

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

NÁVRH SÍŤOVÉHO PRVKU POMOCÍ NEURONOVÉ SÍŤE

NETWORK ELEMENT PROJECT BY MEANS OF NEURAL NETWORK

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE
AUTHOR

PETR POKORNÝ

VEDOUCÍ PRÁCE
SUPERVISOR

doc. RNDr. Ing. JIRÍ ŠTASTNÝ, CSc.

BRNO 2008

ABSTRAKT

Diplomová práce se zabývá řešením prioritního síťového přepínače, jehož model byl vytvořen v simulačním prostředí Matlab - Simulink. Úloha optimálního přepínání je řešena pomocí umělé neuronové Hopfieldovy sítě. Výsledkem práce je model přepínače a srovnání časové náročnosti, při řešení optimalizačního problému, pomocí umělé neuronové sítě a bez využití této sítě.

Tato diplomová práce byla zpracována v rámci vědecko-výzkumného záměru MSM 0021630529 Inteligentní systémy v automatizaci.

ABSTRACT

The diploma thesis deal with a priority network switch whose model was made in programming environment Matlab - Simulink. Problem of optimal switching is solved by Hopfield's artificial neural network. Produce of the diploma thesis is a model of packet switch and time-severity comparison of optimization problem solved with or without artificial neural network.

The thesis was developed in research project MSM 0021630529 Intelligent Systems in Automation.

KLÍČOVÁ SLOVA

Umělá neuronová síť, Hopfieldova neuronová síť, optimalizace, přepínač, paket, prioritní přepínání, simulace, Simulink.

KEYWORDS

Artificial neural network, Hopfield neural network, optimization, packet switch, packet, priority switching, simulation, Simulink.

OBSAH

Abstrakt	7
Abstract.....	7
Klíčová slova.....	7
Keywords	7
Obsah	9
1 Úvod.....	11
2 Popis modelu.....	13
2.1 Podsystem Generátor paketů (Packet Generator).....	15
2.1.1 Stateflow diagram (PGChart).....	16
2.2 Podsystem Volání funkce (Function-Call Beat).....	19
2.2.1 Podsystem Spouštěné volání funkce (Triggered Function Call).....	19
2.3 Podsystem Hlavní manažer dat (Main Data Manager)	20
2.3.1 Podsystem Volání funkce (Function-Call Beat).....	22
2.3.2 Podsystem Spouštěné volání funkce (Triggered Function Call).....	22
2.3.3 Stateflow diagram Rozděl pakety (Sort Packets).....	22
2.3.4 Podsystem Fronta (FIFO).....	25
2.3.5 Podsystem Překladač konfig. vekt. na číslo fronty (Vector_To_FOFO_no)	30
2.4 Blok Sloučení matic (Matrix Concatenation).....	33
2.5 Podsystem Hopfield (Hopfield)	33
2.5.1 Podsystem Úprava priorit (Priority Adaptation)	34
2.5.2 Podsystem Generátor pulsu (Pulse Generator).....	35
2.5.3 Podsystem Hopfieldova neuronová síť (Hopfield NN).....	36
2.5.4 Podsystem Stabilní výsledek (Product Ready).....	39
2.6 Podsystem Standardní výpočet (Standard Computing).....	39
2.6.1 Stateflow tabulka Sériový výpočet (Serial Computing).....	40
2.7 Průběh simulace	43
2.8 Reprezentace výsledků.....	45
3 Hopfieldova neuronová síť	47
3.1 Teoretický popis Hopfieldovy neuronové sítě	47
3.2 Konkrétní použití Hopfieldovy neuronové sítě.....	47
4 Porovnání výsledků.....	51
5 Závěr	55
Seznam obrázků a tabulek	57
Použitá literatura	59

1 ÚVOD

Základním elementem v datových sítích jsou aktivní síťové prvky. Hlavním úkolem těchto síťových prvků je přesměrování datového celku (paketu) od odesílatele k požadovanému příjemci. Při vysokých rychlostních požadavcích, které jsou na síťové prvky kladeny, je největším problémem zajistit rychlý a přitom prioritně optimální výběr paketů.

Rychlost sériového zpracování dat, které se v některých aktivních síťových prvcích používá, je závislá na výkonu centrálního procesoru, jehož vysokovýkonná varianta může být ekonomicky nepřijatelná. Z tohoto důvodu je efektivnější použít větší počet jednodušších paralelně zapojených funkčních bloků. Jeden ze způsobů implementace paralelního zapojení více jednodušších funkčních bloků může být umělá neuronová síť.

Typů umělých neuronových sítí je mnoho, ale každá má specifický obor použití. Pro optimální výběr paketů podle jejich priorit je nutno použít jednu z optimalizačních neuronových sítí, která by byla schopna zpracovat prioritní údaje ze vstupů síťového prvku a optimálně nastavit přepínače tohoto prvku mezi vstupy a požadovanými výstupy. Příkladem takové umělé neuronové sítě je rekurentní Hopfieldova neuronová síť.

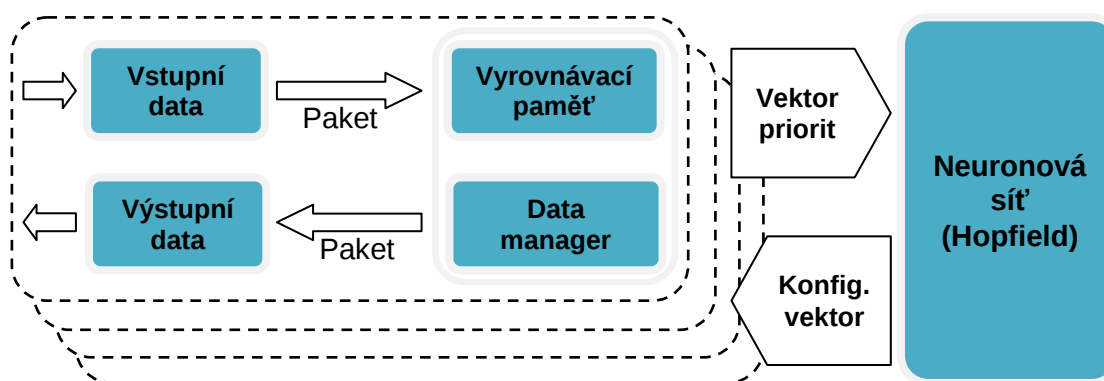
Tato neuronová síť, tedy Hopfieldova, je použita při řešení optimalizace aktivního síťového prvku (síťového přepínače) v této diplomové práci a pro porovnání časové náročnosti je srovnána se sériově numerickým řešením problému. Řešení síťového přepínače s Hopfieldovou neuronovou sítí a jeho simulace jsou prováděny v programu Matlab, respektive v jeho nástavbě Simulink.

Použitý model přepínače má 4 vstupy, do kterých přicházejí datové celky od odesílatele a 4 výstupy, ze kterých datový celek odchází k příjemci. Podle počtu vstupů a výstupů, který musí být totožný, se přepínač nazývá čtyř-portový síťový přepínač (4 port packet switch).

2 POPIS MODELU

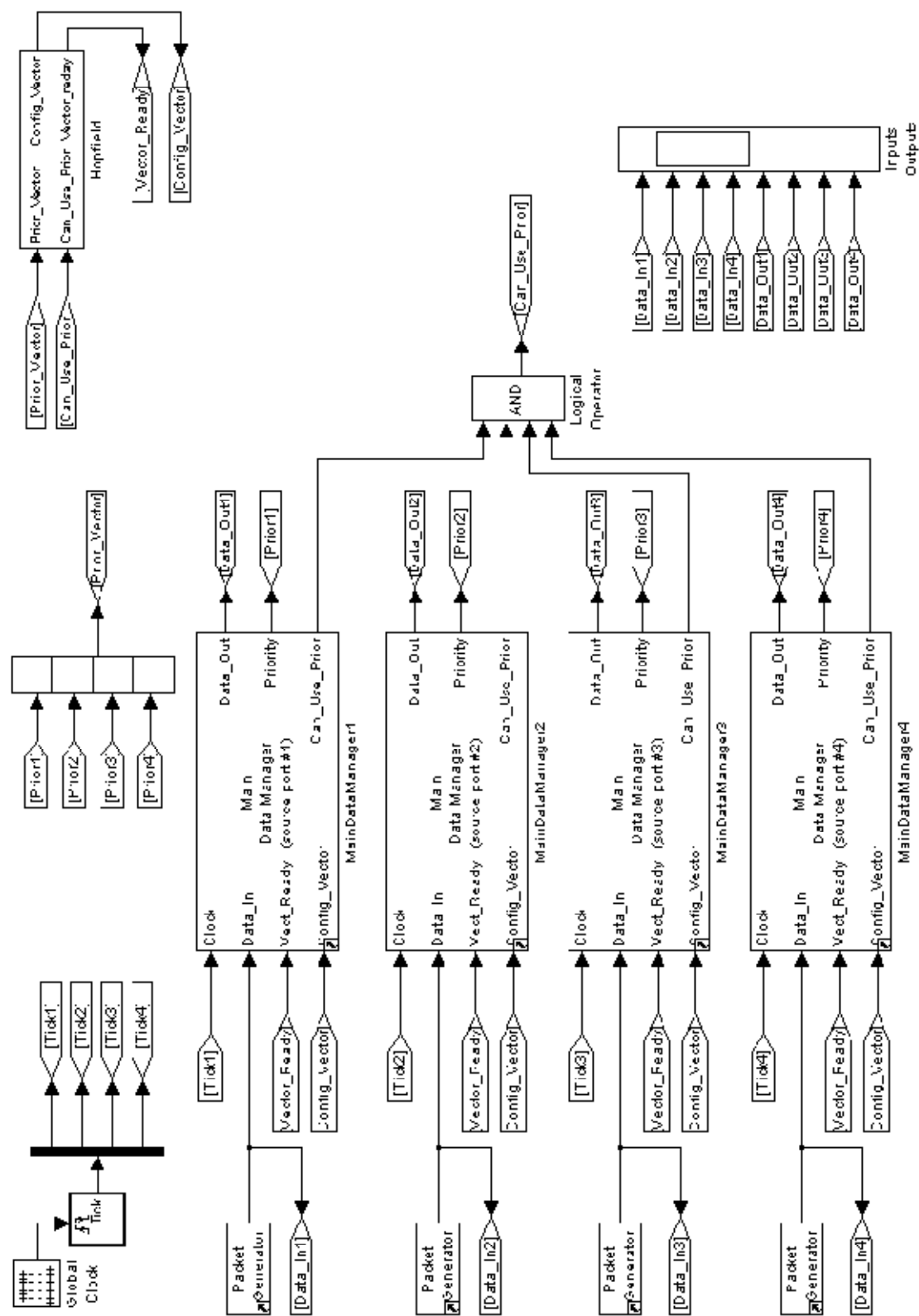
Model čtyř-portového síťového prvku (dále jen přepínač) je vytvořen pomocí základních bloků, Stateflow Toolboxu (Chart bloky) a Neural Network Toolboxu programu Simulink.

Tento model zároveň obsahuje generátor paketů (packet generator), který je náhradou za vstupní data od odesílatele. Data jsou generována náhodně s ohledem na využitelné vlastnosti paketu. Výstupní data, namísto posílání k příjemci, jsou graficky zobrazována v grafech. Struktura přepínače je zobrazena na Obr. 1.



Obr. 1 Struktura přepínače.

Každý ze čtyř vstupů přepínače má svoji vlastní paměť, do které se ukládají příchozí data. Paměť je typu fronta (FIFO). Z fronty jsou postupně přečteny vektory priorit (priority jsou brány v rozmezí 0 až 98), které jsou vstupní hodnotou pro neuronovou síť. Po dokončení výpočtu neuronové sítě je možno použít konfigurační vektor, který je složen z bipolárních hodnot, jež udávají přesměrování daných paketů z každého vstupu na určitý výstup přepínače. Při jednom cyklu jsou tedy poslána data ze všech vstupů (od všech odesílatelů) na všechny výstupy (pro každého příjemce) a to s optimálním využitím priorit. V případě, že některý z odesílatelů neposílá data (fronta je prázdná), odešle se prázdný paket, resp. žádná data.



Obr. 2 Model přepínače.

2.1 Podsystem Generátor paketů (Packet Generator)

Generátor paketů je jedním ze subsystémů modelu. V celém modelu je jeho četnost rovna počtu portů přepínače, jsou zde tedy 4 Generátory paketů (viz Obr. 2). Nemá žádný vstup a disponuje jedním výstupem. Plní funkci odesílatele dat, který by byl v reálné struktuře datové sítě [6].

Pro každý generátor paketů lze nastavit parametry (Obr. 3 Ukázka nastavení parametrů pro blok Packet Generator pomocí masky.), pomocí masky bloku, udávající:

- Zdroj paketu (Source),
- délku paketu (Length Range),
- zpoždění paketu od předchozího (Wait Range),
- výstup, na který je paket určen (Destination Range),
- velikost dat (Data Range),
- maximální možnou prioritu paketu (Max. Priority Number).

Parametry označené jako rozsah (Range) se udávají v potřebném rozsahu a následně je z tohoto rozsahu vybrána hodnota náhodným výběrem.

Priorita může být v rozmezí 0 až Max. Priority Number (< 99) a je zavedena konvence, čím vyšší je prioritní číslo, tím horší je priorita paketu. Je-li naopak prioritní číslo rovno nule, pak je priorita paketu nejlepší a tento paket bude s vysokou pravděpodobností odeslán na určený výstup z přepínače přednostně.

Packet Generator (mask)

Parameters

Source
1 (a)

Length Range [Min Max]
[8 20] (b)

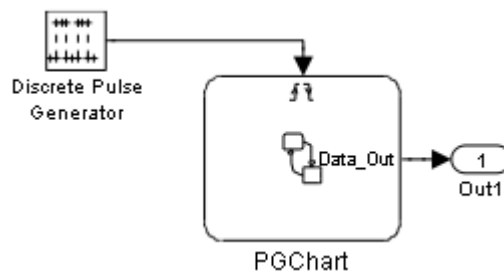
Wait Range [Min Max]
[5 20] (c)

Destination Range [Min Max]
[1 4] (d)

Data Range [Min Max]
[1 5] (e)

Max. Priority Number (<99)
98 (f)

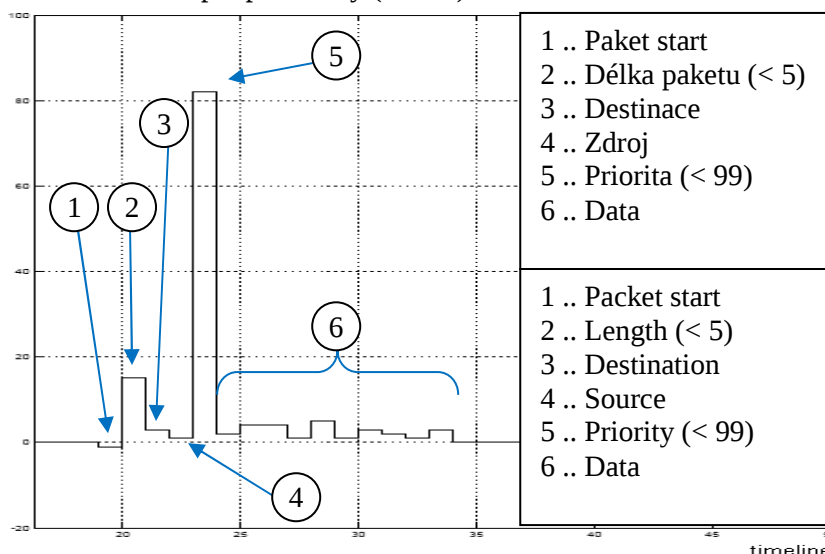
Obr. 3 Ukázka nastavení parametrů pro blok Packet Generator pomocí masky.



Obr. 4 Obsah subsystému Packet Generator.

Obr. 4 ukazuje, že se blok generátoru paketů skládá z Generátoru diskretního pulsu (Discrete Pulse Generator), který udává takt simulace a z tabulky PGChart. Tato tabulka obsahuje tzv. Stateflow diagram, jehož podrobnější popis je možno najít v kapitole 2.1.1.

Výstup z bloku Packet Generator (Out1) obsahuje signál jehož ukázka je na Obr. 5. Prvotní výstupek signálu je tzv. Paket start, podle kterého je možno rozeznat začátek nového paketu. Hodnota Paket start je nastavena na hodnotu -1. Ostatní vrcholy signálu jsou náhodně generované s ohledem na vstupní parametry (Obr. 3).



Obr. 5 Ukázka výstupního signálu z bloku Packet Generator.

2.1.1 Stateflow diagram (PGChart)

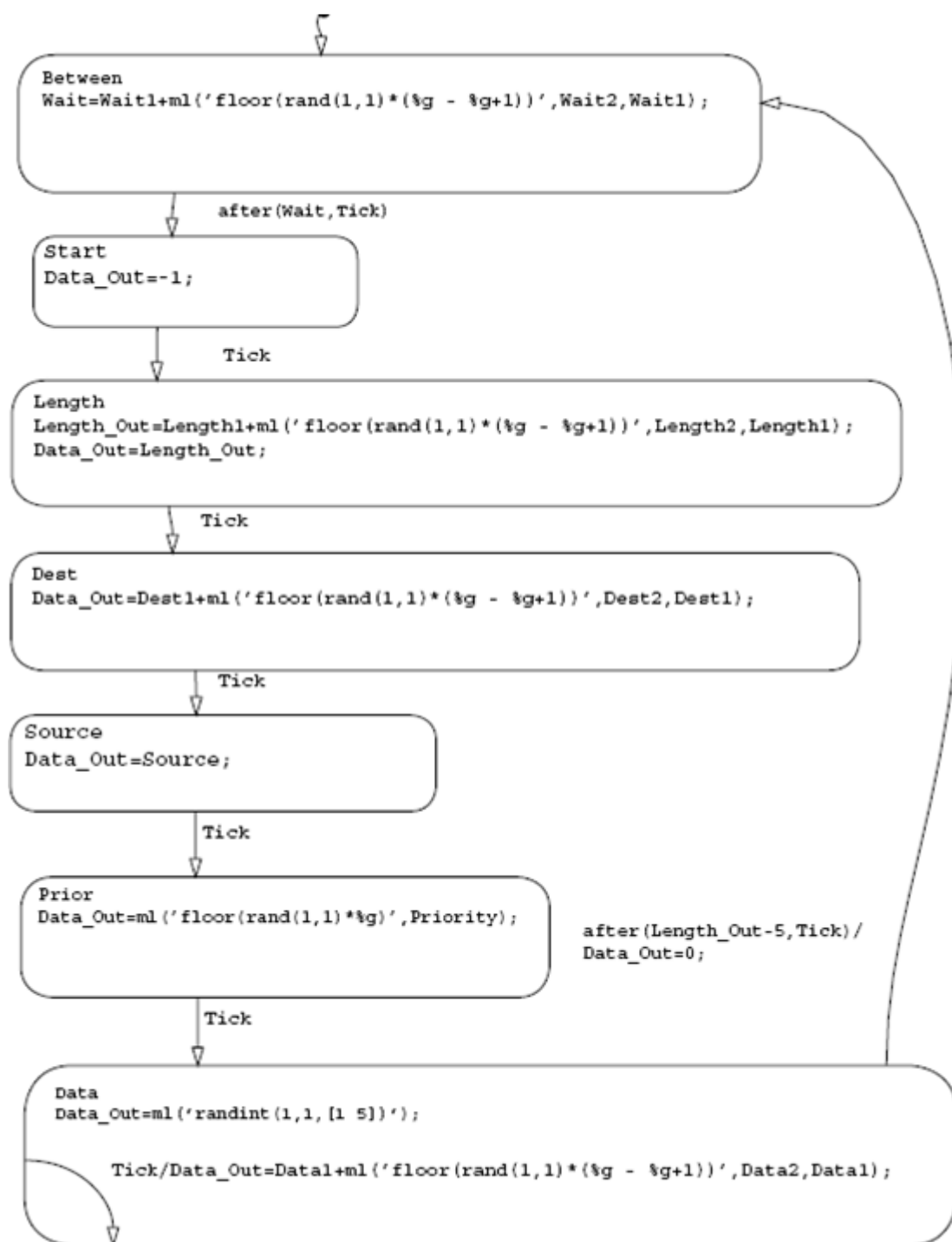
Stateflow Toolbox programu Simulink obsahuje Stateflow diagram (Stateflow Chart). Více informací o funkcích a použití Toolboxu je možno vyhledat v Stateflow helpu [1].

Tabulka PGChart bloku Paket generátor obsahuje 7 stavů (viz Obr. 6), které generují signál příchozího paketu. PGChart disponuje, mimo jiné, proměnnou Data_Out, která je nastavena na výstup z tabulky PGChart a tudíž i na výstup z nadřazeného bloku Packet Generator. Stavby tabulky PGChart jsou následující:

- Between - Při inicializaci tabulky se aktivuje stav nazvaný Between, kde se čeká na uplynutí náhodně vygenerovaného počtu takzvaných Tick pulsů, které jsou generovány v již zmíněném Generátoru diskretního pulsu. Takto náhodně vygenerovaný počet je v rozmezí, které je zadáno jako parametr (Wait Range) v masce tohoto bloku.
- Start - Po uplynutí požadovaného zpoždění paketu jsou generována data prostřednictvím proměnné Data_Out. Stav tabulky, nazvaný Start, generuje signál o hodnotě -1 (začátek paketu).
- Length - Dalším stavem je stav Length. Zde se do proměnné Data_Out a tedy i na výstup z tabulky odešle délka paketu, která je rovněž generována náhodně z rozmezí zadaného v masce bloku.
- Dest - Následující stav nazvaný Dest generuje na výstup hodnotu určující příjemce, tedy destinaci (Destination), kam má být paket přesměrován, opět generovanou náhodně v rozmezí zadaném v masce bloku.
- Source - Tento stav generuje jako výstupní hodnotu z bloku takovou hodnotu, která bude udávat číselné označení odesílatele, ze kterého je paket poslán. Zde je pouze přiřazení parametru Source z masky bloku do proměnné Data_Out.

- f) Prior - Výstupní proměnné Data_Out přiřadí hodnotu požadované priority, která je vybrána náhodně z rozmezí parametru Priority, jež obsahuje maska bloku.
- g) Data - Generuje výstupní signál o náhodné hodnotě vybrané z rozmezí udaném v masce bloku (Data Range). Tento krok se opakuje tolikrát, dokud není splněna podmínka, že délka paketu je rovna požadované délce. Po splnění této podmínky se nastaví proměnná Data_Out na nulovou hodnotu a celý cyklus se opakuje od bodu a).

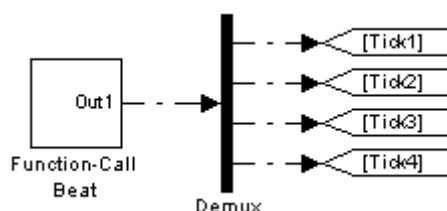
Každý ze stavů tabulky PGChart se provede na začátku nové události (Event), která se v tomto případě jmenuje Tick, to znamená, že pokud přijde do tabulky signál z Discrete Pulse Generatoru, přejde se na následující stav tabulky podle hrany (šipky) grafu. Tímto způsobem je zajištěno časování dat a jejich posloupnost.



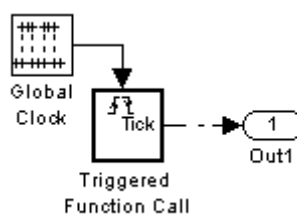
Obr. 6 Stateflow tabulka (PGChart).

2.2 Podstým Volání funkce (Function-Call Beat)

Do tohoto bloku (viz Obr. 7) nejsou přiváděna žádná vstupní data. Výstupem z bloku (Out 1) je pulsující signál, který pulsuje v taktu simulace. Každý jeden takt přivádí na tento výstup informaci o spuštění funkce. Obsah tohoto subsystému je zobrazen na Obr. 8.



Obr. 7 Subsystém Function-Call Beat a zpracování jeho signálu blokem Demux.

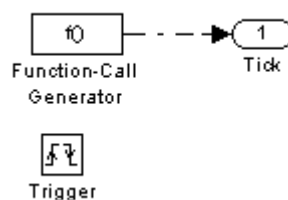


Obr. 8 Obsah subsystému Function-Call Beat.

Na výstupu z bloku Global Clock je diskretní impulsní signál o amplitudě 1. Tento signál je veden to subsystému Triggered Function Call.

2.2.1 Podstým Spouštění volání funkce (Triggered Function Call)

Tento subsystém (Obr. 9) je takzvaně spouštěný (Triggered). Hlavním znakem spouštěného systému je zahájení jeho činnosti až při dodání signálu na vstup Trigger. Je-li v tomto případě na vstup Trigger přiveden signál z bloku Global Clock, pak se provede vygenerování signálu blokem Function-Call Generator a tento signál je veden na výstup ze subsystému Triggered Function Call, tedy i z celého subsystému Function-Call Beat.

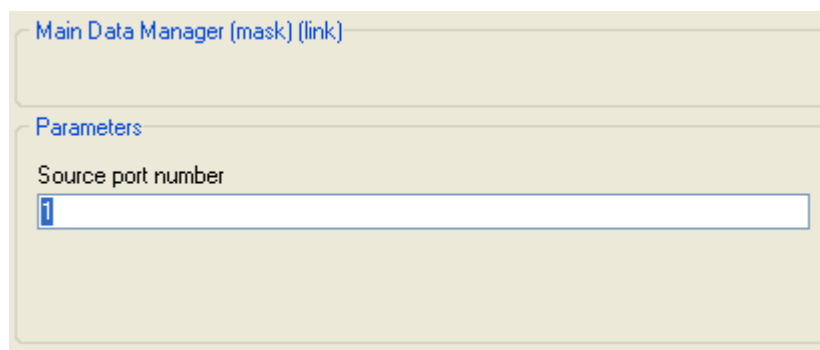


Obr. 9 Obsah subsystému Triggered Function Call.

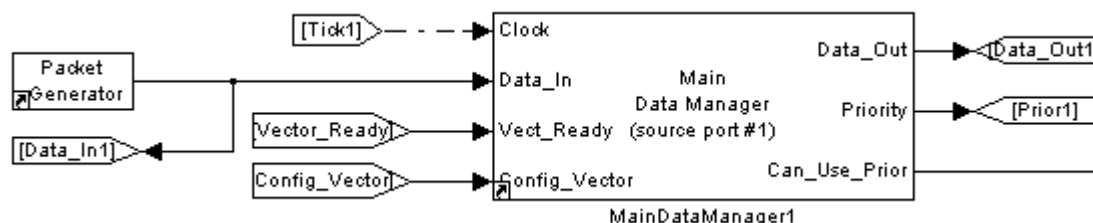
Jakmile je ze subsystému Function Call Beat uvolněn signál následuje jeho kopie na více stejných signálů pomocí bloku Demux.

2.3 Podsystem Hlavní manažer dat (Main Data Manager)

Subsystem Hlavní manažer dat je v modelu tolik, kolik má přepínač portů (viz Obr. 2). Jsou zde tedy 4 stejné subsystemy nazývané Main Data Manager (1, 2, 3 a 4), přičemž každý z nich musí mít v masce bloku zadaný parametr, nazvaný Číslo zdrojového portu (Source Port Number), na unikátní číslo v rozmezí od 1 do počtu portů přepínače. Tento parametr by měl mít hodnotu totožnou s číslem vstupu do přepínače, s nímž je spojen.



Obr. 10 Maska pro nastavení parametru subsystemu Main Data Manager.



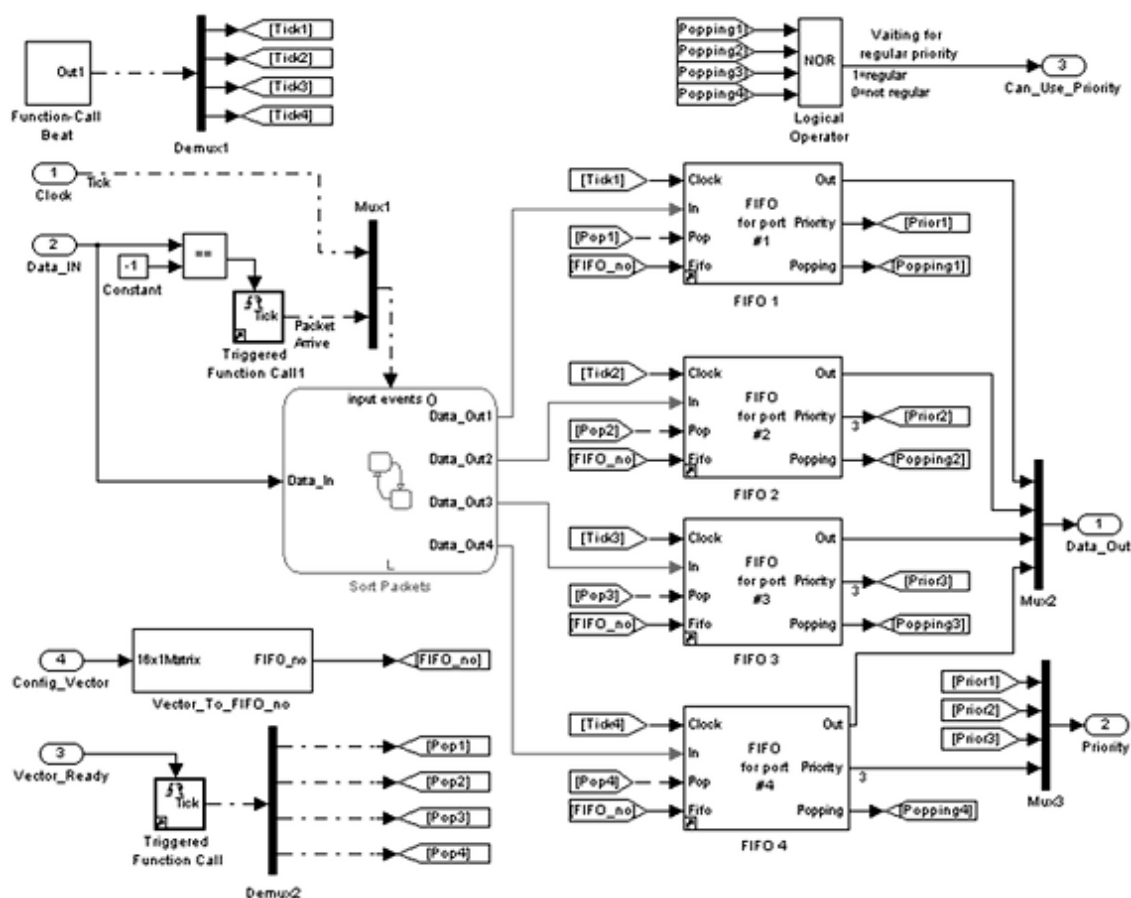
Obr. 11 Jeden ze subsystemů Main Data Manager nastaven na vstupní port číslo 1.

Subsystem Main Data Manager pracuje se čtyřmi vstupy a třemi výstupy (Obr. 11). Názvy vstupů a jejich požadavky jsou následující:

- Clock – Jedná se o diskretní spouštění funkce na základě taktu simulace.
- Data_In – Data přicházející od odesílatele na vstupní port. V případě simulace se jedná o data přicházející z bloku Packet Generator (Popsaný v kapitole 2.1).
- Vector_Ready – Pokud je konfigurační vektor (Config_Vector) připraven k použití (je ukončen jeho výpočet), pak je signál přichází do vstupu Vector_Ready nastaven na logickou hodnotu pravda (true), resp. 1. Tento signál přichází z bloku Hopfield popsany v kapitole 2.5.
- Config_Vector – Vstupem je konfigurační vektor, přicházející z bloku Hopfield (kapitola 2.5).

Popis výstupů z bloku Main Data Manager:

- Data_Out – Na tento výstup jsou posílána data, která obsahují data jednoho paketu, který je právě vybrán a vyslán.
- Priority – Složky výstupního vektoru Priority obsahují hodnoty priorit paketů, čekajících na odbavení.
- Can_Use_Prior – Je – li možno použít vektor priorit na výstupu Priority, pak je tato hodnota logická pravda (true), resp. 1.



Obr. 12 Obsah subsystému Main Data Manager.

Funkce bloku Main Data Manager lze rozdělit do dvou hlavních skupin.

První z nich spočívá v přijetí příchozího paketu a jeho umístění do jedné ze čtyř front (FIFO). O rozhodnutí, do jaké fronty bude paket vložen, se stará Stateflow tabulka nazvaná Sort Packets (viz kapitola 2.3.3). Pakety rozděluje podle jejich požadované destinace (číslo výstupu z přepínače), jejíž hodnota je v paketu uložena. Po rozdělení je paket uložen (PUSH) do fronty FIFO1, FIFO2, FIFO3, nebo FIFO4. Počet těchto front musí být opět roven počtu výstupů (vstupů) z přepínače.

Další, druhá, zásadní funkce bloku je vybrání požadovaného paketu (POP) z fronty. Otázku, z jaké fronty FIFO vybrat paket, řeší subsystém Vector_To_FIFO_no (popsán v kapitole 2.3.5). Vstupem do tohoto subsystému je konfigurační vektor přijatý z bloku Hopfield (blok je popsán v kapitole 2.5. Blok Vector_To_FIFO_no posílá na výstup číselnou hodnotu, která udává číslo fronty FIFO, ze které se má paket vybrat (POP). Vybrání paketu se uskuteční po doručení signálu Vector_Ready, který vyšle blok Hopfield, jakmile jsou data konfiguračního vektoru regulární.

2.3.1 Podsystem Volání funkce (Function-Call Beat)

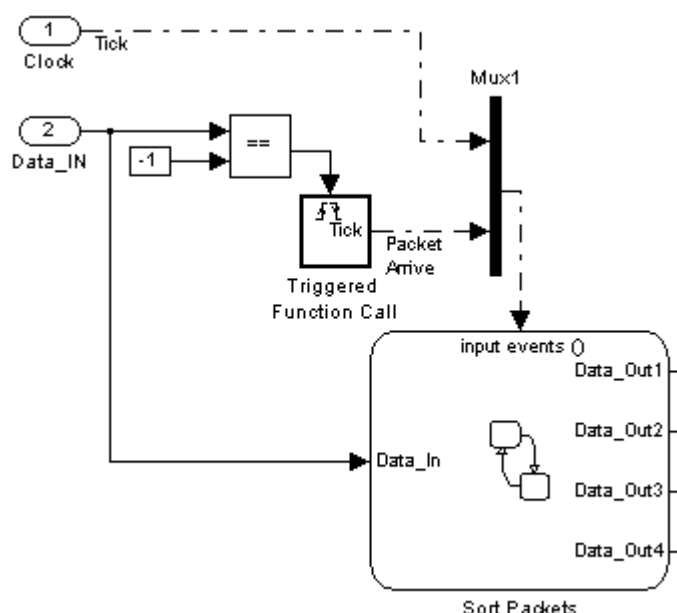
Funkce tohoto subsystému je již popsána v kapitole 2.2.

2.3.2 Podsystem Spouštěné volání funkce (Triggered Function Call)

Funkce tohoto subsystému jsou popsány v kapitole 2.2.1.

2.3.3 Stateflow diagram Rozděl pakety (Sort Packets)

Subsystem Sort Packets (Obr. 13) rozděluje pakety do jednotlivých front FIFO podle jejich destinace. Všechny pakety se stejnou destinací jsou kumulovány do jedné fronty FIFO přes výstupy tabulky Data_Out1, Data_Out2, Data_Out3 a Data_Out4.



Obr. 13 Stateflow blok Sort Packets a bloky pro úpravu jeho vstupních dat.

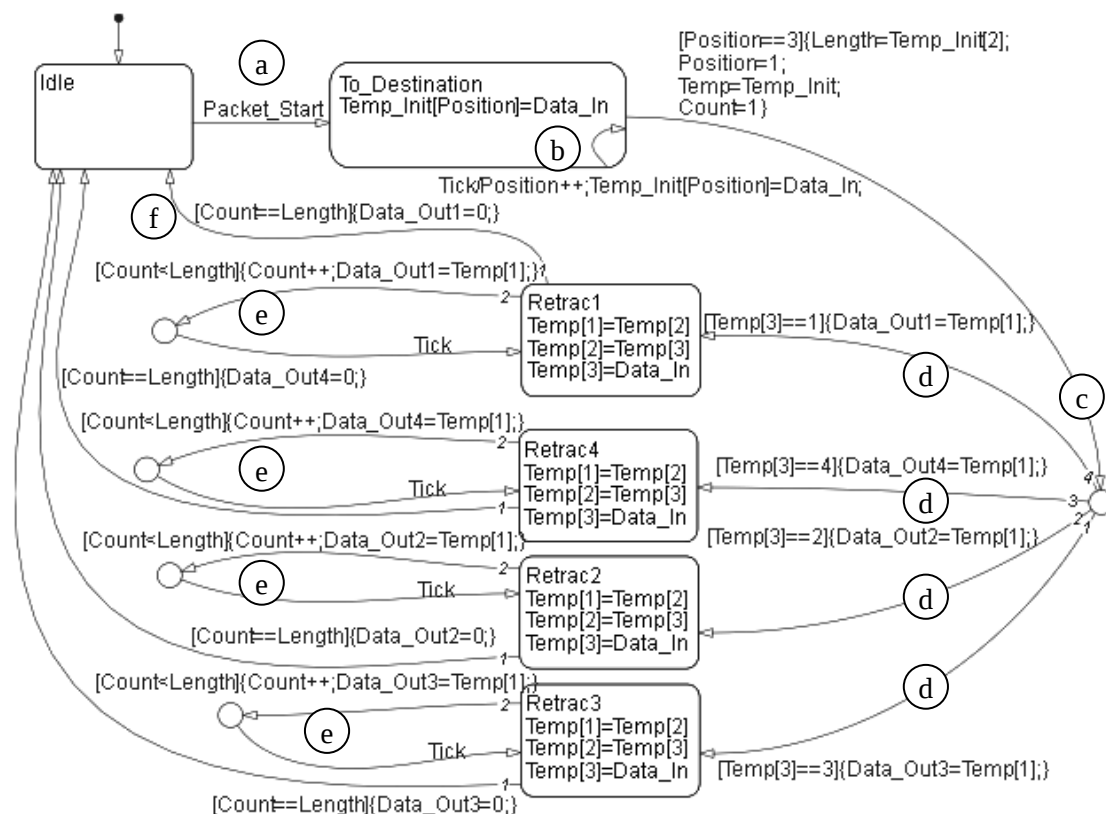
Na vstup tabulky Sort Packets jsou přiváděna data Data_In, která generuje Packet Generátor. Události tabulky (Events) jsou spojeny blokem Mux1 do jednoho signálu. První z událostí je tzv. Tick, která udává takt simulace a je volána jako Function Start přes vstup bloku Main Data Manager nazývaný Clock. Další událost je zavolána také signálem Function Start, ten se produkuje v subsystému Triggered Function Call, na jehož vstup musí být přivedena logická pravda (true, resp. 1). To nastane pouze za předpokladu, že vstupní hodnota Data_IN bude rovna -1, to indikuje začátek paketu. Subsystem Triggered Call Function je použit již v kapitole 2.2.1, kde je vysvětlena jeho funkce.

Na obrázku Obr. 14 Stateflow diagram bloku Sort Packets. je zobrazen diagram tabulky Sort Packets [1].

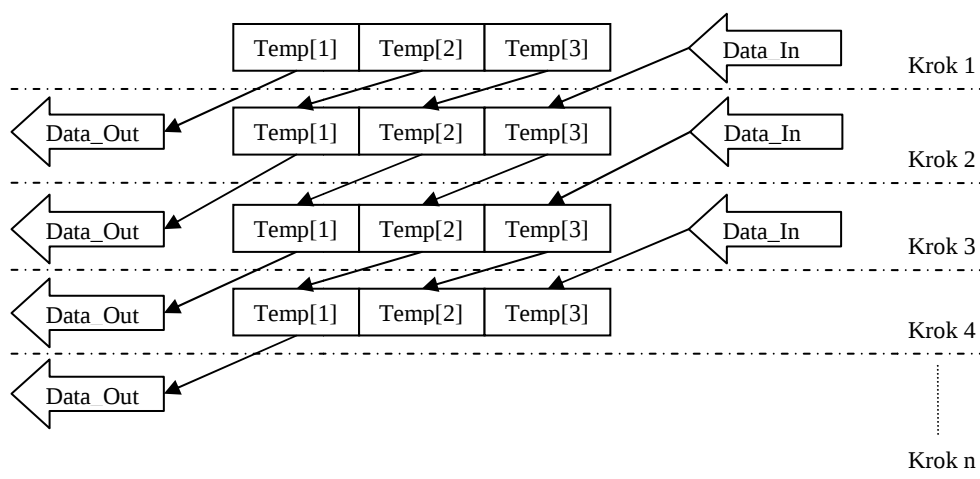
- Při inicializaci se aktivuje stav nazvaný Idle. Po příchodu signálu spuštění funkce (Function Start), jako událost Packet_Start, na vstup Events se aktivuje následný stav
- To_Destination. Zde se do pole proměnných (dále jen pole) Temp_Init vloží na první pozici (Position) první hodnota příchozího signálu paketu. Pokud není proměnná Position rovna 3, pak se bod b) opakuje s inkrementovanou hodnotou proměnné Position v časovém taktu události Tick. V opačném případě jsou v poli

Temp_Init uloženy 3 počáteční hodnoty paketu, všechny se zkopírují do nového pole Temp proměnné Position, Count se nastaví na hodnotu 1 a proměnná Length se nastaví na hodnotu délky paketu, která je uložena v poli Temp_Init na pozici 2. Dále se aktivuje následná tzv. spojka (Junction).

- c) Ze spojky je možno aktivovat 4 různé stavy (Retrac1, Retrac2, Retrac3 a Retrac4), podle toho na jaké číslo výstupního portu přepínače má být paket odeslán. Pro rozhodnutí slouží podmínka, je – li hodnota pole Temp na pozici 3, což je právě destinace paketu, rovna 1, 2, 3, nebo 4. Těsně před aktivací stavu Retrac se na výstup z tabulky Data_Out posílá první hodnota paketu, tedy ta, která je uložena v poli Temp na pozici 1.
- d) Po dosažení jednoho ze stavů Retrac se přeskupí data tak, že se na první pozici pole Temp vloží hodnota druhé pozice pole, na druhou pozici pole se vloží hodnota třetí pozice pole a do třetí pozice se vloží nová hodnota příchozí ze vstupu v proměnné Data_In (Obr. 15).
- e) Pokud vyhovuje podmínka, že hodnota čítače Count je menší než délka paketu Length, pak se hodnota čítače inkrementuje a výstupní proměnné Data_Out se přiřadí data pole Temp na nejnižší pozici, tedy 1 (Temp[1]). Čeká se na událost Tick, po níž se opět aktivuje stav Retrac, cyklus se vrací na bod d).
- f) Pokud podmínka z předchozího bodu e) není splněna, pak je splněna podmínka: Proměnná Count je rovna proměnné Length. V tomto případě je už celý paket odeslán na požadovaný výstup tabulky, proměnná Data_Out se vynuluje a přejde se do stavu Idle, tedy na bod a).



Obr. 14 Stateflow diagram bloku Sort Packets.



Obr. 15 Ukázka přeskupení dat ve stavu Retrac v tabulce Sort Packets.

2.3.4 Podsystem Fronta (FIFO)

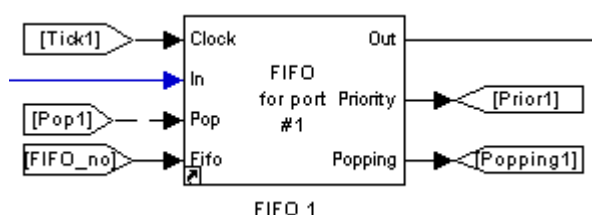
V každém ze čtyř subsystémů Main Data Manager jsou čtyři subsystémy FIFO (FIFO1, FIFO2, FIFO3 a FIFO4). Jedná se o speciální datové fronty, lišící se od obyčejné fronty tím, že je možno nedestruktivně číst potřebná data prvního prvku fronty. Jako prvky fronty jsou zde myšleny pakety a potřebná data jsou především priority paketů.

Subsystém FIFO má 4 vstupní a 3 výstupní proměnné (Obr. 16).

- První ze vstupů je časovač (Clock), do kterého je posílán diskretní signál Function Start z bloku Function-Call Beat (kapitola 2.3.1).
- Další vstup je nazván In. Zde jsou očekávána data paketů přicházející z bloku Sort Packets (kapitola 2.3.3).
- Do vstupu Pop je poslán signál Function Call, generovaný blokem Triggered Function Call (funkce bloku je popsána v kapitole 2.3.2), právě pokud je nutno provést vybrání (POP) z fronty. Tento signál je posílán do všech čtyř front zároveň. Ze které fronty bude proveden výběr uvádí až následující vstupní signál.
- Na vstup Fifo je posíláno číslo fronty (FIFO_no). Signál na tento vstup je generován blokem Vector_To_FIFO_no (kapitola 2.3.5). Pokud je toto číslo totožné s číslem fronty a zároveň je zavolána Function Call na vstupu Pop, pak se provede výběr (POP) z této fronty. Zmiňované číslo fronty je zadáno prostřednictvím masky bloku jako parametr FIFO Number (Obr. 17).

Výstupní data z bloku FIFO jsou následující:

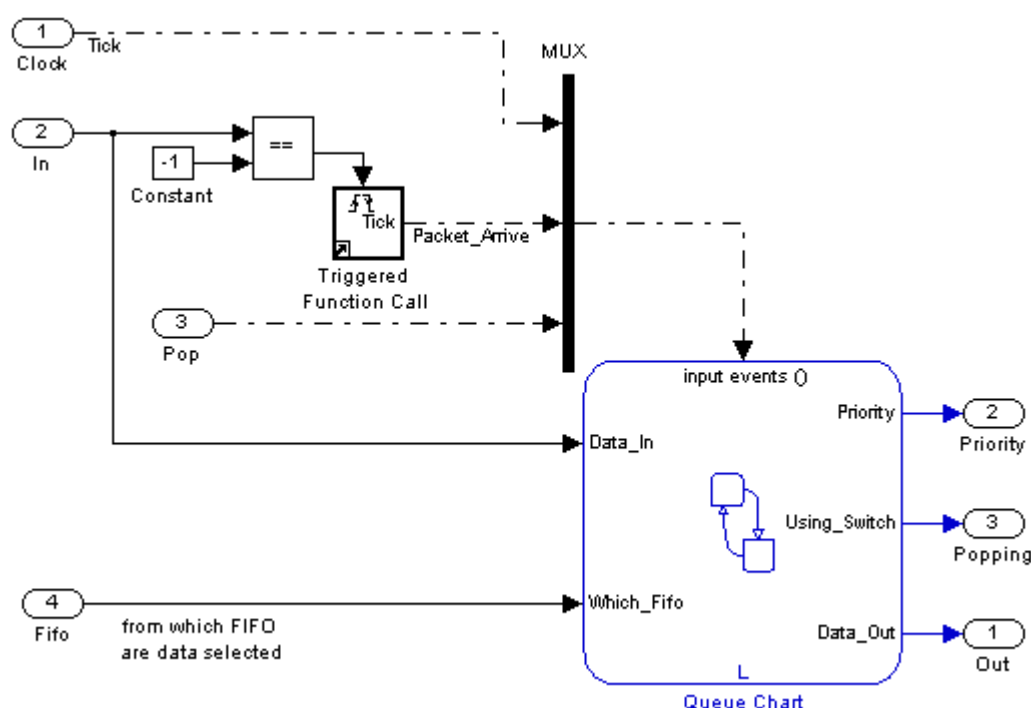
- Out – Přes tento výstup jsou posílána data paketů (v každém cyklu se jedná nejvýše o jeden celý paket), čtená z fronty. Další zpracování těchto dat má za úkol blok Mux2, který spojí všechny signály posílané z každé fronty FIFO do jednoho signálu, respektive do vektoru. Přesto, že je vektor vytvořen ze čtyř datových toků, jedná se vždy pouze o vektor velikosti 1x1. Tato vlastnost je zapříčiněna tím, že se vysílají Data vždy pouze z jedné fronty FIFO každého bloku Main Data Manager.
- Priority – Z výstupu Priority se čte hodnota priority prvního prvku fronty. O další zpracování priorit z každé fronty FIFO se stará blok MUX3, který vytvoří z jednotlivých hodnot priorit vektor priorit o velikosti 1x4 a odešle jej na výstup z bloku Main Data Manager pojmenovaný Priority (Obr. 12). Aby byla data vektoru regulérní je nutno zajistit, aby se priority nepoužili současně, probíhá – li v jakékoli frontě FIFO výběr dat POP (data se odesílají na port Out). Takový stav lze předcházet pomocí následujícího výstupu.
- Popping – Tento výstup vysílá logickou pravdu (true, 1), pokud ve frontě probíhá výběr POP. Pokud výběr (POP) neprobíhá, vysílá logickou nepravdu (false, 0). Od každé ze čtyř front je signál zpracován v logickém operátoru NOR, z něhož vystupuje jediná logická hodnota, která určuje, zda je vektor priorit regulární (lze jej použít pro další výpočty), nebo zda probíhá v jedné z front výběr (POP) a tedy nejsou zaručena regulární data vektoru priorit. Tato logická hodnota je z bloku Main Data Manager odeslána přes výstup Can_Use_Priority (Obr. 12).



Obr. 16 Subsystém FIFO, jeho vstupy a výstupy (konkrétně FIFO 1).



Obr. 17 Zadání parametru FIFO Number do masky bloku FIFO



Obr. 18 Obsah subsystému FIFO.

Na Obr. 18 je znázorněn obsah jednoho ze subsystémů FIFO. Parametr této fronty FIFO je číslo fronty (Obr. 17), respektive číslo příjemce, ke kterému budou pakety z fronty odesílány. Subsystém FIFO obsahuje Stateflow tabulku Fronta (Queue Chart), která je popsána v následující kapitole 2.3.4.1.

2.3.4.1 Stateflow diagram Fronta (Queue Chart)

Tabulka Queue Chart pracuje s dvěma vstupy, třemi událostmi a data odesílá na tři výstupní porty [1].

Funkční události vstupující do tabulky Queue Chart jsou spojeny blokem MUX.

- První událostní funkce se jmenuje Tick a je aktivována při každém taktu simulace. Jedná se o využití bloku Function-Call Beat (popsaný v kapitole 2.3.1), jehož signál je poslán z bloku Main Data Manager přes vstupní port bloku FIFO Clock.
- Další událost se jmenuje Packet_Start a je aktivována blokem Triggered Function Call, jehož funkce je vysvětlena v kapitole 2.3.2. Tento blok odešle signál Function Start, pokud je na jeho vstup Trigger poslána logická hodnota pravda (true, 1). To se

uskuteční, je - li splněna podmínka rovnosti hodnoty na vstupu In (bloku FIFO) s hodnotou -1 (jedná se o počátek paketu).

- c) Třetí z událostí má název Start_Popping. Jedná se o příchozí Function Start signál z bloku Main Data Manager přes vstupní port Pop.

Následuje popis dvou vstupních proměnných tabulky Queue Chart:

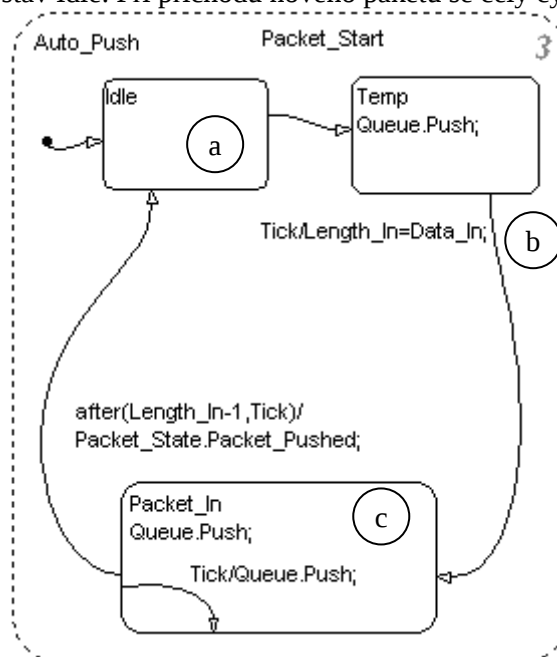
- d) Na proměnnou Data_In se navazují hodnoty příchozí přes vstupní port In bloku FIFO. Jedná se o data příchozích paketů s totožnou hodnotou destinace (číslo portu příjemce).
- e) Proměnná Which_Fifo tabulky Queue Chart může nabývat hodnoty 1 až 4, podle toho, ze které fronty má být paket vybrán (POP). Pokud je hodnota proměnné totožná s parametrem FIFO Number a zároveň je aktivní událost Start_Popping, pak nastane výběr paketu z fronty.

Tři výstupní proměnné tabulky jsou přímo spojeny s třemi výstupy bloku FIFO, které jsou popsány již v předchozí kapitole 2.3.4 (body e, f, g).

Diagram Stateflow tabulky Queue Chart je rozdělen na 4 celky nazvané Auto_Pop, Queue, Auto_Push a Packet_State.

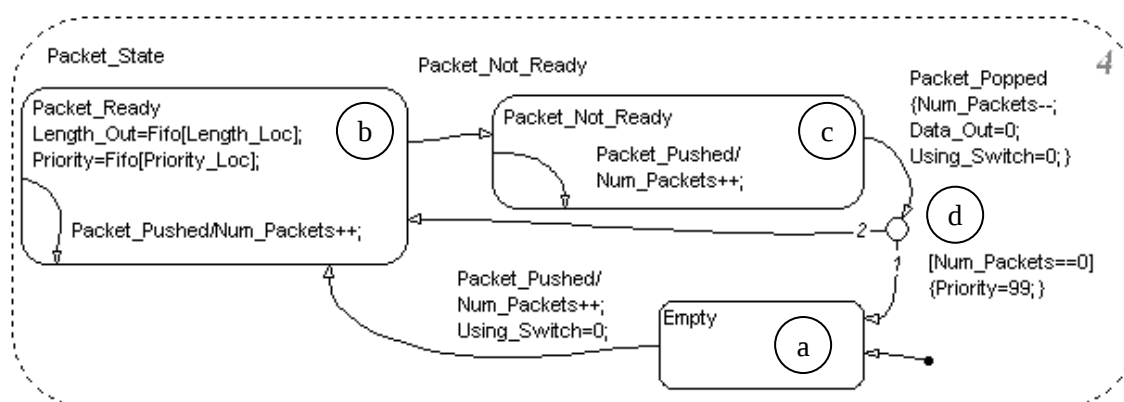
- 1) Auto_Push (Obr. 19) obsahuje 3 stavy: Idle, Temp a Packet_In.

- a) Při inicializaci tabulky se aktivuje stav Idle.
- b) Začátek nového příchozího paketu je indikován událostí Packet_Start, po níž se přejde na stav Temp, zde se provede funkce Queue.Push s návratem na výchozí pozici, tedy stav Temp. Takový zápis s tečkovou notací znamená přejít na celek Queue s událostí Push, kde se uloží hodnota vstupní proměnné Data_In do proměnné typu pole (celek Queue je vysvětlen v jednom z dalších bodů). Při události Tick se proměnné Length_In přidělí hodnota Data_In, která v daný okamžik obsahuje hodnotu velikosti paketu.
- c) Následně se aktivuje stav Packet_In v němž se cyklicky provádí funkce Queue.Push až do doby, kdy je počet spuštěných událostí Tick totožný s hodnotou proměnné Length_In-1. Poté je celý paket uložen ve frontě, nastaví se aktuální stav celku Packet_State na stav Packet_Pushed, příkazem Packet_State.Packet_Pushed a aktivuje se stav Idle. Při příchodu nového paketu se celý cyklus 1) opakuje.



Obr. 19 Celek Auto_Push tabulky Queue Chart.

- 2) Packet_State (Obr. 20) je tvořen třemi stavy Packet_Ready, Packet_Not_Ready a Empty.
- Při inicializaci tabulky se aktivuje stav Empty. Jakmile je vložen první paket do fronty, je aktivována událost Packet_Pushed, inkrementuje se hodnota vnitřní proměnné Num_Packets, která udává počet uložených paketů ve frontě, hodnota výstupní proměnné Using_Switch se změní z logické 1 na logickou 0 a aktivuje se stav Packet_Ready.
 - Packet_Ready - Nastavovaným proměnným Length_Out a Priority bude věnována pozornost v následujícím popisu celku Queue (3f). Ve stavu Packet_Ready se stagnuje do doby, kdy je aktivována událost Packet_State.Packet_Not_Ready z celku Auto_Pop (viz níže).
 - Stav Packet_Not_Ready je aktivní za předpokladu, že se provádí výběr (POP) z fronty. Pokud je v tomto případě nutno vložit paket do fronty (PUSH), pokračuje se z aktuálního stavu přes zpětnou vazbu (šipka uvnitř stavu) s opětovným návratem do stavu Packet_Not_Ready a inkrementací hodnoty Num_Packets. Po dokončení akce POP je v celku Auto_Pop aktivována událost Packet_State.Packet_Poped. V tomto případě se přejde v celku Packet_State do spojky (Junction) s dekrementací hodnoty proměnné Num_Packets, nulováním proměnné Data_Out a přidělením logické 0 do proměnné Using_Switch.
 - Následuje rozhodnutí mezi aktivací stavu Empty, to za předpokladu, že je proměnná Num_Packets rovna nule (fronta je prázdná), a stavu Packet_Ready, to při nesplnění předchozí podmínky. Dále se cyklicky postupuje od bodu 2a, respektive 2b.

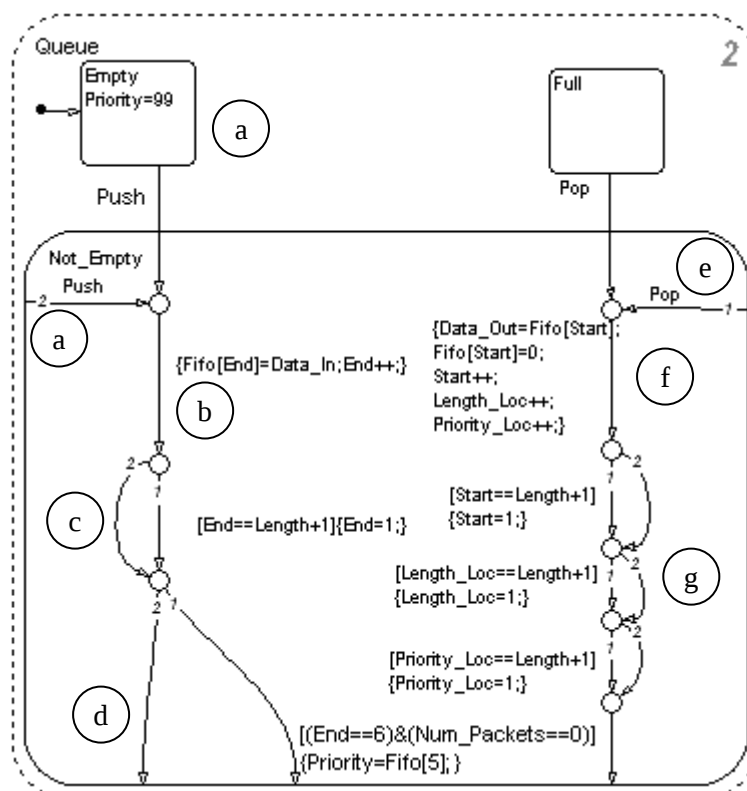


Obr. 20 Celek Packet_State tabulky Queue Chart.

- 3) Diagram celku Queue je zobrazen na Obr. 21. Akce pro vložení dat (PUSH) jsou znázorněna na levé polovině obrázku a jedná se o následující body a) až d). Naopak v pravé polovině se provádějí akce pro výběr dat z fronty (POP), popsané v bodech e) až g).
- Při inicializaci simulace je aktivován stav Empty a proměnná Priority nastavena na nejvyšší možnou hodnotu 98.
 - Při příchodu události Queue.Push z celku Auto_Push (1b) se aktivuje stav Not_Empty. Proveďte se vložení dat do pole Fifo, na pozici, jejíž hodnotu obsahuje proměnná End (Fifo[End]). Vložená data jsou příchozí data ze vstupní proměnné Data_In. A poté se inkrementuje hodnota proměnné End.
 - Následně se zjišťuje podmínka, zda je proměnná Length, což je maximální možná velikost fronty (pole Fifo), rovna pozici End. Při splnění podmínky se proměnná End nastaví zpět na hodnotu 1 a data se zapisují na začátek fronty. V případě nesplnění podmínky se následuje na další spojku (Junction) bez provedení jakékoli

operace.

- d) Poslední podmínkou pro vkládání dat do fronty je podmínka: Je – li hodnota proměnné End = 6 a zároveň Num_Packets = 0 (ve frontě není žádný paket), pak priority = Fifo[6] (Výstupní proměnná Priority je rovna prioritě právě vkládaného paketu). Tato operace se provede pouze, když se vkládá do prázdné fronty nový paket. V případě opačném je výstupní proměnná Priority nastavována vždy po vyzvednutí (POP) paketu z fronty (viz. níže).
- e) Při výběru dat z fronty se provádějí funkce stavů v celku Auto_Pop (viz. odrážka 4), odkud je odeslána událost Queue.Pop. Po příchodu této události do celku Queue je aktivována první spojka (Junction) a následně další akce.
- f) Data_Out = Fifo[Start]. Takový zápis znamená, že se výstupní proměnné Data_Out přidělí hodnota fronty Fifo, jež je na pozici Start (Inicializační hodnota Start je rovna jedné). Hodnota fronty Fifo se na této pozici nuluje (Fifo[Start] = 0). Inkrementuje se hodnota proměnné Start, Length_Loc (pozice hodnoty délky paketu) a Priority_Loc (pozice hodnoty priority paketu). Dvě posledně zmíněné proměnné jsou použity po odebrání celého paketu, kdy se v celku Packet_State aktivuje stav Packet_Ready. Pak jsou nastaveny výstupní proměnné Priority a Length_Out na prioritu a délku následujícího paketu ve frontě.
- g) Následující 3 podmínky kontrolují, zda nedojde ke čtení dat z koncové + první pozice fronty, v takovém případě se přeskočí na začátek fronty a data se čtou odtud. Je to odezva na vkládání paketu na začátek fronty, je – li fronta zaplněna.

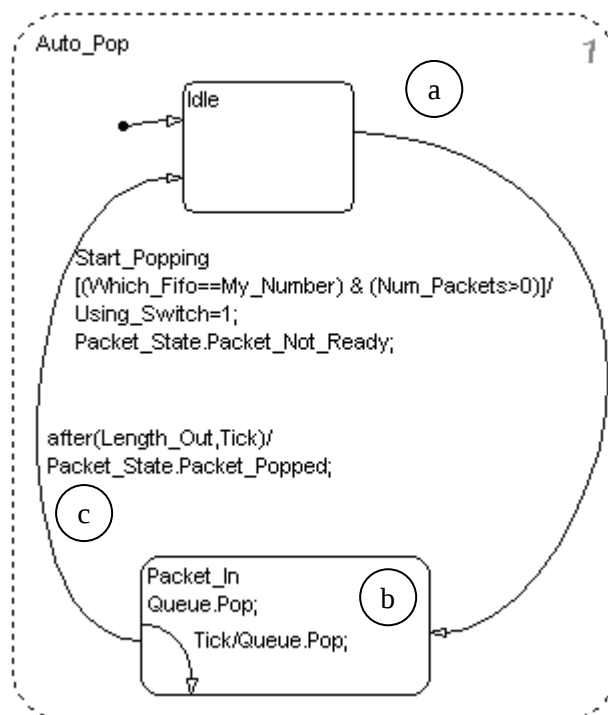


Obr. 21 Celek Queue tabulky Queue Chart.

- 4) Diagram celku Auto_Pop je vyobrazen na Obr. 22.
 - a) Při inicializaci simulace se aktivuje stav Idle. Po přijetí vstupní události Start_Popping je kontrolováno, zda souhlasí číslo bloku FIFO (FIFO Number) s hodnotou na vstupní proměnné Which_Fifo. Pokud je podmínka splněna,

následuje přiřazení logické 1 proměnné Using_Switch, zavolání události Packet_State.Packet_Not_Ready a přechod do stavu Packet_In.

- b) Ve stavu Packet_In se cyklicky provádí událost Queue.Pop v taktu Tick až do doby, kdy je počet opakování Tick roven hodnotě délky paketu Length_Out.
- c) Při dosažení předchozí rovnosti je aktivována událost Packet_State.Packet_Popped a aktivuje se opět stav Idle. Při následujícím požadovaném výběru paketu z fronty se cyklus opakuje od bodu 4a).



Obr. 22 Celek Auto_Pop tabulky Queue Chart.

2.3.5 Podsystem Překladač config. vekt. na číslo fronty (Vector_To_FOFO_no)

Subsystem Vector_To_FIFO_no (Obr. 23) očekává na svůj jediný vstup 16 - ti prvkový konfigurační vektor *c*, který vytváří blok Hopfield (kapitola 2.5). Výstupem ze subsystemu je číselná hodnota v rozmezí 1 až 4, která udává číslo fronty FIFO, z níž má být vybrán (POP) paket.

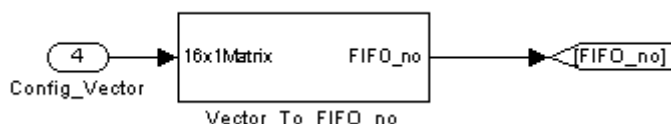
Hodnotami konfiguračního vektoru mohou být pouze čísla 1 a -1. Pro jednodušší představu uveďme, že konfigurační vektor je vytvořen z konfigurační matice *C* o velikosti 4 x 4 tak, že každý řádek matice je vložen za předchozí řádek matice. Pak platí pravidla konfigurační matice:

- a) V každém řádku matice musí být přítomna pouze jedna hodnota 1. Ostatní jsou -1.
- b) V každém sloupci matice musí být přítomna pouze jedna hodnota 1. Ostatní jsou -1.
- c) Mějme řádky očíslované od 1 do 4, pak každý řádek bude zpracováván právě jedním blokem Main_Data_Manager s totožným číselným označením.
- d) Mějme sloupce očíslované od 1 do 4, pak každý sloupec bude zpracován všemi frontami FIFO s totožným číselným označením.

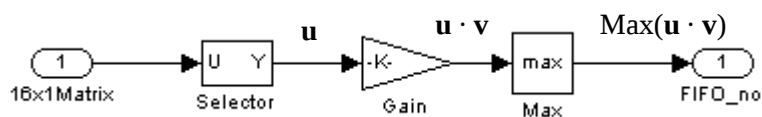
Z těchto pravidel lze určit, že první řádek matice bude propustný pro subsystem Vector_To_FIFO_no prvního bloku Main Data Manager 1, druhý řádek bude propustný pro

subsystém Vector_To_FIFO_no druhého bloku Main Data Manager 2, atd. V každém jednom bloku Main Data Manager je nutno celý řádek (tedy sloupce 1 až 4 téhož řádku) násobit vektorem $\mathbf{v} = [1 \ 2 \ 3 \ 4]$. Dostáváme 3 hodnoty menší jak nula a jednu hodnotu větší nebo rovno nule. Taková hodnota, tedy větší nebo rovna nule, udává číslo fronty (FIFO_no), ze které má být paket vybrán.

Postup podobný je použit v subsystému Vector_To_FIFO_no, pouze se počítá s prioritním vektorem $\mathbf{c}$ namísto prioritní matice $\mathbf{C}$.



Obr. 23 Subsystém Vector_To_FIFO_no.



Obr. 24 Obsah subsystému Vector_To_FIFO_no.

Na Obr. 24 je zobrazen obsah subsystému, který z vektoru priorit určí číslo fronty, ze které má být vybrán (POP) paket.

Prvním blokem za vstupem 16x1Matrix je tzv. Selektor.

Selector dokáže vyfiltrovat pouze požadované prvky vektoru. Nastavení parametrů je ukázáno na Obr. 25. Parametr Number of input dimensions udává počet dimenzí vstupní matice (v tomto případě vektoru se jedná o jednu dimenzi). Index mode určuje, jakým indexem začíná číslování vektoru. Zřejmě nejdůležitější parametr je Index, který určuje, jaké hodnoty (na jakých pozicích) mají být vybrány.

Vysvětlení $((My_Number - 1) \cdot 4) + [1 \ 2 \ 3 \ 4]$:

Proměnná My_Number je unikátně zadána v parametru bloku Main Data Manager a hodnoty v hranaté závorce jsou hodnoty matice o velikosti 4x1. Například při užití bloku Vector_To_FIFO_no, který se nachází uvnitř bloku Main Data Manager 1, bude výpočet následující:

$$((1 - 1) \cdot 4) + [1 \ 2 \ 3 \ 4] = [1 \ 2 \ 3 \ 4].$$

Vyberou se tedy první čtyři prvky vektoru. Dalším příkladem může být užití bloku Vector_To_FIFO_no, který se nachází uvnitř bloku Main Data Manager 2, pak bude výpočet vypadat následně:

$$((2 - 1) \cdot 4) + [1 \ 2 \ 3 \ 4] = [5 \ 6 \ 7 \ 8].$$

Budou tedy vybrány prvky vektoru na pozicích 5, 6, 7 a 8.

Výstupní vektor z bloku Selector označíme jako $\mathbf{u}$, jehož jeden prvek má hodnotu 1 a ostatní tři prvky mají hodnotu -1 (viz. pravidla konfigurační matice).

Selector

Select or reorder specified elements of a multidimensional input signal. The index to each element is identified from an input port or this dialog. You can choose the indexing method for each dimension by using the "Index Option" parameter.

Parameters

Number of input dimensions: 1

Index mode: One-based

	Index Option	Index	Output Size
1	Index vector (dialog)	$[(My_Number-1)*4]+[1\ 2\ 3\ 4]$	Inherit from "Index"

Input port size: 16

Obr. 25 Nastavení parametrů bloku Selector.

Následně je vektor $\mathbf{u}$, zpracováván blokem Opakovač (Gain), který vytvoří skalární součin vektorů $\mathbf{u} \cdot \mathbf{v}$, za předpokladu, že:

$$\mathbf{v} = [1\ 2\ 3\ 4].$$

Výstupem z bloku Gain je tedy vektor:

$$\mathbf{v} \cdot \mathbf{u} = [(1 \cdot u_1)\ (2 \cdot u_2)\ (3 \cdot u_3)\ (4 \cdot u_4)].$$

Tento vektor obsahuje 3 hodnoty menší jak nula a jednu hodnotu větší nebo rovnou nule, z nichž se následně vybere maximum blokem Max, vybere se tedy jediná nezáporná hodnota vektoru $\mathbf{v} \cdot \mathbf{u}$. Po této akci vznikne číslo n , které udává, z jaké fronty FIFO má být paket vybrán.

$$n = \max(\mathbf{v} \cdot \mathbf{u}).$$

Gain

Element-wise gain ($y = K.*u$) or matrix gain ($y = K*u$ or $y = u*K$).

Main | Signal Data Types | Parameter Data Types

Gain: [1 2 3 4] ← vektor $\mathbf{v} = [1\ 2\ 3\ 4]$

Multiplication: Element-wise($K.*u$)

Sample time (-1 for inherited): -1

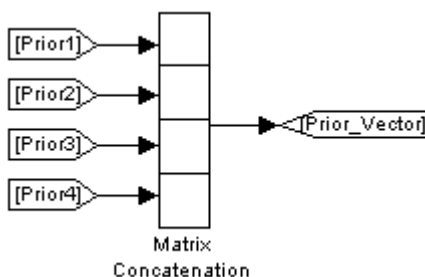
Obr. 26 Nastavení parametrů opakovače Gain.

2.4 Blok Sloučení matic (Matrix Concatenation)

Tento blok (Obr. 27) skládá vektory přijaté na vstupu do jednoho vektoru a to tak, že druhý vektor vloží za první, třetí za druhý atd. Pokud je, v daném případě, velikost každého jednoho ze čtyř vstupních vektorů rovna 4 pak velikost výstupního vektoru bude $4 \cdot 4 = 16$. Matematické vyjádření funkce bloku Sloučení matic může být následující:

$$\begin{aligned} In_1 &= [a_{11} \ a_{12} \ a_{13} \dots a_{1n}], \\ In_2 &= [a_{21} \ a_{22} \ a_{23} \dots a_{2n}], \\ &\vdots \\ In_m &= [a_{m1} \ a_{m2} \ a_{m3} \dots a_{mn}], \\ Out &= [a_{11} \ a_{12} \ a_{13} \dots a_{1n} \ a_{21} \ a_{22} \ a_{23} \dots a_{2n} \dots a_{m1} \ a_{m2} \ a_{m3} \dots a_{mn}], \end{aligned}$$

kde In jsou označeny vstupy do bloku, Out je označen výstup z bloku, a jsou prvky vektorů, n je délka vstupních vektorů a m je počet vstupních vektorů.



Obr. 27 Blok Matrix Concatenation.

2.5 Podsystem Hopfield (Hopfield)

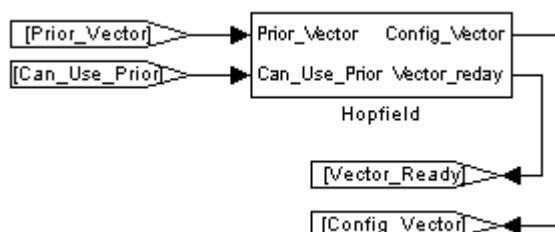
Tento subsystém obsahuje, mimo jiné, Hopfieldovu neuronovou síť, která je řešitelem optimalizačního problému vlastního úkolu.

Subsystém Hopfield má 2 vstupní porty a dva porty výstupní (Obr. 28). Mezi vstupní porty patří:

- Prior_Vector – Na tomto portu je očekáván vektor priorit o 16-ti prvcích, který je získán ze všech čtyř bloků Main Data Manager, respektive ze všech 16-ti front FIFO (popsány v kapitole 2.3.4) a dále je zpracován blokem Sloučení matic (kapitola 2.4).
- Can_Use_Prior – Hodnoty na tento vstup jsou buď logická pravda (true, 1) nebo logická nepravda (false, 0). Přicházejí, stejně jako v případě a), z bloků Main Data Manager, respektive front FIFO a dále jsou zpracovány logickým operátorem AND, který odešle hodnotu true, pokud je každá z hodnot od bloků Main Data Manager true. S vědomostí z předchozích kapitol lze psát tvrzení: Pokud v žádné ze 16-ti front FIFO nedochází k výběru dat (POP), pak je hodnota Can_Use_Prior rovna 1 (true). V opačném případě je hodnota Can_Use_Prior rovna 0 (false).

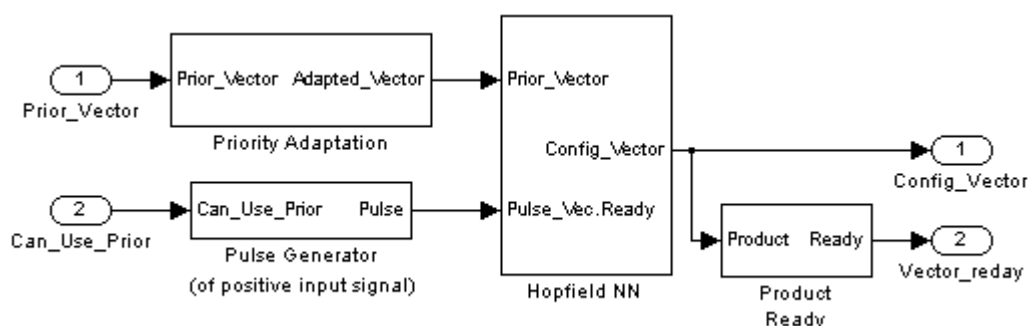
Výstupy subsystému Hopfield jsou dva:

- Vector_Ready – Pokud je dokončen optimalizační výpočet (výsledek je stabilní) Hopfieldovou neuronovou sítí, pak je výstup roven logické pravdě (true, 1). V opačném případě je na výstupu logická nepravda (false, 0).
- Config_Vector – Tímto výstupem je odeslán konfigurační vektor, do bloků Main Data Manager, vypočten Hopfieldovou neuronovou sítí. Jedná se o vektor nastavení výběru paketů z front, jejichž prioritní součet je optimální.



Obr. 28 Subsystém Hopfield.

Obsah subsystému Hopfield je zobrazen na obrázku Obr. 29. Vyskytují se zde další 4 subsystémy, které jsou popsány v následujících podkapitolách.



Obr. 29 Obsah subsystému Hopfield.

2.5.1 Podsyntém Úprava priorit (Priority Adaptation)

Tento subsystém má jeden vstup (Prior_Vector), na který jsou posílána data z prioritního vektoru a jeden výstup (Adapted_Vector). Zabývá se následujícími vlastnostmi Hopfieldovy neuronové sítě (dále jen síť).

- Vstupní data pro síť musejí být z intervalu $< -1 \ 1 >$.
- Nejlepší funkčnost sítě je zajištěna při využití celého pásma hodnot z tohoto intervalu.
- Vyšší hodnota vstupních dat má nižší energetickou hodnotu pro přechod k optimálnímu stavu.

Teoretická podpora těchto tvrzení je v kapitole 3.2.

Z těchto bodů vyplývá, že je nutno vstupní vektor priorit upravit tak, aby jeho hodnoty byly rozloženy do intervalu $< -1 \ 1 >$ a lepší priorita (tedy menší hodnota priority) odpovídala vyššímu číslu z intervalu. Pro extrémní body platí, že nejlepší priorita s ohodnocením 0 by měla vstupovat do sítě s hodnotou 1 a nejhorší priorita, tedy 98, by měla odpovídat hodnotě -1.

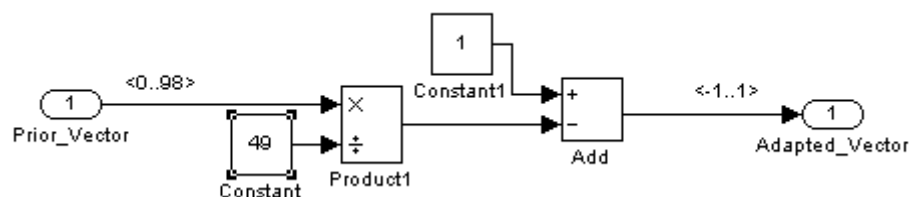
Mechanismus úpravy priorit je:

- Podíl priority konstantní hodnotou 49 ($98/2$).
- Od čísla 1 odečíst výsledek z bodu a).

Matematicky lze funkci subsystému zapsat:

$$y = 1 - \frac{x}{49}$$

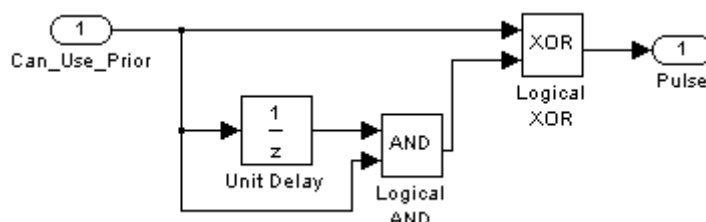
kde y je požadovaná hodnota z intervalu $< 0 \ 98 >$ a x je vstupní hodnota v intervalu $< -1 \ 1 >$. Tento výpočet je nutno provádět pro každý prvek vektoru. Poté je na výstupu Adapted_Vector takový vektor, který je možno použít v síti.



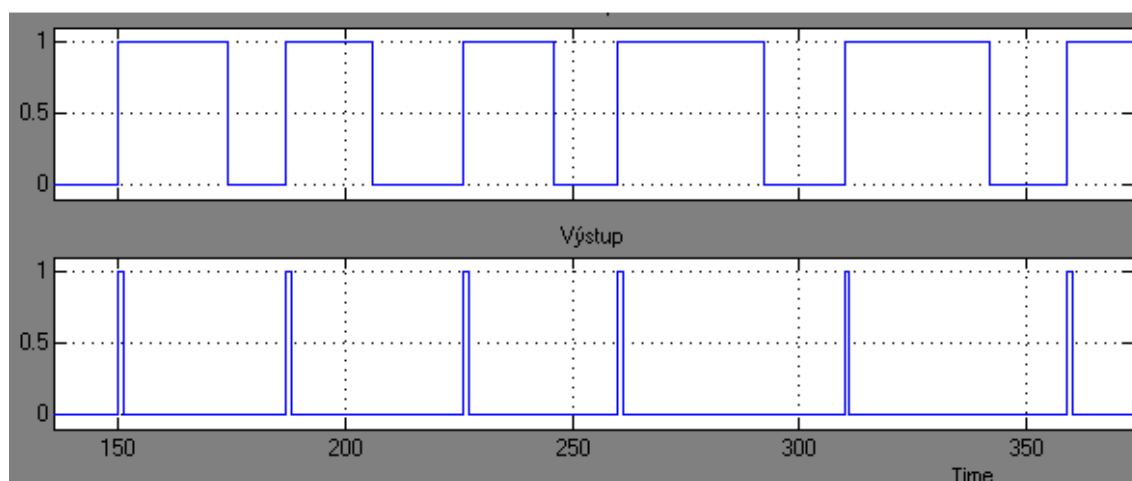
Obr. 30 Obsah subsystému Priority Adaptation.

2.5.2 Podsystem Generátor pulsu (Pulse Generator)

Na vstup tohoto subsystému je přiváděna logická hodnota true (1) nebo false (0) podle toho, zda je možno použít vektor priorit, nebo nikoli. Tento subsystém reaguje pouze na pozitivní změnu tohoto signálu (z false na true) a generuje diskrétní impuls amplitudy true (1). Příklad signálů je zobrazen na Obr. 32 a pravdivostní tabulka ukazující závislost amplitudy v čase $A(t-1)$ a amplitudy v čase $A(t)$ na výstupu $y(t)$ je tabulka Tab. 1.



Obr. 31 Obsah subsystému Pulse Generator.



Obr. 32 Ukázka generování pulsů při pozitivní změně na vstupu.

$A(t-1)$	$A(t)$	$Y(t)$
0	0	0
1	1	0
1	0	0
0	1	1

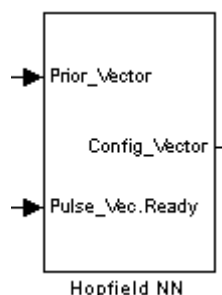
Tab. 1 Pravdivostní tabulka generátoru pulsů.

Obr. 31 znázorňuje obsah subsystému Pulse Generator. Vstupní signál je přiveden do bloku XOR, zároveň do bloku Unit Delay, kde se signál zpožďuje o jeden časový okamžik, a

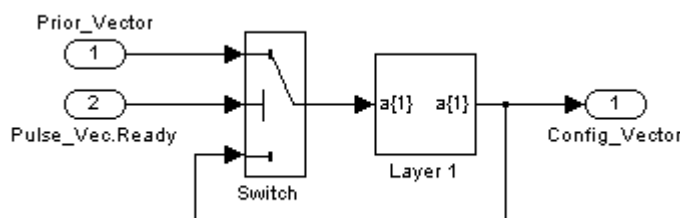
zároveň do bloku AND. Blok AND posílá na výstup true při stavu, kdy je stávající signál true a předchozí byl taktéž true, v ostatních případech posílá false. Při skoku vstupního signálu z false na true dává blok AND hodnotu false a zašle ji na druhý vstup bloku XOR, kde je na prvním vstupu stávající hodnota true. XOR se vyhodnotí jako true a vznikne skok na výstupu. Další časový okamžik je na první ze vstupů bloku XOR přivedena hodnota true a AND dává hodnotu taktéž true. XOR se vyhodnotí jako false a tím vznikne impuls při pozitivním skoku vstupního signálu.

2.5.3 Podsystem Hopfieldova neuronová síť (Hopfield NN)

Vstupy do tohoto subsystému jsou dva. Na první z nich (Prior_Vector) jsou posílána data adaptovaného vektoru priorit a na druhý (Pulse_Vec.Ready) je poslán diskretní impuls, založený na pozitivní změně signálu Can_Use_Prior (je možno použít prioritní vektor p). Výstupem ze subsystému je konfigurační vektor c , jehož hodnoty je možno použít až po ustálení výpočtu, tedy po několika rekurentních krocích Hopfieldovy neuronové sítě. Ustálení výpočtu hlídá subsystém Product Ready, který je popsán v kapitole 2.5.4.



Obr. 33 Podsystem Hopfield NN, jeho vstupní a výstupní porty.



Obr. 34 Obsah subsystému Hopfield NN.

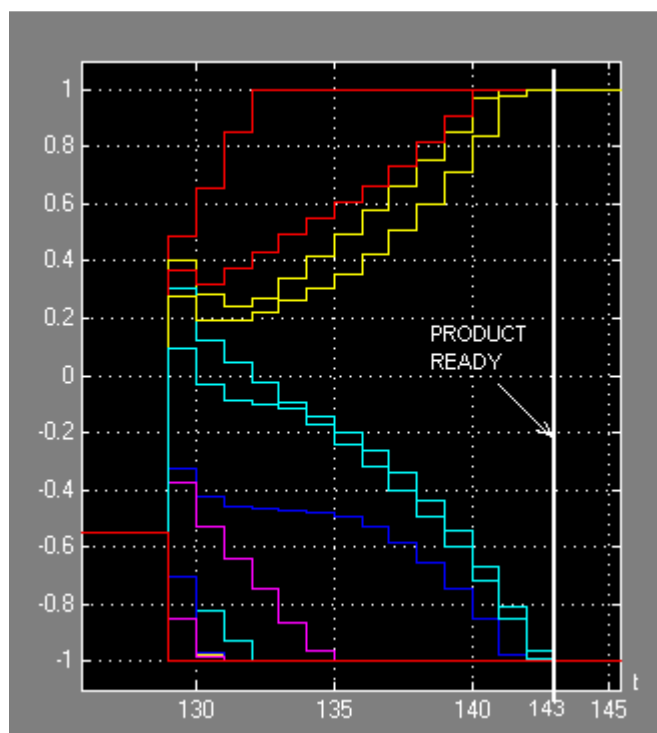
Na obrázku Obr. 34 Obsah subsystému Hopfield NN. je zobrazen obsah subsystému Hopfield NN, který sestává z bloku Switch a jedné vrstvy Layer 1. Blok Switch má funkci přepínače mezi dvěma vstupními signály na jeden výstupní. Prostřední, řídicí vstup udává, který ze vstupních signálů bude přepnut na výstup, pokud je na řídicí vstup přivedena hodnota 0, pak je spojen vstup 3 s výstupem, v opačném případě je spojen vstup 1 s výstupem (Vstupy jsou číslovány od shora bloku vzestupně dolů).

V případě této simulace je na řídicím vstupu bloku Switch nenulová hodnota (true), pokud je možno použít vektor priorit. Jedná se o diskretní hodnotu, tudíž bude přivedena na vstup pouze v době trvání nejmenšího časového okamžiku simulace. V této době Switch propojí data prioritního vektoru do vrstvy Layer 1 Hopfieldovy neuronové sítě. Následující časový okamžik je hodnota Pulse_Vec.Ready opět nulová a Switch propojí vstup 3 s výstupem. Na vstup 3 je přivedena zpětná vazba z výsledku, vypočteného blokem Layer 1. Tato zpětná vazba umožňuje rekurentnost výpočtu, protože je vedena zpět na vstup bloku Layer 1.

V případě této simulace je použita Hopfieldova neuronová síť s jednou vrstvou, tedy

blokem Layer 1, a 16-ti neurony, které jsou vnořeny v této vrstvě. Další informace o použité neuronové síti jsou v kapitole 3.

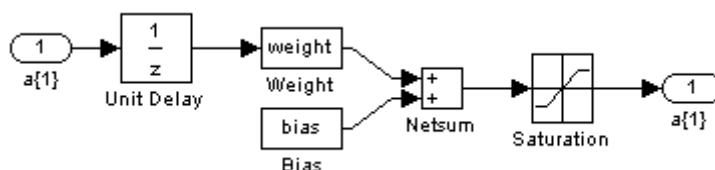
Počáteční data pro neuronovou síť jsou interpretována adaptovaným vektorem priorit, z nichž je rekurentně počítán konfigurační vektor. Graf ukazující postupný výpočet konfiguračního vektoru závislý na čase t je zobrazen na Obr. 35. Každý prvek vektoru je zobrazen jednou křivkou s barevným rozlišením. Přímka Product Ready označuje začátek stability výpočtu a tedy možnost použití konfiguračního vektoru v dalších výpočtech.



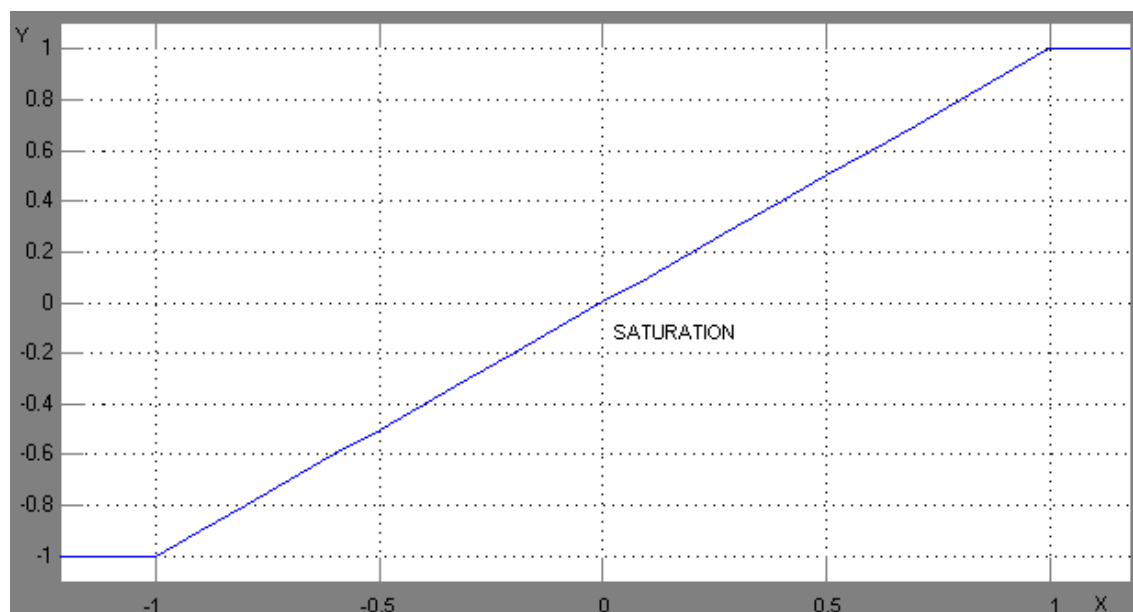
Obr. 35 Ukázka postupného výpočtu konfiguračního vektoru.

2.5.3.1 Podsystem Vrstva 1 (Layer 1)

V subsystému Layer 1 (Obr. 36) přijde signál nejdříve do bloku Unit Delay. Tento blok zpožďuje signál o jeden krok simulace, to je nutno provádět kvůli časování subsystému při použití zpětné vazby. Dále se signál šíří do bloku Weight, jehož činnost je popsána níže. Výstup z bloku Weight, což je vektor o počtu prvků 16, je sečten s vektorem Bias. Blok Bias je standardní blok konstanty (Constant), který produkuje vektor 16 – ti hodnot Biasů. Po sečtení obou vektorů je výsledný vektor přiveden na vstup bloku Saturation. Blok Saturation je v podstatě lineární saturační funkce (Satlin), zobrazená na obrázku Obr. 37, jejíž obor hodnot je $< -1 \ 1 >$. Výsledek ze subsystému Layer 1 je zpět veden na vstup tohoto subsystému zpětnou vazbou, tím vzniká rekurentní výpočet.



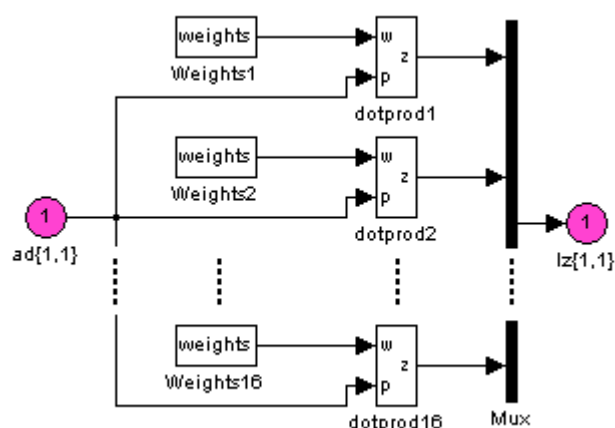
Obr. 36 Obsah subsystému Layer 1.



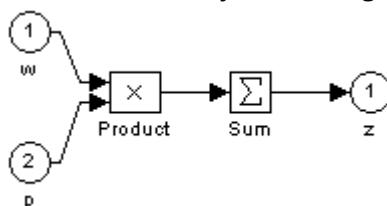
Obr. 37 Graf lineární saturační funkce (Satlin).

Váhy Hopfieldovy neuronové sítě jsou obsaženy v subsystému Weight. Použitá neuronová síť je tvořena 16 - ti neurony a každý z neuronů má 16 hodnot vah. Vstupní signál, tedy vektor o počtu prvků 16, je následně zpracováván takto:

- Vektor je veden do každého z 16 – ti bloků nazvaných Dotprod 1 až 16 (Obr. 38).
- V každém bloku Dotprod se násobí tento vektor s vlastním vektorem vah a následně se všechny hodnoty výsledného vektoru sečtou (Obr. 39). Vzniká jedna číselná hodnota.
- Bod b) se provádí pro každý blok Dotprod, vzniká tedy 16 nových hodnot, ze kterých se dále vytvoří vektor blokem MUX. Tento vektor je výstupem z bloku Weight.



Obr. 38 Obsah subsystému Weight.



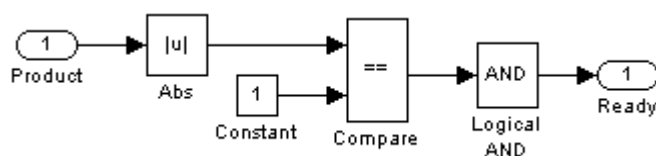
Obr. 39 Obsah subsystému Dotprod.

2.5.4 Podsystem Stablní výsledek (Product Ready)

Výpočet Hopfieldovy neuronové sítě se provádí neustále. Při zadání nových počátečních hodnot do sítě začne výsledek konvergovat k jednomu z nastavených vzorů. Jakmile je dosaženo výsledku korespondujícího se vzorem, je možno výsledek prohlásit za stabilní a můžou se jeho hodnoty poskytnout pro další zpracování.

V případě této simulace je nastaveno 24 vzorových matic (resp. vektorů), jejichž prvky nabývají hodnot 1 nebo -1. Z tohoto faktu lze stabilní výsledek jednoduše rozpoznat a to tak, že každý prvek vektoru výsledku bude mít hodnotu 1 nebo -1.

Blok Product Ready (Obr. 40) má jeden vstup, na kterém je přijímán konfigurační vektor vypočtený Hopfieldovou neuronovou sítí a jeden výstup, který nabývá logických hodnot true (1) nebo false (0), podle toho, zda je konfigurační vektor stabilní (true) nebo nikoli (false).



Obr. 40 Obsah bloku Product Ready.

Blok Abs přiřadí každému prvku vektoru jeho absolutní hodnotu, takový vektor je následně porovnán blokem Compare vůči konstantě 1. Je-li absolutní hodnota prvku vektoru rovna 1, pak blok Compare na tuto pozici vektoru vloží hodnotu true (1). Posledním blokem je logický operátor AND (Logical AND), do kterého vstupují všechny prvky vektoru z bloku Compare, v případě, že mají všechny prvky hodnotu true (1), pak je výsledkem logického operátoru AND taktéž hodnota true (1). V jiném případě vrací AND hodnotu false (0). Tyto hodnoty jsou reprezentovány výstupem ze subsystému Product Ready.

Matematický popis subsystému lze zapsat následovně:

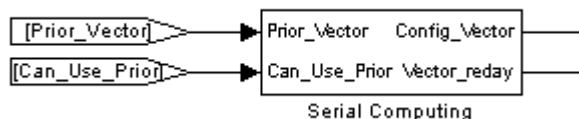
$$\mathbf{a} = [a_1 \ a_2 \ .. \ a_{16}],$$

$$y = \begin{cases} true, & \text{když } \mathbf{a} - \mathbf{1} = \mathbf{0} \\ false, & \text{když } \mathbf{a} - \mathbf{1} \neq \mathbf{0} \end{cases}$$

kde y je výstupní hodnota, $\mathbf{a}$ je vstupní vektor a $\mathbf{0}$ je nulový vektor.

2.6 Podsystem Standardní výpočet (Standard Computing)

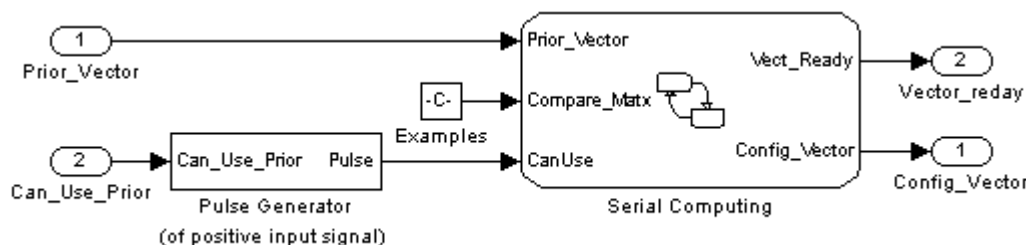
Tento subsystém (Obr. 41) je alternativou pro subsystém Hopfield. Jedná se o blok, který standardním způsobem (sériově) počítá optimální výběr konfiguračního vektoru ze vzorů konfiguračních vektorů. Jedná se o prvek simulace, který je určen pouze pro porovnání časových náročností optimalizačního výpočtu pomocí Hopfieldovy neuronové sítě s výpočtem bez neuronové sítě.



Obr. 41 Alternativní subsystém Serial Computing.

Na vstupní porty se přivádějí stejná data jako u subsystému Hopfield. Jedná se tedy o vektor priorit (Prior_Vector) a logickou hodnotu, zda je možno vektor priorit použít

(Can_Use_Prior). Výstupy ze subsystému jsou opět totožné se subsystémem Hopfield, jedná se o konfigurační vektor na prvním z výstupů (Config_Vector) a logickou hodnotu, zda je možno konfigurační vektor použít (Vector_ready).



Obr. 42 Obsah subsystému Standard Computing.

Obsah subsystému Standard Computing je zobrazen na Obr. 42. Pro generování diskretního pulsu je zde použit blok Pulse Generator, který je popsán již v kapitole 2.5.2. Konstanta nazvaná Examples obsahuje matici vzorů 24 konfiguračních vektorů. Tato matice má tedy 24 řádků a 16 sloupců, přičemž každý řádek udává jednu z 24 možných kombinací konfiguračního vektoru (Více o konfiguračních vektorech v kapitole 2.6.1 a Konkrétní použití Hopfieldovy neuronové sítě). Hlavním výpočetním procesorem tohoto subsystému je Stateflow tabulka Serial Computing.

2.6.1 Stateflow tabulka Sériový výpočet (Serial Computing)

Tato tabulka má za úkol vybrat z matice vzorů konfiguračních vektorů právě jeden, který optimálně odpovídá počátečním podmínkám, respektive Prioritnímu vektoru. Výrazem „optimálně“ je v tomto případě myšleno, že součet vektoru priorit, podle jednoho z 24 vzorů, bude nejmenší, vůči součtům, podle ostatních vzorů.

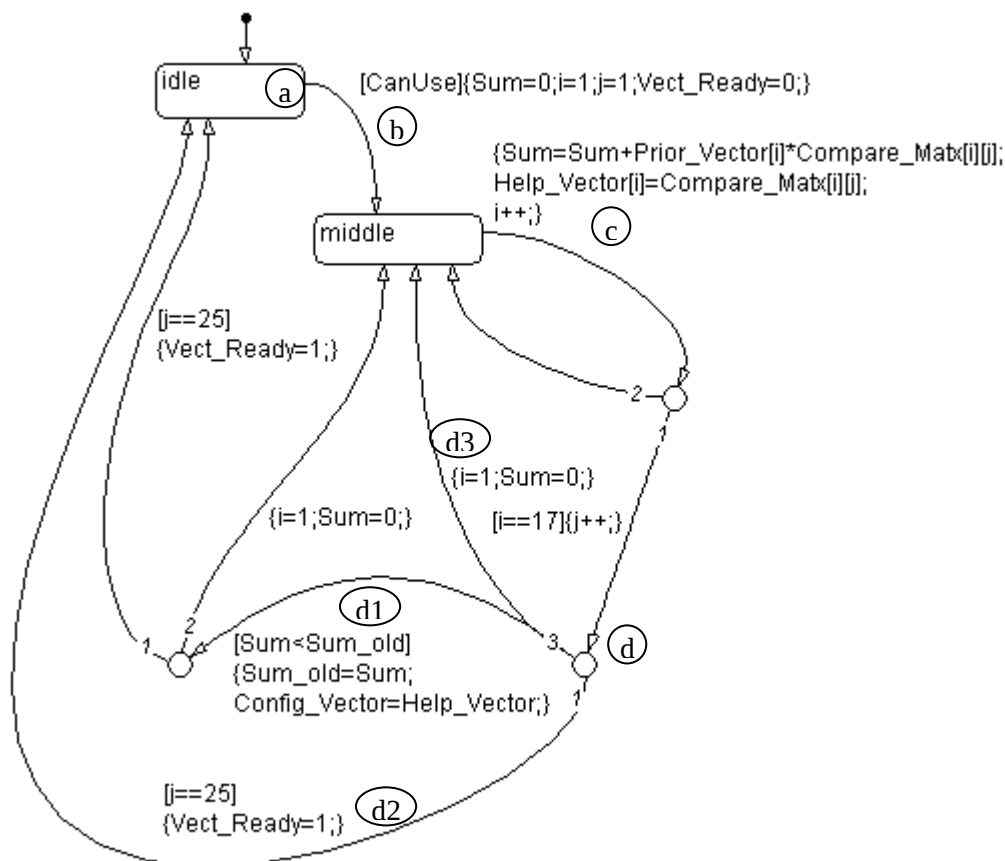
Vzory (Examples) jsou do tabulky posílány přes vstup Compare_Matx, prioritní vektor se objevuje na vstupu Prior_Vector, jeho hodnoty jsou z intervalu $< 0 \ 98 >$, není tedy upraven adaptací, jak je tomu u bloku Hopfield a posledním vstupem je CanUse, kam je posílána diskretní logická hodnota, zda je možno prioritní vektor použít. Přes výstupní porty je odesílána logická hodnota, zda je možno použít konfigurační vektor (Vector_ready) a vypočtená data samotného konfiguračního vektoru [1].

Průběh výpočtu konfiguračního vektoru je zobrazen na Obr. 43.

- Při inicializaci modelu se aktivuje stav Idle.
- Po příchodu signálu true na vstup CanUse se přejde do stavu Middle při nulování vnitřní proměnné Sum, nastavení proměnných i a j na hodnotu 1 a nastavení výstupní proměnné Vect_Ready na hodnotu 0.
- Dále se násobí každý prvek prioritního vektoru s každým prvkem jednoho řádku matice vzorů (Compare_Matx). Matice Compare_Matx nemá hodnoty -1 a 1, jak je tomu u vzorů konfiguračních matic, ale namísto hodnoty -1 je vložena hodnota 0. Vynásobené prvky se sečtou a přičtou se k proměnné Sum. Do pole proměnných Help_Vector je uložen právě násobený řádek matice Compare_Matx.
- Po vynásobení celého řádku matice ($i = 17$) se inkrementuje hodnota proměnné j a učiní se rozhodnutí:
 - Pokud je součet prvků vektorů Sum menší jak předešlý nejmenší součet (Sum_Old), pak se hodnota proměnné Sum_Old ztotožní s proměnnou Sum a výsledné proměnné Config_Vector se přiřadí Help_Vector. Poté záleží na stavu, zda jsou propočteny všechny řádky matice Compare_Matx ($j = 25$), pak se aktivuje stav Idle a Vect_Ready je 1 (výpočet je ukončen), nebo pokud nebyly propočteny všechny

řádky matice vzorů, pak se přechází zpět na stav Middle s nulováním proměnné Sum a přiřazením hodnoty 1 do proměnné i. Pokračuje se bodem c).

- Pokud jsou propočteny všechny řádky matice vzorů, pak je hodnota $j = 25$. Přejde se do stavu Idle a proměnná Vect_Ready se nastaví na 1. Výpočet je ukončen.
- Pokud není proměnná Sum menší jak Sum_Old a zároveň j je různé od 25, pak se ze spojky (Junction) pokračuje na stav Middle s přiřazením hodnoty 1 do proměnné i a nulováním proměnné Sum. Cyklus se opakuje od bodu c).



Obr. 43 Stateflow tabulka Serial Computing.

Matice vzorů Compare_Matx je totožná s maticí vzorů Examples na Obr. 44. Vektor priorit může vypadat následovně:

$$\mathbf{p} = [p_{11} \ p_{12} \ p_{13} \ p_{14} \ p_{21} \ p_{22} \ p_{23} \ p_{24} \ p_{31} \ p_{32} \ p_{33} \ p_{34} \ p_{41} \ p_{42} \ p_{43} \ p_{44}].$$

Vektorový zápis je pro zřejmost smyslu nevhodný, jasněji lze vysvětlit smysl prvků při maticovém zápisu vektoru priorit, který je následující:

$$\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{pmatrix}.$$

V matici P odpovídá každý řádek prioritám z jednoho vstupu do přepínače, jedná se tedy o jeden prioritní vektor z daného bloku Main Data Manager a každý sloupec odpovídá prioritám daného výstupu z přepínače. Jako příklad je možno uvést paket, který byl přijat vstupním portem přepínače číslo 1 (Source = 1) a jeho číslo výstupního portu z přepínače je 4 (Destination = 4). Tento paket má hodnotu priority uloženou na pozici p_{14} matice P , respektive vektoru p .

Examples =

1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0
1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1
1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0
1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1
1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	1
0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1
0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0	0	0	1	1	0	0	0
0	1	0	0	0	0	0	1	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0
0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1
0	0	1	0	1	0	0	0	0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	1
0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0
0	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0
0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0	0	0	1	0	1	0	0	0
0	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0
0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0

Obr. 44 Matice vzorů **Examples**.

Při sériovém výpočtu, jak již bylo zmíněno v popisu bloku Serial Computing, je násoben každý řádek matice **Examples** jako vektor vektorem p a hodnoty výsledného vektoru jsou sečteny. Pokud je hodnota výsledku právě počítaného řádku matice **Examples** menší než jakákoli předchozí, pak se tento řádek uloží jako výsledný konfigurační vektor.

Matematické vyjádření výpočtu optimálního konfiguračního vektoru může být následující:

$$E = \text{Examples},$$

$$E = \begin{pmatrix} E_{1,1} & \cdots & E_{1,16} \\ \vdots & \ddots & \vdots \\ E_{24,1} & \cdots & E_{24,16} \end{pmatrix},$$

$$\text{sum} = p \cdot (E)^T,$$

$$\text{sum} = (s_1 \ s_2 \ .. \ s_i \ .. \ s_{24}),$$

$$\min(\text{sum}) = s_i,$$

$$c = (E_{i,1} \ E_{i,2} \ .. \ E_{i,16}).$$

Kde E je matice vzorů, $\mathbf{sum}$ je vektorem součtů, $\min(\mathbf{sum})$ dává prvek vektoru $\mathbf{sum}$ s nejnižší hodnotou a $\mathbf{c}$ je výsledný konfigurační vektor.

2.7 Průběh simulace

Celá simulace čtyř portového přepínače (Switch) je založena na funkcích menších simulačních celků, takzvaných subsystémů. Funkce a vlastnosti každého subsystému jsou podrobně popsány v předešlých kapitolách. Tato kapitola pojednává o šíření signálu (dat) v simulaci, respektive o propojení jednotlivých subsystémů.

Data, v případě přepínače pakety, která jsou v reálné síti zasílána na vstup přepínače, jsou přijímána od jiného aktivního síťového prvku. Může se jednat o síťovou kartu počítače, jiný přepínač (Switch), router atd. V případě uvažované simulace přepínače není model propojen s žádnými reálnými síťovými prvky, příchozí pakety jsou z tohoto důvodu nasimulovány přímo v modelu simulace přepínače. Simulované pakety obsahují, stejně jako reálné pakety, informace o příjemci paketu, odesílateli paketu, délce paketu, prioritě paketu a jiné. O generování zmíněných simulovaných paketů se starají 4 bloky (subsystémy), odpovídající čtyřem vstupním portům přepínače, s názvem Packet Generator (Kapitola 2.1). Ukázka signálu vycházejícího z bloku Packet Generator je na Obr. 5.

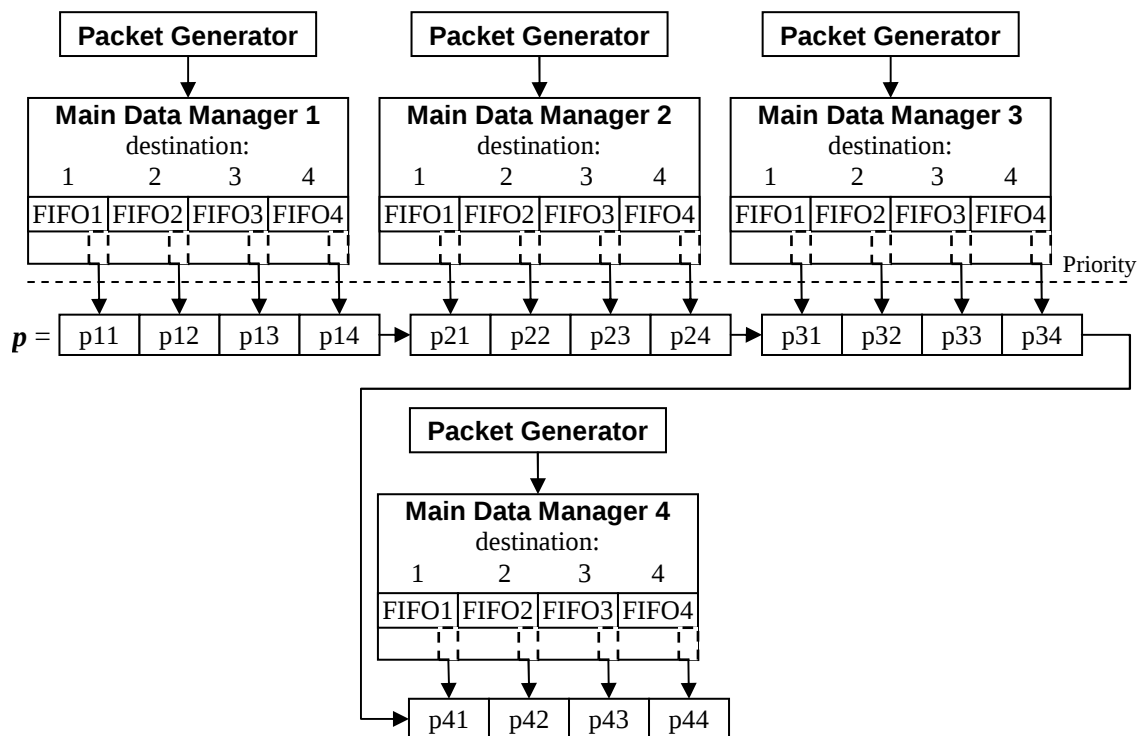
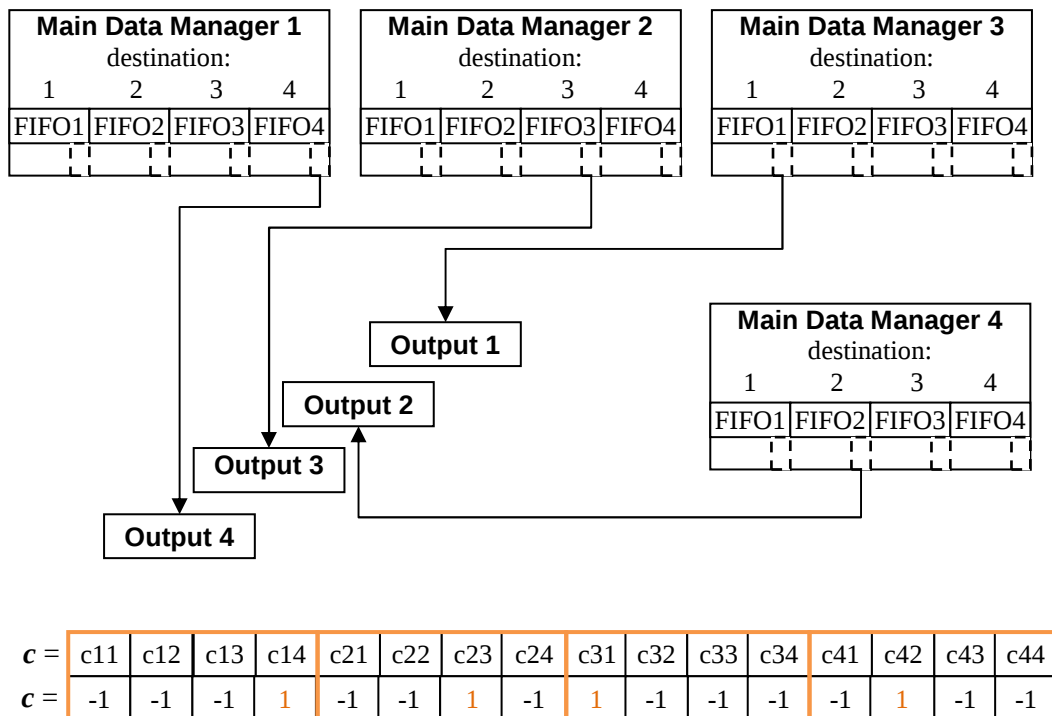
Z bloků Packet Generator putuje signál do bloků Main Data Manager 1, 2, 3 a 4. Pro každý vstup přepínače je zde jeden z bloků Main Data Manager, pokud je tedy generován signál paketu z 1. vstupu přepínače, pak paket putuje do bloku Main Data Manager 1 a podobně tak pro ostatní 3 vstupy. Tento blok rozdělí příchozí paket do jedné ze čtyř front (FIFO), které obsahuje. Fronty jsou opět označeny pořadovými čísly 1 až 4. Klíč pro rozdělení paketů je následující: Zjistí se, na kolikátý výstupní port má být paket poslán (taková hodnota se nazývá destinace) a podle této hodnoty je zařazen do jedné ze čtyř front, právě takové, která shromažďuje pakety s touto destinací. Popsaný děj, rozdělování paketů, se odehrává v každém bloku Main Data Manager. Deduktivně lze psát, že počet front FIFO je v modelu použito 16, a to 4 pro každý blok Main Data Manager.

Poznámka: Od reálného paketu se hodnota destinace liší tak, že reálný paket používá pro určení destinace adresu, jejíž hodnota nebývá pouze jedno-číselná. Adresy příjemců, resp. odesílatelů, dat jsou ukládány do tabulky adres, ze které se následně vychází při určování přepínání mezi odesílatelem a příjemcem.

Rozdělené pakety jsou uloženy v příslušných frontách, z nichž je dále čtena pouze část paketů, jedná se o prioritu všech 16 – ti paketů uložených na začátcích front. Priority jsou kumulovány do vektoru priorit $\mathbf{p}$ (Obr. 46).

Vektor priorit obsahuje základní data pro určení následného výběru jednoho paketu z každého bloku Main Data Manager, respektive jedné z jeho front. Adaptované hodnoty vektoru jsou zpracovány Hopfieldovou neuronovou sítí (dále jen síť), obsaženou blokem Hopfield. Síť je předem naučena přirovnávat svůj výsledek k jednomu z 24 vzorů (Obr. 44) a to tak, že je přirovnán ke vzoru za využití minima energetické funkce. Jeden ze vzorů je tedy výsledkem sítě, reprezentován jako konfigurační vektor $\mathbf{c}$.

Po získání stabilních hodnot konfiguračního vektoru je tento vektor zaslán do každého bloku Main Data Manager. Konfigurační vektor má 16 prvků, z toho každá čtveřice, braná jako 4 prvky za sebou, obsahuje pouze jednu hodnotu 1 (ostatní -1) a náleží právě jednomu bloku Main Data Manager. Ukázka rozdělení konfiguračního vektoru mezi bloky Main Data Manager a odeslání paketů na základě tohoto vektoru je na (Obr. 45).

Obr. 46 Vznik vektoru priorit p .Obr. 45 Ukázka použití konfiguračního vektoru c .

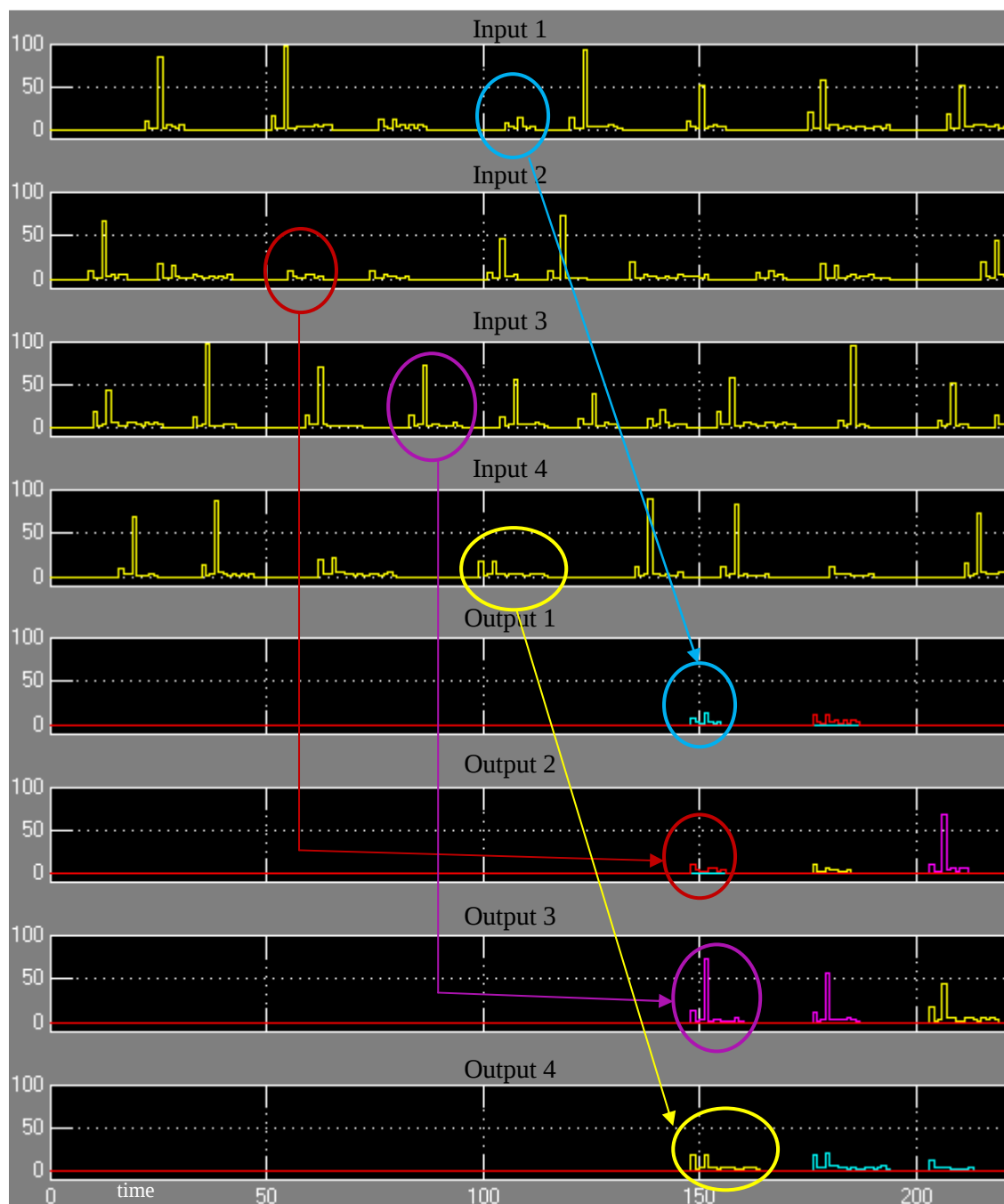
Z každého bloku Main Data Manager je následně vybrána, podle konfiguračního vektoru, jedna z front a její první paket je odeslán na výstup přepínače. Z vlastnosti konfiguračního vektoru takové, že v každé čtveřici prvků po sobě braných musí být pouze a právě jedna hodnota 1, lze vyvodit závěr, že na každý výstupní port (Output) musí být odeslán paket z nějaké fronty (FIFO), vždy však z jedné z kompetence jednoho bloku Main Data Manager. Pokud nastane stav, že data na jeden (nebo více) výstupní port nejsou právě v žádné frontě uložena a přesto konfigurační vektor dává příkaz z této fronty vybrat paket, pak se jedná o tzv. posláni nulového paketu, respektive žádných dat.

2.8 Reprezentace výsledků

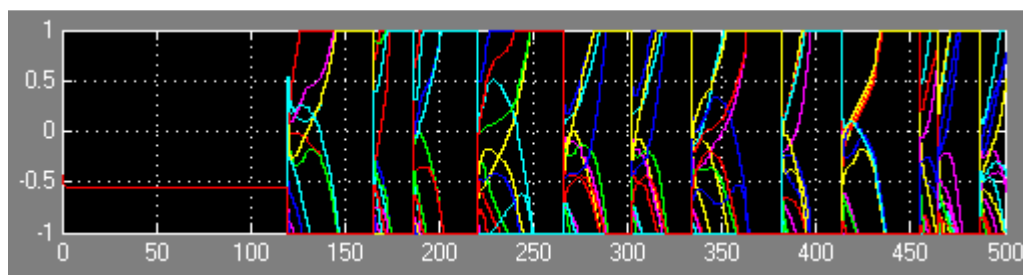
Jak již bylo předesláno, tato simulace není propojena s reálným aktivním síťovým prvkem, není tedy možno pakety přijímat z okolí, ani je do okolí vysílat. Jako generátor paketů, namísto reálného síťového prvku, je použit blok Packet Generator. Data, která produkuje, tedy pakety, lze zobrazit jako signál. V tom případě pakety, které jsou určeny, jako výstupní pakety z přepínače jsou téhož typu. Pro zobrazování signálů je v programu Simulink určen blok Scope. Jako hlavním reprezentantem výsledků je v simulaci použit Scope s názvem Inputs/Outputs, který má 8 vstupních portů. Shora první 4 jsou pro pakety na čtyřech vstupech přepínače a další 4, pro výstupy z přepínače. Scope tedy bude zobrazovat 8 různých signálových křivek v závislosti na čase simulace. Ukázka signálů je na Obr. 47.

Signálový blok, který je zakroužkován na Obr. 47 ukazuje jeden paket. Z prvních čtyř křivek (Input) byly vybrány pakety, které jsou zakroužkovány a byly odeslány na výstupy, jak je vidět na dalších čtyřech křivkách (Output) a jejich zakroužkovaných blocích.

Obr. 48 ukazuje příchozí signály z Hopfieldovy neuronové sítě do bloku Main Data Manager. Každý z jedenácti zobrazených celků má na svém konci za následek výběr dat z fronty FIFO. Jedná se tedy o grafické zobrazení jedenácti vypočtených konfiguračních vektorů. Prodleva mezi jednotlivými výpočty Hopfieldovy sítě je zapříčiněna časovou náročností výběru paketu z fronty.



Obr. 47 Ukázka paketů vstupujících a vystupujících z přepínače.



Obr. 48 Výsledný signál z Hopfieldovy sítě.

3 HOPFIELDOVA NEURONOVÁ SÍŤ

3.1 Teoretický popis Hopfieldovy neuronové sítě

Hopfieldova neuronová síť, pojmenovaná podle jejího tvůrce Johna Hopfielda (1982), je jednovrstvá umělá neuronová síť se stejným počtem vstupů, výstupů a neuronů. Každý výstup z neuronu je zpětnou vazbou spojen se všemi vstupy neuronů včetně sama sebe [4]. Jedná se tedy o rekurentní síť [3]. Vstupní i výstupní hodnoty jsou bipolární (1 a -1).

Síť se učí podle zadaných vzorů. Učením se rozumí nastavení hodnot vah a biasů neuronů. Po naučení neuronové sítě se využívají tyto hodnoty pro výpočet výsledku totožného se vzorem z počátečních hodnot vstupujících do sítě. Za předpokladu, že matice vah je $\mathbf{w}$, vektor biasů je $\boldsymbol{\theta}$, $\mathbf{a}(t)$ je vektor vstupních hodnot do sítě (počátečních hodnot) v čase t a f je aktivační funkce, lze psát matematickou formulaci výpočtu prováděného neuronovou sítí:

$$\begin{aligned}\mathbf{a}(0) &= \text{vektor počátečních hodnot,} \\ \mathbf{a}(t) &= f(\mathbf{w} \cdot \mathbf{a}(t-1) - \boldsymbol{\theta}).\end{aligned}$$

Při stavu $\mathbf{a}(t) = \mathbf{a}(t-1)$ je výsledek stabilní a je ukončen výpočet, výsledkem je vektor $\mathbf{a}(t)$.

3.2 Konkrétní použití Hopfieldovy neuronové sítě

Hopfieldova neuronová síť použita pro simulaci tohoto přepínače byla vytvořena a naučena programem Matlab [2] a následně generována příkazem `simgen` do nástavby Simulink. Z vygenerované sítě jsou použity pouze hodnoty vah a biasů pro tento problém. Následuje ukázka kódu pro vytvoření, zároveň i naučení, a vygenerování nové Hopfieldovy sítě:

```
Examples= [1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1;-...
-1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1;-...
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 -1 -1;-...
-1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 1 1 1;-...
1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1;-...
-1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1;-...
-1 -1 -1 1 1 -1 -1 -1 -1 1 1 -1 -1 -1 1 1 -1 -1 -1 -1;-...
-1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1 -1 1 -1 -1 -1 1 1 -1;-...
-1 -1 1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 1 1 -1 -1 1;-...
1 -1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 -1;-...
-1 1 -1 1 -1 -1 -1 1 -1 -1 -1 1 -1 1 -1 -1 -1 -1 -1 -1;-...
-1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 1 -1 -1 1 -1 1 -1 1 -1;-...
-1 -1 1 1 -1 1 -1 -1 -1 -1 -1 1 -1 1 -1 1 -1 -1 1 -1;-...
-1 1 -1 1 -1 -1 1 1 -1 1 1 -1 -1 -1 -1 1 1 -1 1 -1;-...
1 -1 1 -1 -1 1 1 -1 1 1 -1 1 1 -1 -1 -1 -1 -1 -1 -1];
net = newhop(Examples);
simgen(net);
```

Examples je matice vzorů pro novou síť, generována za předpokladu, že každý konfigurační vektor $\mathbf{c}$ může být zapsán jako konfigurační matice $\mathbf{C}$. Těchto matic, respektive vektorů, je tolik, kolik je možno utvořit kombinací maticových prvků takových, že v jednom řádku a zároveň jednom sloupci může být pouze jeden prvek s hodnotou 1, ostatní s hodnotou -1. Pravidlo pro přepis konfiguračního vektoru na konfigurační matici je ukázáno dále:

$$c = [c_{11} \ c_{12} \ c_{13} \ c_{14} \ c_{21} \ c_{22} \ c_{23} \ c_{24} \ c_{31} \ c_{32} \ c_{33} \ c_{34} \ c_{41} \ c_{42} \ c_{43} \ c_{44}].$$

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{pmatrix}.$$

Pro automatické generování vzorů, tedy všech možností konfiguračních vektorů, byl využit programovací jazyk Delphi. Naprogramovaná funkce generující výpis matice vzorů do objektu memo2 je následující:

```

procedure ExampesGen();
var i,j,k,l,m,n,poc:integer;
    pole:array of integer;
    vypis:string;
begin
    poc:= 0;
    setLength(pole,16);
    for i:= 1 to 4 do
        for j:= 1 to 4 do
            for k:= 1 to 4 do
                for l:= 1 to 4 do
                    if ((i <> j) and (i <> k) and (i <> l) and (j <> k) and (j <> l) and (k <> l)) then
                        begin
                            vypis:= "";
                            for m:= 0 to 15 do
                                pole[m]:= -1;
                                pole[i-1]:= 1;
                                pole[j-1+4]:= 1;
                                pole[k-1+8]:= 1;
                                pole[l-1+12]:= 1;
                                for m:= 0 to 3 do
                                    begin
                                        if m= 0 then poc:= poc+1;
                                        for n:= 0 to 3 do
                                            vypis:= vypis+' '+inttostr(pole[m*4+n]);
                                        end;
                                        memo2.Lines.Add(vypis+' ');
                                    end;
                                end;
                            end;
                        end;
                    end;
                end;
            end;
        end;
    end;
end;

```

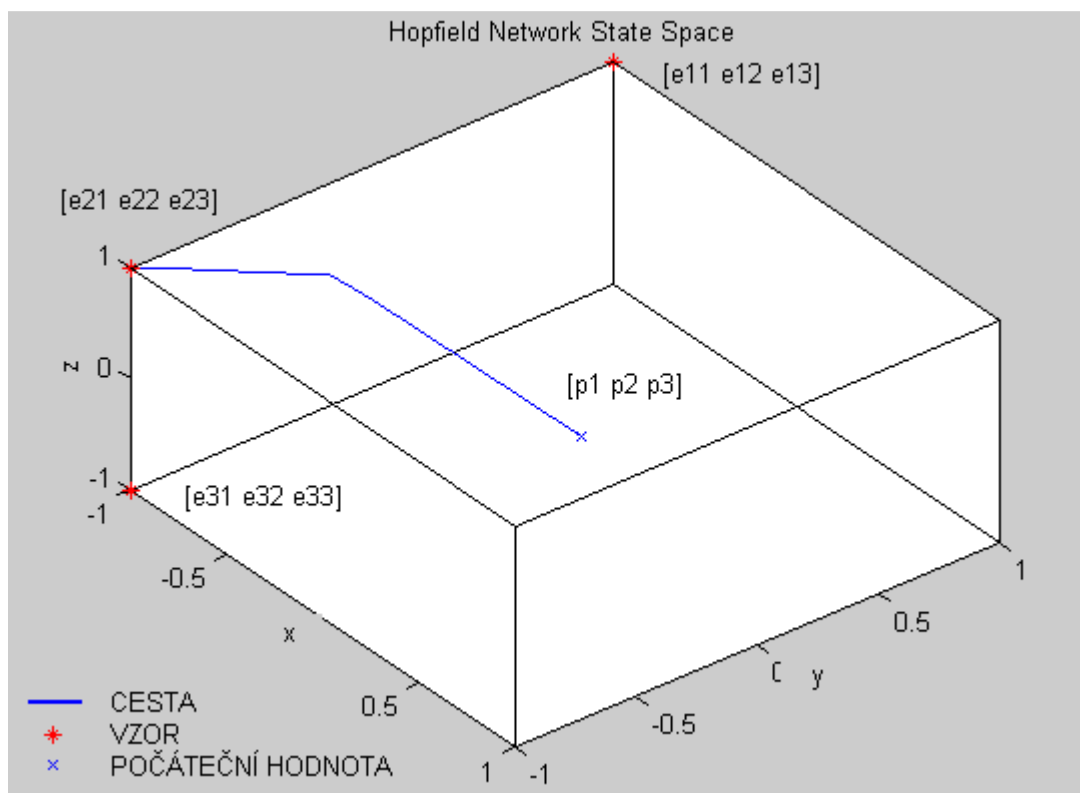
Realizace použité sítě je založena na teorii z kapitoly 3.1. Jedná se tedy o vzorec:

$$\begin{aligned} a(0) &= p, \\ a(t) &= f(w \cdot a(t-1) - \Theta), \end{aligned}$$

kde p je adaptovaný vektor priorit. Konečný výsledek, tedy konfigurační vektor c , je zpracováván až po stabilizaci vektorů $a(t) = a(t-1)$, poté je konfigurační vektor $c = a(t)$.

Jak již bylo předesláno, Hopfieldova neuronová síť je optimalizační. Optimalizace je vedena za účelem minima energetické funkce [5]. Jedná se tedy o nejméně energeticky náročnou cestu z bodu a_i do bodu c_i jednoho z konfiguračních vektorů. Jako zjednodušený příklad výpočtu Hopfieldovy sítě lze vytvořit 3D problém se třemi vzory ve třech rozích krychle

a náhodně vloženým bodem v prostoru krychle [2]. Tento bod bude mít souřadnice uloženy jako vektor $\mathbf{p}$ a rohy krychle, tedy vzory, budou uloženy v matici vzorů $\mathbf{E}$. Jako výsledek dostaneme konfigurační vektor $\mathbf{c}$, který je vyjmut jako jeden z řádků matice $\mathbf{E}$. Body a výpočtová křivka jsou zobrazeny na Obr. 49.



Obr. 49 Ukázka výpočtu Hopfieldovy neur. sítě v 3D prostoru.

V příkladu uvedeném na obrázku byla použita následující matice vzorů:

$$\mathbf{E} = \begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix} = \begin{pmatrix} -1 & 1 & 1 \\ -1 & -1 & 1 \\ -1 & -1 & -1 \end{pmatrix}$$

a počáteční hodnota jako vektor souřadnic $\mathbf{p}$:

$$\mathbf{p} = [p_1 \ p_2 \ p_3] = [0.254 \ -0.133 \ 0.134].$$

Výsledný bod je určen vektorem se souřadnicemi z druhého řádku matice $\mathbf{E}$, tedy:

$$\mathbf{c} = [c_1 \ c_2 \ c_3] = [e_{21} \ e_{22} \ e_{23}] = [-1 \ -1 \ 1].$$

Podobným způsobem si lze představit realizovanou optimalizaci přepínače, s tím rozdílem, že namísto tří-dimenzionálního (3D) prostoru, ukázaného v tomto příkladě, je použit prostor šestnácti-dimenzionální (16D) a počet vzorů je 24. Matice $\mathbf{E}$ pak má rozměr 16x24, vektory $\mathbf{p}$ a $\mathbf{c}$ počet prvků 16.

4 POROVNÁNÍ VÝSLEDKŮ

V modelu přepínače je vytvořen blok Standard Computing (kapitola 2.6). Jedná se o standardní sériový výpočet konfiguračního vektoru z vektoru priorit. Výpočet probíhá krok po kroku a zjišťuje, jaký vzor je z matice vzorů nutno vybrat, aby byl výběr paketů optimální.

Stejnou úlohu řeší Hopfieldova neuronová síť simulovaná blokem Hopfield (kapitola 2.5), kde jsou, stejně jak u sériového výpočtu, vstupními daty priority čekajících paketů, respektive prioritní vektor. Hopfieldova síť provádí výpočet jako paralelní počítání jednodušších celků, ze kterých je stvořen výsledný konfigurační vektor. Výpočty vah a biasů se při běhu simulace neprovádějí, jejich hodnoty jsou fixně dány již při učení sítě, v tomto případě při tvorbě bloku Hopfield. Výpočet probíhá rekurentně a je ukončen až v době, kdy jsou výstupní data ustálená.

Úlohu výpočtu optimálního konfiguračního vektoru lze řešit oběma způsoby, tedy jak pomocí bloku Standard Computing, tak pomocí bloku Hopfield. Díky tomu lze provést vyjádření, jaká z těchto metod je rychlejší nebo přesnější.

O přesnosti výpočtu lze prohlásit, že je u obou případů stejná. Je to způsobeno stejnými maticemi vzorů. Výsledný konfigurační vektor musí být v obou případech výňatkem jednoho z řádků matice vzorů. Jiný výsledek není očekáván. Pokud se jedná o přesnost z pohledu optimálního výběru, pak by tato přesnost měla být opět pro oba případy totožná. Hopfieldova síť, díky nastaveným vahám a biasům, zvolí vždy svoji nejmenší energetickou funkci, tedy i optimální řešení problému. Blok Standard Computing vybere konfigurační vektor, vždy ten, s nejmenším součtem prvků, tedy i optimálním výběrem paketů. Rozpor mezi výsledky obou metod může nastat pouze v případě kolizních součtů priorit podle konfiguračních vektorů. Jednat se může o stav, kdy vektor priorit umožňuje provádět dva nebo více součtů podle matice vzorů takových, že jejich součet je optimální. V tomto případě může vybrat jiný vzor Hopfieldova síť a jiný standardní výpočet, oba vzory ale zabezpečují stejnou optimálnost výběru. Ukázka kolizního vektoru (matice) priorit je následující:

$$\mathbf{p} = [30 \ 1 \ 8 \ 6 \ 9 \ 2 \ 13 \ 4 \ 10 \ 28 \ 5 \ 0 \ 3 \ 6 \ 16 \ 9],$$

z toho matice priorit

$$\mathbf{P} = \begin{pmatrix} 30 & 1 & 8 & 6 \\ 9 & 2 & 13 & 4 \\ 10 & 28 & 5 & 0 \\ 3 & 6 & 16 & 9 \end{pmatrix}.$$

V tomto příkladu je kolizní hodnota optimální priority rovna 13. Lze stvořit nejmenší součet priorit, tedy 13, dvěma způsoby.

Způsob 1:

$$\mathbf{C} = \begin{pmatrix} -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 \\ -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 \end{pmatrix}.$$

Součet priorit

$$\text{sum} = (1+4+5+3) = 13.$$

Způsob 2:

$$\mathbf{C} = \begin{pmatrix} -1 & -1 & 1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 \\ 1 & -1 & -1 & -1 \end{pmatrix}.$$

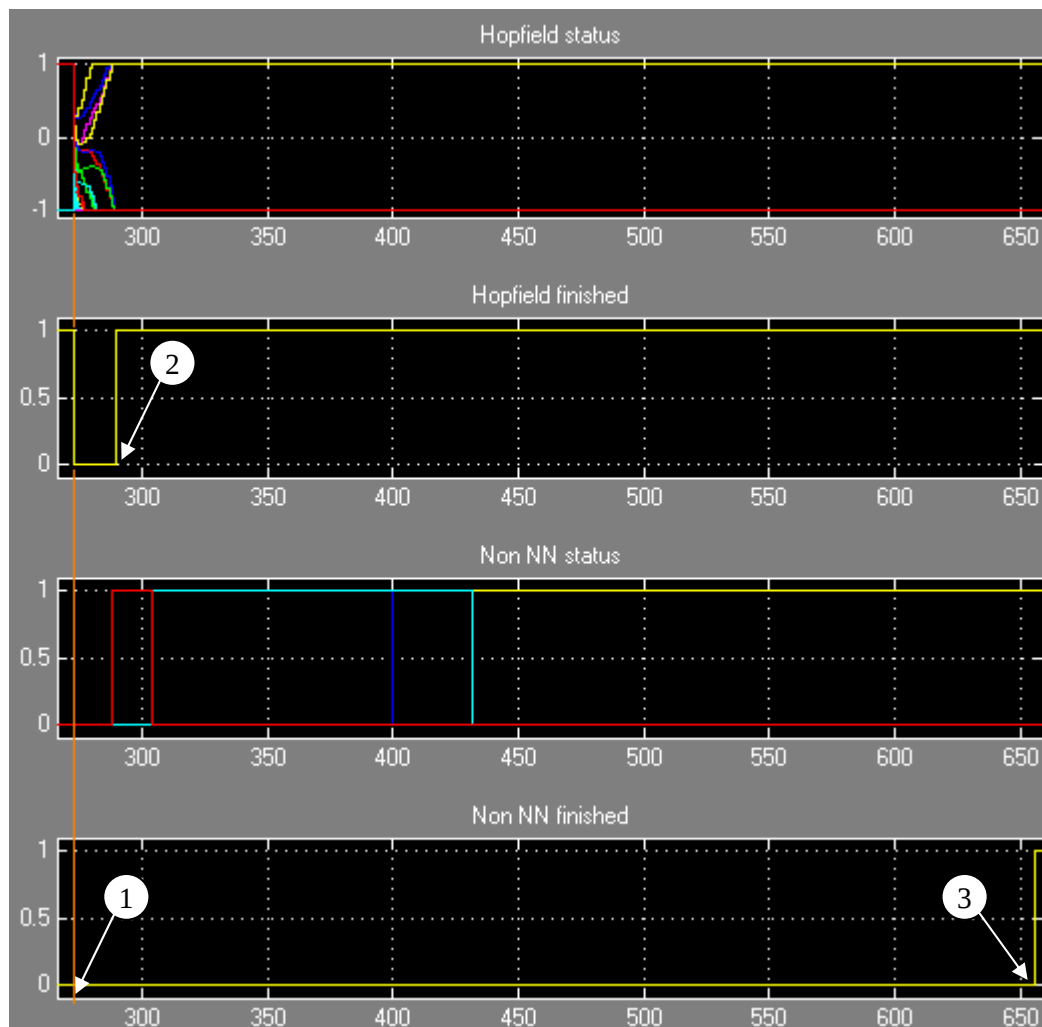
Součet priorit

$$\text{sum} = (8+2+0+3) = 13.$$

Tvrzení týkající se rychlosti výpočtu konfiguračního vektoru jsou podložena simulací. Oba způsoby výpočtu jsou simulovány zároveň. Výsledná doba trvání výpočtu za využití bloku

Standard Computing je pokaždé stejná. Doba výpočtu Hopfieldovou neuronovou sítí se může výrazně měnit, záleží na vstupních hodnotách vektoru priorit. Obrázek vystihující příklad časové náročnosti jednotlivých způsobů výpočtu je na Obr. 50.

Poznámka: Časové hodnoty simulace nejsou uváděny v sekundách, jedná se spíše o počet provedených taktů simulace, proto je uvažována jednotka času simulace [takt].



Obr. 50 Časová náročnost výpočtu konfiguračního vektoru.

- 1) V časovém okamžiku naznačeném oranžovou svislou čarou začíná zároveň výpočet Hopfieldovou sítí i blokem Standard Computing. Tento časový okamžik má hodnotu $t(1) = 273$ takt.
- 2) Při dokončení výpočtu Hopfieldovou sítí se signál Hopfield finished změní z hodnoty 0 na 1. Tento časový okamžik má hodnotu $t(2) = 290$ takt.
- 3) Ukončení výpočtu blokem Standard Computing signalizuje změna signálu nazvaného Non NN finished z hodnoty 0 na 1. Hodnota tohoto časového okamžiku je $t(3) = 656$ takt.

Z naměřených hodnot lze vypočítat, o kolik je rychlejší výpočet blokem Hopfield od výpočtu blokem Standard Computing.

$$\begin{aligned}
 t_{Hop} &= t(2) - t(1) = 290 - 273 = 17 \text{ takt}, \\
 t_{NNN} &= t(3) - t(1) = 656 - 273 = 382 \text{ takt}, \\
 \alpha &= \frac{t_{NNN}}{t_{Hop}} = \frac{382}{17} = 22.5,
 \end{aligned}$$

kde t_{Hop} je čas výpočtu Hopfieldovou sítí, t_{NNN} je čas výpočtu blokem Standard Computing a α je poměr rychlostí, který udává, kolikrát je výpočet Hopfieldovou neuronovou sítí rychlejší než standardní výpočet. Tato hodnota se může obecně lišit, záleží na vstupních datech do Hopfieldovy neuronové sítě, přesto by nikdy neměla klesnout na hodnotu 1, natož menší. Další ukázka získaných hodnot je v následující tabulce:

č. výpočtu	t_{NNN}	t_{Hop}	α	průměrné α
1	382	17	22.47059	20.88397
2		35	10.91429	
3		18	21.22222	
4		12	31.83333	
5		18	21.22222	
6		18	21.22222	
7		22	17.36364	
8		13	29.38462	
9		31	12.32258	

Tab. 2 Srovnání rychlostí výpočtu pomocí Hopfieldovy sítě a výpočtu bez ní.

Průměrná hodnota α z Tab. 2 činí 20.9, tedy výpočet konfiguračního vektoru blokem Hopfield je asi 20.9 krát rychlejší než výpočet blokem Standard Computing.

5 ZÁVĚR

Cílem diplomové práce bylo vytvořit simulační model aktivního síťového prvku s možností prioritního odesílání paketů. Pro rozhodování výběru optimálního odeslání paketů dle jejich priorit byla využita optimalizační umělá neuronová síť a to Hopfieldova. Pomocí simulace bylo zjištěno, že přepínač může být asi 20x rychlejší s implementovanou neuronovou sítí, než při standardním výpočtu, bez ní. Právě tato rychlost rozhodování optimálního výběru činí velkou část z rychlosti funkce celého síťového prvku. Na aktivní prvky jsou kladeny čím dál vyšší nároky ohledně jejich prostupnosti, tedy rychlosti, s jakou pracují. Je to zapříčiněno velkým rozmachem celé škály síťových komunikací a potřeb, zasílat po nich co nejvíce dat za co nejkratší možný čas. Například je možno uvést videokonferenční přenosy, multimediální přenosy, stahování objemných datových souborů atd.

Model je zkonstruován tak, aby se do něj mohly jednoduše implementovat nové optimalizační prvky. Dalším využitím modelu by tedy mohla být simulace jiné optimalizační neuronové sítě, nebo optimalizace za využití evolučních metod, pro porovnání časových náročností těchto způsobů řešení optimalizace. Model lze taktéž brát jako inspiraci při technické konstrukci reálného přepínače.

SEZNAM OBRÁZKŮ A TABULEK

Obrázky dokumentu:

Obr. 1 Struktura přepínače.	13
Obr. 2 Model přepínače.	14
Obr. 3 Ukázka nastavení parametrů pro blok Packet Generator pomocí masky.	15
Obr. 4 Obsah subsystému Packet Generator.	15
Obr. 5 Ukázka výstupního signálu z bloku Packet Generator.	16
Obr. 6 Stateflow tabulka (PGChart).	18
Obr. 7 Subsystém Function-Call Beat a zpracování jeho signálu blokem Demux.	19
Obr. 8 Obsah subsystému Function-Call Beat.	19
Obr. 9 Obsah subsystému Triggered Function Call.	19
Obr. 10 Masky pro nastavení parametru subsystému Main Data Manager.	20
Obr. 11 Jeden ze subsystémů Main Data Manager nastaven na vstupní port číslo 1.	20
Obr. 12 Obsah subsystému Main Data Manager.	21
Obr. 13 Stateflow blok Sort Packets a bloky pro úpravu jeho vstupních dat.	22
Obr. 14 Stateflow diagram bloku Sort Packets.	24
Obr. 15 Ukázka přeskupení dat ve stavu Retrac v tabulce Sort Packets.	24
Obr. 16 Subsystém FIFO, jeho vstupy a výstupy (konkrétně FIFO 1).	25
Obr. 17 Zadáání parametru FIFO Number do masky bloku FIFO.	26
Obr. 18 Obsah subsystému FIFO.	26
Obr. 19 Celek Auto_Push tabulky Queue Chart.	27
Obr. 20 Celek Packet_State tabulky Queue Chart.	28
Obr. 21 Celek Queue tabulky Queue Chart.	29
Obr. 22 Celek Auto_Pop tabulky Queue Chart.	30
Obr. 23 Subsystém Vector_To_FIFO_no.	31
Obr. 24 Obsah subsystému Vector_To_FIFO_no.	31
Obr. 25 Nastavení parametrů bloku Selector.	32
Obr. 26 Nastavení parametrů opakovače Gain.	32
Obr. 27 Blok Matrix Concatenation.	33
Obr. 28 Subsystém Hopfield.	34
Obr. 29 Obsah subsystému Hopfield.	34
Obr. 30 Obsah subsystému Priority Adaptation.	35
Obr. 31 Obsah subsystému Pulse Generator.	35
Obr. 32 Ukázka generování pulsů při pozitivní změně na vstupu.	35
Obr. 33 Podsytem Hopfield NN, jeho vstupní a výstupní porty.	36
Obr. 34 Obsah subsystému Hopfield NN.	36
Obr. 35 Ukázka postupného výpočtu konfiguračního vektoru.	37
Obr. 36 Obsah subsystému Layer 1.	37
Obr. 37 Graf lineární saturační funkce (Satlin).	38
Obr. 38 Obsah subsystému Weight.	38
Obr. 39 Obsah subsystému Dotprod.	38
Obr. 40 Obsah bloku Product Ready.	39
Obr. 41 Alternativní subsystém Serial Computing.	39
Obr. 42 Obsah subsystému Standard Computing.	40
Obr. 43 Stateflow tabulka Serial Computing.	41
Obr. 44 Matice vzorů Examples	42
Obr. 45 Ukázka použití konfiguračního vektoru c	44
Obr. 46 Vznik vektoru priorit p	44
Obr. 47 Ukázka paketů vstupujících a vystupujících z přepínače.	46

Obr. 48 Výsledný signál z Hopfieldovy sítě.	46
Obr. 49 Ukázka výpočtu Hopfieldovy neur. sítě v 3D prostoru.	49
Obr. 50 Časová náročnost výpočtu konfiguračního vektoru.	52

Tabulky dokumentu:

Tab. 1 Pravdivostní tabulka generátoru pulsů.	35
Tab. 2 Srovnání rychlostí výpočtu pomocí Hopfieldovy sítě a výpočtu bez ní.	53

POUŽITÁ LITERATURA

- [1] **The MathWorks, Inc.** *Stateflow® and Stateflow® Coder™ 7 : User's Guide*. [Dokument] 2008.
- [2] **The MathWorks, Inc.** *Neural Network Toolbox™ 6 : User's Guide*. [Dokument] 2008.
- [3] **Medsker, L. R. a Jain, L. C.** *Recurrent Neural Networks : Design and Applications*. London : CRC Press, 2001.
- [4] **Kasabov, Nikola K.** *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*. London : The MIT Press, 1995.
- [5] **Volná, E.** *Neuronové sítě 1*. [Dokument] Ostrava, 2002.
- [6] **Roupec, J.** *Počítačové sítě*. VUT, Brno, 2002.