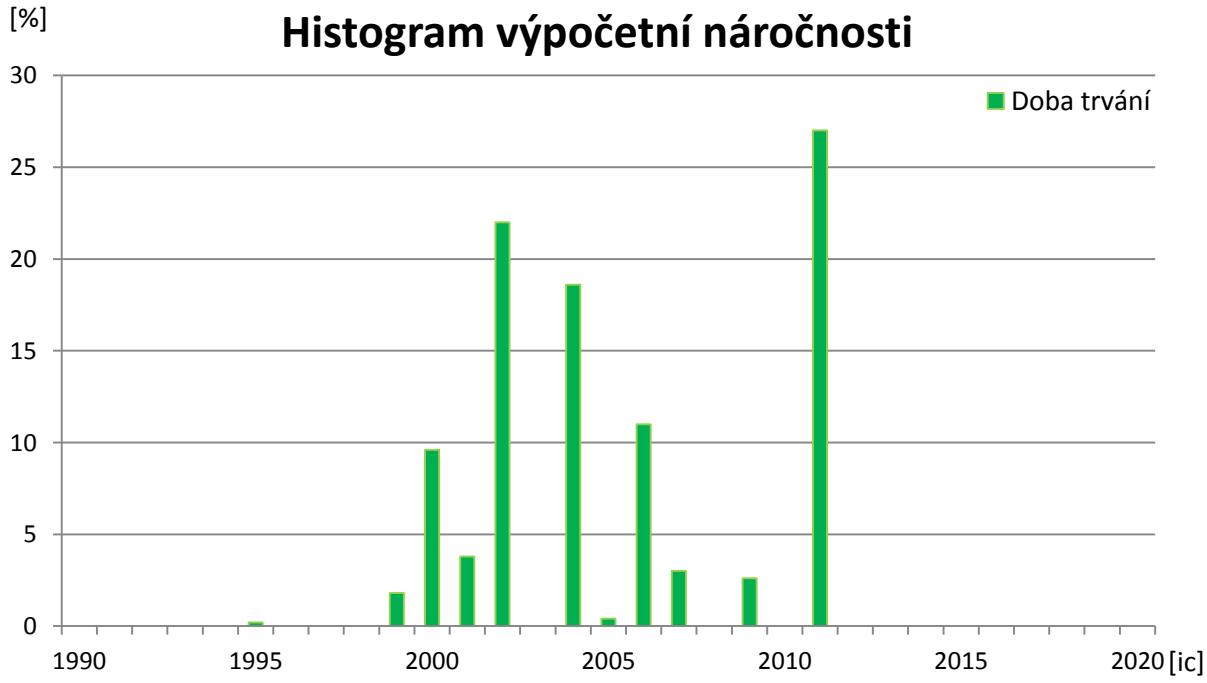
Příloha H



Grafické znázornění sítě S53 s organizací 2-2-2-2

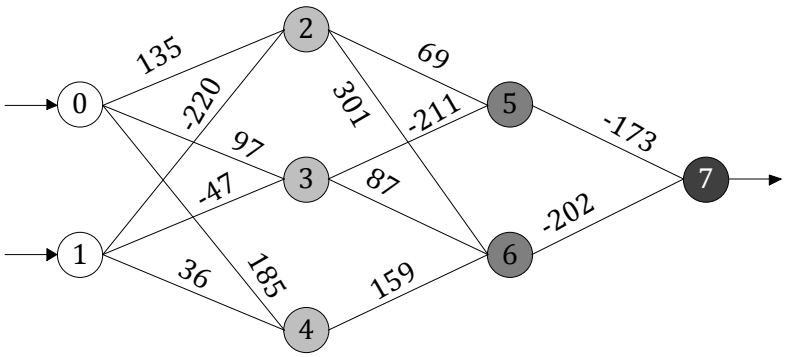
S53;8;
0;0;
0;0;
0;2;240;255;135;0;220;128;
0;2;16;255;97;0;47;128;
2;2;125;0;185;0;36;0;
2;2;62;255;69;0;211;128;
4;2;22;0;45;1;87;0;
4;2;213;0;173;128;202;128

Data pro zavedení sítě do systému „S-příkazem“



Procentuální četnost výpočtů s danou náročností uvedené neuronové sítě

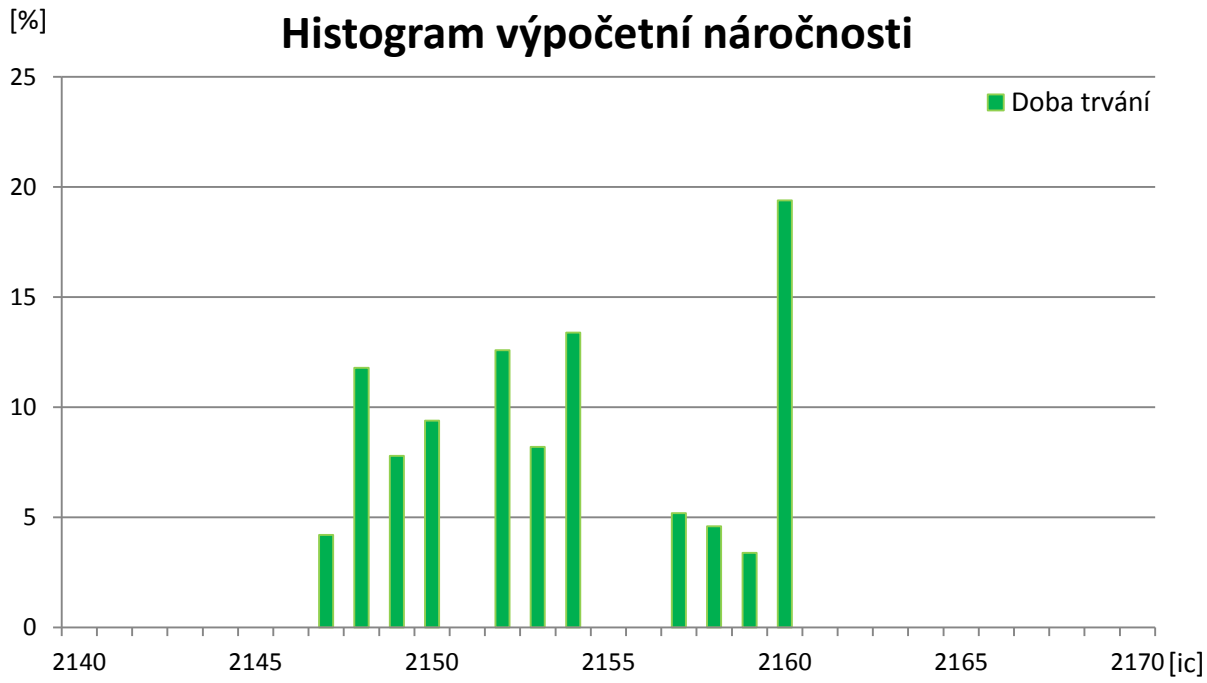
Příloha I



Grafické znázornění sítě S57 s organizací 2-3-2-1

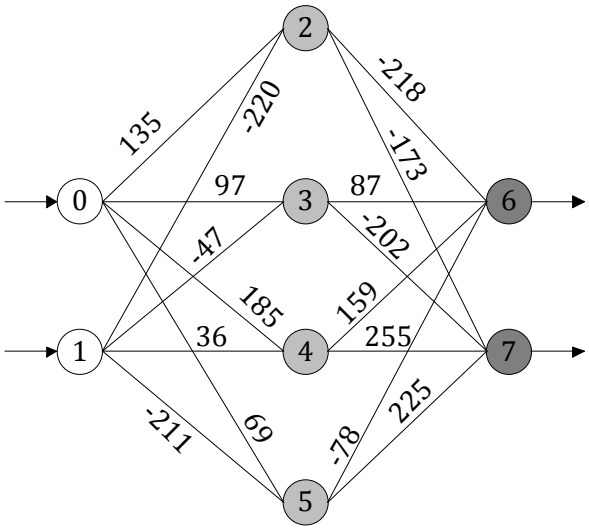
S57;8;
0;0;
0;0;
0;2;240;255;135;0;220;128;
0;2;16;255;97;0;47;128;
0;2;125;0;185;0;36;0;
2;3;62;255;69;0;211;128;0;0;
2;3;22;0;45;1;87;0;159;0;
5;2;213;0;173;128;202;128

Data pro zavedení sítě do systému „S-příkazem“



Procentuální četnost výpočtů s danou náročností uvedené neuronové sítě

Příloha J



Grafické znázornění sítě S61 s organizací 2-4-2

S61;8;

0;0;

0;0;

0;2;240;255;135;0;220;128;

0;2;16;255;97;0;47;128;

0;2;125;0;185;0;36;0;

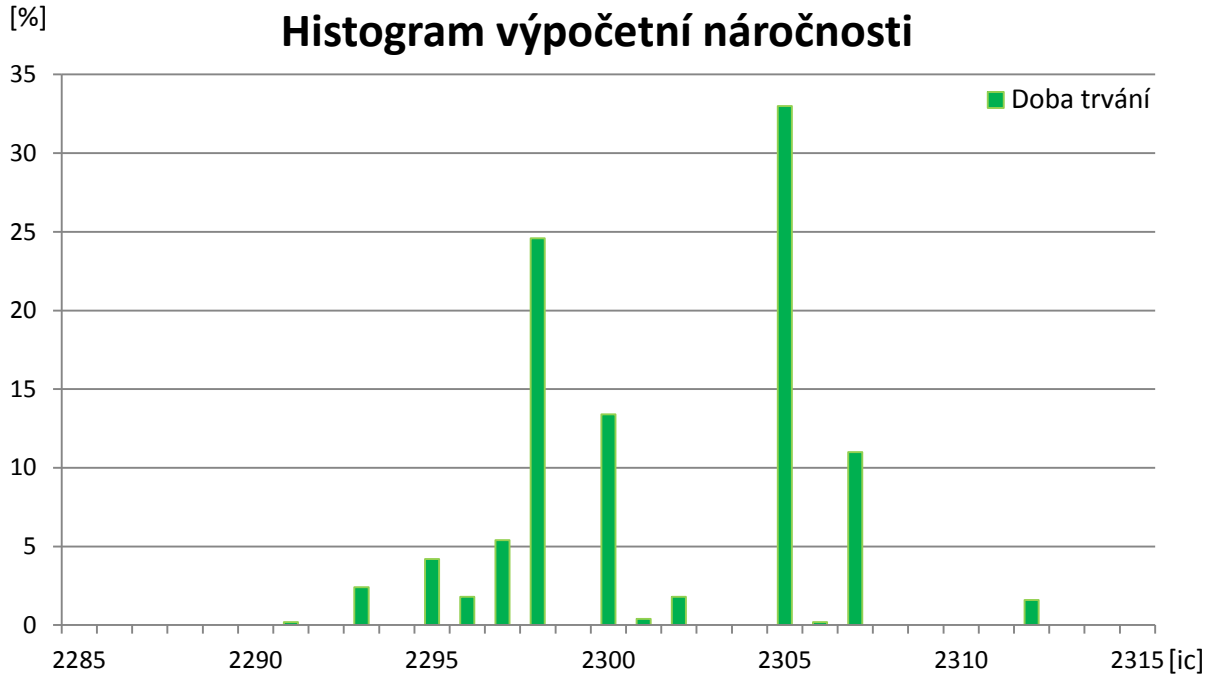
0;2;62;255;69;0;211;128;

2;4;22;0;218;128;87;0;159;0;78;128;

2;4;213;0;173;128;202;128;255;0;225;

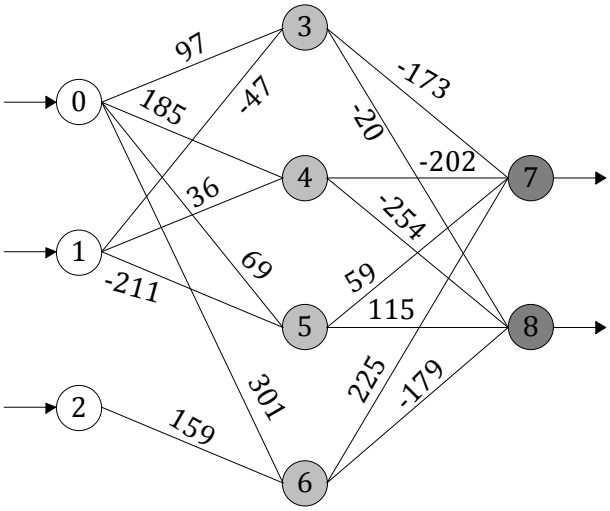
0

Data pro zavedení sítě do systému „S-příkazem“



Procentuální četnost výpočtů s danou náročností uvedené neuronové sítě

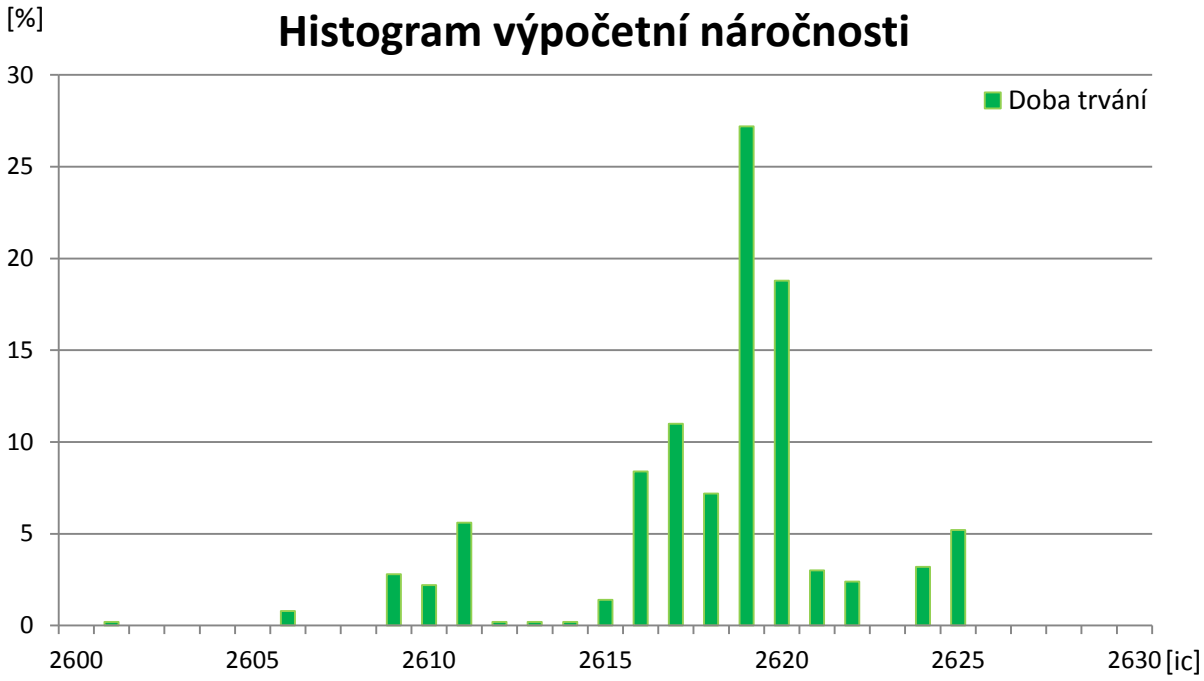
Příloha K



Grafické znázornění sítě S71 s organizací 3-4-2

S57;8;
0;0;
0;0;
0;2;240;255;135;0;220;128;
0;2;16;255;97;0;47;128;
0;2;125;0;185;0;36;0;
2;3;62;255;69;0;211;128;0;0;
2;3;22;0;45;1;87;0;159;0;
5;2;213;0;173;128;202;128

Data pro zavedení sítě do systému „S-příkazem“



Procentuální četnost výpočtů s danou náročností uvedené neuronové sítě