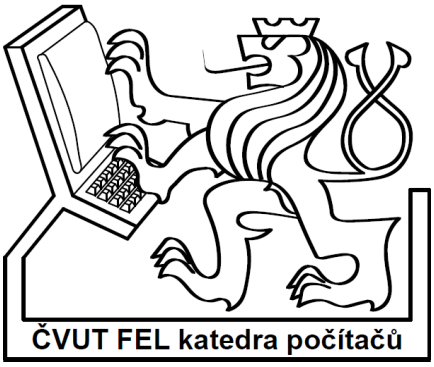
České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

Samoorganizující se neuronové sítě na FPGA

Bc. Tomáš Kunc

Vedoucí práce: Ing. Miroslav Skrbek Ph.D.

Studijní program: Elektronika a informatika strukturovaný magisterký

Obor: Výpočetní technika

leden 2010

Poděkování

Tímto bych rád poděkoval vedoucímu mé diplomové práce Ing. Miroslavu Skrbkovi Ph.D. za vedení práce, přínosné konzultace a poskytnutí podpůrných studijních materiálů.

Prohlášení:

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 SB., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 4. 1. 2010 .....

Abstract

The aim of this thesis is to design SOM (Self Organizing Map) artificial neural network in VHDL language for the use in the FPGA device Xilinx Virtex 4 which is contained in the DRC system. The design and implementation include usage of the approximated functions to compute more complex arithmetical operations.

Abstrakt

Tato diplomová práce se zabývá návrhem samoorganizující se umělé neuronové sítě typu SOM (Self Organizing Map). Cílem je návrh a implementace sítě v jazyce VHDL pro FPGA Xilinx Virtex4 obsažené v systému DRC. Návrh zahrnuje využití aproximovaných funkcí pro realizaci složitějších aritmetických operací.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Diplomová práce se zbývá návrhem implementace samoorganizující se umělé neuronové sítě typu SOM na programovatelných obvodech typu FPGA.

Umělé neuronové sítě se snaží matematickým modelem „simulovat“ chování reálných biologických neuronových struktur. Výpočty realizované těmito sítěmi jsou z podstaty paralelní, probíhající v mnoha neuronech současně a často nad stejnou sadou dat. Jsou tedy ideálními kandidáty na paralelní zpracování v hardwaru.

Neuronová síť typu SOM (Kohonenova mapa) slouží ke shlukové analýze dat. Síť má schopnost rozpoznávat podobnosti ve vícedimenzionálních vstupních datech a vytvářet projekci do dvoudimenzionálního prostoru neuronové mřížky. Vznikají tak separované shluky neuronů reagující na podobná data.

Cílem práce je návrh neuronové sítě v obvodu FPGA, umístěném v systému DRC, což umožní akceleraci výpočtů neuronové sítě. Zvláštností je ověření možnosti využití aproximovaných funkcí. Jedná se o funkce, pomocí nichž je možné složité operace jako násobení, dělení a mocnina za určitých okolností nahradit jednoduššími obvody za cenu nižší přesnosti. Výsledkem je realizace sítě v jazyce VHDL. Systém DRC je speciální počítač typu PC, na jehož základní desce se nachází obvod FPGA Xilinx Virtex 4 spojený s procesorem AMD Opteron sběrnicí HyperTransport.

1.1 Popis kapitol

Práce je rozdělena do 7 hlavních kapitol. První úvodní kapitola stručně shrnuje náplň diplomové práce.

Kapitola 2 se zabývá teoretickým popisem neuronové sítě typu SOM. Zmíní se také o aproximovatelných funkcích (shif-add aritmetika) , použitých v aritmetických obvodech navrhované sítě typu SOM.

Kapitola 3 přiblíží koncepci systému DRC (popíše jeho základní bloky a schopnosti) s důrazem kladeným na části, které jsou využitelné v rámci této diplomové práce. Kompletní popis systému DRC lze najít v uživatelském manuálu [3].

Další, 4. kapitola nazvaná „Návrh řešení“ pojednává o metodách a koncepci, kterou jsem navrhl pro následnou implementaci. Zde je blokově i funkčně popsáno k čemu slouží a jak se chovají jednotlivé části sítě. Jsou tu upřesněny i některé detaily týkající se aproximovaných funkcí a návrhu softwarového modelu sítě v jazyce Java, který je také významnou částí práce a slouží k ověření funkce a realizovatelnosti sítě.

Kapitola 5 se věnuje implementaci sítě na FPGA. Pojednává o detailech mnou provedené implementace sítě SOM v jazyce VHDL. Je zde uvedeno schéma ukazující organizaci komponent a zabývá se jednotlivými obvody, návrhem pipeline a řadiče. Zmíněny jsou pouze nejpodstatnějších části, nejedná se o kompletní popis zdrojového kódu v jazyce VHDL řádek po řádku, ten je možné nalézt v okomentované podobě na přiloženém médiu CD/DVD .

V kapitole 6 je popis provedené simulace, jako metody k ověření dosažené funkce sítě. Jsou zde vyobrazeny simulační průběhy signálů během aritmetických výpočtů. Simulace je provedena programem Modelsim 6.5.

Kapitola 7 se zabývá shrnutím a zhodnocením dosažených cílů.

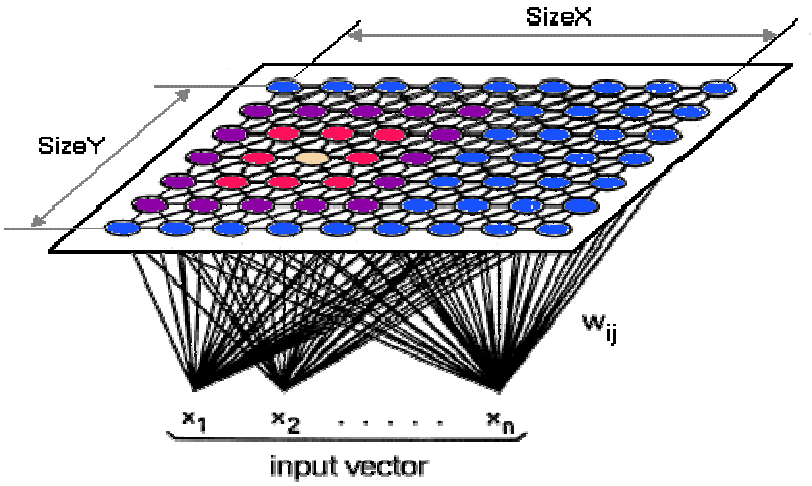
2 Teoretický úvod

2.1 Neuronová síť SOM

Neuronové sítě se obecně vyznačují poměrně velkou výpočetní náročností po paměťové i časové stránce. Tato skutečnost vyplývá z jejich struktury-sítě se mohou skládat z velkého množství výpočetních jednotek (neuronů). Každý neuron musí provádět matematické operace definované daným typem sítě a současně obsahuje i datové struktury uchovávající jeho vnitřní konfiguraci. Neurony v ideálním případě pracují paralelně. Neuronová síť je poměrně snadno realizovatelná v objektovém programování, avšak díky sekvenční povaze takové realizace je nutno počítat s pomalejší odezvou. Paralelní povaha sítí však poukazuje na vhodnost implementace v hardwaru, který takovou paralelnost bez pochyb umožňuje. Výhodu v tomto případě představuje především vyšší rychlost zpracování díky paralelismu výpočtu i přístupu do paměti. Většinou je možné přiřadit každé jednotce samostatné paměťové lokace (registry) a tudíž se vyhnout nutnosti přístupu do sdílené paměti. Nevýhodou je naopak omezený počet jednotek daný omezenými fyzickými prostředky FPGA obvodů.

SOM (*Self-Organizing Maps*) jinak také nazývané Kohonenovy mapy podle svého objevitele, patří mezi tzv. samoorganizující se typy neuronových sítí. Implementují metodu učení bez učitele. Této vlastnosti se využívá v aplikacích, kde je nutná tzv. clusterizace vstupních dat. Jedná se o hledání shluků ve vícedimenzionálních datech. Za shluky můžeme považovat data (vektory), která mají k sobě určitý vztah, tudíž jsou určitou mírou korelovaná. Sít SOM realizuje projekci vícedimenzionálních dat do prostoru o méně dimenzích při zachování určité míry topologie. SOM lze využít např. pro identifikaci různých druhů živočichů na základě mnoha vlastností (váha, rozměry, barva, …) za předpokladu, že tyto vlastnosti lze vyjádřit numericky.

Na Obr. 1 je vyobrazena základní struktura sítě. Jedná se o jednovrstvou 2D mřížku neuronů, ve které je každý neuron spojen se vstupem sítě. Vyobrazení spojů mezi neurony navzájem je abstraktní a poukazuje na skutečnost, že se sousední neurony ovlivňují. Tvar mřížky bývá ve většině případů pravoúhlý a rozměry x, y jsou volitelné. Není tedy podmínkou čtvercový tvar. Vstupní data sítě SOM mají podobu n-dimenzionálních vektorů $(x_1 x_2 \ldots x_n), x_n \in R$. Každý neuron obsahuje datové struktury, které udržují informaci o jeho vnitřním stavu. Jedná se o konstantní registry, udržující informaci o pozici (neměnná) a právě jeden tzv. váhový vektor (proměnný). Váhový vektor je ve stejném formátu jako vektory vstupních dat. Tento vektor se během učení neuronu modifikuje a výsledná konfigurace sítě je dána sadou těchto vektorů. Váhové vektory určují, jak intenzivně bude daný neuron reagovat na vektor vstupních dat.



Obr. 1 Síť SOM (Převzato z http://www.lohninger.com/helpcsuite/img/kohonen1.gif)

2.1.1 Etapy výpočtu

U neuronových sítí SOM se setkáváme se 3 základními fázemi (režimy).

1. Inicializace
2. Učení
3. Vybavování

2.1.2 Fáze inicializace

Inicializace složí k počátečnímu nastavení vah jednotlivých neuronů. Toto nastavení může být provedeno náhodně nebo předem připravenými vektory vah. Každému vektoru se v rámci této fáze přidělí právě jeden váhový vektor, vytvoří se počáteční konfigurace sítě.

2.1.3 Fáze učení

Během fáze učení musí všechny neurony v síti modifikovat své váhy tak, aby reprezentovaly rozložení vstupních dat. Inicializované síti se předkládají vektory dat. Mohou to být např. vektory z trénovací množiny. Síť implementuje učení bez učitele, tudíž se nevkládají žádná data o příslušnosti vektoru k určité třídě. Všechny neurony mají povolenu změnu vah. Váhy pro daný neuron se přizpůsobují na základě vzdálenosti neuronu od neuronu nazývaného BMU(Best Matching Unit). Jedná se o neuron, jehož váhový vektor je v n-dim. prostoru nejblíže ke vstupnímu vektoru. Z toho mimo jiné plyne, že dimenze datových a váhových vektorů musí být identická. Pro potřeby úprav vah je nutné určit pro každý vstupní vektor BMU, což vyžaduje, aby každý neuron individuálně vypočetl vzdálenost vstupního datového vektoru od svého dosavadního váhového vektoru. Veškeré neurony počítají individuálně tuto vzdálenost. Nejčastěji využívaná metrika pro výpočet vzdálenosti mezi dvěma vektory je definovaná:

$$\mathrm{dk(t)} = \sum_i (\mathrm{x_i} - \mathrm{w_i})^2 \qquad (1)$$

Kde $dk(t)$ je vzdálenost $\vec{X}$ a $\overrightarrow{W}$ a Xi , Wi jsou složky vektoru dat resp. vah.

Pro výpočet vzdálenosti mezi dvěma neurony v 2D mřížce lze využít tzv. Manhattanskou vzdálenost definovanou:

$$d = |x_1 - x_2| + |y_1 - y_2| \tag{2}$$

Pro úpravu vah existují dva základní matematické postupy. Jedná se o tzv. online a offline algoritmus [2] [3]. Nejčastější a původní Kohonenův (autor sítě SOM) algoritmus je první zmíněný.

2.1.4 Online algoritmus

Váhy neuronů se během učení upravují po každém předloženém vektoru dat. Postupuje se dle následujících kroků.

I. Určení minima vzdálenosti od vstupního vektoru, neuron s tímto minimem je označován jako BMU (Best Matching Unit).

$$dc(t) = min\, dk(t) \tag{3}$$

II. Úprava vah všech neuronů

$$W_k(t+1) = W_k(t) + \alpha(t) h_{ck}(t)[x(t) - W_k(t)] \tag{4}$$

kde $0 < \alpha(t) < 1$ je koeficient závislý na čase (monotónně klesající), $h_{ck}(t)$ je funkce okolí. Většinou se jedná o variantu Gaussovy funkce. Výraz $[x(t) - W_k(t)]$ představuje vzdálenost vektoru dat od vektoru vah daného neuronu v čase t.

Tyto kroky se cyklicky opakují pro každý vstupní vektor. Na konci epochy (sada vektorů během fáze učení) učení tak získáme konečné nastavení vah. Jak je vidět lze tento algoritmus použít i v situaci, kdy nemáme v čase t k dispozici celou sadu vstupních datových vektorů pro danou epochu učení. Váhy se mění po každém vektoru a máme tedy k dispozici jejich aktuální stavy, proto se také tomuto postupu anglicky říká „online“.

2.1.5 Offline algoritmus - Batch SOM

Batch algoritmus se liší od předešlé varianty v definici okamžiku změny vah. Algoritmus provádí změnu váhových vektorů až na konci epochy. Z toho plyne, že je předurčen pro aplikace, ve kterých jsou finální váhy potřebné až na konci epochy, do této chvíle zůstávají beze změn. Určení vzdálenosti vektorů i určení BMU zůstává stejné jako u online algoritmu. Při každém předložení vektoru je zjištěna BMU a v rámci každého neuronu je aktualizován čitatel a jmenovatel vzorce (5) a po zpracování posledního vektoru následuje modifikace vah pro každý neuron a to vydělením každé složky naakumulovaného vektoru v čitateli jmenovatelem, čímž se získá nová váha. Formálně se tedy úprava vah provádí za využití vzorce:

$$W_k(t_f) = \frac{\sum_{t'=t_0}^{t'=t_f} h_{ck}(t')\, x(t')}{\sum_{t'=t_0}^{t'=t_f} h_{ck}(t')} \tag{5}$$

Kde čas t_f je čas konce epochy a t_o je čas začátku epochy, $W_k(t_f)$ je nová váha na konci epochy, $x(t')$ je vstupní vektor v čase t' h_{ck} je funkce okolí. Jedná se o monotónní klesající funkci. Argumentem této funkce je prostorová vzdálenost daného neuronu od BMU v 2D mřížce. Pro výpočet vzdálenosti se využívá např. Euklidovská nebo Manhattanská metrika. Výsledek metriky se

využije jako argument funkce h_{ck}, která často bývá ve tvaru Gaussovy funkce $e^{\frac{-||d1-d2||^2}{\sigma(t)^2}}$, kde čitatel exponentu představuje vzdálenost a jmenovatel monotónní klesající funkci, která se mění v čase mezi epochami učení a zajišťuje postupný útlum vlivu BMU jednotek (zužování okolí). Při posledním opakování epochy může dojít ke zmenšení okolí na nulu, tzn. jen BMU mění své váhy (BMU je vždy ve vzdálenosti 0).

2.1.6 Fáze vybavování.

Naučená síť v této fázi vybavování poskytuje jako odpověď na vstupní vektor pozici neuronu v mřížce, který má nejmenší vzdálenost od předloženého vektoru. Sděluje tedy pozici BMU. Pozice se využívá k dalšímu zpracování, například ke klasifikaci dat do tříd. To již není tématem práce, proto dále není následnému zpracování věnována pozornost.

2.2 Datové typy

Většina neuronových sítí pracuje s vektorovými daty. Nejinak je tomu u sítě SOM. Vektorem $X = (x_1, x_2, x_3 \dots x_n)$ rozumíme skupinu několika čísel (složek, elementů vektoru) s přesně daným pořadím. Počet složek vektoru X je dán počtem dimenzí n, vstupního prostoru se kterým neuronová síť pracuje a tedy i shodný s počtem složek váhových vektorů jednotlivých neuronů. Počet dimenzí byl stanoven na hodnotu 16. Složka vektoru bývá z oboru reálných čísel. V případě mojí implementace byla zvolena varianta normalizovaných desetinných čísel v rozsahu intervalu $(-1.0\,;\,+1.0)$. Pro implementaci takového datového typu lze využít více různých formátů, počínaje pevnou řádovou čárkou až po složitý formát s plovoucí řádovou čárkou. Byla zvolena varianta formátu čísel s pevnou řádovou čárkou a to z důvodů menší složitosti a použití aproximovaných funkcí, předpokládajících tento formát vstupů. Formát čísel je v rámci dokumentace dále označován také názvem f*racval*. Fracval se v síti vyskytuje ve 2 variantách a to fracval16 a fracval32, což je rozšířená verze pro zvýšení přesnosti při operaci dělení v síti SOM. Tvar řádové mřížky fracval16 je uveden na obr Obr. 2 a fracval32 na Obr. 3. Fracval je formát, u kterého je nejvyšším bitem znaménkový bit, za ním následuje řádová čárka a ostatní bity představují záporné mocniny základu soustavy. Jednotka řádové mřížky fracval16 je 2^{-15} a modul je $2^0 = 1$, situace u fracval32 je analogická jednotka 2^{-31}, modul $2^0 = 1$. Existují dva typy fracval čísel, lišící se kódováním záporných hodnot – v doplňkovém a přímém kódu. U přímého kódování určuje bit S (sign) znaménko (1=záporné číslo) a zbylé datové bity představují absolutní hodnotu čísla. U doplňkového kódu lze „zápornost“ určit také na základě nejvyššího bitu, avšak zbylé bity nepředstavují absolutní hodnotu, pokud je bit S logická 1.

S	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}

Obr. 2 Formát čísel fracval16

S	2^{-1}	bity 29:1	2^{-31}

Obr. 3 Formát čísla fracval32

Některé operace v síti jsou založeny na logaritmech, a pro výsledky těchto operací byl zaveden vlastní datový typ nazvaný *logval*. Číslo se skládá z celé části a části za řádovou čárkou. Na Obr. 4 a Obr. 5 je celá část zvýrazněna zelenou barvou. Kódování záporných hodnot čísla logval je pouze ve variantě přímého kódu. Jsou použity dvě varianty a to *logval 20* (šířka 20 bitů) a *logval 37* (šířka 37 bitů). Způsob kódování i vyhodnocení obou typů je identický, liší se pouze v bitových šířkách racionální části (za řádovou čárkou). Modul *logval 20* resp. *logval 37* je 2^5 resp. 2^6 , jednotka 2^{-14}, resp. 2^{-30}.

S	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}

Obr. 4 Formát čísla logval 20

S	bity 35:30	bity 29:0

Obr. 5 Formát čísla logval 37

Pro vyjádření pozice neuronu v síti (mřížka) je použita celočíselná hodnota o šířce 5 bitů pro x a 5 bitů pro y pozici. Maximální rozměr x nebo y je $2^5 = 32$ a maximální počet neuronů v síti je tedy $2^{5+5} = 1024$. Hodnota 1024 neuronů v síti je samozřejmě na FPGA v praxi nedosažitelná, uvádím ji jen jako teoretickou mez při využití uvedených dat. typů.

2.3 Aproximované funkce

Implementace neuronové sítě v hardware je náročná především díky nutnosti provádět složitější matematické operace jako je násobení, mocnina a dělení, jak je vidět ze vzorců v kapitole 2.1. Implementace těchto funkcí na rozdíl od sčítání/odčítání a mocnin 2 je náročnější na množství prostředků FPGA (LUTů a registrů). Současně tyto komplexní obvody způsobují zpomalení výpočtu (větší perioda hodin nebo více taktů). Delší periodu hodin lze částečně kompenzovat pipelinovaným návrhem. To ovšem vede k větší latenci naplnění a vyprázdnění pipeline. Z tohoto důvodů byly pro návrh sítě využity aproximované funkce [1]. Dalším důvodem byla potřeba ověřit, zda je síť typu SOM těmito funkcemi realizovatelná s rozumnou chybou, nebo je tato cesta neuskutečnitelná.

Aproximované funkce [1] umožňují výpočet mocnin, logaritmu, exponenciál, násobení a dělení za použití základních obvodů typu součet, posuv. Cenou za toto usnadnění je určitá míra nepřesnosti v řádech jednotek procent. Bližší informace o jednotlivých funkcích jsou uvedeny v následujících kapitolách.

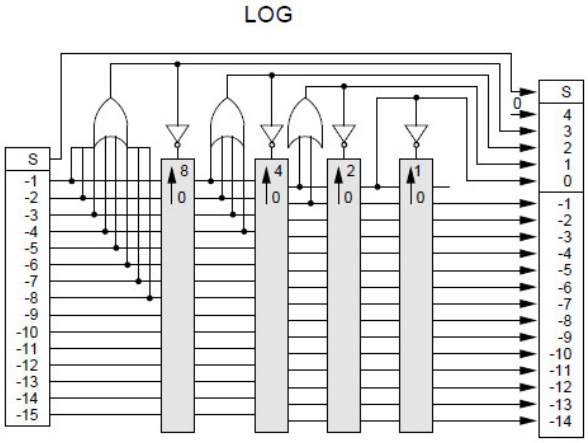
2.3.1 Logaritmus $\log_2(x)$

Funkce aproximovaný logaritmus počítá přibližný výsledek dvojkového logaritmu z čísla typu fracval. Výstupem funkce je číslo formátu logval. Funkci pro celá čísla x lze popsat výrazem:

$$LOG_2(x) = int(log_2 x) + \frac{x}{2^{int(log_2 x)}} - 1 \tag{6}$$

kde int(x) značí celočíselnou část.

Princip obvodu pro výpočet dvojkového logaritmu čísla typu fracval spočívá v nalezení nejlevější jedničky v řádové mřížce. Pozice této jedničky se objeví jako hodnota celočíselné části datového typu logval. Část vstupního čísla za nejvyšší jedničkou se stane racionální částí výstupního logval čísla. K nalezení nejvyšší jedničky se využívá hledání pomocí hradel logický součet a barrel shifteru,který posouvá vstupní číslo a tím současně zajistí správnou pozici ve výstupu. Hloubka obvodu je logaritmus z délky racionální části vstupu. Nejvyšší bit celočíselné části výstupu je nulován, jelikož maximální pozice jedničky je 15 (pozice jsou číslovány od 1). Schéma obvodu realizujícího funkci je uvedeno na Obr. 6.



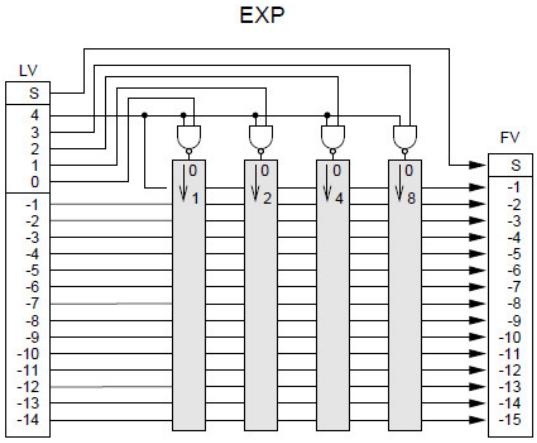
Obr. 6 Aproximovaný logaritmus (Převzato z [1])

2.3.2 Exponenciála

V aproximovaném tvaru lze funkci vyjádřit následujícím předpisem:

$$\text{EXP(x)} = 2^{int(x)} * \left(1 + frac(x)\right) \tag{7}$$

kde $int(x)$ je celá část logval čísla x a $frac(x)$ je část za řádovou čárkou. Vstupem je číslo logval, výstupem je číslo fracval. Princip tohoto obvodu lze přímo odvodit ze vzorce (7). Výraz $1 + frac(x)$ představuje součet racionální části a celého čísla 1, lze tedy snadno implementovat tak, že se ze vstupního fracval čísla vezme racionální část a před ní se předsadí bit s hodnotou 1. Výraz $2^{int(x)}$ představuje posuv doleva, to ale předpokládá umístění čísla $1 + frac(x)$ úplně napravo za řádovou čárkou. Pro úsporu HW se více osvědčí zapojení uvedené na Obr. 7. Zde se číslo posouvá směrem doprava.



Obr. 7 Aproximovaná funkce EXP(x) (Převzato z [1])

Je důležité si povšimnout zapojení nejvyššího bitu. Tento bit při hodnotě 0 způsobí odsunutí celého čísla doprava a nulovost výsledku. Pokud se podíváme na zapojení funkce logaritmus, je vidět, že na této pozici výsledku vždy vystavuje právě hodnotu 0, což znamená, že přímé zřetězení funkce log a exp za sebe nevede ke stejnému výsledku, jako byl vstup. Funkce tedy nejsou k sobě inverzní, tak jak by se mohlo matematicky zdát, ale jsou navrženy přímo pro účel využití v neuronové síti, pro operaci násobení nebo dělení. Např. při násobení se předpokládá, že výsledky logaritmů se sečtou a teprve potom dojde k aplikování exponenciální funkce pro zpětnou transformaci odlogaritmováním.

2.3.3 Aproximované násobení

Pro násobení dvou čísel lze použít logaritmy a exponenciály (2^x). Využívá se známých matematických vztahů:

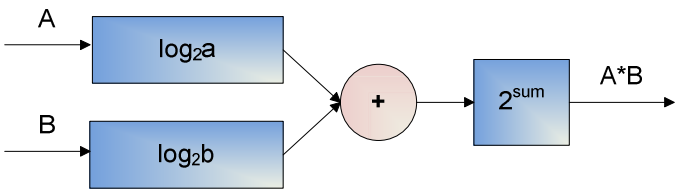
$$\log_2 (a * b) = \log_2 a + \log_2 b \tag{8}$$

$$2^{log_2(a*b)} = a * b \tag{9}$$

Ze vztahů 1 a 2 plyne:

$$a * b = 2^{(log_2 a + log_2 b)} \tag{10}$$

Ze vzorce (10) přímo vyplývá, jaká hardwarová struktura může násobení realizovat. Blokové schéma ukazuje Obr. 8.



Obr. 8 Násobička

V kapitole 2.2.2 byla zmíněna poměrně důležitá skutečnost a to, že funkce $log_2 X$ a $EXP(x)$ nejsou vůči sobě inverzní. Důvod spočívá v tom, že funkce pracují zapojené uvnitř neuronové sítě, kde se počítá s hodnotami fracval (na to jsou tyto funkce koncipovány). Transformace z desetinného čísla a v intervalu (0; 1) do hodnoty A fracval je definována takto:

$$A = a * 2^n \tag{11}$$

kde a je číslo před transformací, A je transformovaná fracval hodnota a n je počet bitů racionální části čísla fracval, standardně u typu fracval16 se jedná o 15 bitů, tedy n=15. Násobení s ohledem na tuto transformaci a na skutečnost, že výsledkem musí být opět číslo ve stejné transformaci vypadá následovně:

$$A * B = (a * 2^{15}) * (b * 2^{15}) = (a * b) * 2^{30} \tag{12}$$

Pro převod do stejné transformace je nutno výsledek podělit 2^{15}:

$$C = (A * B) * 2^{-15} \text{ , a poté platí } c = C * 2^{-15} \tag{13}$$

Aplikací na vzorec (10) získáme

$$2^{(\log_2 A + \log_2 B) * \mathbf{2^{-15}}} = 2^{\log_2 (a*b*2^{30})} * \mathbf{2^{-15}} = (a * b * 2^{15}) \tag{14}$$

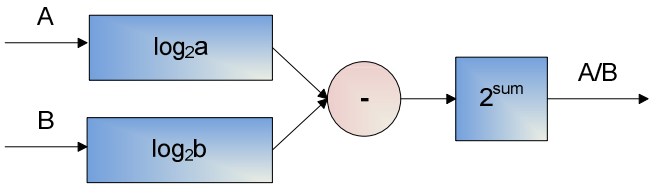
Srovnáním s realizací exponenciály v kapitole 2.3.2 vidíme, že obvod při všech číslech které mají ve 4. bitu celočíselné části na vstupu 0, způsobí posuv o 15, odsune všechny bity čísla doprava a tudíž výsledek je nula, čísla logval s celočíselnou částí menší než 16 jsou odsunuta. Toto zajistí navrácení do transformace fracval16. Díky hradlům typu XOR je současně provedena negace čtyř dolních bitů celočíselné části (vytváří se tak doplněk do 15 ze 4 dolních bitů), což znamená, že výsledný počet posunutí doprava lze definovat vzorcem:

$$15 - (logval\ int\ part - 16) = 31 - logval\ int\ part \tag{15}$$

Obvod v podstatě provede jak výpočet exponenciály, tak i navrácení hodnoty čísla do původní transformace viz vzorec (13). Předpokladem správné funkce obvodu je skutečnost, že číslo, které je na vstupu, vzniklo jako výsledek součtu dvou logval hodnot a obvod by měl dokončit operaci násobení. Z výše uvedeného vyplývá, že se obvod nechová jako klasická inverzní funkce k funkci logaritmus.

2.3.4 Aproximované dělení

Operace dělení je založena na stejném principu jako násobení, jen je zaměněna operace sčítání za operaci odčítání. Tato operace před samotným zahájením prací na síti nebyla dostupná, byla vytvořena až v rámci implementace. Návrh operace předpokládá, že čitatel bude menší než jmenovatel, což bude v bloku, ve kterém bude dělení využito, splněno. Více se o dělení zmíním v kapitole 5.2.6.



Obr. 9 Dělička

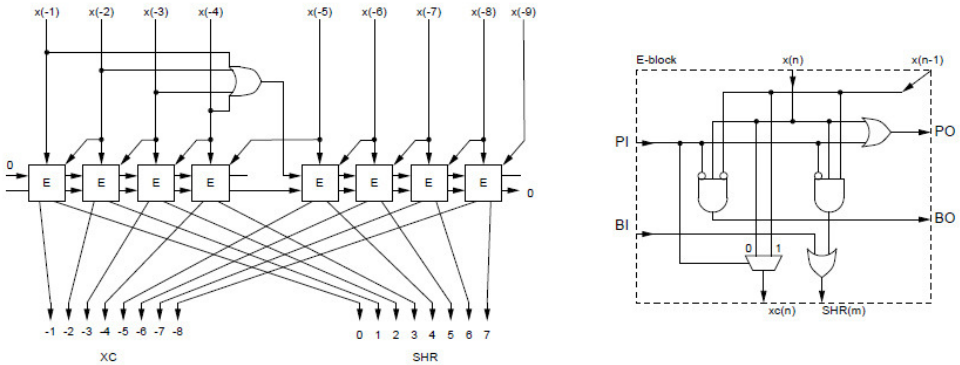
2.3.5 Funkce mocnina x²

Funkce mocniny může být implementována jako násobení. Je však možno použít optimálnější řešení, které popisuje operaci výrazem:

$$sqr(x) = (a + 2c) \ll (n + x_{n-1}) \tag{16}$$

kde $a = 2^n$, n je pozice nejlevější jedničky, $b = x_{n-1}2^{n-1}$, x_{n-1} je hodnota bitu, který je pravým sousedem bitu s nejvyšší jedničkou, $c = x - a - b$. Operace << představuje logický posuv doleva.

Funkce je vyhodnocována ve dvou krocích, v prvním kroku je vypočteno a, b, c, $n+x_{n-1}$. Ve druhém kroku je proveden logický posuv doleva o $n+x_{n-1}$. Pokud je operace aplikována na čísla z intervalu (0,1) potom záporný argument mění směr posuvu. Detekce nejlevější jedničky a výpočet výrazu 2c může být vyhodnocen zřetězením bloků viz Obr. 10. Funkce *sqr* má za vstup hodnotu fracval16. Bity 2^{-1} až 2^{-9} ze vstupu jsou zpracovány řetězem bloků *E-block*. Poté je vypočteno x+2c a nastaveny výstup XC a současně je aktivován jeden ze SHR výstupů. Tento výstup slouží jako příkaz pro barrel shifter. XC bity jsou dále doplněny o bity 2^{-9} až 2^{-15} a kompletní číslo je posláno skrze barrel shifter.



Obr. 10 Funkce SQR (Převzato z [1])

2.3.6 Přesnost aproximovaných funkcí

Přesnost aproximovaných funkcí se dle [1] pohybuje v rozmezí, které uvádí Tab. 1.

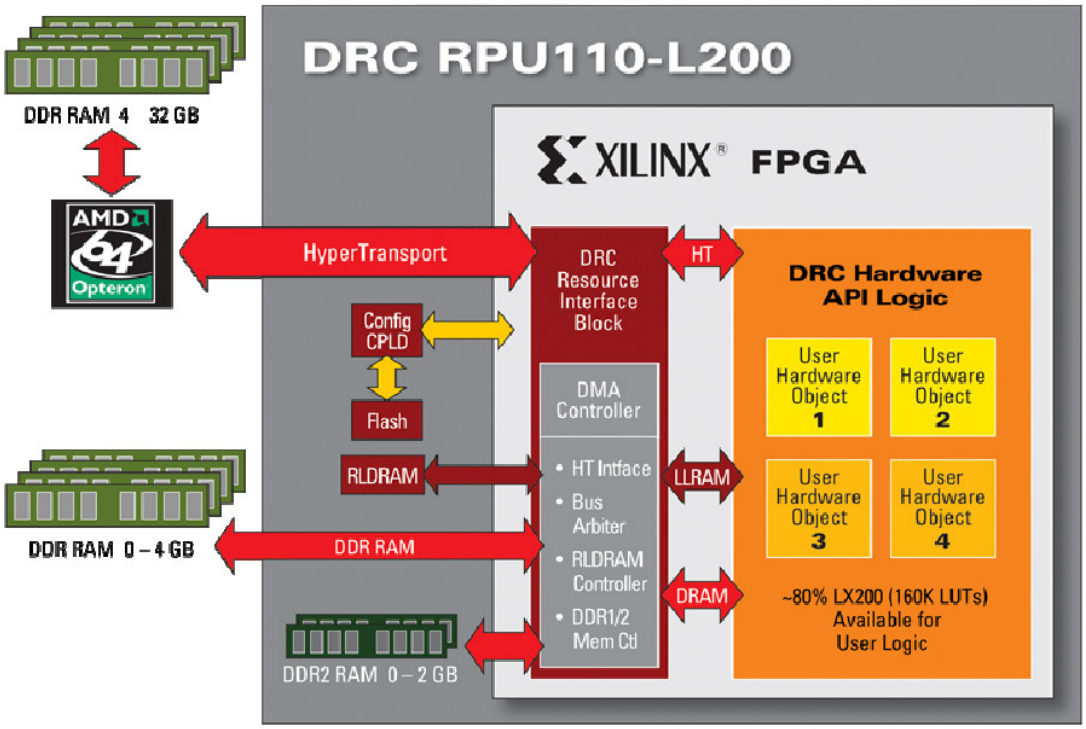
Funkce	ε_{max}	$\mu(\varepsilon)$	$\sigma^2(\varepsilon)$
$EXP_2(x)$	0.086	-	-
$LOG_2(x)$	-0.086	-	-
SQR(x)	0.043	-	-
x*y	0.0625	0	$2.268*10^{-4}$

Tab. 1 Přesnost aproximovaných funkcí

3 Popis HW

3.1 Systém DRC

Systém DRC se sestává ze speciální základní desky, na které je instalován obvod FPGA Xilinx Virtex 4 LX200. Systém je postaven na platformě AMD Opteron. Na Obr. 11 je vyobrazeno blokové schéma DRC a kompletní specifikace jsou uvedeny v Tab. 2. Zmíním se pouze o částech systému souvisejících s dalšími kapitolami, kompletní popis systému je možné nalézt v uživatelském manuálu systému DRC[5], případně v datasheetu obvodu Xilinx Virtex4 LX200-11 (XC4VLX200) [6].



Obr. 11 Struktura systému DRC (Převzato z [5])

Při bližším pohledu je vidět, že FPGA má možnost přístupu do paměti DDR2 a je schopno komunikovat přímo s procesorem po sběrnici HyperTransport. Pro tuto diplomovou práci je podstatná převážně část týkající se samotného FPGA a okrajově sběrnice HT. Jako rychlý komunikační prostředek mezi procesorem a obvodem FPGA je využita již zmíněná sběrnice HyperTransport, která představuje dvoubodové spojení typu point-to-point. Nejedná se tedy o klasickou sběrnici, ke které jsou „paralelně“ připojena zařízení, nýbrž o systém až 32 dvoubodových spojů. Pro přenos se využívá metoda DDR, kdy jsou data přenášena náběžnou i sestupnou hranou hodinového signálu. Šířka jedné datové linky se liší v závislosti na provedení v rozsahu 2-32b. K přenosu informace je použito LVDS (Low Voltage Differential Signalling). Jedná se o symetrické vedení, kde se logická 1 a 0 vyhodnocuje na základě rozdílového napětí na páru vodičů a napěťová úroveň je stanovena na 1.2V. Výrobce základní desky DRC dodává jako součást

zařízení komponenty pro komunikaci se sběrnicí HT. Tyto komponenty existují v podobě netlistu, který se využívá při procesu mapování a rozmístění cílového uživatelského HW návrhu pro FPGA. Dodaný netlist poskytuje paralelní vstup a výstup bez nutnosti psaní vlastního řadiče sběrniče. V dalších kapitolách je proto tato část někdy nazývána řadičem sběrnice HyperTransport.

Jak je dále vidět z obrázku, lze standardní obsah FPGA v systému DRC rozdělit na dvě části nazývané „DRC HW API Logic“ a „DRC Resource Interface Block“. První z uvedených obsahuje uživatelský návrh, což je právě ta část, kde lze implementovat například neuronovou síť či jiný HW návrh. Druhá část obsahuje řadič HT a pamětí a představuje tedy vlastně rozhraní pro komunikaci se zbytkem sytému.

Řadič HT poskytuje signály pro komunikaci se sběrnicí HT uvedené v Tab. 3. Přes tyto signály lze komunikovat s uživatelským HW pomocí dodaného driveru, který poskytuje API v jazyce C. Driver podporuje přenosy po slovech a dávkové burst přenosy. Řadič současně generuje i hodiny pro uživatelský návrh a je možno parametrem při syntéze nastavit několik hodnot frekvence (signál usr_clk), díky čemuž lze přizpůsobit rychlost schopnostem návrhu:

- 33MHz
- 50MHz
- 100MHz
- 133MHz
- 167MHz
- 200MHz

Bližší popis HW komunikačního protokolu protokolu je uveden v kapitole 5.1.

FPGA[1]	
Xilinx Virtex™-4	LX200-11 (XC4VLX200)
Number of LUTs	200,448
Block RAMs (w/ ECC) (18Kb each)	336 (6,048Kbits)
Digital Clock Managers (DCMs)	12
XtremeDSP™ Slices	96
HyperTransport™	
Number of Physical Links	3 (HT0/1/2)
Link Frequency and Width	400MHz x 8bits (DDR)[2]
Per Link Bandwidth	1.6GB/s per direction, theoretical 3.2GB/s aggregate, theoretical
Total Bandwidth	9.6GB/s aggregate, theoretical
Motherboard DRAM	
Number of Physical Buses	1
Type and Size	DDR DIMMs, 0-4GB
Bus Frequency and Width	200MHz x 288bits (DDR w/ ECC[3])
Per Bus Bandwidth	6.4GB/s peak
Total Bandwidth	6.4GB/s peak
Accessible from User Logic	Yes
Accessible from User Application	No[3]
RPU DRAM[3]	
Number of Physical Buses	2
Type and Size	DDR2 miniDIMMs, 0-2GB
Bus Frequency and Width	200MHz x 144bits (DDR w/ ECC[3])
Per Bus Bandwidth	3.2GB/s peak
Total Bandwidth	6.4GB/s peak
Accessible from User Logic	Yes[3]
Accessible from User Application	No[3]
RPU Low Latency RAM	
Number of Physical Buses	2
Type and Size	RLDRAM, 128MB (2x64MB)
Bus Frequency and Width	200MHz x 18bits (DDR w/ ECC[3])
Per Bus Bandwidth	800MB/s peak
Total Bandwidth	1.6GB/s peak
Accessible from User Logic	Yes
Accessible from User Application	Yes
Non-Volatile Memory	
Type and Size	Flash, 256Mbits
Accessible from User Logic	No
Accessible from User Application	Yes[3]

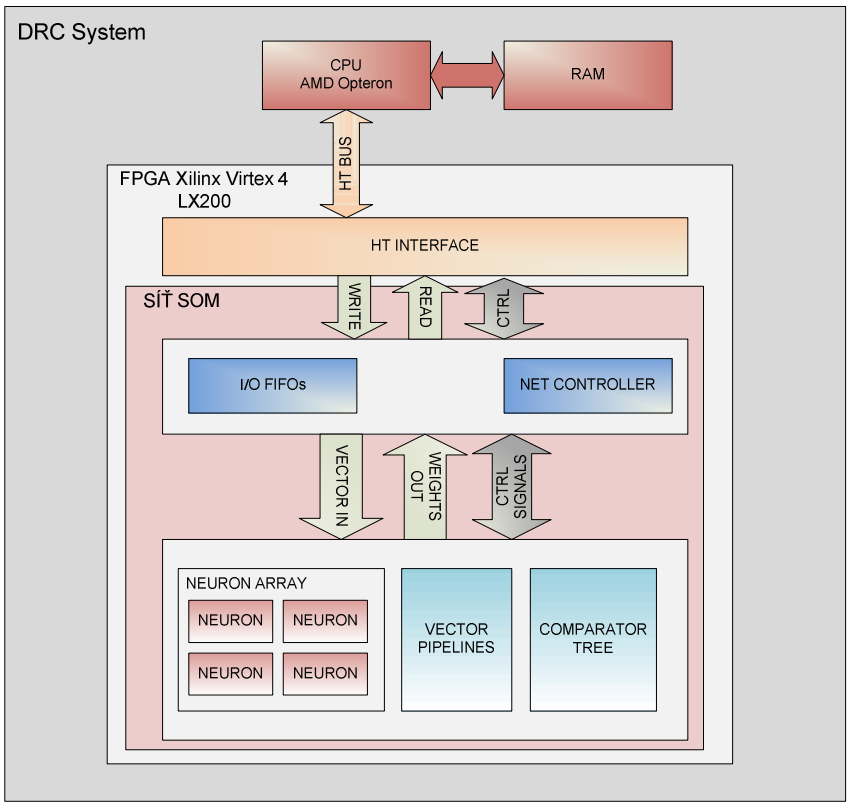
Tab. 2 Parametry systému DRC (Převzato z [5])

směr HT→user logic	
ht_wr_addr(47:3)	adresa pro zápis dat do uživ. Designu
ht_wr_data(63:0)	data do uživ. Designu
ht_wr_mask(7:0)	maska platnosti bajtu v 64b slově
ht_wr_req	log. 1 značí platnost adresy a dat a požadavek zápisu od HT
ht_wr_wait	log. 1 značí nepřipravenost uživ.designu pro příjem dat-čekací cyklus
směr user logic→HT	
ht_rd_addr(47:3)	adresa pro čtení dat z uživ. Designu
ht_rd_data(63:0)	data od uživ. Designu
ht_rd_req	log. 1 značí platnost adresy a dat a požadavek čtení od HT
ht_rd_wait	log. 1 značí nepřipravenost uživ. designu pro odeslání dat-čekací cyklus
ht_rd_data_val	platná data od uživ. designu
společné signály	
usr_clk	Hodiny
usr_rst	Reset

Tab. 3 Signály řadiče HyperTransport

4 Návrh řešení

V rámci návrhu řešení implementace sítě SOM na FPGA byl vytvořen koncept pro návrh hardwarové verze a současně i softwarového modelu v jazyce Java - tento model slouží pro verifikaci výsledků získaných ze simulace FPGA verze. Návrh hardware vychází z požadavku dosažení co nejvyšší frekvence obvodu a současně využití pokud možno co nejmenšího počtu prostředků FPGA, aby se na čip vešlo dostatečné množství neuronů. K implementaci aproximovaných aritmetických funkcí je použit zvláštní typ obvodů – tzv. shift-add aritmetika, která umožní výpočet funkcí jako mocnina, násobení, dělení za pomocí snadno realizovatelných a rychlých obvodů typu posuv (barrel shifter) a sčítačky/odčítačky. Design využívá velmi intenzivně pipelining pro dosažení vyšší frekvence. Na Obr. 12 je znázorněno zjednodušené blokové schéma hardwaru neuronové sítě v prvním přiblížení.



Obr. 12 Blokové schéma sítě v systému DRC

Neuronová síť je umístěna v obvodu Xilinx Virtex 4 LX200, který je součástí základní desky systému DRC. Toto FPGA je napojeno na sběrnici HyperTransport přes kterou komunikuje s procesorem AMD Opteron, na kterém běží driver zajišťující komunikaci se sítí. Rozhraní zpřístupňující sběrnici HT (*HT interface* neboli řadič HT) je dodáváno k základní desce jako netlist, tudíž ho není třeba implementovat. Nevýhodou tohoto řešení je samozřejmě nemožnost zasáhnout do samotného řadiče.

Neuronová síť se musí starat o komunikaci s řadičem HT, tzn. přijímat a odesílat data dle stanoveného protokolu a provádět interní výpočty. Lze ji rozdělit na výpočetní část a část řadiče.

Část s řadičem obsahuje vstupně výstupní paměti FIFO, adresové dekodéry, multiplexory pro směrování dat mezi řadičem HT a frontami a stavovými/řídícími registry. Dále obsahuje hlavní stavový automat řídící přechody mezi jednotlivými fázemi výpočtu. Samotná řídící funkce je distribuovaná mezi tento stavový automat a struktury uvnitř datové části - nelze tedy zavést nějaké striktní prostorové dělení řadiče.

Výpočetní část obsahuje pole neuronů, strom komparátorů pro hledání neuronu s minimem vzdálenosti od vstupního vektoru, pipeline pro uchování vektorů pro pozdější využití a mnoho pomocných struktur pro synchronizaci a řízení pipeline.

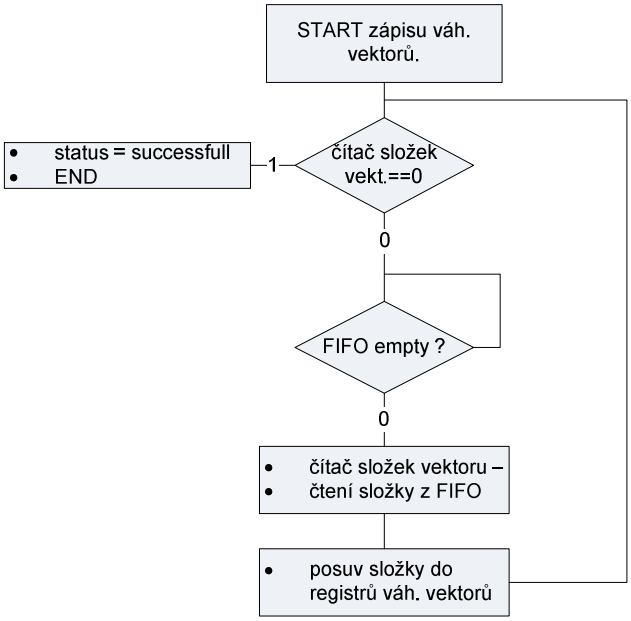
4.1 Výběr algoritmu učení

Vzhledem k existenci více algoritmů učení, viz kap. 2.1, jsem v první fázi musel vybrat jeden konkrétní. Online algoritmus je výhodný tím, že učení může být pozastaveno kdykoliv během učící epochy a přesto získáme nové váhové vektory. Z jeho principu ale plyne, že po každém vektoru je nutné změnit váhy a ty znovu využít při práci s dalším vektorem, což v důsledku vede k datovým závislostem. Datové závislosti představují zásadní problém při návrhu hardware, protože vyžadují pozastavení činnosti pipeline a různé druhy forwardingu dat (předávání dat zpět z vyšší části pipline od nižší). Závislosti každopádně vedou k vyšší složitosti návrhu a zpomalení činnosti.

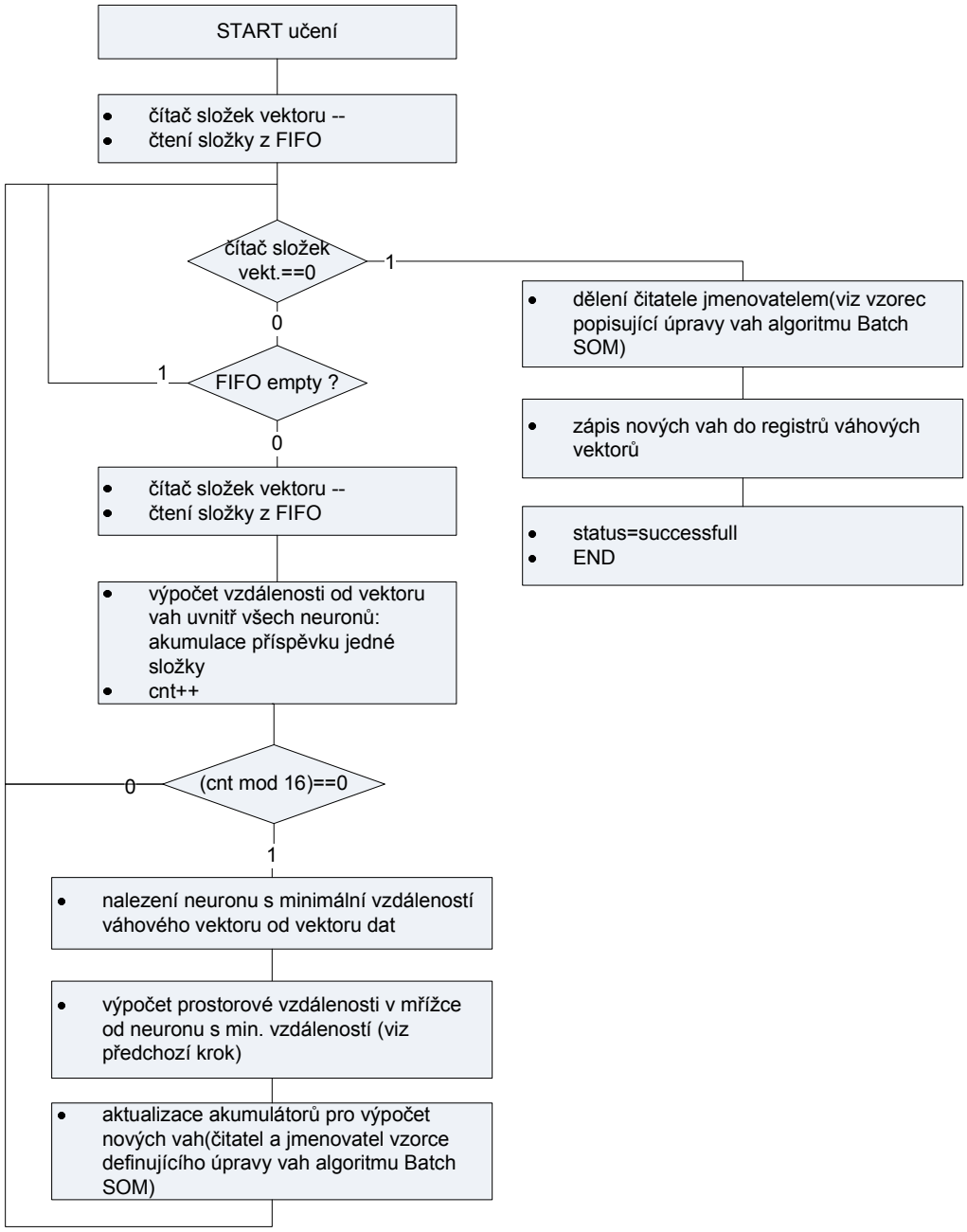
Z výše uvedených důvodů jsem se rozhodl implementovat algoritmus ve variantě *Batch SOM*, který eliminuje datové závislosti.

4.2 Obecný postup výpočtu

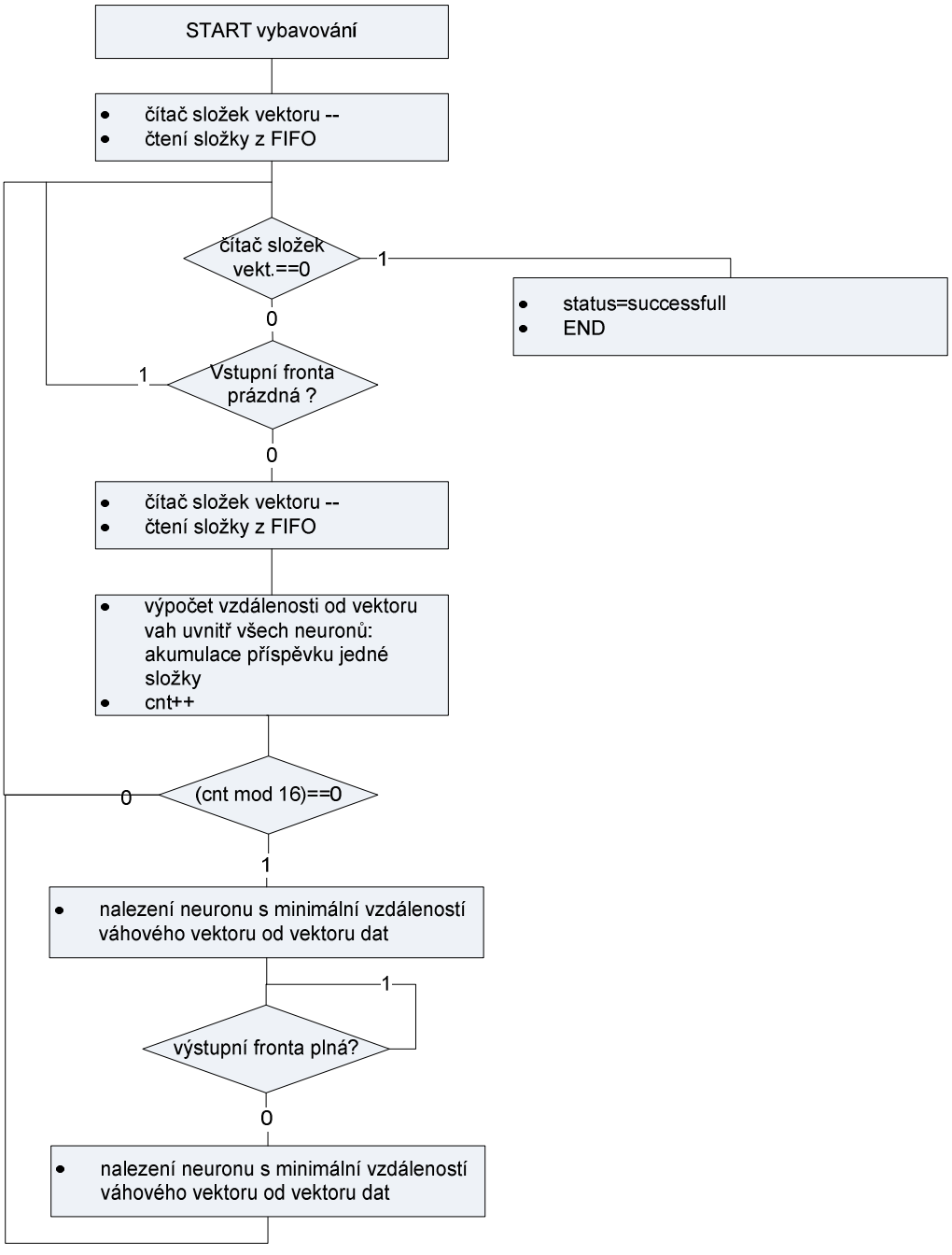
Vývojové diagramy na Obr. 13, Obr. 14, Obr. 15 postupně ukazují obecný princip činnosti sítě během nahrávání vah, učení a vybavování. Z těchto úvah vycházel další návrh sítě.



Obr. 13 Vývojový diagram nahrávání váhových vektorů



Obr. 14 Vývojový diagram učení sítě



Obr. 15 Vývojový diagram vybavování

4.3 Softwarový model sítě v jazyce Java

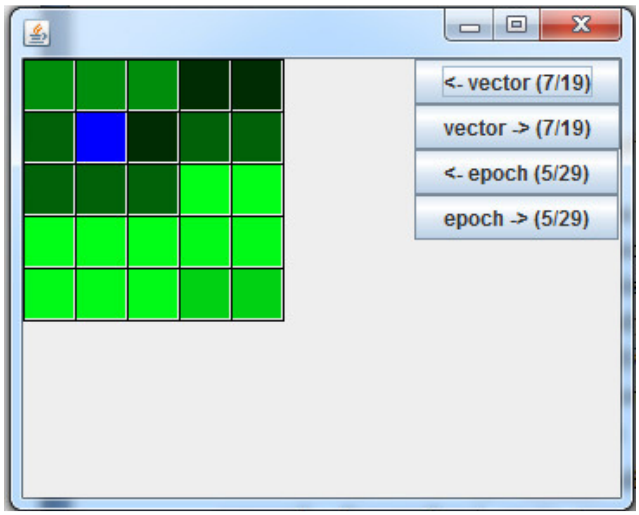
V rámci návrhu sítě jsem vytvořil softwarový model sítě v jazyce Java pro ověření funkčnosti sítě za požití aproximovaných funkcí. Jedná se o aplikaci simulující výpočet v budoucím hardware. Program byl vytvořen s ohledem na co nejvěrnější podobnost funkcí s předpokládanou hardwarovou realizací.

Model je schopen načíst data v hexadecimálním formátu ze souboru a provést inicializaci váhových vektorů a následně učení sadou vektorů (epocha). Slouží pouze k ověření realizovatelnosti sítě a ověření chyb aproximovaných funkcí. Je identický pouze po aritmetické stránce, neobsahuje stejný řadič ani příkazovou sadu jako hardwarové realizace. Implementovány jsou i neaproximované funkce. Přepnutí je dáno proměnnou ve zdrojovém kódu.

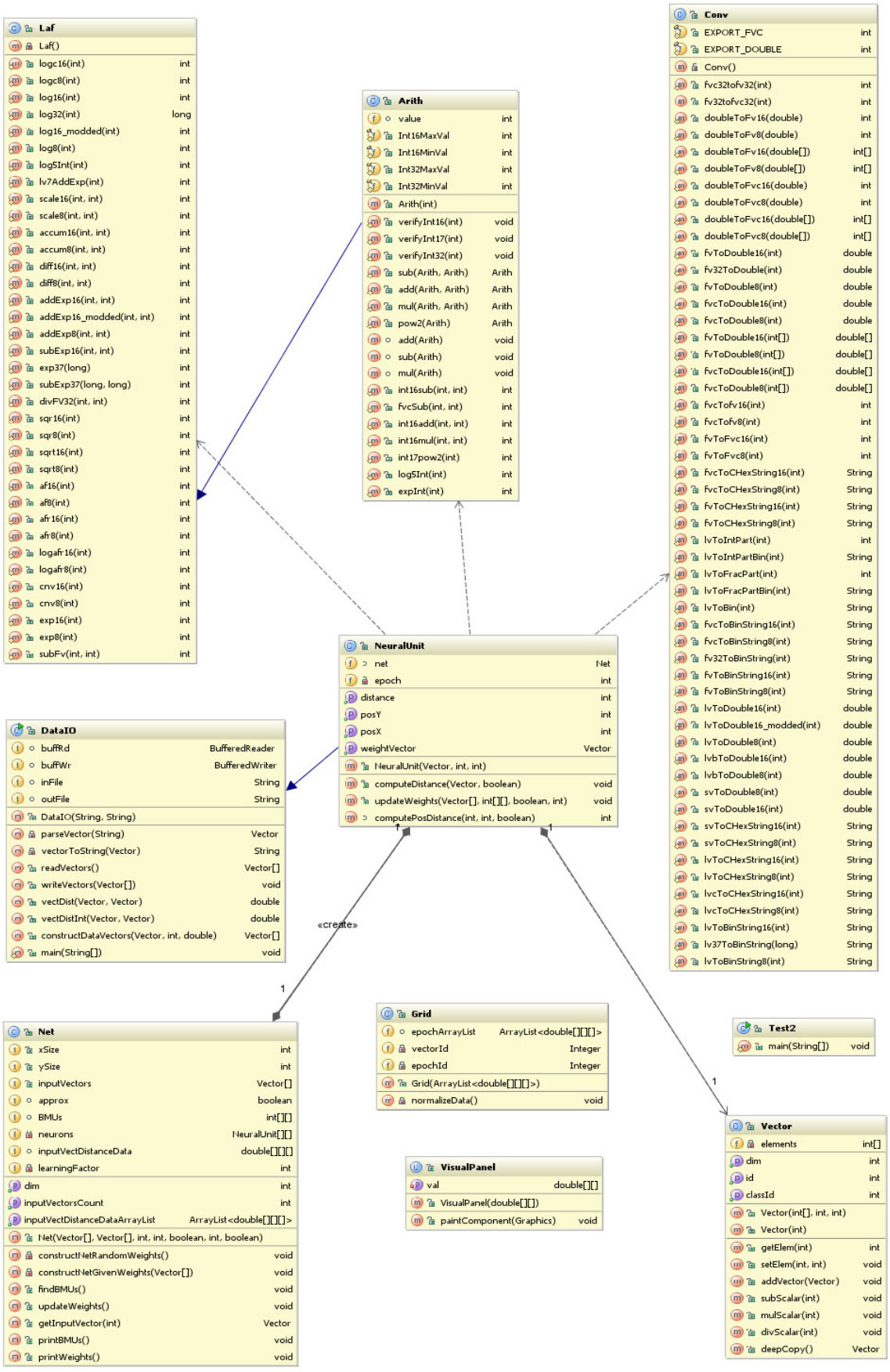
Výstupem programu je konzolový výpis, ze kterého lze zjistit průběh hodnot váhových vektorů v průběhu učení. Současně také vypisuje do konzole procentuální chyby aproximovaného dělení, které je bezesporu nejsložitější aproximovanou funkcí sítě.

Ukázka grafického rozhraní je na Obr. 16. Mapa ukazuje vzdálenost vektoru od váhových vektorů neuronů v síti. Každá buňka odpovídá jednomu neuronu. Čím je zelená barva sytější, tím je větší vzdálenost. Modrá buňka zobrazuje BMU. Je možno listovat náhledy mezi neurony i mezi epochami. Barvy jsou normalizovány pro každý obrázek zvlášť, není tedy přesně definován vztah mezi dvěma stejnými odstíny ve dvou různých náhledech.

Diagram tříd vzhledem k neuronové jednotce je uveden na obrázku Obr. 17.



Obr. 16 Java SOM rozhraní



Obr. 17 Diagram tříd Java SOM

4.4 Sdílená část neuronové sítě

Zjednodušené blokové schéma neuronového pole je na Obr. 18. Neuronové pole se skládá ze sady neuronů. Počet neuronů se zadává pomocí dvou konstant při překladu sítě. Důraz je kladen na možnost syntézy sítě libovolné velikosti (za předpokladu dostatku prostředků na FPGA). Neexistuje tedy žádné omezení na rozměry X a Y (nemusí to být mocniny dvou ani sudá čísla). Každý neuron nese svou identifikaci, která je tvořena dvojicí X, Y, což představuje geometrickou pozici v souřadném systému, tyto konstanty jsou automaticky generovány při překladu zdrojových kódů.

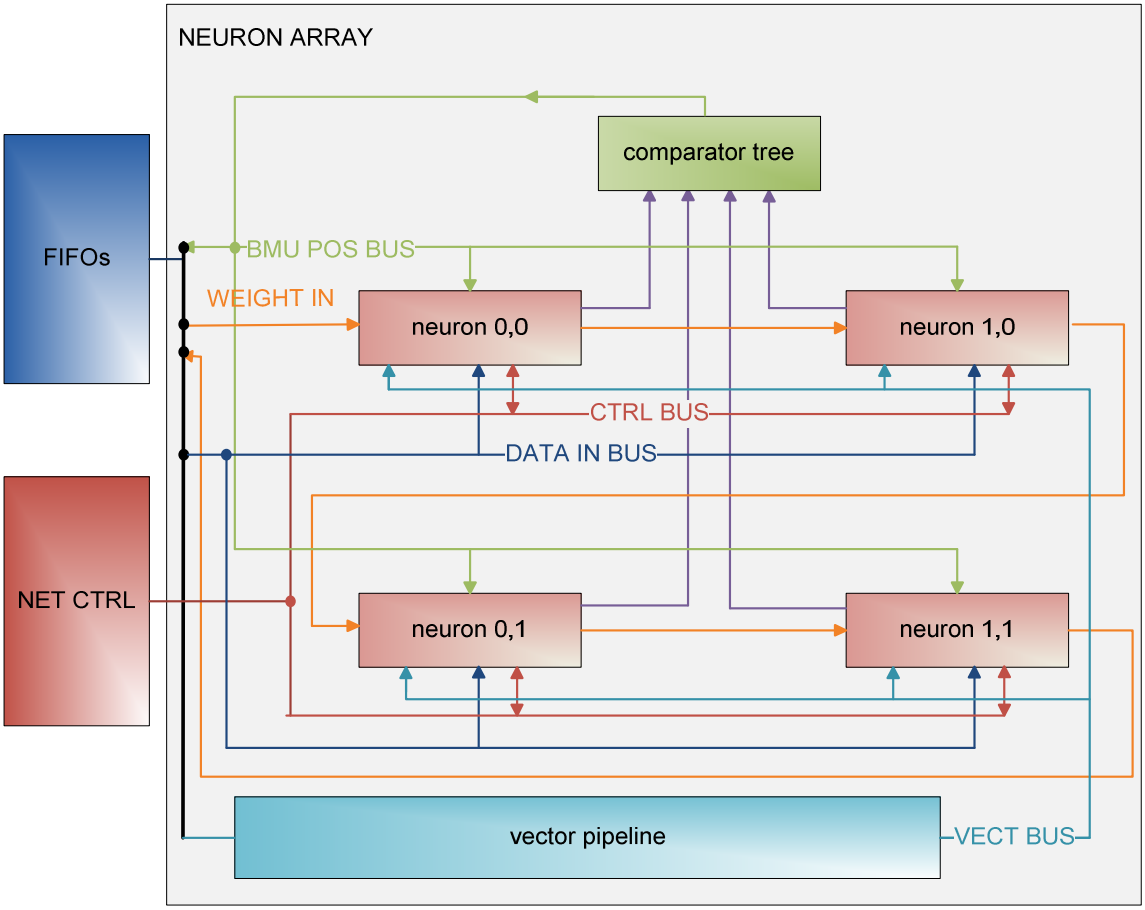
Existují zde tři typy sběrnic-sdílené paralelní sběrnice, které dodávají data směrem do neuronů, dvoubodové spoje typu bod-bod (přenos dat směrem z neuronů) a dále je zde „zřetězená" sběrnice tvořená posuvnými registry, která propojuje sousední neurony a slouží k nahrávání a vyčítání vah (pojem sousední zde neznamená sousední na základě euklidovské vzdálenosti!). Sběrnice prochází neurony v řádcích, nejdříve po dimenzi X a pak přeskok na další hodnotu Y (řádková organizace).

- 1. Sdílené paralelní sběrnice
  - Vstupní datová sběrnice
    - Přenos elementů vektorů do neuronů.
  - Řídící sběrnice
    - Řídící signály pro neurony.
  - Sběrnice pro pozice BMU
    - Přenáší pozice X, Y vítězného neuronu ze stromu komparátorů.
- 2. Dvoubodové spoje
  - Přenáší informaci o vzdálenosti vstupního vektoru od váhového vektoru daného neuronu a současně pozici v síti. Dvoubodové spoje končí ve stromu komparátorů, který slouží k nalezení neuronu s nejmenší vzdáleností.
- 3. Zřetězená sběrnice váhových vektorů
  - Slouží k transferu váhových vektorů mezi neurony a jejich inicializaci nebo vyčtení. Zřetězená sběrnice váhových vektorů pracuje na podobném principu jako např. sběrnice SPI. Jedná se o zřetězení multifunkčních registrů. Pojmem multifunkční je použit, protože mohou pracovat v režimu rotačního nebo posuvného registru. V případě posuvného registru dochází k postupnému nasouvání vah přes všechny neurony (nebo případně v rámci jednoho neuronu při výpočtu nových vah), v režimu rotace se elementy váhového vektoru točí individuálně v rámci neuronu.

Blok *vector pipeline (VP)* slouží ke zpoždění původního nezměněného vstupního vektoru. Uvnitř každého neuronu je pipeline a na konci této pipeline je nutné pro výpočet využít původní vstupní vektor. Pipeline v této situaci dodává složky vstupního vektoru synchronně s potřebami neuronové jednotky. Protože vektor je pro všechny neurony stejný, bude blok umístěn do sdílené oblasti, aby se nemusel zbytečně redundantně opakovat v každém neuronu, čímž dojde k významnému ušetření registrů vzhledem k tomu, že každý vektor má celkem 16x16 bitů a v pipeline se v jeden okamžik nachází více než jeden celý vektor (jeden vektor+hloubka stromu komparátorů+zpoždění jednotky pro výpočet vzdálenosti v 2D mřížce).

Mezi sdílené součásti patří také bloky sloužící k řízení činnosti pipeline registrů, tyto části jsou okomentovány v rámci kapitoly 5.

.

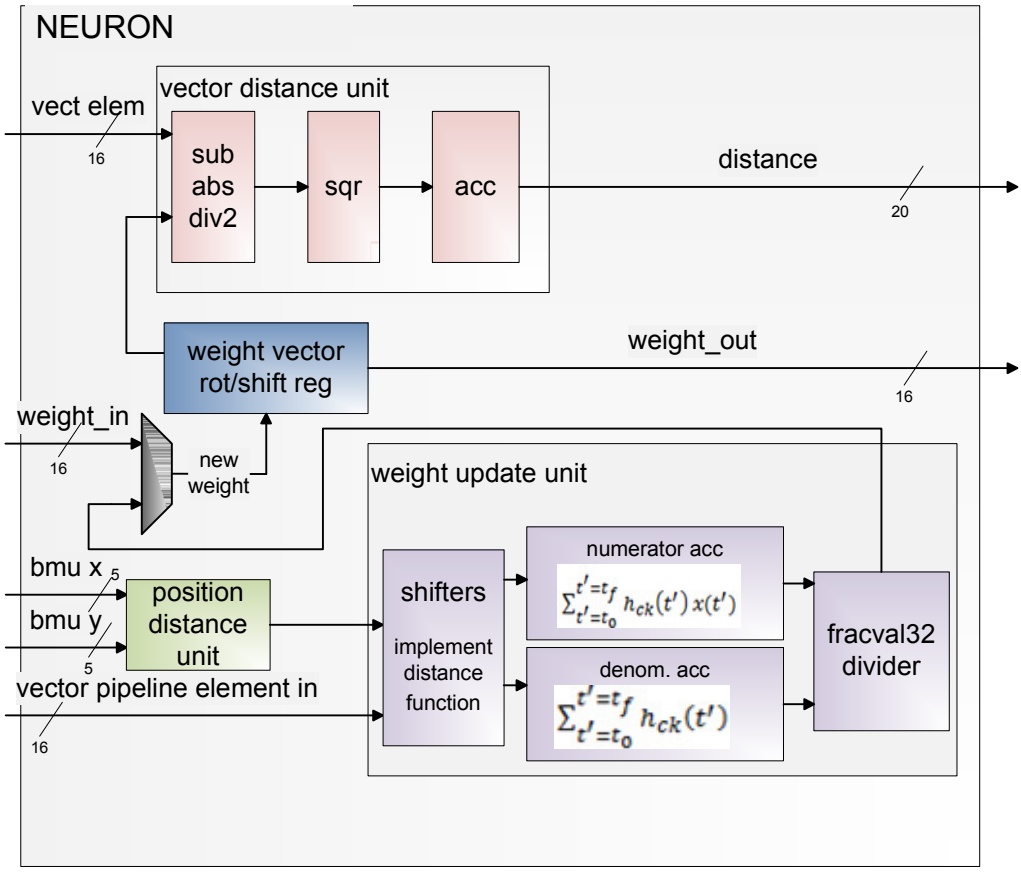


Obr. 18 Neuronové pole (příklad pro síť 2x2)

4.5 Neuronová jednotka (neuron)

Neuronová jednotka je blokem, kde se provádí většina numerických výpočtů neuronové sítě. Jedinou další jednotkou, kde dochází ke zpracování mezivýsledků je strom komparátorů (*comparator tree*) , kde se vyhodnocuje minimum vzdálenosti od vstupních vektorů (vzdálenost je jedním z výsledků výpočtů neuronu). Kromě aritmetických jednotek obsahuje neuron registry váhového vektoru.

Neuron zajišťuje výpočet vzdálenosti vstupního vektoru od váhového vektoru, dále počítá Manhattanskou vzdálenost od BMU (kdo je BMU určuje strom komparátorů) a provádí výpočet nových vah viz vzorec (5). V rámci aritmetických jednotek jsou využity aproximované funkce zmíněné v kapitole 2.3. Blokové schéma neuronu je ukázáno na Obr. 19. Každá jednotka je pipelinovaná (v některých případech jsou pipeline registry i mezi jednotkami, to je však implementační záležitost), aby se dosáhlo dostatečné pracovní frekvence. Pipeline registry nejsou v blokovém obrázku naznačené.



Obr. 19 Blokové schéma neuronu

Vstupní vektory do neuronové jednotky jsou ve formátu fracval v doplňkovém kódu. Tyto vektory přicházejí z fronty FIFO umístěné ve sdílených částech sítě. Fronta nedodává data v každém taktu ale jen když data opravdu obsahuje. Protože nelze zajistit, aby nadřazený systém byl rychlejší, než samotná neuronová sí,ť je nutné hlídat, zda FIFO obsahuje data a jen v kladném případě povolit činnost obvodů neuronové jednotky (konkrétně pipeline registrů). Princip řízení bude uveden v kapitole 5.

4.5.1 Registry neuronu

V každém neuronu musí výt přítomen registr váhového vektoru. Jedná se o rotační registr. Tento registr uchovává aktuální hodnoty vektoru vah. Pracuje v režimech:

- Posuv doprava s načtením nových hodnot.
  - Používá se při zavádění nových vah nebo úpravě vah během učení.
- Rotace doprava
  - Využívá se při učení a vyhodnocování vektorů. Konkrétně při výše uvedené operaci výpočtu vzdálenosti vstupního vektoru od váhového vektoru je nutné dodávat jednotlivé složky vah na vstup jednotky, která vzdálenost počítá.

Kromě váhových registrů jsou v neuronu i pomocné registry umístěné v aritmetických jednotkách.

4.5.2 Aritmetické jednotky neuronu

Všechny aritmetické operace vyjímaje sčítání a odčítání jsou realizované aproximovanými funkcemi z kapitoly 2.3.

Vzdálenost vstupního vektoru od váhového vektoru je počítaná v obvodu označovaném jako *vector distance unit (VDU)*. Tato jednotka je navržena k realizaci vzorce (1). Na její vstupy se přivádí pro každý vektor šestnáct dvojic složek vektorů. Jedna položka dvojice patří vstupnímu vektoru a druhá patří internímu váhovému vektoru. Jednotka obsahuje celočíselnou odčítačku s převodem na absolutní hodnotu a za ní je připojen posuv napravo implementující dělení dvěma. Z rozdílu vznikne mezivýsledek v intervalu (0; 2), který se dělením dvěma opět dostane do intervalu odpovídajícímu nezápornému rozsahu fracval (0; 1). Po dělení následuje funkce aproximovaná mocnina dvěma (sqr) a za ní akumulátor. Vstup i výstup funkce sqr má stejnou šířku a to 16 bitů. Je to dané tím, že se mocní čísla formátu fracval, jejichž hodnota je menší než celé číslo 1, tudíž dochází umocněním ke zmenšení absolutní hodnoty, nikoliv ke zvětšení. Zachováním stejné šířky nemůže dojít k velké odchylce od správného výsledku, dojde jen k zanedbání výsledků menších než jednotka mřížky fracval16. Umocněné hodnoty fracval16 jsou následně střádány v akumulátoru. Po nastřádání všech šestnácti hodnot (každé dimenze vektoru odpovídá jeden sčítanec) je celkový součet označen za platný a využívá se ve stromě komparátorů. Platnost hodnoty je zajištěna řetězcem posuvných registrů, ve kterých se bude posouvat bit s hodnotou log. 1. Tento bit je generován čítačem, který po šestnáctinásobné inkrementaci těchto jedniček vypustí na jeden takt bit log. 1 do posuvných registrů pipeliny.

Jednotka *position distance unit (PDU)* provádí výpočet vzdálenosti v mřížce mezi BMU a vektorem, ve kterém je umístěna. Pozici svého neuronu získává ze dvou generických konstant (x,y), které se automaticky nastaví při překladu neuronové sítě. Vzdálenost se počítá Manhattanskou metrikou, dle vzorce

$$d = |x - x_{BMU}| + |y - y_{BMU}| \tag{17}$$

Jedná se tedy o dvě celočíselné 5 bitové odčítačky s převodem na absolutní hodnotu následované sčítačkou. Tyto bloky jsou samozřejmě odděleny pipeline registry. Výsledek je 6 bitové číslo vedené přes pipeline registry do bloku pro výpočet nových vah.

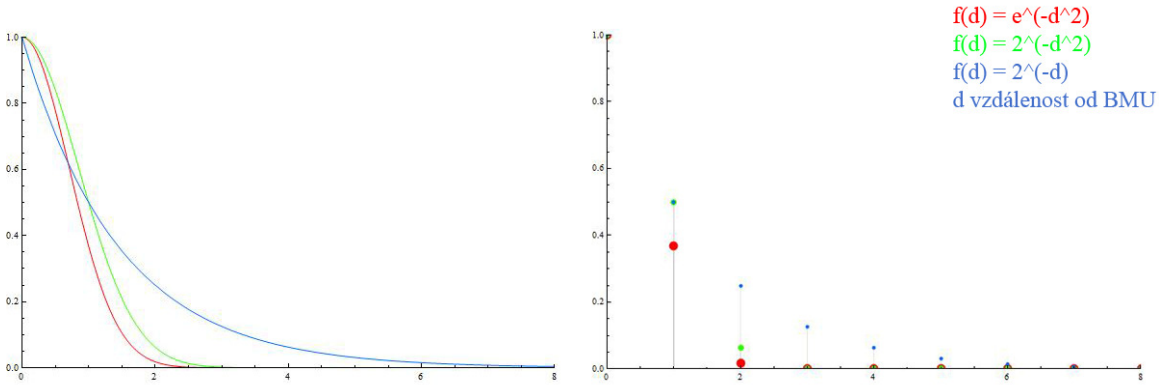
Blok označovaný jako *weigt update unit WUU* (jednotka pro úpravu vah) je nejsložitější obvodem neuronové jednotky. Na jeho vstupech se objevuje vzdálenost od BMU a původní vstupní vektor pro který byla vzdálenost vypočítána. Tento vektor v nezměněné podobě lze získat z jednotky nazvané *vector pipeline(VP). VP* je řetězec posuvných registrů o šířce jedné složky vektoru a jeho hloubka musí být precizně spočítána, což je netriviální problém. Vzorec pro hloubku vypadá přibližně takto:

$$vectDim + compTreeDepth + posDistUnitDepth \tag{18}$$

kde *vectDim* je dimenze vektoru (což je vždy 16), *compTreeDepth* je hloubka stromu komparátorů (dvojkový logaritmus počtu neuronů) a *posDistUnitDepth* je hloubka pipeline v oblasti jednotky PDU (tzn. v oblasti mezi stromem komparátorů a vstupem jednotky WUU). Vraťme se ale zpět k jednotce pro úpravu vah. Jednotka dostane z vector *pipeline* postupně původní hodnoty složek vektoru a vzdálenost neuronu od BMU náležející tomuto vektoru. V této situaci má jednotku veškerá data potřebná pro výpočet vzorce (5). Výraz představuje v podstatě vážený průměr. Čitatel představuje akumulaci složek vektoru a mírou akumulace je vzdálenost od BMU. Pro implementaci čitatele je tedy nutný rotační registr o 16 složkách (složka je 16 bitové číslo) tvořící dohromady celý konstruovaný váhový vektor. Pro realizaci jmenovatele postačí normální 16 bitový registr.

Nyní je již třeba zmínit se o realizaci funkce okolí. Parametrem funkce je vzdálenost od BMU. Funkce je monotónní a klesající a zajišťuje útlum vlivu BMU na vzdálenější neurony. Funkce způsobuje změnu váhy pro daný vektor ve váženém průměru. Většinou se využívá varianta Gaussovy funkce $e^{-x^2/\sigma}$, kde σ se mění v čase (přesněji řečeno mezi epochami v případě algoritmu učení batch SOM) tak, aby rozsah okolí postupně klesal. Gaussova funkce je poměrně složitě implementovatelná v hardwaru, tudíž jsem byl nucen najít velmi zjednodušenou náhradu Gaussovy fce, která by vyžadovala málo HW prostředků. V grafech uvedených na Obr. 20 jsou vyobrazeny varianty funkcí, které by šlo použít. Předpokladem snadné implementace je užití mocniny o základu 2, proto jsem rovnou zavrhl původní Gaussovu funkci. Jako nejsnazší a nejrychlejší vychází aplikace funkce 2^{-d}, kde d bude vzdálenost. Tato funkce má dle Obr. 20 o něco méně strmý sestup. Funkce se uplatní při sčítání hodnot do čitatele, kdy se provede operace $element\ vektoru * 2^{-distance}$, což se transformuje v obvod konající operaci $element\ vektoru \gg distance$. Akumulátor čitatele sčítá tyto transformované hodnoty. Akumulátor jmenovatele sčítá hodnoty samotné funkce vzdálenosti tzn. hodnoty $1 \gg distance$. Typ fracval hodnotu 1 nedokáže zobrazit, místo této hodnoty se využívá nejvyšší platná kladná hodnota čísla fracval *0x7FFF* =~ *0.9999389648*.

Hodnota *distance* uvedená v předchozích úvahách je běžně získávána z PDU na základě vzorce 12. Tato hodnota je ještě upravena na základě hodnoty registru nazvaného *učící koeficient* (*learning factor - LF*). Tento koeficient slouží jako argument funkce $distance \ll learning\ factor$. Učící koeficient tedy slouží ke zvětšení vzdálenosti a rozsáhlějšímu utlumení vlivu BMU. Koeficient lze měnit zápisem do registru a jeho hodnotu lze měnit před započetím epochy učení.



Obr. 20 Variante funkce okolí

Oba akumulátory mají shodnou datovou šířku 32 bit (datový typ fracval32 v doplňkovém kódu), což znamená, že síť může provést epochu o počtu max. 2^{16} vektorů (což se rovná počtu akumulací čísla fracval16 bez rizika přetečení akumulátorů). Platí, že čitatel má vždy menší hodnotu než jmenovatel.

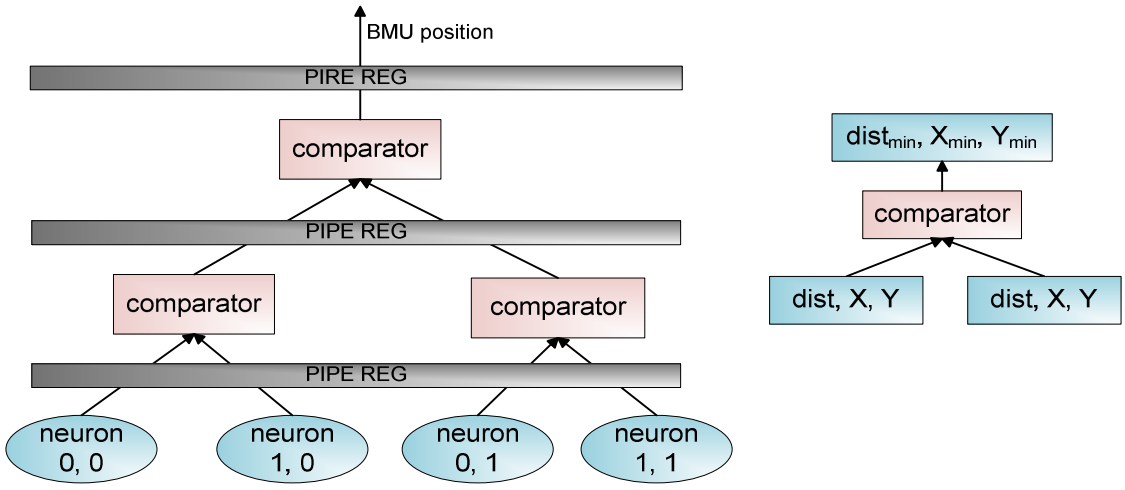
Na konci epochy dochází k dělení čitatele jmenovatelem. K této operaci slouží dělička hodnot fracval32. Dělení probíhá pro 16 složek vektoru, protože elementů vektoru v čitateli je 16. Epocha končí po zapsání všech podílů do váhového registru.

4.6 Strom komparátorů

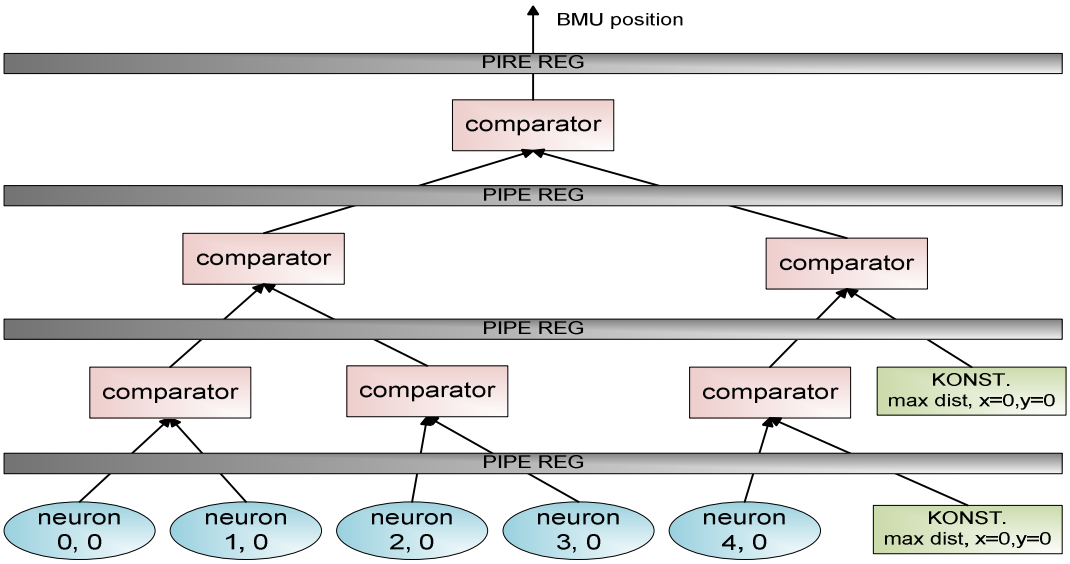
Při výpočtu sítě je nutno zjišťovat neuron s nejmenší vzdáleností od vstupního vektoru (vzdáleností je myšlena vzdálenost mezi vstupním vektorem a vektorem vah dle vzorce (1)). Takový neuron se nazývá BMU a neurony v jeho blízkosti akumulují pro změnu vah vstupní vektor ve váženém průměru s nejvyšším vlivem viz vzorec (5). Vzdálenosti jsou kladné hodnoty, proto je lze snadno porovnat komparátorem. Schéma stromu komparátorů je uvedeno na Obr. 21. Komparátor v uvedeném stromě je obvod, který porovná dvě hodnoty a vrací menší z nich, současně obsahuje multiplexor, který vrátí pozici menšího neuronu, který obsahuje menší hodnotu. Platí priorita zleva, tzn. pokud existují dva neurony se stejnou hodnotou vzdálenosti a rozdílnou pozicí, vrací se pozice levého podstromu. Jednotlivá patra stromu jsou oddělena pipeline registry, hloubka je dvojkovým logaritmem počtu listů, každý list může, nicméně nemusí být reprezentován fyzickým neuronem, v určitých případech je nahrazen konstantami vzdálenosti a pozice.

Rozměry sítě mohou být zcela libovolné, z toho vyplývá, že počet neuronů a tedy listů stromu nemusí být násobkem dvou. Komparátor je obvod, který porovnává vždy dvě hodnoty. Pokud je tedy počet neuronů lichý, jsou některé podstromy nahrazeny konstantou, která generuje maximální vzdálenost a pozici 0,0. Díky přednosti zleva bude vždy propagována jen levá validní hodnota příslušející fyzickému neuronu, nikoliv smyšlené maximální konstantě. Obr. 22 ukazuje příklad stromu pro lichý počet neuronů. V případě sudého počtu, který ale není mocninou dvou je situace obdobná. Například pro 6 neuronů bude nejhlubší vrstva komparátorů obsahovat 3 komparátory a o patro vyšší bude obsahovat dva, přičemž druhý bude zprava doplněn konstantou. Liší se tedy pouze v místech, kde bude konstanta dosazena.

Strom bude generován během syntézy, nebude třeba zasahovat do VHDL kódu.



Obr. 21 Strom komparátorů, vlevo základní varianta stromu, vpravo komparátor



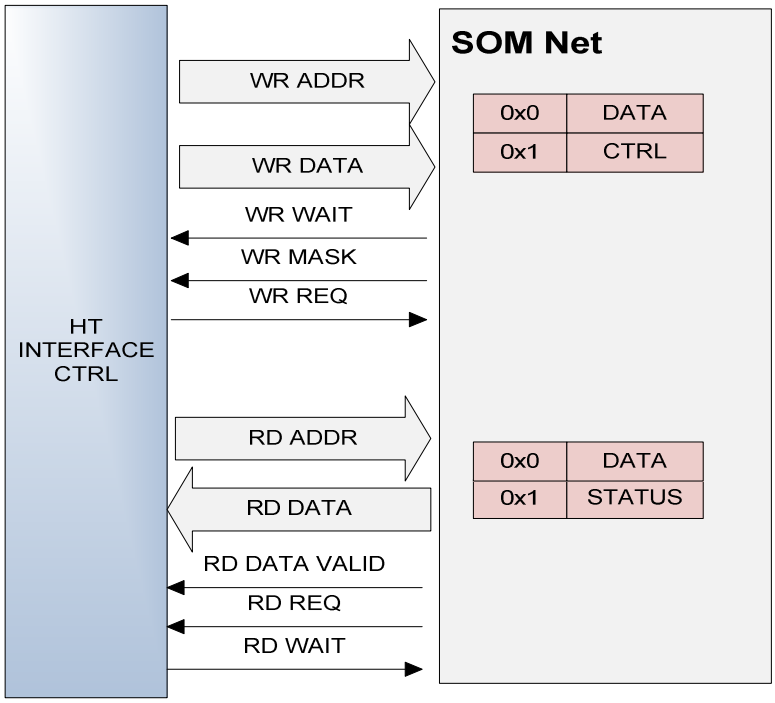
Obr. 22 Strom komparátorů pro lichý počet neuronů, ukázka pro 5 neuronů

4.7 Vstup a výstup sítě

Komunikace sítě s nadřazeným procesorem AMD Opteron probíhá přes sběrnici HyperTransport. Ovládání sběrnice a převod dat na jednoduše zpracovatelná paralelní data a adresy provádí v rámci FPGA komponenta dodávaná v podobě netlistu. Signály, které tato komponenta poskytuje, jsou uvedeny v Tab. 3. Datová šířka přenosu je 32 nebo 64 bitů a bity platnosti určují, které bajty jsou platné – podobný systém využívá například u sběrnice PCI. Zvolil jsem možnost využívat pouze 64b přenos s platností všech bajtů. Vstup a výstup vektorů probíhá přes fronty typu FIFO. Z důvodu úspory prostředků v rámci jednotlivých neuronů jsem se rozhodl, že interně budu zpracovávat jen jednu dimenzi naráz => vstupní datová šířka neuronu bude 16 bitů. Převod z 64b do sekvence 4 x 16b a naopak zajišťuje asymetrické vstupní a výstupní fronty FIFO (rozdílná šíře datového vstupu a výstupu komponenty FIFO).

4.7.1 Registry

Pro řízení sítě jsou určeny řídící a stavové registry na pevných adresách. Pro zápis a čtení dat je definován pomyslný datový registr na adrese 0. Rozhraní zobrazuje Obr. 23 a Tab. 4.



Obr. 23 Rozhraní se sběrnicí HT

ADRESA	SMĚR	NÁZEV	POPIS
0	čtení/zápis	datový registr (data reg.)	čtení zápis dat
1	zápis	řídící registr (control reg.)	zadávání příkazů sítě
1	čtení	stavový registr (status reg.)	info o stavu sítě
2	zápis	registr učícího koeficientu (learning factor reg.)	nastavení faktoru útlumu funkce okolí pro danou epochu

Tab. 4 Vstupně-výstupní registry

Datový registr (Data register) je 64bitový registr, který slouží ke čtení a zápisu vektorů. Vektor má velikost šestnácti 16 bitových komponent vektoru. Pro zápis jednoho vektoru je nutno provést 4 zápisy na adresu datového registru. Zápisy se provádí v pořadí od nejnižší čtveřice dimenzí po nejvyšší čtveřici. Tab. 5 popisuje pořadí zápisu dimenzí (a jejich rozmístěná ve slově) vstupního vektoru do datového registru. Každý řádek představuje zápis jednoho 64b slova, celá tabulka pak představuje zápis jednoho vektoru.

pořadí zápisu	(63:48)	(47:32)	(31:16)	(15:0)
1.	dim1	dim2	dim3	dim4
2.	dim5	dim6	dim7	dim8
3.	dim9	dim10	dim11	dim12
4.	dim13	dim14	dim15	dim16

Tab. 5 Pořadí zápisu dimenzí vektoru do datového registru

Při inicializaci sítě nahráváním počátečních váhových vektorů je nutné dodržet pořadí vektorů. První zapsaný vektor se dostává do posledního neuron [X-1; Y-1] a poslední zapsaný vektor se dostává do neuronu [0; 0]. Toto chování je dané sériovou konstrukcí registrů váhových vektorů.

Při klasifikaci neznámých vektorů je nutné vyčítat BMU příslušné zadaným vektorům. Výstup dat registru je 64 bitové slovo. Počet klasifikovaných vektorů musí být násobkem 4, v případě menšího počtu je nutno doplnit počet vektorů na násobek 4 libovolnými daty a BMU pro tyto data pak ignorovat. Výstupní formát ukazuje Tab. 6. Souřadnice BMU_X a BMU_Y mají šířku 5 bitů, jsou doplněny 3 nulami zleva, kompletní identifikace je tedy tvořena dvojicí BMU X a Y a tvoří 16 bitů. V každém 64 bitovém slově se přenesou 4 pozice BMU. Tabulka ukazuje pouze nižších 32b. V horní polovině registru se situace opakuje.

63:48	31:24	23:16	15:0	7:0
…..	000&BMU_X	000&BMU_Y	000&BMU_X	000&BMU_Y

Tab. 6 Datový registr - čtení BMU při klasifikaci

Registr učícího koeficientu (Learning factor register) je 64b registr, který svou hodnotou určuje útlum funkce okolí v dané učící epoše neuronové sítě. Povolená hodnota je v rozsahu 0-4, stačí tedy jen 3 spodní bity. Vyšší musí být nulové.

Řídící registr (Control register) je 64b registr, kde spodní bity reprezentují příkazy pro síť a vyšších 32b udává počet dat, které budou sítí zpracovány. Příkazy jsou v kódu 1 z N, tzn. vždy musí být maximálně jeden bit aktivní. V opačném případě stejně dojde k provedení jen jednoho příkazu na základě priority dané způsobem HW implementace. Rozdělení registru ukazuje Tab. 7.

63:32	31:8	7	6	5	4	3	2	1	0
data cnt	-	classify	-	learn	-	wload	wread	-	reset

Význam položky *data cnt* v závislosti na příkazu:

Příkaz	Hodnota „data cnt“
wload (nahrání vah)	počet váhových vektoru*16 – 1
wread (čtení vah)	počet váhových vektorů*16 – 1
learn (učení)	počet vstupních vektorů*16
Classify	počet vstupních vektorů*16

Tab. 7 Řídící registr

Stavový registr (Status register) je 64b registrem, který je pouze pro čtení, zápis není definován. Svou hodnotou signalizuje stav sítě. Pokud je síť v klidovém stavu (*idle*) nebo ve stavu nedokončené činnosti (*busy*), nemá jeho čtení žádný vedlejší efekt. Pokud se síť nachází ve stavu ukončené operace (*successfull*), způsobí čtení registru jeho výmaz na hodnotu idle. Driver, který bude síť shora obsluhovat, tedy tímto vyčte stav a v případě dokončené činnosti síť automaticky přejde do stavu *idle*. Tab. 8uvádí přiřazení hodnot stavového registru stavům sítě. Stavy budou popsány podrobněji kapitole 4.8

hodnota	význam
0x0001	IDLE (READY)
0x0010	BUSY
0x0100	SUCCESSFULL

Tab. 8 Význam hodnot stavového registru

4.8 Stavy sítě

Činnost neuronové sítě SOM je rozdělena do několika fází, které jsou určovány řadičem sítě na základě příkazu zaslaného do řídícího registru. Základní fází, ve které se síť nachází po resetu obvodu je stav *idle.* V tomto stavu je síť připravena k přijetí příkazu.

Další stavy sítě jsou *weight_load, weight_read, learn_st0, learn_pipe_flush, learn_update_weights, classify, classify_pipe_flush, fin.* Tyto stavy nejsou přímo viditelné ze stavového registru, jedná se o vnitřní stavy. Stavový registr informuje jen o tom, zda právě probíhá operace nebo je dokončena, nebo je síť v klidovém idle stavu. Bližší informace není pro vyšší SW vrstvu nutná. Příkaz se vždy skládá z příkazového bitu v dolní polovině řídícího registru a počtu dat, která má příkaz zpracovat v horní polovině řídícího registru. Význam počtu dat uvádí Tab. 7 v kapitole 4.7.1.

Prvním příkazem po resetu sítě by mělo být nahrání počátečních vah do neuronů. Do stavu nahrávání vah(weight load) se síť dostává zapsáním příkazu do řídícího registru. Váhové rotační registry neuronů jsou ve stavu nahrávání nových vah přepnuty do režimu posuvu a sada váhových registrů všech neuronů tvoří řetězec. Princip je podobný, jako u sběrnic typu SPI, váhy nahráváme

sériovým způsobem po jednotlivých složkách vektoru (složka=fracval16 v doplňkovém kódu). Kolik vah je nutno zapsat se určí zápisem čísla do horní poloviny řídícího registru. Po nahrání daného počtu vah se síť dostává do stavu *fin(final).* V tomto stavu je nutno přečíst stavový registr, ze kterého obdržíme hodnotu *successfull* a síť přechází automaticky do stavu *idle*. Stavem *fin* končí všechny režimy sítě.

Po nahrání vah je typickým postupem učení (stav *learn*). Před tímto stavem je nutné v případě potřeby zapsat *learning_factor* registr Po přechodu do tohoto stavu následuje nahrání zadaného počtu vektorů do vstupní FIFO(přesněji řečeno zadaného počtu elementů vektoru, což se provede zápisem do horních 32 bitů řídícího registru současně s bitem příkazu v nižších 32bitech). Po dokončení nahrávání zůstává chvíli síť ve stavu busy, dokud nezpracuje veškeré vektory. Stav zjistíme čtením stavového registru. Pokud je jeho hodnota *busy,* nesmí driver zahájit žádnou další operaci zapsáním do řídícího registru. Pokud se objeví hodnota *successfull* a tato je vyčtena, dojde k přesunu do stavu *idle* a driver současně získává informaci, že byla dokončena fáze učení zadané sady vektorů.

Dalším krokem může být opakování učení, nebo vyčtení vah do výstupní fronty. Pro příkaz vyčtení vah platí podobná pravidla jako pro zápis s tím rozdílem, že po skončení (stav *successfull*) musí ovladač z výstupní fronty přečíst zadaný počet váhových elementů.

Po naučení sítě je možno provést klasifikace neznámých vektorů. Ta se provede opět pomocí příkazu řídícího registru, zadáním počtu elementů klasifikovaných vektorů a provedením daného počtu zápisů do datového registru. Následně musí být driverem vyčítán status registr do doby, než se objeví hodnota *successfull.* Poté je možno vyčíst z výstupní fronty daný počet odpovědí (každému zaslanému datovému vektoru odpovídá jedno BMU).

5 Implementace

Kapitola implementace se zabývá detailnějším popisem konkrétní implementace sítě SOM v obvodu FPGA jazykem VHDL. Bude zde uvedeno rozdělení na komponenty a u vybraných složitějších bloků bude i schéma. Velmi podstatnou částí sítě je řadič a detaily týkající se pipeline, obě tyto záležitosti jsou úzce svázány se způsobem realizace, proto budou zmíněny také až v této kapitole.

5.1 Komunikační protokol síť - řadič HT

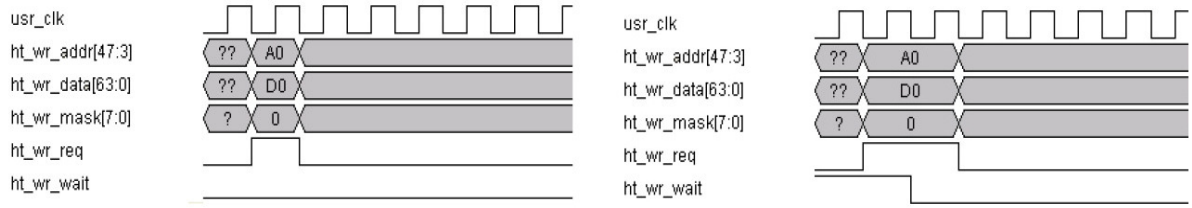
Kapitola se zabývá popisem komunikace neuronové sítě s řadičem sběrnice HT. Především zde budou uvedeny průběhy signálů, které jsou definovány komponentou řadiče dodávaného formou netlistu. Tyto průběhy je nutné velmi exaktně dodržet, jsou pevně dané a nelze je nijak uživatelsky ovlivnit.

Řadič HT umožňuje 32 i 64 bitové přenosy. Kapitola popisuje pouze 64 bitový přenos. 32 bitový přenos není popisován, neboť nebyl využit vzhledem ke své nižší efektivitě.

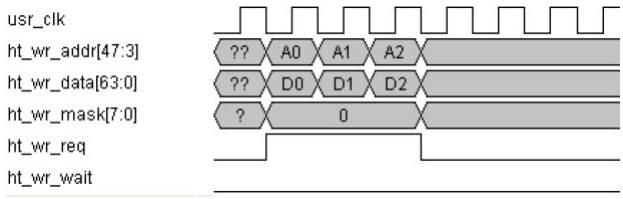
Přenos je synchronní, uživatelský hardware by měl vzorkovat řídící signály na náběžných hranách, řadič HT mění hodnoty signálů na sestupných hranách hodin. Přenosy dělíme striktně podle směru přenosu vzhledem k neuronové síti na zapisovací (HT→SOM) a čtecí (SOM→HT).

Síti je umožněno vkládat během zápisu čekací takty pomocí *wait* signálu. Obr. 24 popisuje průběhy signálů při 64 b zápisu do neuronové sítě, vlevo bez čekacích taktů, vpravo s čekáním po dobu jednoho taktu hodinového cyklu. V případě vložení čekacího taktu je držen signál *ht_wr_req* v aktivní úrovní a data ani adresa se nemění, čeká se na uvolnění signálu *ht_wr_wait*. Jakmile je tento uvolněn, je změna zaregistrována řadičem HT a signál požadavku čtení je deaktivován. Adresa a data se stávají neplatnými. Obrázek ukazuje, že signál *ht_wr_wait* mění síť na sestupné hraně, to však není bezpodmínečně nutné a správnost funkce to neovlivní, je tedy možné měnit signál i na náběžné hraně hodin.

Existuje ještě jeden režim zápisu do sítě, který se nazývá *burst* režim. Jedná se o blokový přenos většího množství dat. Režim se liší od předchozích v možnosti ponechat signál *ht_wr_req* v aktivní úrovni po více hodinových cyklů a na každé náběžné hraně hodin přenést 1 slovo. Přenos je opět možno ze strany sítě pozastavit aktivováním signálů *ht_wr_wait.* Burst režim je tedy z HW pohledu v podstatě variantou klasického čtení jednoho slova několikrát zřetězeného za sebe v čase.



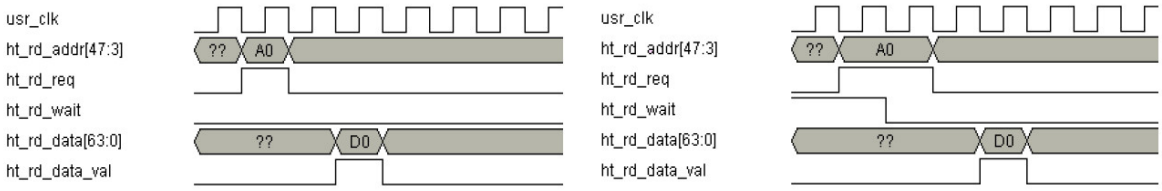
Obr. 24 Zápis do uživatelského HW, směr HT → neuronová síť SOM (Převzato z [5])



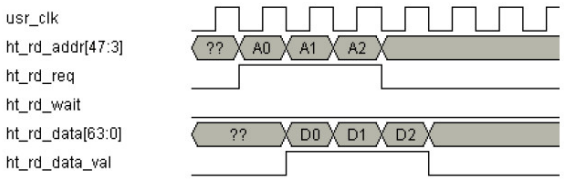
Obr. 25 Burst zápis do uživatel. HW, směr HT → neuronová síť SOM (Převzato z [5])

V případě čtení je situace obdobná. Nejprve řadič požádá o čtení. Na adresovou a datovou sběrnici vystaví adresu, ze které požaduje data. Síť SOM může vystavit signál *ht_rd_wait* čímž vloží čekací cykly. Pokud tato situace nenastane, následující takt řadič HT deaktivuje signál *ht_rd_req* a adresa se stane neplatnou. Následně SOM odpovídá platnými daty, což signalizuje aktivní úrovní portu *ht_rd_data_val.* Zpoždění vystavení platných dat po přijatém požadavku se může lišit. Obr. 26 ukazuje situaci, kdy zpoždění činí 2 hodinové takty a v pravé části obrázku je situace kdy navíc síť vystavuje čekací signál.

Podobně jako v případě zápisu i zde existuje *burst* režim viz Obr. 27, kdy se přenáší sled více slov za sebou.



Obr. 26 Čtení z uživatelského HW, směr neuronová síť SOM → HT (Převzato z [5])

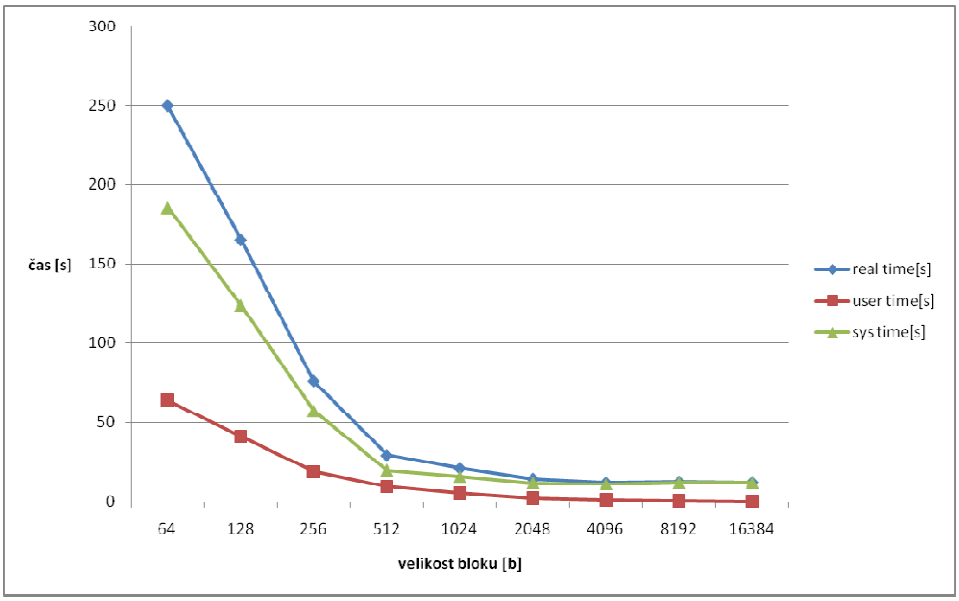


Obr. 27 Burst čtení z uživatel. HW, směr neuronová síť SOM → HT (Převzato z [5])

Při přípravě implementace jsem změřil rychlost přenosu dat mezi sítí a driverem běžícím na procesoru systému DRC. Výsledky mého měření přináší tabulka Tab. 9 a Obr. 28. Měření jsem provedl pro různé velikosti bloku dat přenášeného na jedno zavolání funkce driveru *send*. Z grafu je vidět větší efektivita při přenosu větších bloků.

velikost bloku	real time[s]	user time[s]	sys time[s]
64	249,543	63,836	185,736
128	165,173	41,271	123,92
256	76,164	18,857	57,312
512	29,336	9,565	19,777
1024	21,032	5,304	15,729
2048	14,465	2,16	11,853
4096	12,292	1,272	11,021
8192	12,672	0,736	11,937
16384	12,412	0,308	12,105

Tab. 9 Přenos 1GB dat s proměnnou velikosti přenášeného bloku



Obr. 28 Přenos 1GB dat s proměnnou velikosti přenášeného bloku

5.2 Neuronová jednotka (neuron)

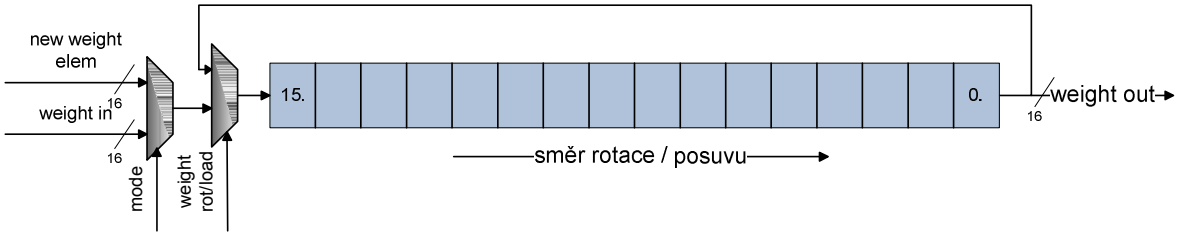
Neuronová jednotka neboli neuron, se skládá z několika komponent (viz Obr. 19), jejichž vnitřní implementace je popsána v této kapitole. Některé moduly a signály jsou pojmenovány anglickými názvy, které souvisí s názvy použitými ve zdrojových kódech. Na obrázcích ukazujících jednotlivé jednotky (komponenty) budou někdy vyobrazeny názvy VHDL entit/komponent kurzivou a obdélníkem v nazelenalé barvě.

Všechny aritmetické jednotky jsou navrženy s ohledem na pipelinovaný návrh, kdy jsou delší kombinační cesty rozděleny na více úseků. Po zkrácení hloubky kombinační logiky je možné použít kratší periody hodin a dosáhnout vyšší hodinové frekvence celého obvodu. Můj návrh počítá s jednou hodinovou doménou návrhu a tudíž je hodinová perioda tak dlouhá, jak vyžaduje nejdelší souvislá část kombinační logiky včetně zpoždění registrů oddělující tento úsek (setup time, hold time…). Pipeline registry mají symbol modrého obdélníku s nápisem „R“.

5.2.1 Váhové registry

Každý neuron obsahuje interní váhový vektor. Vektor je sada šestnácti fracval16 hodnot v doplňkovém kódu. Tyto hodnoty se využívají při učení i klasifikaci jako vstup do jednotky VDU. Jednotlivé složky mají pevně dané pořadí a je potřeba je vystavovat po jedné. Současně je nutné mít možnost vektor inicializovat počátečními hodnotami případně tyto hodnoty přepisovat. Vstup nových hodnot do neuronu je buď přímo z vyhrazeného vstupního kanálu, a nebo z interní jednotky WUU. Jako ideální prvek pro implementaci takové datové struktury se nabízí rotační registr, který je navržen tak, aby umožňoval i posuv. Obvod je vyobrazen na

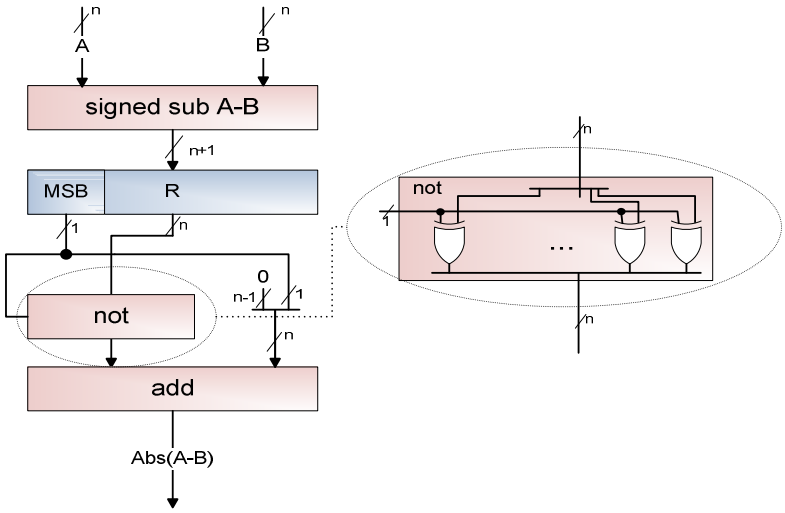
Obr. 29. Každá buňka váhového rotačního registru představuje jedno číslo typu fracval16 v doplňkovém kódu. Signál *mode* slouží k výběru vstupu, signál weight load/rot vybírá režim činnosti. Oba signály jsou řízeny řadičem. Výstup registru (signál weight out) je propojen s následujícím neuronem (následujícím v uspořádání řádek, sloupec) a současně je veden jako vstup do jednotky VDU. Obvod je implementován generickou (konfigurovatelnou) entitou *rotate_reg*.



Obr. 29 Váhový rotační registr

5.2.2 Odčítačky s absolutní hodnotou

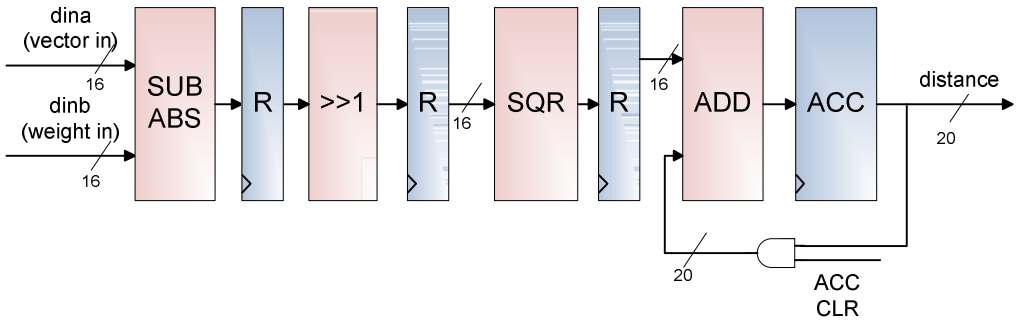
V mnoha obvodech je využita odčítačka kombinovaná s výpočtem absolutní hodnoty rozdílu. Blokové schéma obecného obvodu je na Obr. 30. Obvod je realizován ve dvou variantách představovaných entitami *sub5_unsig_abs* (použita v jednotce PDU) a v jed VDU v rámci entity *fvc_sub_abs_div_2*.



Obr. 30 Odčítačka s absolutní hodnotou

5.2.3 Vector distance unit

Jednotka *vector distance unit (VDU)* viz Obr. 31, je blok, který počítá vzdálenost interního váhového registru od vstupního datového vektoru. Funkce jednotky je popsána vzorcem (1). V rámci zdrojového kódu je entita nazvána *neural_unit_distance_module*. Jedním vstupem je signál z fronty FIFO umístěné ve sdílené části sítě, druhým je výstup z již zmíněného váhového rotačního registru. Nejprve se provede rozdíl těchto dvou vstupů typu fracval16 v doplňkovém kódu (složka vstupního vektoru a složka váhového vektoru). Každá hodnota fracval16 nabývá v desetinných číslech hodnot (-1; 1), rozdíl tedy může být v intervalu (-2; 2). Z rozdílu je vypočtena absolutní hodnota, která je z intervalu <0; 2). Tento kladný výsledek je posléze vydělen dvěma (posuv vpravo o 1 pozici), vznikne tak hodnota z intervalu <0; 1) což je opět fracval. Tento fracval je umocněn v bloku *SQR* a veden do akumulátoru. Akumulátor má šíři 20bitů, což je dostatečné pro posčítání 16 dimenzí vektoru bez rizika přetečení. Mezi každými dvěma jednotkami je pipeline registr. Jedná se o proudově pracující jednotku, která podává výsledky až po naplnění všech částí pipeline. Efektivního využití lze docílit jen v případě, že vektory budou dodávány neustále a nebude nutné opakovaně vyprazdňovat a plnit pipeline. Po každých 16 elementech jednoho vektoru následuje hned první dimenze dalšího vektoru, platný součet tedy vznikne každých 16 taktů. V 16. taktu se deaktivuje signál *acc_clr* a součinové hradlo způsobí přerušení zpětné vazby akumulátoru. To způsobí, že v 17. taktu se objeví v akumulačním registru první dimenze nového vektoru. Tímto způsobem je zajištěno smazání předchozího výsledku a současně načtení nových dat. Signál mazání řídí čítač modulo 16, který při hodnotě 0 způsobuje zmíněné mazání akumulátoru. Tento čítač je umístěn ve sdílené části sítě, protože všechny neurony pracují synchronně a je zcela redundantní používat v každém neuronu stejný samostatný čítač. Modul pro výpočet rozdílu včetně dělení dvěma je v entitě *fvc_sub_abs_div_2*.

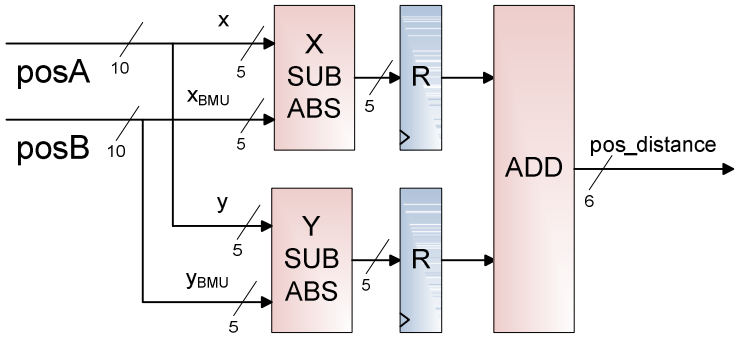


Obr. 31 Vector distance unit

5.2.4 Position distance unit

Jednotka, nazvaná *position distance unit (PDU)* slouží k výpočtu vzdálenosti mezi neurony v dvojdimenzionální mřížce a její zdrojový kód je reprezentován entitou *neural_unit_pos_distance_module*. Tento výpočet je nutný ve fázi učení pro výpočet funkce vzdálenosti ve vzorci (5) a stejný úkol má i ve fázi vybavování. Každý neuron má pozici definovanou dvěmi hodnotami, jedna pro dimenzi X (řádky), druhá představuje dimenzi Y (sloupce). Každá dimenze používá 5bitové celé kladné číslo pro zakódování polohy. Vzdálenost počítaná touto jednotkou se nazývá Manhattanská a je definována vzorcem

$$d = |x - x_{BMU}| + |y - y_{BMU}| \tag{19}$$



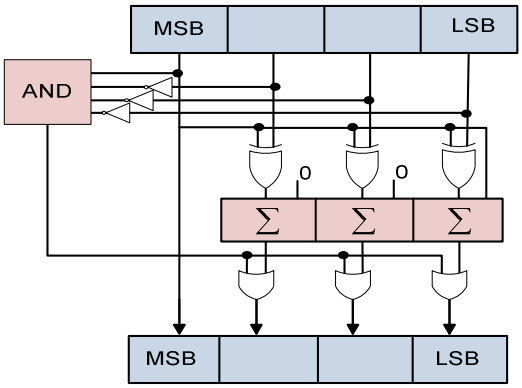
Obr. 32 Position distance unit

5.2.5 Konverzní obvody datových typů

Obvody slouží pro konverzi mezi hodnotami *fracval* v doplňkovém kódu a přímém kódu. Číslo v doplňkovém kódu může nabývat hodnot z intervalu $< -\frac{Z}{2}; \frac{Z}{2})$, zatímco číslo v přímém kódu se pohybuje v symetrickém intervalu $(-\frac{Z}{2}; \frac{Z}{2})$. Konstanta Z představuje v obou případech modul řádové mřížky.

Obvody jsou implementovány pro převod z doplňkového kódu do přímého i opačně. Při převodu z přímého kódu je situace jednoduchá, na základě nejvyššího bitu se provede negace všech bitů v oblasti řádové mřížky obsahující absolutní hodnotu (všechny bity kromě nejvyššího). Za negací následuje přičtení tzv. horké jedničky. Celý obvod je možné realizovat pomocí hradel xor pro negaci a jedné sčítačky.

V případě převodu z doplňkového kódu je situace komplikována nesymetrií intervalu hodnot doplňkového kódu. Je vhodné ošetřit situaci, kdy číslo nabývá hodnoty – Z/2. V tomto případě se nejedná o platnou hodnotu čísla v přímém kódu. Pokud bychom aplikovali předchozí postup, dojde v tomto případě k velké chybě. Z čísla – Z/2 se stane číslo 0, což představuje chybu půlky rozsahu čísla. Řešením je detekce této situace. Pokud má číslo v doplňkovém kódu na pozici nejvyššího bitu hodnotu log. 1 a ostatní bity jsou nulové, musí být výsledkem převodu číslo v přímém kódu s minimální (maximální zápornou) hodnotou. Takové číslo má ve všech bitech jedničky. Schéma obvodu realizující převod je na Obr. 33. Obvody jsou ve zdrojovém kódu realizovány entitami *fv_to_fvc*, *fvc32_to_fv32*.



Obr. 33 Převod doplňkový kód → přímý kód

5.2.6 Weight update unit

Jednotka *weight_update_unit (WUU)* slouží pro výpočet nových váhových vektorů. Entita implementující tento obvod má název *weight_update_module.* Jedná se o nejsložitější jednotku neuronu. Základními vstupy obvodu jsou vstupní vektor po složkách, vzdálenost od BMU, učící koeficient. Výstupem jsou složky nového váhového vektoru. Vstupní vektor se získává z jednotky nazývané *vector pipeline*. Jedná se o nezměněné vstupní vektory dané epochy opožděné tak, aby byly na vstupu jednotky ve správný čas současně se vzdáleností od BMU pro daný vektor.

Definice batch algoritmu učení neuronové sítě SOM stanovuje úpravu váhových vektorů na základě vzorce:

$$W_k(t_f) = \frac{\sum_{t'=t_0}^{t'=t_f} h_{ck}(t')\, x(t')}{\sum_{t'=t_0}^{t'=t_f} h_{ck}(t')} \tag{20}$$

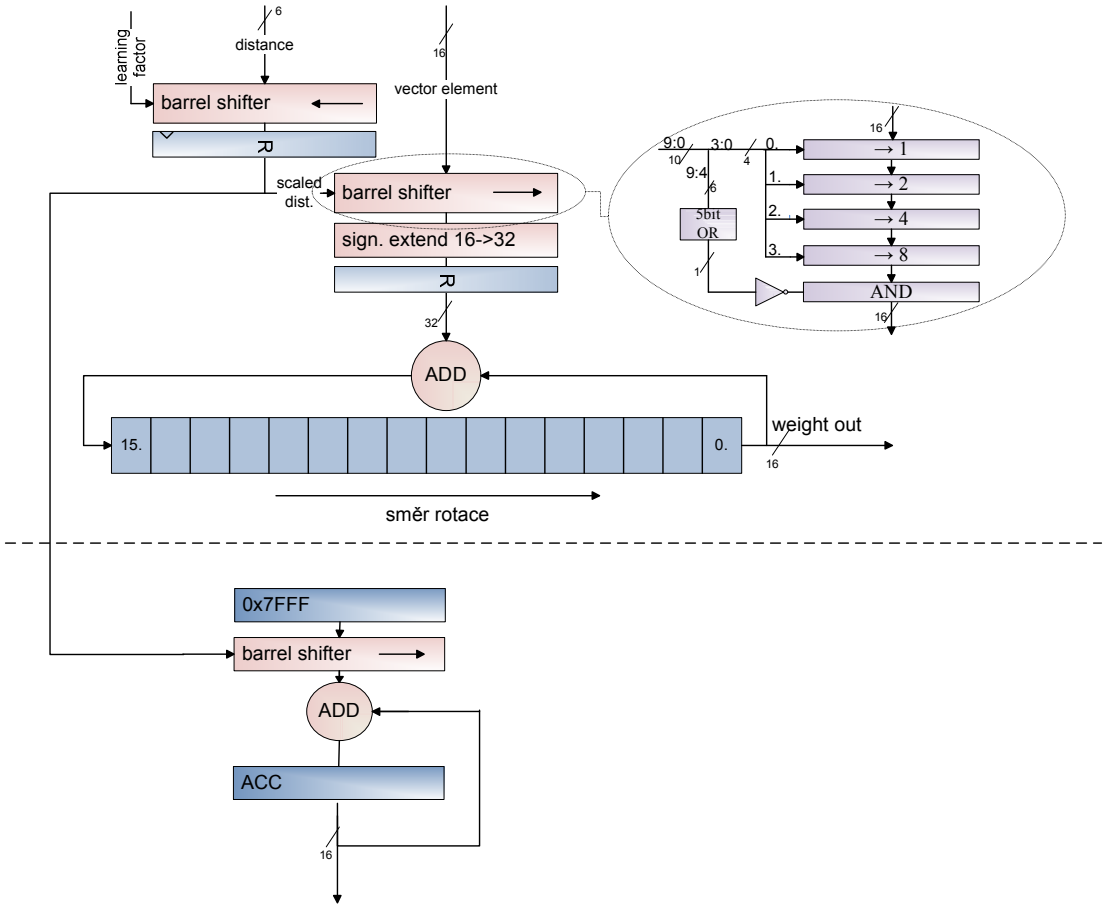
Levá strana výrazu představuje nový váhový vektor v čase t_f (konec epochy učení). Pravá strana je předpisem, jak tento vektor získat. V čitateli je suma váhového vektoru (v čase t') s funkcí okolí h_{ck}. Časy t' jsou časy příchodů jednotlivých vstupních vektorů. Z uvedeného vyplývá, že operace v čitateli je součin vektoru a skaláru, tudíž je čitatel sám o sobě vektorem, a zůstává jím v rámci všech mezivýsledků sumy. Ve jmenovateli se nachází suma hodnot funkcí okolí *h*, což představuje skalární hodnotu. Výraz jako celek představuje vážený průměr složek vektorů podle funkce vzdálenosti o BMU a je opět vektorem, ale pro jeho využití potřebujeme získat složky po jedné.

Nejprve se věnujme funkci okolí h_{ck}. Jak již bylo zmíněno v kapitole 4.5.2, je využita funkce mocnina 2^{-x}, kde x představuje vzdálenost. Situace ale není tak jednoduchá. U sítě SOM je definován učící koeficient, neboli hodnota, která mění prostorový dosah funkce okolí mezi epochami. Výraz x tedy není přímo vzdálenost braná z výstupu jednotky *PDU*, nýbrž je nejdřív upravena. Funkce okolí tedy vznikne složením dvou funkcí a lze ji definovat výrazem:

$$2^{-(distance*2^{l})} \tag{21}$$

kde l představuje učící koeficient. Funkce $distance*2^{l}$ je realizována barrel shifterem, který má jako řídící signál zmiňovaný učící koeficient a jako vstup má vzdálenost, která vstupuje do jednotky pro úpravu vah (WUU) z obvodu pro výpočet vzdálenosti (PDU). Hodnota koeficientu představuje počet posuvu doleva a tím zvětšování hodnoty. Toto zvětšení má za následek, že ve stejné vzdálenosti bude po transformaci funkcí okolí h menší vliv jednotky BMU na okolní neurony v případě většího koeficientu.

Nyní se věnujme čitateli. Jedná se o sumu v rámci dimenzí vektoru. Lze si to představit jako 16 individuálních rotačních akumulátorů. Akumulátory byly zvoleny rotační z důvodu úspor prostředků. Výpočet probíhá po složkách během 16 kroků. V případě paralelního zpracování by bylo nutno využít šestnáct 32bitových sčítaček. Čitatel je implementován rotačním registrem o 16 složkách typu fracval32. Šířka rotačních akumulátorů přímo určuje, kolik vektorů může být zpracováno v rámci jedné epochy, v tomto případě jich může být až 2^{16} bez rizika přetečení. Mezi akumulátorem nejnižší a nejvyšší dimenze je ve zpětné vazbě sčítačka, která k původní hodnotě přičítá příslušné elementy vstupního vektoru vynásobené hodnotou funkce okolí. Datové cesty realizující čitatele jsou na Obr. 34 v horní polovině (v dolní polovině je obvod realizující jmenovatele).



Obr. 34 Implementace akumulátoru jednotky WUU

V horní polovině obrázku jsou dva barrel shiftery. První upravuje vzdálenost posuvem o 0-4 pozice doleva na základě hodnoty učícího koeficientu (learning factor). Druhý barrel shifter provádí aritmetický posuv doprava elementu vstupního vektoru na základě výsledku posuvu z prvního barrel shifteru, tím provádí operaci h∗ x_i, kde h je funkce vzdálenosti, x je vektor a i je aktuálně násobená dimenze vektoru. Řídící signál barrel shifteru (scaled distance) má víc než 4 bity, takže teoreticky by barrel shifter měl mít hloubku odpovídající šířce řídícího signálu v bitech. Vzhledem k tomu, že vstupní data mají šíři pouze 16bitů, jakýkoliv posuv větší nebo roven 16 způsobí nulování daných dat, jsou ostatní úrovně nahrazeny 16 bitovým hradlem AND které v případě jakéhokoli většího posuvu než 16 způsobí nulování výstupu.

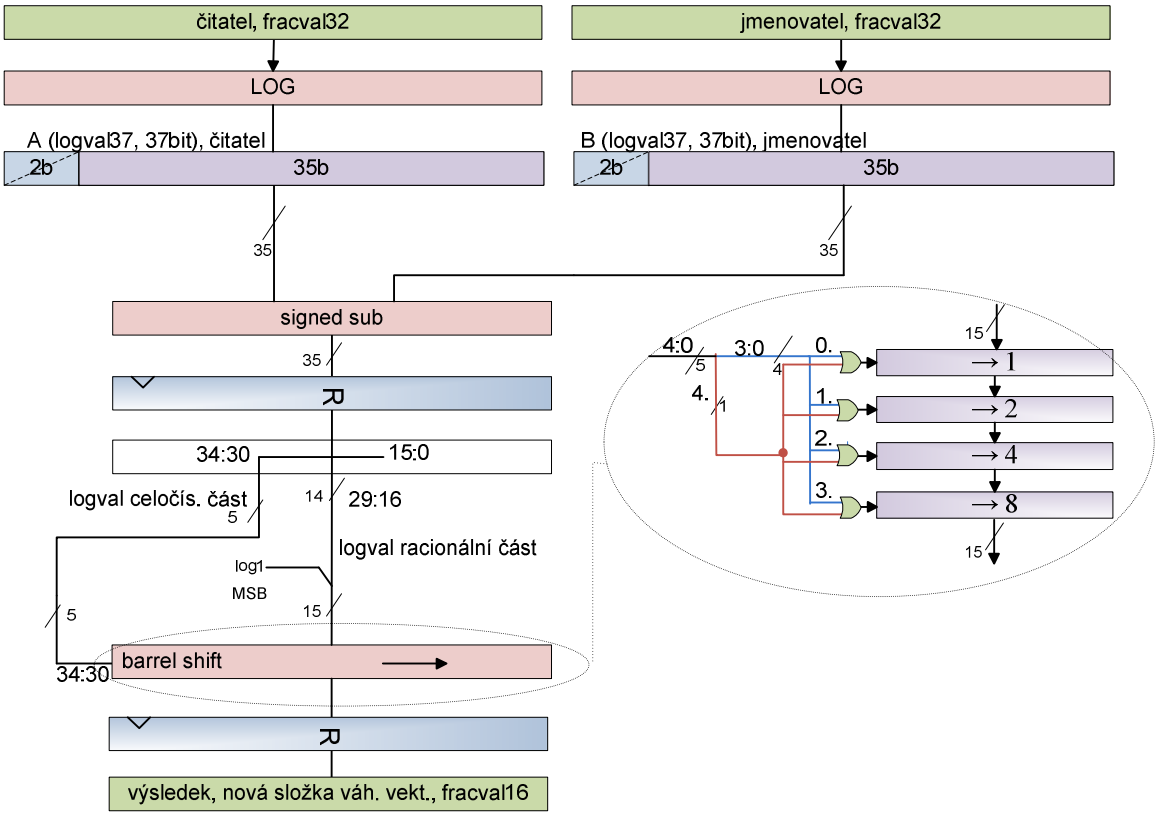
Dolní polovina Obr. 34 představuje čitatele. Obvod implementující čitatele je jednodušší, poněvadž se jedná o akumulátor skalární hodnoty (šířka 32b stejně jako čitatel). Akumulovaná hodnota představuje výsledek funkce $2^{-x} = 1 * 2^{-x}$. Tudíž se jedná o posuv čísla 1 doprava. Číslo 1 není možno v konkrétní reprezentaci fracval16 vyjádřit, proto byla nahrazena maximální platnou hodnotou fracval16, což je 0x7FFF.

Každý element vektoru v čitateli je nutné pro získání nové váhy vydělit hodnotou jmenovatele. Obě hodnoty jsou ve formátu fracval32 v doplňkovém kódu a dělička je navržena na stejnou vstupní datovou šířku. Výsledek je ale ve formátu fracval16. Dělení je založeno na platnosti výrazů

$$\log_2 (a/b) = \log_2 a - \log_2 b \tag{22}$$

$$2^{log_2(a/b)} = a/b \tag{23}$$

Oba vstupy děličky je nezbytné převést do typu fracval v přímém kódu (datový typ se kterým pracují aproximované funkce). Konverze se provede pomocí převodních obvodů, o kterých hovoří kapitola 5.2.5. Během operace je nejdříve nutné převést hodnoty na hodnoty typu logval37. Transformace je provedena logaritmátory uvedenými v kapitole 2.3.1 s tím rozdílem, že jejich datová šířka je 32 bitů místo 16bitů a jejich výstup je typ logval37 nikoliv logval20. Dvě hodnoty logval37 jsou od sebe odečteny pomocí signed odčítačky (obvod pracuje s datovými typy se znaménkem). Následně je nad výsledkem rozdílu provedena operace exponenciála $2^{x.}$ Tentokrát se ale nejedná o obvod identický s exponencielou v kapitole 2.3.2, který je určen výhradně pro operaci násobení. Obvod neobsahuje hradla XOR, která by odčítala konstantně hodnotu danou nejvyšším bitem celočíselné části hodnoty logval. Výsledkem exponenciální funkce je hledaný podíl. Chyba implementované děličky je dle měření modelovou aplikací Java SOM <11.89%.



Obr. 35 Dělička

5.3 Sílené části sítě a pipeline

Navrženou neuronovou síť lze koncepčně rozdělit na dvě základní části. Jedná se o paralelní systém se sdílenými jednotkami a samostatnými neuronovými jednotkami, které pracují paralelně a víceméně nezávisle (mimo výpočet BMU, který se provádí ve stromě komparátorů). Mezi sdílené

komponenty patří především řadič, vstupně výstupní systém (fronty) a dále sada posuvných registrů pro řízení pipeline na základě platnosti dat.

Takovýto design je umožněn tím, že neurony pracují zcela synchronně, tzn. každý zpracovává stejná data, liší se jen vnitřními „proměnnými“, a je tedy možné řídit všechny neurony společnými signály.

5.3.1 Pipeline

Základním cílem při řízení činnosti sítě je určení, kdy by pipeline měla pracovat a kdy naopak by měla být pozastavena. Jelikož je zvolen algoritmus učení, který nezpůsobuje datové závislosti, není nutné pozastavovat pipeline z důvodů datových závislostí. Je však nutné zajistit návaznost činnosti neuronů na dostupnost dat uvnitř vstupní fronty FIFO. Fronta slouží jako vyrovnávací paměť mezi sítí a driverem běžícím na straně procesoru. Není zaručeno, že data budou vždy k dispozici, je možné že síť bude data zpracovávat rychleji, než je driver stihne dodávat. Nyní postupně proberu jednotlivé základní části řízení pipeline. Zjednodušené blokové schéma výpočetních jednotek a sdílené pipeline je na Obr. 36.

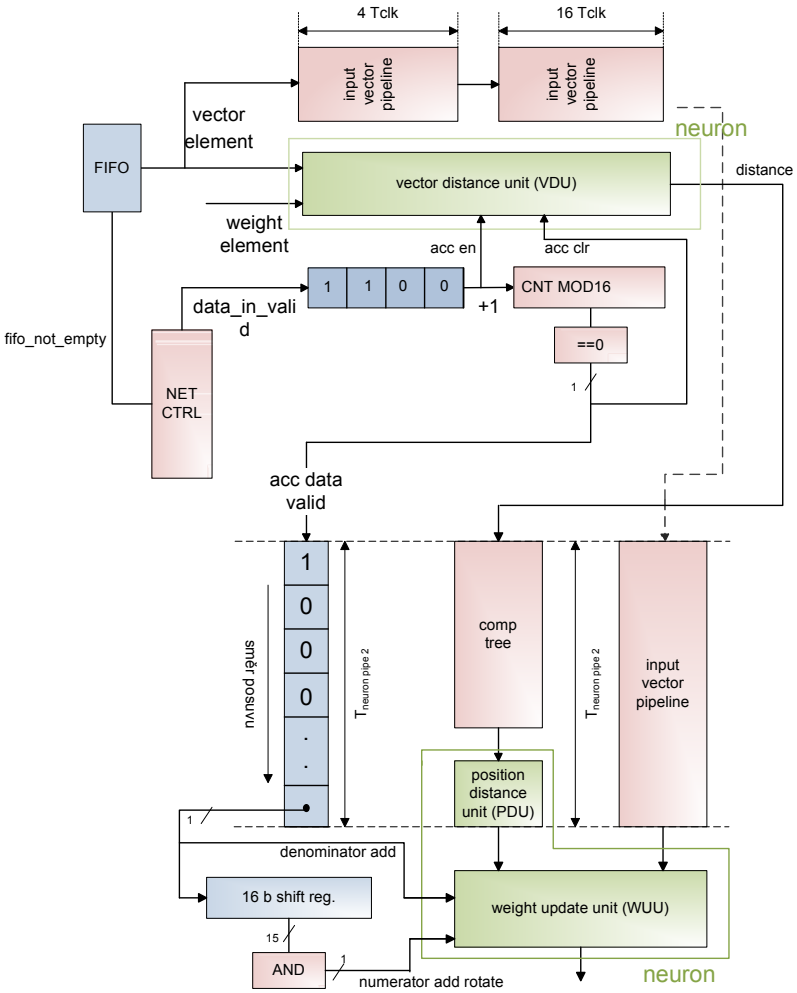
První část pipeline uvnitř neuronových jednotek je zmíněna v kapitole 5.2.3, jedná se o pipeline registry jednotky VDU. Tato jednotka přímo přebírá data ze vstupní fronty. Její hodinový signál je identický s centrálními hodinami, povolení činnosti registrů pipeline uvnitř této jednotky je trvale spuštěné. Povolení rotace váhového registru je určeno na základě signálu z fronty FIFO, který udává, zda má k dispozici data nebo ne (*fifo_empty*). Pokud je fronta plná, generuje se v daném taktu logická 1 do posuvného registru ve sdílené části sítě, který má stejnou délku jako má část pipeline uvnitř neuronu před akumulátorem jednotky VDU. Tato jednička putuje spolu s platnými daty, a když dorazí až k akumulátoru, řídí signál pro povolení akumulace hodnoty. Tímto je zajištěno, že data, která akumulátor přičte, budou vždy platná. Posuvná jednička současně způsobí inkrementaci čítače modulo 16, který slouží k určení taktu, ve kterém má již akumulátor součet všech 16 dimenzí vektoru a současně zajišťuje výmaz akumulátoru. Čítač je společný pro všechny neuronové jednotky. Zmíněná jednička také způsobí aktivaci registrů pipeline následujících jednotek za VDU (comparator tree,PDU, WUU) a dojde k posuvu dat v pipeline o jednu pozici.

Druhá sada posuvných registrů souvisí s řízením činnosti jednotky PDU a jednotky WUU. Tyto dva obvody jsou specifické tím, že vstupy nezískávají z předchozí vnitřní jednotky VDU, nýbrž ze stromu komparátorů. Obě jednotky mají povolenu činnost (aktivované pipeline registry) vždy, když do sítě přichází platná data z fronty. Jednotka pro určení vzdálenosti je proudovou jednotkou a není zde třeba žádného dalšího řízení. Diametrálně odlišná situace nastává u modulu pro WUU. Zde je třeba řídit okamžik načtení jmenovatele a okamžik akumulace a rotace čitatele. K tomu jednotka potřebuje pro činnost dva signály *numerator_add_rotate* a *denominator_add*. První zmíněný slouží k povolení akumulace jednoho elementu vektoru čitatele výrazu (5) a druhý slouží k přičtení hodnoty do jmenovatele. Tyto signály je nutné generovat v přesně stanovených časových okamžicích. K tomuto účelu slouží druhá sada posuvných registrů s bitem platnosti (zvýrazněny modře v Obr. 36). Systém je implementován tak, aby bit platnosti v posuvných registrech určoval místo kde se nachází platný výsledek. Platným výsledkem je vzdálenost mezi vektorem vah a vektorem vstupních dat, která je vypočítána akumulátorem v jednotce VDU. Platnost dat, jak již bylo zmíněno, je indikováno přetečením čítače modulo16, který generuje na výstupu log. 1 a ta postupuje do druhé sady posuvných registrů. Současně s tím postupuje vypočtená vzdálenost do stromu komparátorů, kde dochází k pipelinovanému výpočtu minima ze vzdáleností. Druhou sadou posuvných registrů je log. 1 posouvána paralelně se stromem

komparátorů a s *vector_pipeline*(VP). To znamená, že jednička v tuto chvíli označuje místo ve VP, kde je první složka vektoru vstupních dat, pro který byla počítána vzdálenost a současně označuje místo ve stromě komparátorů, kde se nachází platné mezivýsledky. Jakmile pozice neuronu s minimální vzdáleností opustí strom komparátorů, je vedena zpět do neuronů, kde se využívá v jednotce PDU, která počítá Manhattanskou vzdálenost od BMU v 2D mřížce. Logická 1 ve sdílených posuvných registrech platnosti stále označuje aktuální místo, kde jsou platné hodnoty (nebo mezivýsledky). V taktu, kdy je vypočtena platná vzdálenost od BMU se nachází jednička nakonci posuvných registrů. Signál ,vedoucí z posledního posuvného registru a oznamující příchod platné vzdálenosti a první složky vektoru vstupních dat z jednotky VP, slouží ke dvěma účelům při řízení jednotky pro úpravu vah. Řídí přičtení vzdálenosti k akumulátoru jmenovatele vzorce (5). Současně je veden do posuvného registru, který má šířku 16bit, na který je napojen obvod pro detekci jedničky (tvořen hradly log. součet). Tento údaj sděluje obvodu signálem *numerator_add*_rotate, že má probíhat akumulace složek vektoru do rotačního akumulátoru čitatele vzorce (5).

Vektor má 16 složek, které jdou v pořadí za sebou uvnitř jednotky VP. V posuvných registrech určujících platnost dat se tedy tvoří sled 16 hodnot, kde první je jednička následovaná patnácti nulami. Celkový efekt celého zapojení je takový, že první jednička aktivuje akumulaci jmenovatele a současně první jednička a následujících 15 nul způsobuje akumulace a rotace čitatele. Dochází tedy ke kompletnímu zpracování jednoho vektoru v rámci epochy učení.

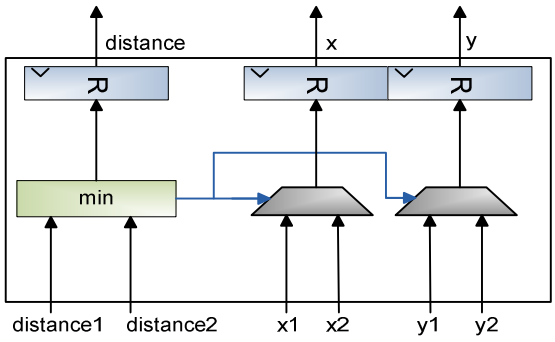
Ve stavu nahrávání vah nebo čtení vah není třeba pipeline řídit, v akci jsou pouze registry váhových vektorů. Ve stavu klasifikace nás zajímá pouze část končící stromem komparátorů a signály pro řízení jednotky WUU jsou hlavním řadičem sítě odpojeny.



Obr. 36 Pipeline

5.3.2 Strom komparátorů

Strom komparátorů slouží k nalezení neuronu s minimální vzdáleností. Strom komparátorů je generován v závislosti na velikosti sítě při překladu. Základním prvkem je komparátor, který porovnává dvě celočíselné hodnoty a multiplexor pro přepnutí dvou vstupů pozic na jeden výstup a pipeline registry. Obr. 37 představuje schéma jednoho komparátoru.



Obr. 37 Komparátor

Různé varianty stromu komparátorů ukazuje Obr. 21 a Obr. 22 v kapitole 4.6. Velikost je závislá na počtu neuronů a ten je závislý na rozměrech sítě, tzn. počet neuronů=X*Y. VHDL kód generující tento strom obsahuje smyčky generate (jedná se o úseky konfigurovatelného kódu generující určité zapojení), které připojují jednotlivé komparátory na pole signálů, která slouží k propojení úrovní stromu a napojení neuronů na odpovídající signály. Entita implementující strom se jmenuje *comparator_tree.vhd*.

5.3.3 Fronty FIFO

Komunikace mezi sítí a řadičem sběrnice HyperTransport probíhá přes fronty typu FIFO. Pro implementaci jsem využil asymetrické fronty, které lze vygenerovat pomocí systému *Xilinx Core Generator*. Vstupní fronta má vstupní šířku 64bitů a směrem k síti výstupní šířku 16bitů. Výstupní fronta je organizovaná v opačném poměru. Fronta vystavuje data na užší straně v pořadí od nejvyššího slova k nejnižšímu slovu. Proto je důležité dodržovat v driveru správné pořadí zápisů a čtení dimenzí vektorů, tak jak bylo uvedeno v kapitole 4.7. Fronta poskytuje signály pro hodiny, reset, vsup a výstup dat a také velmi důležité signály určující prázdnost a plnost fronty, jedná se o signály *fifo_empty* a *fifo_full*. Poslední dva zmíněné signály jsou využity pro řízení pipeline uvnitř sítě a pro řízení komunikace s HT řadičem. U vstupní fronty je signál *fifo_full* použit pro generování signálu *ht_wr_wait*, který způsobuje pozastavení zápisu směrem od driveru. Signál *fifo_empty* je veden na řídící vstup fronty *read_enable,* čímž je zajištěno, že fronta vystaví ihned platná data. Platná data se ale na výstupu fronty objevují se zpožděním jednoho taktu (registrovaný výstup). Ze signálu *fifo_empty* se vytváří negací a zpožděním o jeden takt signál *not_empty_delayed*. Tento signál je použit pro řízení pipeline v podobě signálu *data_in_valid* (ten je vytvořen ze signálu *fifo_empty_delayed* až v řadiči sítě), který jak bylo zmíněno v kapitole 5.3.1 emituje jedničky do posuvných registrů platnosti dat. U odchozí fronty je její signál *empty* po negaci použit jako signál pozastavující čtení a signal *full* slouží k pozastavení činnosti pipline při klasifikaci nebo vyčtení vah, protože je plná odchozí fronta.

5.3.4 Řadič

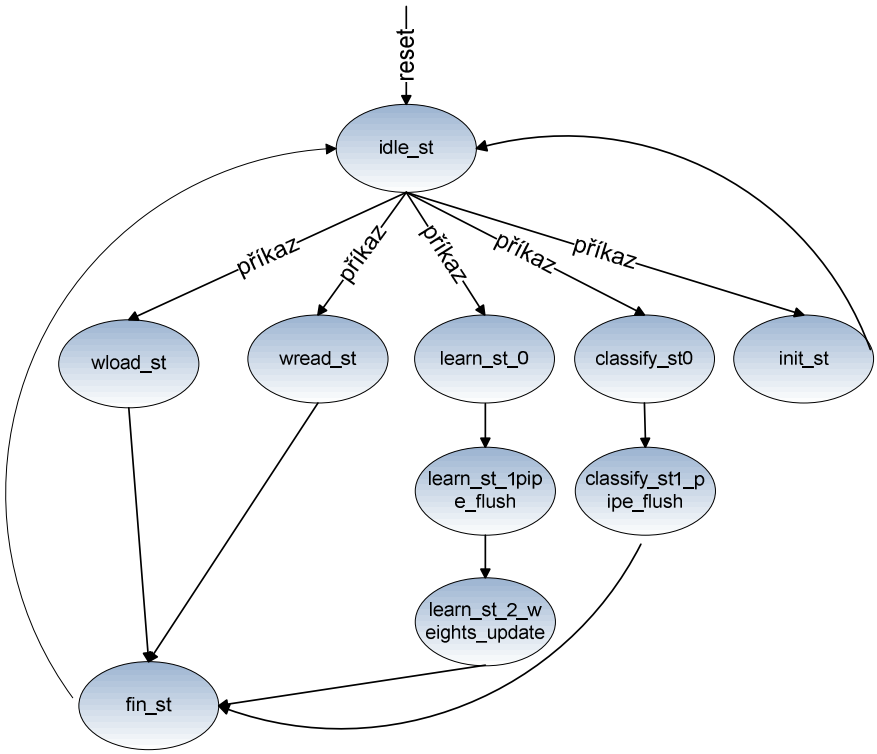
Neuronová síť je řízena pomocí příkazů zasílaných na adresu příkazového registru. Tento registr ve skutečnosti není implementován v podobě registru, nýbrž přímo řídí přechody hlavního řadiče sítě ze stavu klidu (IDLE) do jiného stavu. Síť má po stránce vnitřní implementace více stavů než bylo uvedeno v kapitole návrhu. Jako řadič je použit konečný stavový automat typu *Mealy* (výstup je daný stavem a vstupními signály z ostatních komponent). Tento druh automatu jsem zvolil proto, že automat modifikuje stavové signály ze sítě a front FIFO a dál je využívá k řízení, způsob modifikace je na základě stavu. Například signál pro zapsání do vstupní fronty není veden přímo ze vstupu sběrnice HyperTransport, ale je veden přes stavový automat, který v případě stavu klidu tento signál „odpojí“.

Prvním stavem automatu je klidový stav (IDLE), v tomto stavu síť čeká na nový příkaz. Automat čeká na aktivaci signálů *ht_wr_req, ctrl_reg_write_sel* a na jeden z bitů signálu *ht_wr_data.* Signál ht_wr_data je vstup dat do sítě ze sběrnice HT. Jak bylo uvedeno v kapitole 4, každý příkaz je představován jedním bitem. Na základě toho, který bit je aktivní, se automat přesune do jednoho z dalších stavů, které představují počátek některého z režimů. Všechny režimy kromě režimu INIT končí stavem FIN, který slouží k tomu, aby driver mohl vyčíst stav sítě. Tento stav indikuje dokončení příkazu. Během tohoto stavu se vystavuje na signál *status* hodnota signalizující dokončení činnosti. Nyní stavový automat čeká na signál *status_reg_read*, který

signalizuje, že došlo ke čtení stavového registru (čtení stavového signálu). Poté přechází do stavu *IDLE*.

Mezi sekvence stavů patří:

- *INIT_ST*
- *WLOAD_ST, FIN_ST*
- *WREAD_ST,FIN_ST*
- *LEARN_ST0,LEARN_ST0_PIPE_FLUSH,*
- *EARN_ST2_WEIGHTS_UPDATE, FIN_ST*
- *CLASSIFY_ST0,CLASSIFY_ST1_PIPE_FLUSH, FIN_ST*



Obr. 38 Stavový diagram řadiče

INIT_STATE slouží k resetu sítě. Tím se smažou pipeline registry. Do stavu *WLOAD_ST* se přechází, když je požadováno čtení vah. Při zápisu na adresu virtuálního příkazového registru se současně do vyšší poloviny zapíše počet dat ke zpracování (počet 16b hodnot fracval). V případě vah je to o jedna méně, než je skutečný počet (způsobeno čítačem, který povoluje rotaci vah i hodnotě 0). Počet je zapsán do čítače, který se dekrementuje při každém taktu, kdy je schopna vstupní fronta dodat data (aktivní signál *data_in_valid*). Tím je zajištěno, že po načtení daného počtu složek váhových vektorů přejde řadič na základě podtečení čítače do stavu *WLOAD_FIN.*

Ve stavu *WLOAD_ST* je především nastaven signál *mode* (odvozen od stavu fronty FIFO), který nastavuje správný režim činnosti váhových registrů (posuv), signál *neural_unit_data_in_valid* je nastaven na hodnotu log. 0 (aritmetická pipeline není aktivní), a signál *in_fifo_wr_en* (zápis do vstupní fronty FIFO) je odvozen jako logický součin signálů *ht_wr_req* (požadavek zápisu ze sběrnice HT) , *in_fifo_wr_sel*(jedná se o adresu FIFO), not *in_fifo_full*(FIFO není plné). Ve výsledku tato kombinace způsobí, že můžeme nahrávat váhy do FIFO a současně se FIFO vyčítá

směrem do posuvných váhových registrů. Na status signál je vystavena hodnota reprezentující činnost sítě. Při četní ze stavového registru nedochází k žádnému přechodu. Po podtečení čítače počtu dat (vect_elem_cnt) přejde řadič do stavu *FIN*.

Stav *WREAD_STAT*, je stavem ve kterém má docházet k zápisu vah do výstupního FIFO. Současně je vhodné, aby se váhy zachovaly, tzn. výstup řetězce registrů by se měl spojit se vstupem. Opět je nastaven signál mode, tentokráte je jeho nižší bit odvozen od stavu výstupní fronty (signál *fifo_full*). Signál říká, zda se mají posouvat registry váhových vektorů. Signál pro zápis do vstupní fronty je deaktivován a signál *out_fifo_wr_en*(zápis výst. fronty) je spojen se signálem not *out_fifo_full* (fronta není plná). Čítač počtu dat je nastaven stejně jako v případě zápisu a složí ke stejnému účelu.

Význam *WREAD_FIN* je stejný jako v případě režimu zápisu vah.

Stav LEARN_ST0 je prvním ze sekvence stavů, které jsou nutné pro naučení sítě a přechod do něj je inicializován zápisem do řídícího registru ze stavu *IDLE*. V tomto stavu je povolen zápis do vstupní fronty a na konci fáze bude zaručeno, že budou vyčteny všechny vektory, které byly do FIFO zapsány (přesněji musí dojít k vyčtení všech složek vstupních vektorů). Počet složek pro vyčtení se podobně jako u vah objeví při zápisu řídícího slova, nyní je však počet rovný počtu složek a není o jedna menší jako v případě vah. V průběhu této učící fáze je povolena rotace váhových registrů uvnitř neuronů (váhy nepřechází mezi neurony). Povolení rotace je zajištěno signálem *weight_rot*, který je odvozen ze signálu *neural_unit_data_in_valid*. Signál *neural_unit_data_in_valid* je v případě učící fáze nastaven nikoliv na 0, nýbrž je řadičem propojen se signálem *in_fifo_not_empty_delayed*, který oznamuje, že má vstupní FIFO platný element vektoru. Tímto signálem jsou emitovány jedničky do posuvných registrů, které jsou využívány pro určení platnosti dat v daném místě numerické pipeline a načítání složek do jednotky VP. Vždy když má FIFO platná data dojde k vyslání jedničky do jednotky VDU uvnitř neuronu. Tato jednička řídí akumulátor zmíněné jednotky a také povoluje posuv *vector_pipeline.* Po vyčtení všech dat z FIFO ještě není možné přejít do konečného stavu. Data v pipeline se pohybují na základě signálu *in_fifo_not_empty_delayed*, který se po vyčerpání FIFO deaktivuje. To ovšem znamená, že celá numerická část přestane pracovat díky zastavení pipline registrů. Tomu je nutné předejít, poněvadž numerické obvody stále obsahují platná data a výpočet je nutné dokončit. Řešením je vynucený posuv pipeline. Pro tuto úlohu jsem zavedl další stav zvaný *LEARN_ST1_PIPE_FLUSH*.

Během uvedeného stavu *LEARN_ST1_PIPE_FLUSH* nastaví automat signál *neural_unit_data_in_valid* na hodnotu 1 a zůstane v této konfiguraci po dobu nutnou k vyprázdnění pipeline (tu odpočítává čítač *pipe_flush_cnt* a je rovna době nutné k vyprázdnění celé jednotky vector_pipeline). Po skončení této fáze jsou všechny mezivýsledky již naakumulovány v akumulátoru jmenovatele a čitatele v jednotce pro úpravu vah(Weight Update Unit - WUU). Pro účel změny hodnot ve váhových registrech byl zaveden stav *LEARN_ST2_UPDATE_WEIGHTS.* V tomto stavu je nutné provést podělení každé složky vektoru v čitateli jmenovatelem a přenést tuto hodnotu do registrů váhových vektorů. Také je aktivní signál *input_vect_accum_rotate,* který způsobuje rotaci složek vektoru v čitateli. Operace dělení trvá více hodinových cyklů než se naplní pipeline děličky platnými daty a než je v děličce vypočítán první platný element nového váhového vektoru. Je tedy nutné zpozdit povolení lokálního posuvu registru váhového vektoru neuronu. Toto je provedeno zařazením posuvného registru do cesty vyššího bitu signálu *mode.* Tento bit slouží k povolení posuvu váhových registrů a zvolení výstupu jednotky WUU jako vstupu do těchto registrů. Tím se začnou nahrávat nové váhy do registrů. Po tomto stavu již následuje stav *LEARN_FIN_ST* se stejným významem jako u režimu načítání nebo zápisu vah. Po přečtení stavového registru se síť překlopí do stavu *IDLE.*

Posledním režimem je vybavování. V tomto režimu se k předloženému vstupnímu vektoru hledá pozice BMU. V podstatě se jedná o jednodušší variantu režimu učení, kdy není nutné provádět závěrečnou úpravu váhových vektorů. Je ale nutné povolit zápis souřadnic BMU do výstupní fronty FIFO. Automat řadiče tedy přepne vstup FIFO na výstup stromu komparátorů (ten určuje BMU). Zápis do FIFO je povolen signálem generovaným na základě logického součinu signálů out_fifo_bmu_pos_write(získává se z konce řetězu posuvných registrů bitu platnosti) a *pipe_enable_out*.

5.3.5 Popis zdrojových kódů

Zdrojový kód sítě je napsaný v jazyce VHDL. Kód je strukturálně rozdělen do entit a komponent. Každá z entit představuje určitý funkční blok/jednotku. Konfigurace sítě se nachází v souboru pojmenovaném *constants.vhd*. V tomto souboru je především nastavení rozměrů sítě. Jsou zde i šířky a definice veškerých datových typů. Všechny definice kromě rozměrů sítě jsou především pro přehlednost implementace a neměly by být měněny bez změn v ostatních entitách, kde jsou konstanty použity. Je to především z důvodu, že aritmetické obvody (např. barrel shiftery) nejsou generické, jsou optimalizovány pro konstantní šířku dat.

Všechny složky kromě front FIFO jsou napsány platformě nezávislým způsobem, je možné je syntetizovat pro libovolná FPGA. Zmíněné fronty jsou generovány systémem Xilinx Core Generator, lze je tedy použít jen se syntézními prostředky Xilinx. Při nutnosti konverze na jinou platformu postačí zaměnit vnitřek komponent za jinou implementací.

Každá entita obsahuje kód procesů a paralelní prostředí a současně má většinou vloženu i jinou komponentu. Tab. 10 popisuje rozdělení a hierarchii (vnoření) základních komponent. Tabulka je organizována zleva doprava od nejvyšších komponent k nejnižším.

<table>
<tr><td rowspan="6">top_ctrl_net</td><td rowspan="5">top_neural_units</td><td rowspan="3">neural_unit</td><td>neural_unit_distance_module</td><td>fvc_sub_abs_div_2
sqr_func</td></tr>
<tr><td>neural_unit_pos_dist_module</td><td>sub5_unsig_abs</td></tr>
<tr><td>neural_unit_weight_update_module</td><td>fvc32_to_fv32
fv_to_fvc
log32_func
sub_exp37_func</td></tr>
<tr><td>comparator_tree</td><td>comparator</td><td></td></tr>
<tr><td colspan="3">input_vectors_pipeline</td></tr>
<tr><td colspan="4">out_fifo
in_fifo</td></tr>
</table>

Tab. 10 Hierarchie komponent

6 Simulace, syntéza a ověření výsledků

Po dohodě s vedoucím práce bylo stanoveno, že ověření činnosti sítě bude v podobě simulace. Vzhledem k tomu, že test neprobíhal na fyzickém hardware, bylo nutno vytvořit simulační model systému DRC. Tento model má za úkol číst data ze souboru, zasílat je do neuronové sítě a případně vyčítat data z neuronové sítě a porovnávat s předpokládanými daty. Model má identické porty jako komponenta HyperTransport řadiče dodávaná k systému DRC jako netlist.

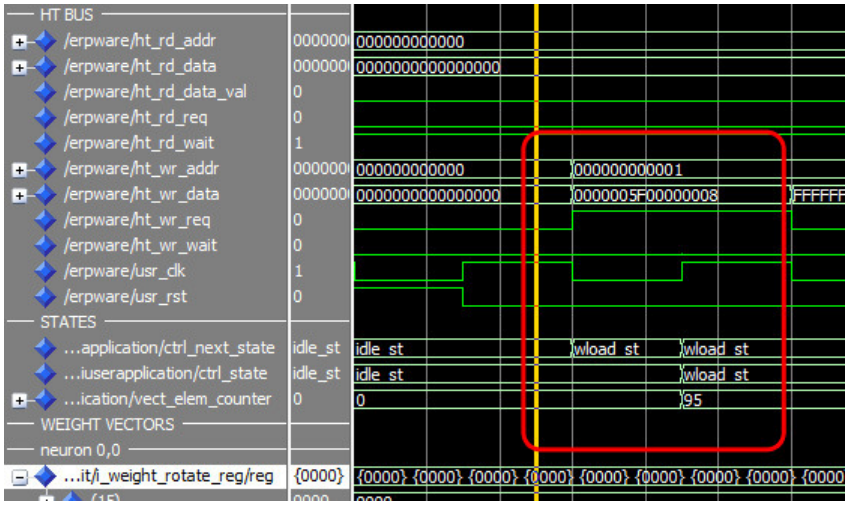
Při vytváření modelu bylo využito průběhů signálu uvedených v technické dokumentaci systému DRC. Jsou uvedeny i v rámci kapitoly 5.1. Tyto průběhu bylo nutné dodržet proto, aby simulační model co nejvěrněji popisoval reálného chování DRC.

Při ověřování funkce jsem vycházel v první fázi především z velmi podrobného studia průběhů signálů s použitím simulátoru Modelsim 6.5. Po odladění chyb nalezených tímto postupem ověření jsem využil možnost automatizované simulace, vytvořil jsem simulační skript, který provedl epochu učení a ověřil, že data odpovídají datům ze softwarového modelu zmíněného v kapitole 4.3. Ověření dopadla úspěšně.

6.1 Simulace

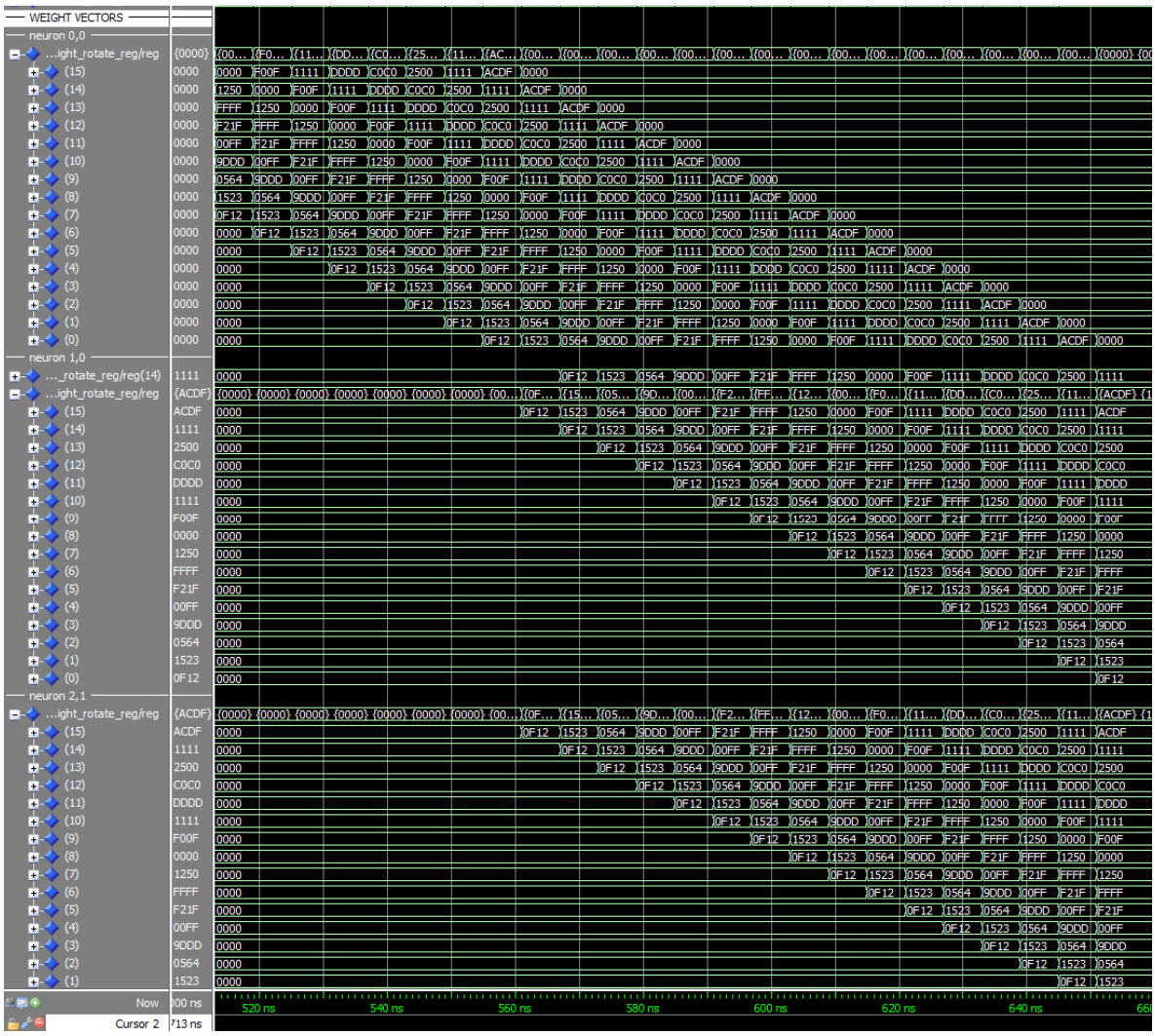
Na následujících řádcích prezentuji ukázky s popisem jednotlivých zajímavých částí průběhů signálu ze simulace učení. Obrázky se vztahují k síti o velikosti 3x2 (X x Y) neurony.

Obr. 39 ukazuje přechod ze stavu *IDLE* do stavu načítání váhových vektorů. K přechodu dojde na základě zapsání příkazu (*ht_wr_data*) na adresu 0x1 (*ht_wr_addr*). Horní polovina příkazového slova představuje počet složek všech váhových vektorů a tento počet se zapíše do čítače *vector_elem_counter* .V následujícím taktu se změní signál *ctrl_state* na hodnotu *wload_st*, která označuje stav načítání váhových vektorů.



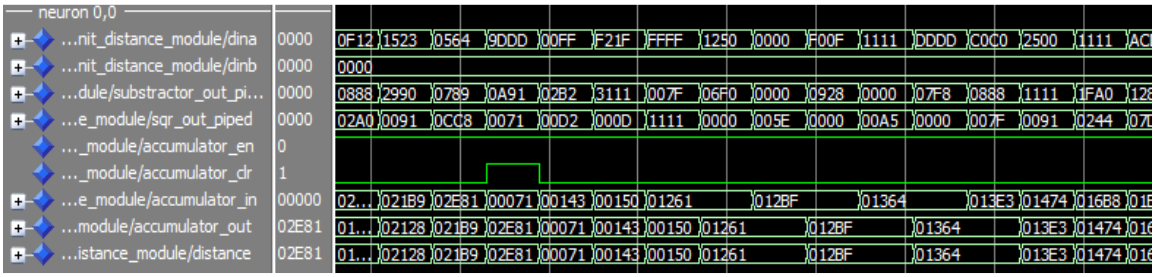
Obr. 39 Přechod do stavu načítání vah (weight load)

Obr. 40 ukazuje průběh načítání váhových vektorů do neuronů V pořadí odshora dolů jsou neurony na pozicích [0,0], [1,0] a [2,1]. Je zobrazena situace v posledních několika taktech. Váhy neuronů [1,0] a [2,1] jsou nenulové a identické. Tomu odpovídá výsledek. Je vidět, jak složky elementů prochází skrz posuvný registr v neuronu [0,0] do cílového neuronu a na konci fáze jsou váhy správně v neuronech [1,0] a [2,1].



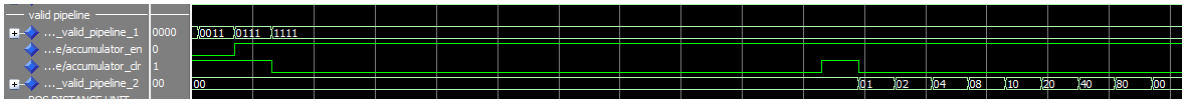
Obr. 40 Inicializace váhových vektorů (sériové načítání složek skrz neurony)

Část průběhu signalů jednotky VDU (vector distance unit) je uvedena na Obr. 41. Vstupy *dina, dinb* reprezentují vstupy složek datového a váhového vektoru. Signal *substractor_out_piped* je pipelinovaný výstup obvodu pro výpočet podílu a dělení dvěma (entita *fvc_sub_abs_div_2*). Podobně *sqr_out_piped* je výstupem obvodu výpočtu druhé mocniny. Signál *accumulator_en* a *accumulator_clr* slouží k povolení zápisu akumulačního registru a druhý zmíněný signál slouží k nulování. *Accumulator_in* je výstup sčítačky, která sčítá výstup akumulátoru se vstupem do akumulačního obvodu. *Accumulator_out* je výstup akumulátoru a představuje vzdálenost vektorů, je platný vždy v taktu, kdy je nastaven signál pro nulování akumulátoru.



Obr. 41 Vector distance unit

Obr. 42 zobrazuje průběhy posuvných jedniček v registrech planosti dat. Na obrázku je průběh *valid_pipelin_1* a *valid_pipeline_2*. V obou případech se jedná o sadu posuvných registrů o šířce 1b. Délka první sady je pevně daná, délka druhé sady je vypočítaná během překladu na základě hloubky stromu komparátor. Vstupem první sady je výstup z fronty, který indikuje platná data, vstupem druhé sady je výstup čítače, který indikuje platný součet dat akumulátoru jednotky VDU, což potvrzuje simulace(za impulsem na signálu CLR následuje platná jednička, která postupuje pipeline2), která používá pro řízení jednotky WUU(weight update unit).



Obr. 42 Valid „bits“

Jednotka WUU obsahuje rotační registr akumulátor čitatele, akumulátor jmenovatele a děličku a pomocné obvody pro převod fracval z dopňkového kódu do přímého kódu a zpět. Ukázka výpočtu rotačního akumulátoru čitatele a akumulátoru jmenovatele je na Obr. 43. Je vidět, že do jednotlivých sekcí akumulátoru jsou nasouvány postupně složky vektoru ze vstupu, což se děje v první iteraci, kdy počáteční hodnota je nulová. Před začátkem další iterace (další vstupní vektor) je akumulátor naplněn prvním vektorem a během následující iterace dochází k přičítání hodnot složek dalšího vektoru. V tomto případě se přičítá opět stejná hodnota, protože vstupní vektor je stejný a vzdálenost je 0. Tento okamžik je vidět v čase 1005 ns. Signál *numerator_add_rotate* slouží k povolení rotace a úprav akumulátorů. Signál *denominator_add* slouží k aktualizaci hodnoty jmenovatele přičtením hodnoty funkce vzdálenosti *h* (hodnota $0x7FFF*2^{-scaled\_distance}$). Pro každý vstupní vektor se provede jedna akumulace jmenovatele a šestnáct akumulací čitatele (každá akumulace je jedna dimenze vektoru).

Obr. 43 dále ukazuje průběhy na děličce, jejímž úkolem je dělení čitatele jmenovatelem na konci epochy učení (po zpracování všech vektorů z trénovací množiny). Po skončení platnosti signálu *numerator_add_rotate* se skončí fáze akumulace a začne fáze výpočtu podílu, při které rotují hodnoty čitatele a každá se dělí jmenovatelem. Zpoždění jednotky pro dělení je dáno obvodem pro transformaci doplňku do přímého kódu, logaritmováním, provedením rozdílu a exponenciály a na závěr převodem do fracval v doplňkovém kódu. Výsledné váhy po dělení jsou zvýrazněny obdélníkem. Nové váhy jsou zapsány zpět do rotačních váhových registrů. Tímto končí režim učení.

Obr. 43 Jednotka úpravy vah (WUU)

6.2 Výsledky syntézy

Syntéza byla provedena pro síť velikosti 3x3 neurony a 5x5. Syntéza byla provedena nástrojem Xilinx ISE 11. Výsledky jsou uvedeny v Tab. 11 a Tab. 12.

Podrobnější syntézní zprávy lze nalézt na přiloženém médiu CD/DVD spolu s veškerými zdrojovými kódy.

SOM 3x2 6 Neurons			
Logic Utilization	Used	Available	Utilization
Number of Slices	4094	89088	4%
Number of Slice Flip Flops	4656	178176	2%
Number of 4 input LUTs	7081	178176	3%
Number of bonded IOBs	139	960	14%
Number of GCLKs	1	32	3%
Estimated frequency:			170 MHz

Tab. 11 Syntéza sítě rozměru 3x2 neurony

SOM 5x5 25 Neurons			
Logic Utilization	Used	Available	Utilization
Number of Slices	15923	89088	17%
Number of Slice Flip Flops	18338	178176	10%
Number of 4 input LUTs	27752	178176	15%
Number of bonded IOBs	139	960	14%
Number of GCLKs	1	32	3%
Estimated frequency:			140 MHz

Tab. 12 Syntéza sítě rozměru 5x5 neuronů

7 Závěr

Úkolem této diplomové práce byl návrh implementace hardwarové varianty samoorganizující se neuronové sítě s využitím aproximovaných funkcí, které umožňují optimalizaci hardwarového návrhu pro dosažení vyšší rychlosti zpracování dat. Vedoucím práce byla zadána varianta sítě nazývaná SOM (Self Organizing Maps). Jedná se o síť, implementující tzv. učení bez učitele a umožňující shlukovou analýzu dat. Cílovou platformou pro implementaci je obvod FPGA Xilinx Virtex 4 umístěný v systému DRC.

V první fázi práce jsem se seznámil s principem sítě a možnými variantami algoritmů řešících výpočty váhových vektorů neuronové sítě. Jednalo se o iterativní a dávkovou (Batch SOM) variantu algoritmů. Pro nepřítomnost datových závislostí během výpočtu se ukázal jako vhodnější dávkový algoritmus nazývaný Batch SOM, který umožnil efektivní implementaci hardwarové pipeline. Dále jsem se seznámil s problematikou implementace aproximovaných funkcí, které bylo zadáno použít.

S využitím vybraného algoritmu Batch SOM jsem vytvořil softwarový model v jazyce Java SE. Aplikace umožňuje číst váhy a vstupní vektory, zpracovat je a graficky i textově zobrazit výsledky. Implementuje aproximované i neaproximované funkce, čímž umožňuje porovnat výsledky učení sítě za použití obou variant funkcí. Ověřil jsem, že síť je schopna přizpůsobit své váhové vektory topologii předkládaných dat.

Druhá fáze práce spočívala v návrhu a implementaci sítě pro obvod FPGA s využitím jazyka VHDL. V rámci návrhu jsem dokončil některé aritmetické obvody, využívající aproximované funkce. Vytvořil jsem datový model vstupu a výstupu sítě, tzn. datové a řídící registry. Provedl jsem blokové rozdělení sítě do funkčních celků. Implementační část práce spočívala v konkrétní implementaci jednotlivých aritmetických jednotek a řadiče sítě v jazyce VHDL. Podstatnou roli zde hrála implementace pipeline a to jak vnitřních datových pipeline registrů výpočetních jednotek, tak i řízení této pipeline a synchronizace se vstupní a výstupní frontou typu FIFO. Implementace sítě byla provedena s důrazem na možnost snadné konfigurace rozměrů sítě při překladu a syntéze, díky čemuž je možné síť přizpůsobit aplikaci a typu FPGA.

Na základě dohody s vedoucím práce bylo stanoveno, že finální implementace nebude spouštěna fyzicky na systému DRC. Pro potřeby verifikaci implementace bylo tedy nutné vytvořit simulační model sytému DRC v jazyce VHDL, který byl využit pro emulaci rozhraní systému DRC v programu Modelsim 6.5. Pro jeho tvorbu jsem dodržel rozhraní se systémem DRC definované v uživatelském manuálu [5]. Vytvořil jsem simulační skript, který načítá data v hexadecimální podobě a provádí porovnání dat získaných softwarovým modelem sítě s hodnotami získanými z hardwarové varianty sítě.

Verifikace sítě proběhla úspěšně na testovací množině dat. Síť je možné použít jako komponentu v návrhu pro systém DRC, kde lze využít výhody integrace FPGA v systému typu PC a využít výsledky této práce v podobě akcelerátoru neuronové sítě typu SOM.

8 Literatura

[1] Skrbek M.: **Fast neural network implemntation**, 1999

[2] T. Kohonen: **Self-Organizing Maps**, Springer 2001

[3] Bruno Silva and Nuno Marques: **A Hybrid Parallel SOM Algorithm for Large Maps in Data-Mining**
URL: <http://ssdi.di.fct.unl.pt/~nmm/MyPapers/SM2007.pdf>

[4] Šnorek M.: **Neuronové sítě a neuropočítače**, Vydavatelství ČVUT, 1996

[5] DRC Computer: **DRC coprocessor system – user guide,** 2007

[6] Xilinx, Inc.: **Virtex-4 FPGA user guide**, 2008
URL: <http://www.xilinx.com/support/documentation/user_guides/ug070.pdf>

A Seznam použitých zkratek

SOM	Self Organizing Map
FPGA	Field Programmable Gate Array
BMU	Best Matching Unit
2D	2 dimension
HW	Hardware
SW	Software
HT	HyperTransport
b	bit
Log. 1	Logická jednička
VHDL	Very-high-speed hardware description language
LUT	Look Up Table
SHR	Shift Right
DDR	Double Data Rate
LVDS	Low Voltage Differential Signalling
API	Application Programming Interface
VDU	Vector Distance Unit
PDU	Position Distance unit
VP	Vector Pipeline
WUU	Weight Update Unit
SPI	Serial Pheripherial Interface
SQR	square function
FIFO	First In First Out
LF	Learning Factor

B Obsah CD/DVD

Data jsou rozdělena do následujících adresářů:

java_som\	softwarový model sítě SOM
src\	zdrojové kódy
som_net_fpga\	implementace sítě SOM pro FPGA
src\	
synth\	syntézní zdrojové kódy
sim\	simulace
doc	dokumentace
readme.txt	bližší popis jednotlivých adr.