SLOVENSKÁ TECHNICKÁ UNIVERZITA
FAKULTA CHEMICKEJ A POTRAVINÁRSKEJ TECHNOLÓGIE

Ústav informatizácie, automatizácie a matematiky
Oddelenie informatizácie a riadenia procesov

DIPLOMOVÁ PRÁCA

Spracovanie Simulinkových schém v Matlabe

Vypracoval: Bc. Gabriela Beňová
Vedúci diplomovej práce: Ing. Michal Kvasnica, PhD

Bratislava 2008

Slovenská technická univerzita v Bratislave
Oddelenie informatizácie a riadenia procesov

Fakulta chemickej a potravinárskej technológie
Akademický rok: 2007/2008

S T U
F C H P T

ZADANIE DIPLOMOVEJ PRÁCE

Autorka práce:	**Bc. Gabriela Beňová (16139)**
Študijný program:	chemické inžinierstvo a riadenie procesov
Zameranie:	riadenie procesov
Študijný odbor:	5.2.17 chemické inžinierstvo
Vedúci práce:	Ing. Michal Kvasnica

Názov témy: **Spracovanie Simulinkových schém v Matlabe**

Rozsah práce: 50

Riešenie zadania práce od: 18. 02. 2008
Dátum odovzdania: 23. 05. 2008

L. S.

Bc. Gabriela Beňová
riešiteľka diplomovej práce

prof. Ing. Dr. Miroslav Fikar
vedúci pracoviska

prof. Ing. Vladimír Báleš, DrSc.
garant študijného programu

ČESTNÉ PREHLÁSENIE

Čestne prehlasujem, že diplomovú prácu som vypracovala samostatne, na základe zdrojov uvedených v literatúre, pod vedením vedúceho diplomovej práce a s pomocou vedomostí získaných počas štúdia.

V Bratislave 23. 05. 2008 .....

podpis

POĎAKOVANIE

Ďakujem svojmu školiteľovi Ing. Michalovi Kvasnicovi, PhD. za odborné vedenie pri získavaní vedomostí potrebných k napísaniu tejto práce. Zároveň by som chcela poďakovať Ing. Michalovi Blahovi za cenné rady a pripomienky.

Abstrakt

Diplomová práca sa zaoberá parsovaním SIMULINKových schém v MATLABe. Výsledkom práce je skupina funkcií, ktoré analyzujú danú SIMULINKovú schému, zistia počet a typ blokov v schéme a vytvoria tabuľku prepojení medzi jednotlivými blokmi. Naviac dané funkcie analyzujú informácie vložené v jednotlivých blokoch a poskytnú tieto údaje užívateľovi. Výsledky práce sa dajú využiť pri prevode schém modelovaných pomocou SIMULINKu do iných reprezentácií, prípadne do iných simulačných nástrojov.

V práci je podrobne komentovaný zdrojový kód programu pričom postup riešenia problémov je vysvetlený na konkrétnych príkladoch.

Abstract

This master thesis deals with parsing of SIMULINK schemas in MATLAB. The result of this work is a set of functions which analyze a given SIMULINK model, figure out the number and types of blocks used in the model and create a table of interconnections between individual blocks. Moreover, the program analyzes the numerical parameters of individual blocks and return these data to the user. The results of this thesis can be used to convert SIMULINK models into different representations, or, alternatively, into alternative modeling tools.

The thesis presents a detailed description of the code and the operation principle is explained on concrete examples.

Obsah

Zoznam obrázkov

Úvod

Matematické modelovanie na počítačoch sa stalo účinnou metódou riešenia vedecko - technických problémov. Počítačový model zostavený na základe matematickej analýzy predstavuje náhradnú vzorku skúmaného objektu. S modelom môžeme vykonávať experimenty podobné tým na reálnom objekte no s nižšími nákladmi a máme možnosť experimenty niekoľko krát opakovať.

So stále narastajúcim počtom aplikácii stúpa aj náročnosť experimentov a niekedy je nutné zjednodušiť manipuláciu s nimi. Práve pre tento prípad je vytvorený nasledovný program. Cieľom diplomovej práce je vytvoriť program pre spracovanie zložitých SIMULINKových schém, ktorý popíše štruktúru celého systému vrátane parametrov blokov. Program je vytvorený ako jedna funkcia s viacerými subfunkciami preto ho bude možné použiť ako súčasť iných aplikácii pre prácu so SIMULINKovými schémami. Funkcia vytvára štruktúru pozostávajúcu zo všetkých blokov a ich parametrov, tabuliek portov a liniek a prepojenia portov.

Program je vytvorený v MATLAB verzii 7.1 a podporuje vyššie verzie.

Diplomová práca pozostáva z 3 kapitol:

Kapitola 1 je úvodom do MATLABu a SIMULINKu pre objasnenie niektorých výrazov používaných v texte.

V kapitole 2 je uvedený podrobný scenár vývoja programu na parsovanie SIMULINKových schém aj s príkladmi riešenia.

V kapitole 3 sú jednoduché ukážky ako sa dá program ďalej využiť a ako je možné čítať parametre zo štruktúry vytvorenej v kapitole 2.

1 MATLAB a SIMULINK

1. 1 MATLAB

Výpočtový systém MATLAB sa behom posledných rokov stal celosvetovým štandardom v oblasti technických výpočtov a simulácii. Je považovaný za prelom nielen z hľadiska rozsahu, integrácie a kvality produktu, ale predovšetkým z hľadiska vzťahu k užívateľovi a jeho pohodliu pri práci**[1]**.

MATLAB poskytuje svojím užívateľom nielen mocné grafické a výpočtové nástroje, ale aj rozsiahle knižnice spolu s výkonným programovacím jazykom štvrtej generácie. Knižnice sú svojím rozsahom využiteľné prakticky vo všetkých oblastiach ľudskej činnosti. Možnosti aplikácie MATLABu**[1]**:

- technické výpočty
- modelovanie a simulácia
- meranie a testovanie
- riadiaca technika
- komunikácia
- spracovanie obrazu a videa
- vizualizácia dát
- finančné modelovanie a analýzy

Názov MATLAB vznikol spojením *MATrix LABoratory*. Je to výpočtové prostredie ktorého základný dátový objekt je matica. To dovoľuje riešiť technické výpočtové problémy a hlavne tie, ktoré sú formulované vo vektorovom a maticovom tvare rýchlejšie v porovnaní s klasickým prístupom, ktorý ponúkajú programovacie jazyky C alebo Fortran**[2]**.

MATLAB je systém balíkov (programovací balík) od firmy The MathWorks, ktorý pozostáva z nasledovných častí**[3]**:

1. MATLAB – je to základ pre všetky ostatné časti. Pozostáva z piatich častí

    - programovací jazyk MATLABu – je to najvyšší jazyk, ktorý pracuje s maticami/poliami, je objektovo orientovaný (od verzie 2008a je plná podpora objektov)
    - pracovné prostredie MATLABu – nástroje na vývoj, spracovanie, odladenie aplikácií, prácu s premennými, import a export dát
    - grafika – je to grafický systém v MATLABe, ktorý v sebe zahŕňa od príkazov na vytvorenie 2D a 3D vizualizácie dát, spracovanie obrazov, animácie a prezentačnú grafiku, až po základné príkazy, ktoré umožňujú užívateľovi prispôsobenie grafiky podľa jeho predstáv a vybudovanie grafického užívateľského rozhrania pre užívateľské aplikácie v MATLABe.
    - knižnica matematických funkcií MATLABu – je to veľká zbierka výpočtových algoritmov od základných funkcií až po zložité algoritmy (napr. Fourierová transformácia)
    - Application Programming Interface (API) – knižnice, ktoré užívateľovi umožňujú vytvárať programy v jazykoch C a Fortrane, ktoré spolupracujú s MATLABom

2. Rozšírenie MATLABu – voliteľný nástroj na podporu implementácie systémov vyvinutých v MATLABe (kompilátor, C/C++ matematické knižnice)

3. Toolboxy – voliteľné knižnice špecializovaných funkcií MATLABu, ktoré umožňujú prispôsobenie MATLABu na riešenie špecifických problémov užívateľa. Tieto knižnice sú otvorené a užívateľ si môže pozrieť a vytvoriť vlastné nové funkcie s využitím kódu pôvodných funkcií.

4. SIMULINK – voliteľný grafický interaktívny program na simuláciu lineárnych a nelineárnych dynamických systémov. Umožňuje modelovanie systémov pomocou grafických blokových schém.

5. Blocksets – kolekcia blokov pre SIMULINK pre špecializované aplikácie (DSP, návrh nelineárneho riadenia)

6. Stateflow – nástroj pre modelovanie udalosťami riadených systémov

1.2 SIMULINK

SIMULINK je grafickou nadstavbou programového balíka MATLAB. Pri simuláciách využíva jeho výpočtové jadro a dokáže pristupovať k celej škále rozširovacích modulov – toolboxov.

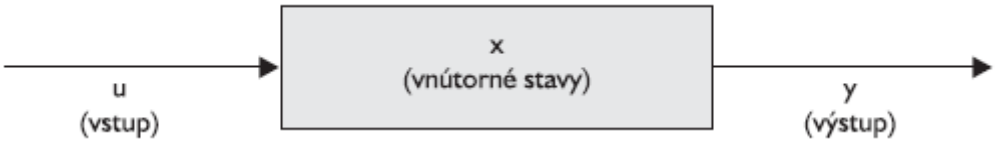
SIMULINK dovoľuje jednoducho a rýchlo vytvárať modely dynamických sústav vo forme blokových schém alebo rovníc. Pomocou SIMULINKu a jeho grafického editora je možné vytvárať modely lineárnych a nelineárnych, v čase diskrétnych alebo spojitých systémov obyčajným presúvaním funkčných blokov z knižníc a ich prepájaním myšou. SIMULINK je nadstavba MATLABu a využíva jeho algoritmy pre numerické riešenie nelineárnych diferenciálnych rovníc, ktoré simulačné modely popisujú**[4]**.

Významným prvkom je otvorená architektúra, ktorá dovoľuje užívateľovi vytvárať si vlastné funkčné bloky a užívateľské knižnice a rozširovať takto už jestvujúce knižnice. Hierarchická štruktúra modelov umožňuje koncipovať i veľmi zložité systémy pre prehľadné sústavy subystémov prakticky bez obmedzení počtu blokov. Grafické možnosti SIMULINKu sú na takej úrovni, že vytvorenú schému je možné použiť priamo ako dokumentáciu pri vývoji systému**[4]**.

Je určený na časové riešenie – simuláciu – chovania sa dynamického systému pokiaľ poznáme matematický popis. S jeho pomocou je možné určiť časové priebehy

výstupných veličín (a všetkých vnútorných) v závislosti od časového priebehu veličín vstupných a počiatočného stavu**[5]**.

Na tvorbu modelov netreba programovať zložité diferenciálne alebo diferenčné rovnice, ale pomocou blokových schém sa skladajú jednotlivé funkčné bloky, ktoré sa vzájomne prepájajú a tým tvoria medzi sebou vazby. Modelované systémy môžu byť spojité, diskrétne, prípadne ich kombináciou sa vytvárajú modely hybridných systémov. Simulačná schéma pozostáva z blokov a signálnych väzieb (obr. 1).



Obr.: 1.1 Blok simulačnej schémy

Do bloku vstupuje vstupný signál *u*. V bloku sa spracuje jeho hodnota a generuje sa výstupný signál *y*. Ak blok modeluje určitú dynamiku, tak výstup *y* sa počíta na základe stavov *x* a vstupu *u* **[6]**.

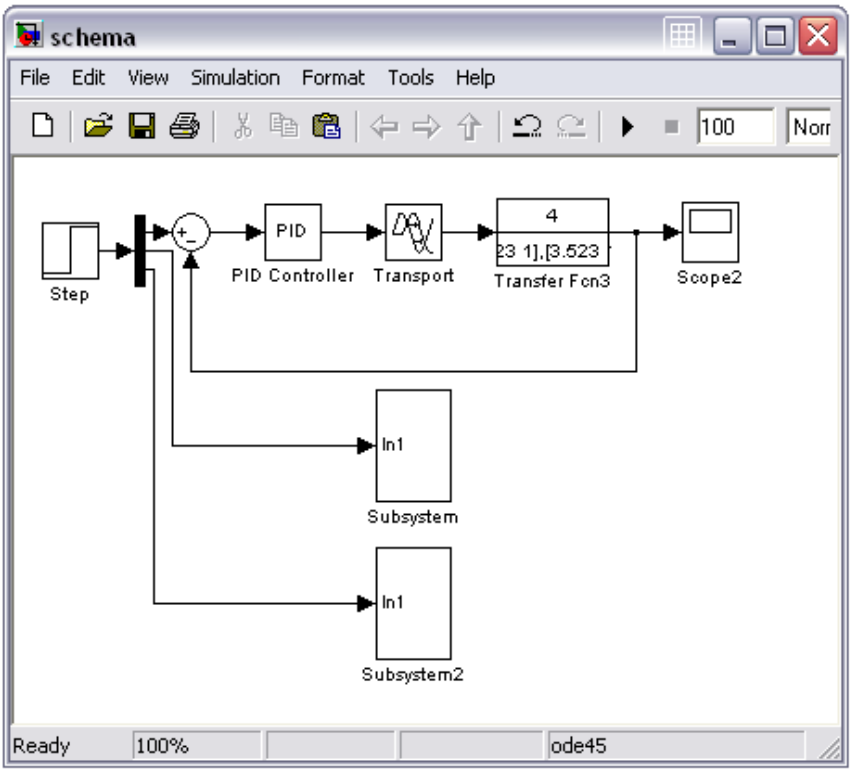
2 Vytváranie prepojení medzi funkčnými blokmi

2.1 Modelová schéma a niektoré dôležité príkazy pre prácu

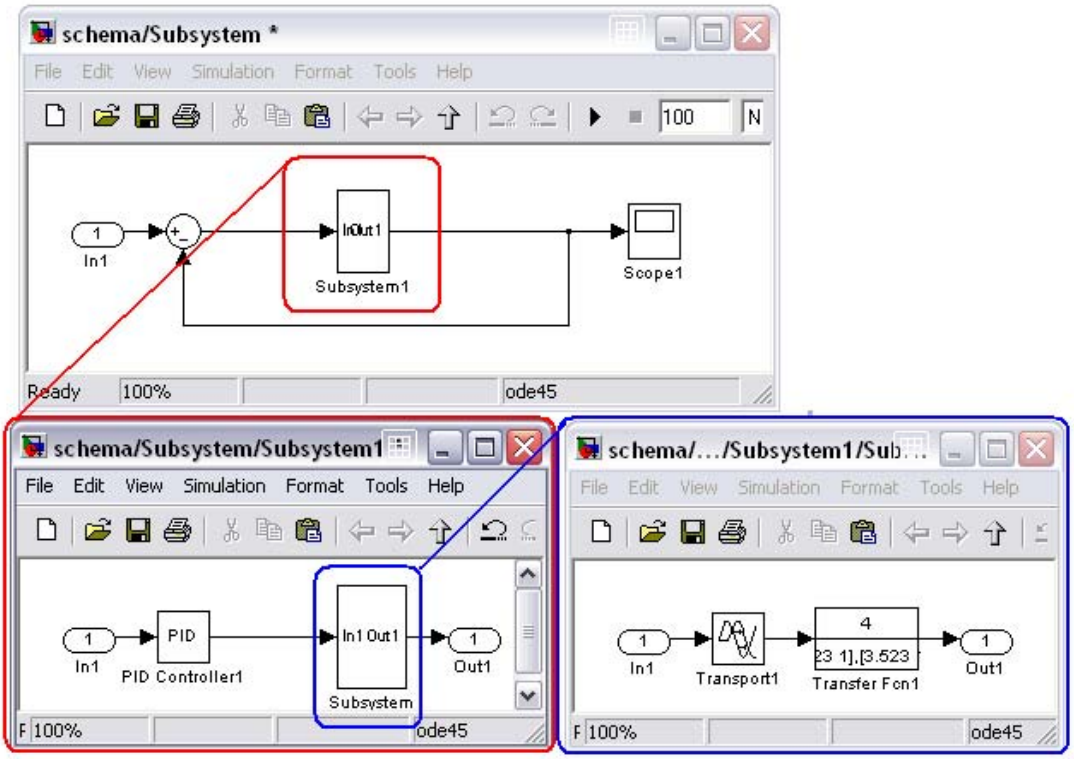
Každý blok v SIMULINKovej schéme má jeden alebo viac vstupných a výstupných portov. Tieto porty sú očíslované čo umožní prácu s ich handlermi. Rovnako sú očíslované spojenia (linky) spájajúce jednotlivé funkčné bloky.

Vytvorili sme si jednoduchú schému spätnoväzbového riadenia pre získavanie parametrov funkčných blokov v schéme. Schéma obsahuje aj masky subsystémov (podsystémy) a viacnásobné maskovanie týchto subsystémov. Celková schéma so všetkými blokmi je na obr. 2.1. Jednotlivé podsystémy bloku *Subsystem* sú zobrazené na obr. 2.2, pričom podsystém v červenom rámiku je už podsystémom podsystému a podsystém v modrom rámiku je 3 krát maskovaný. Blok *Subsystem2* je maskovaný úplne analogicky ale iba do druhého stupňa (obr. 2.3). Ako vidno z obrázkov v jednotlivých maskách sa môžu opakovať názvy blokov. Tak isto môžu mať bloky aj iné názvy ako sú názvy predvolené SIMULINKom. Takže užívateľ si môže premenovať jednotlivé bloky alebo subsystémy v schéme podľa vlastných potrieb.

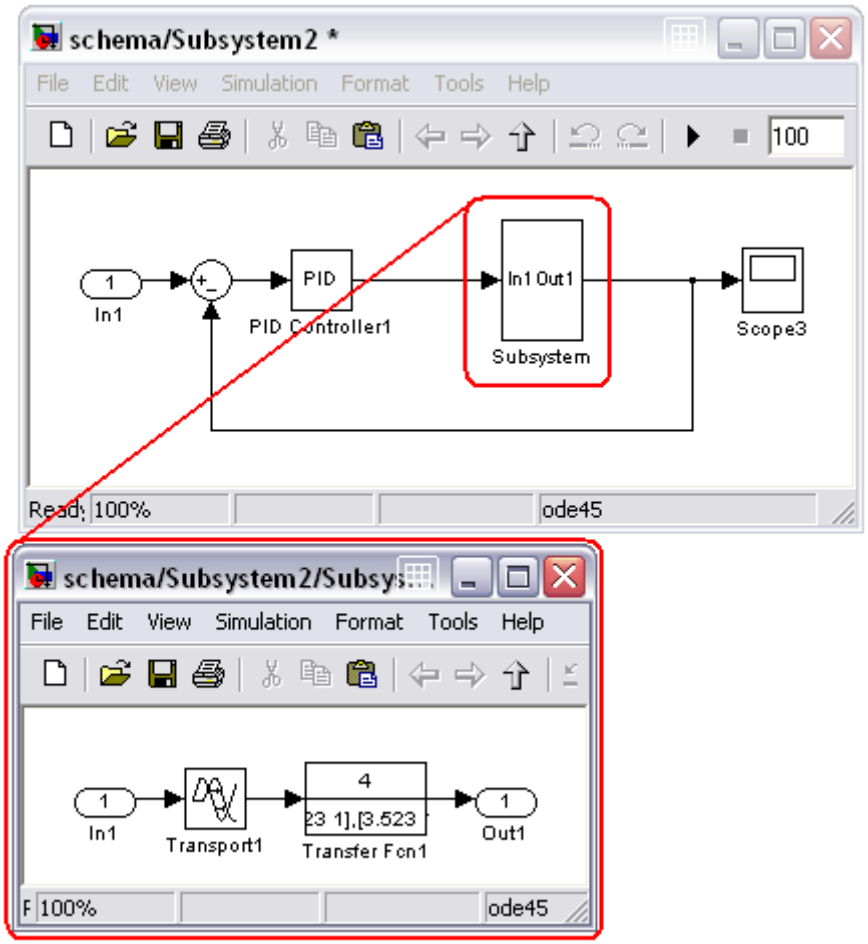
Schéma na obr. 2.1 je iba modelová schéma slúžiaca pre vývoj programu. Je plne funkčná a viacnásobne maskovaná aby sme v programe ošetrili aj vyberanie parametrov z maskovaných subsystémov. Úmyselne sme nemenili názvy jednotlivých blokov aby bolo jasné o aký blok sa jedná a aby bolo vidno, že sa tieto názvy môžu opakovať v jednotlivých úrovniach maskovania.



Obr. 2.1: Bloková schéma s subsystémami



Obr. 2.2: Maskovanie bloku *Subsystem*



Obr. 2.3: Maskovanie bloku *Subsytem2*

Aby sme získali špecifikáciu systému, resp. jednotlivých blokov alebo subsystémov využijeme nasledovné príkazy **[7]**:

- find_system – vráti špecifikáciu systému alebo bloku
- get_param – zistí parametre systému alebo bloku
- get – zobrazí vlastnosti jednotlivých objektov
- set_param – umožní nastavenie parametrov systému
- gcb – vráti cestu ku aktuálnemu bloku
- gcs – vráti cestu ku aktuálnemu systému

Ak by sme chceli v našej SIMULINKovej schéme s názvom *schema* obr. 2.1 nájsť všetky bloky s názvom *Scope* zadáme príkaz *find_system* nasledovne:

```
blok = find_system('schema','BlockType','Scope')
```

príkaz nám vráti všetky bloky zo systému *schema* ktorých blokový typ je *Scope*.
Aby sme zistili všetky bloky nachádzajúce sa v danej schéme vyzeral by príkaz nasledovne:

```
bloky = find_system('schema', 'FindAll','on','type','block')
```

príkaz vráti čísla všetkých blokov nachádzajúcich sa v schéme vo vektorovom tvare. Pre získanie parametrov jednotlivých blokov použijeme príkaz get(). Aby sme zistili parameter napr. prvého bloku z premennej bloky príkaz napíšeme v tvare:

```
a = get(bloky(1))
```

vráti nám všetky parametre daného bloku. S týmito je možné nasledovne pracovať a vkladať ich do premenných:

```
a1 = a.Path
```

Analogicky môžeme zistiť aj parametre portov a čiar (liniek):

```
porty = find_system('schema', 'FindAll', 'on', 'type', 'port')
linky = find_system('schema', 'FindAll', 'on', 'type', 'line')
```

2. 2 Volanie funkcie z MATLABu

V kapitole 2.1 sme v príkazoch volali vždy systém *schema* a to v každom príkaze osobitne. Aby sme sa vyhli tomuto prepisovaniu je užitočné vložiť si názov schéma do premennej a volať funkciu s príkazmi priamo z príkazového riadka. Toto nám bude užitočné pri práci s subsystémami.

Funkcia *main12* je volaná z príkazového riadku so vstupnou premennou *nazov* do ktorej vkladáme názov systému teda našej SIMULINKovej schémy.

Čo ak zadáme do premennej *nazov* názov schémy v tvare nezodpovedajúcom samotnému názvu schémy? Napr.:

```
>> nazov = 'schema/Subsystem1'
```

Preto je nutné z názvu volaného do funkcie odfiltrovať iba názov samotnej schémy. Na obr. 2.4 je časť kódu funkcie slúžiaca práve na tento účel. Príkaz `strfind(nazov,'/')` hľadá v názve všetky lomítka a vkladá ich do premennej *t12*. Lomítko je oddeľovač medzi systémom a jeho podsystémom. Ak názov obsahuje lomítka je dlhý a pomocou príkazu:

```
t23 = regexp(nazov, pat, 'tokens');
```

nájde všetky časti názvu nachádzajúce sa pred nejakým lomítkom a priradí ich do premennej *t23*. V nasledujúcom kroku vyčleníme z *t23* iba prvú hodnotu čo je iba názov samotnej schémy. Keďže MATLAB je stavaný pre prácu s maticami je nutné prekonvertovať názov z dátového tvaru *cell array* do tvaru maticového pomocou príkazu:

```
nazov_schemy=cell2mat(nazov12);
```

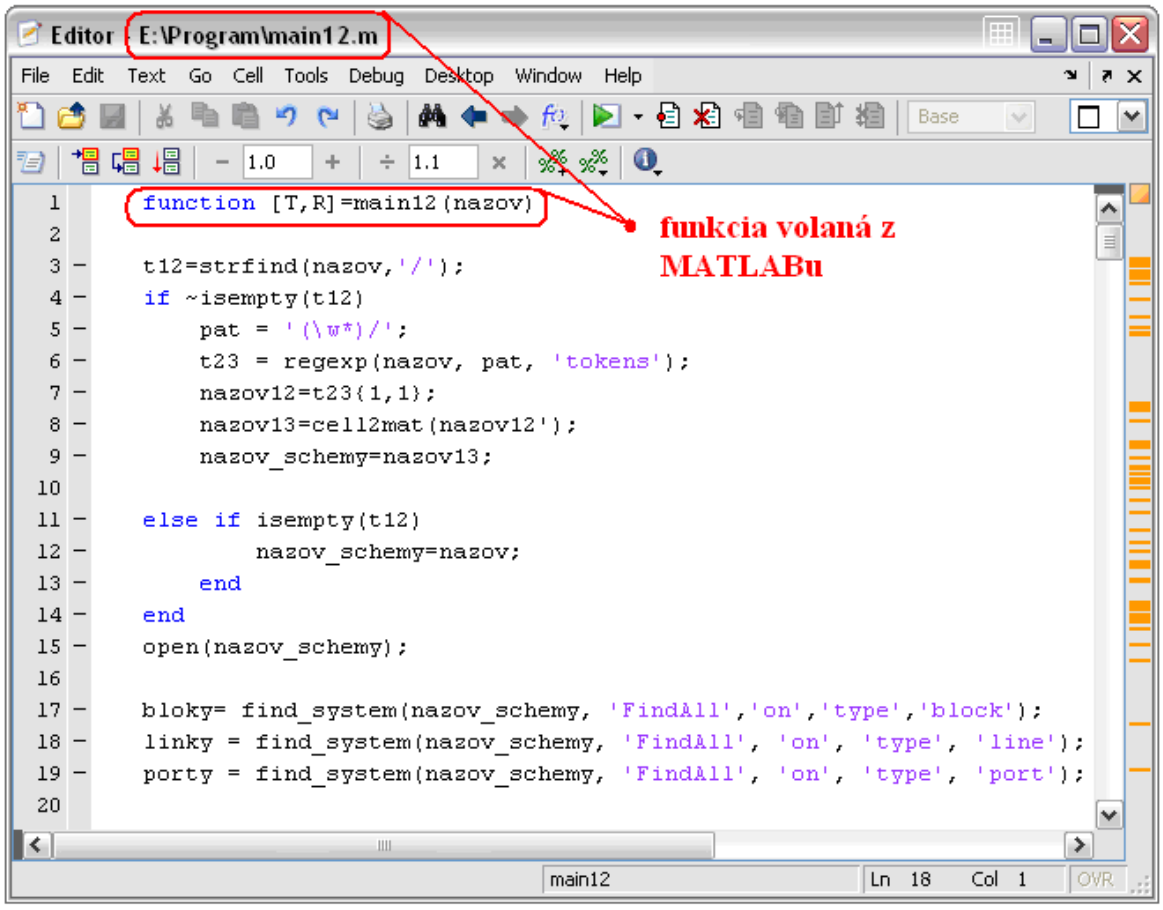
Potom môžeme v každom nasledujúcom príkaze, ktorý volá názov systému volať iba premennú s názvom schémy. Na otvorenie schémy použijeme príkaz:

```
open(nazov_schemy);
```

Rovnako by sme mohli použiť príkaz:

```
sim(nazov_schemy);
```

ale pre vytváranie prepojení a získavanie parametrov blokov nie je nutné schému spúšťať. Zbytočne by to spomaľovalo program.

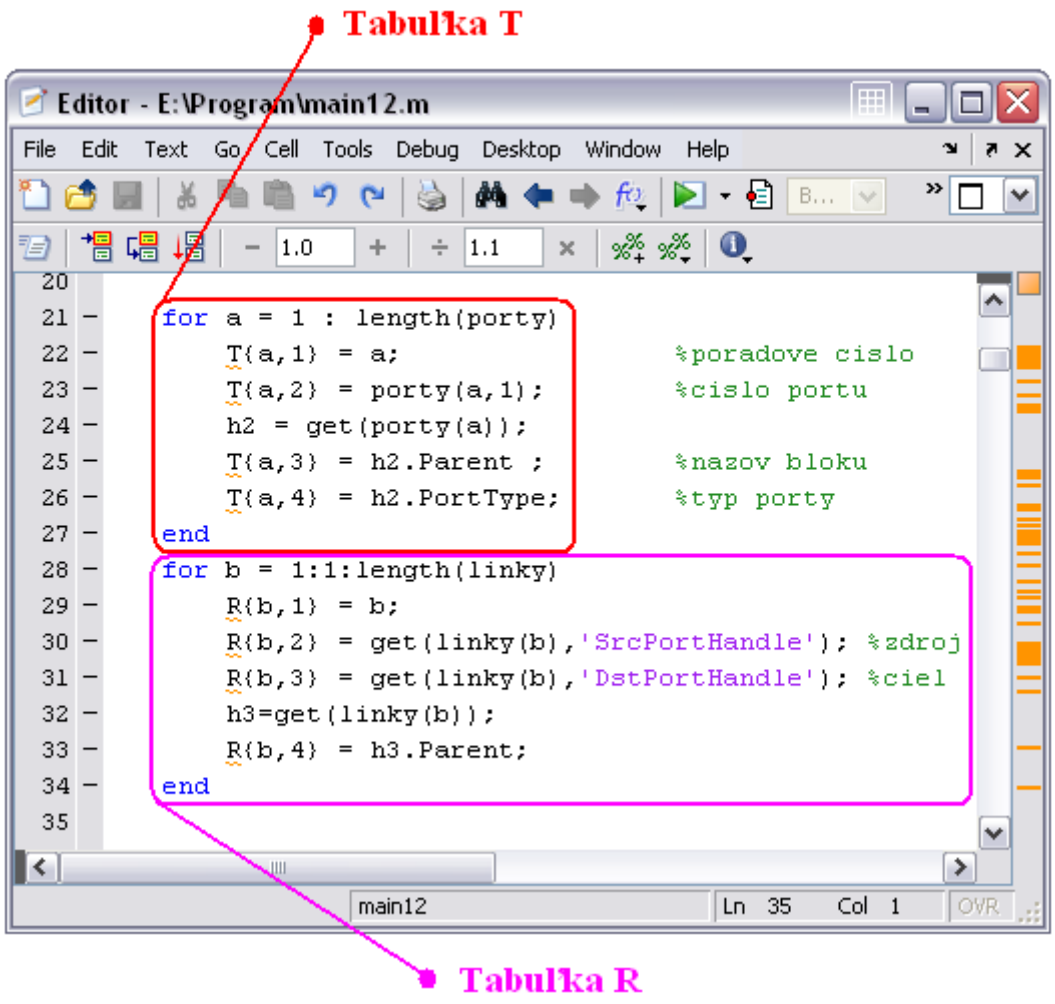


Obr. 2.4: Volanie funkcie z MATLABu

2. 3 Tabuľky portov a liniek

Aby sme mohli s parametrami, ktoré sme získali z danej schémy pracovať je vhodné vložiť ich do tabuliek a ďalej volať tieto parametre z vytvorených tabuliek. Vytvoríme si tabuľku parametrov portov *T* a tabuľku parametrov liniek *R*. Je nutné vytvoriť obe lebo liniek je menej ako portov. V nasledujúcom kroku ich budeme potrebovať na vytvorenie prepojenia medzi jednotlivými portami.

Na obr. 2.5 je zdrojový kód ktorý vytvára obe tabuľky. Využijeme na to *handle* portov a liniek.



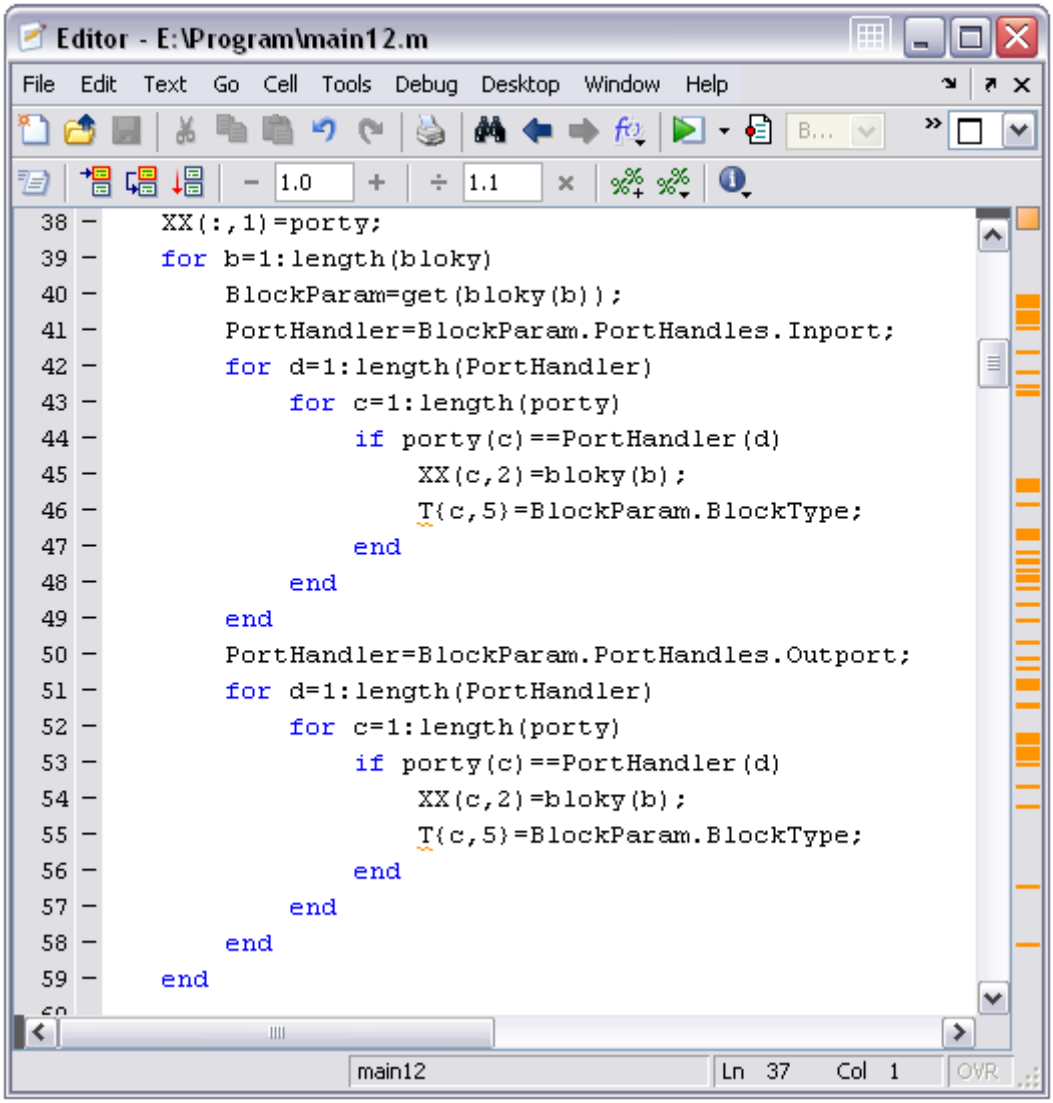
Obr. 2.5: Vytvorenie tabuliek portov a liniek

Obe tabuľky vytvárame pomocou cyklov, do ktorých voláme jednotlivé parametre. Význam niektorých z týchto parametrov je **[7]**:

- Type – pre tabuľku T je to port a pre tabuľku R je line
- Parent – celý názov podsystému alebo bloku kde sa nachádza port s daným číslom alebo odkiaľ vychádza čiara
- Handle – číslo portu alebo čiary
- SrcPortHandle – číslo zdrojového portu
- DstPortHandle – číslo zdrojového bloku

Teraz sme vytvorili dve tabuľky ktoré zobrazujú všetky *porty* a všetky *linky* nachádzajúce sa v danej schéme. Tabuľka *T* pozostáva z poradového čísla jednotlivého portu, čísla portu, názvu bloku ktorý je k danému portu priradený a typu portu (vstupný alebo výstupný). Tabuľka *R* pozostáva z poradového čísla linky, čísla zdrojového portu (odkiaľ linka vychádza), čísla cieľového portu (kam linka vstupuje) a názvu jednotlivého bloku. Tu sa môžu jednotlivé bloky opakovať, pretože schéma sa rozvetvuje a obsahuje spätnú väzbu. Preto jeden zdrojový port môže byť priradený ku viacerým cieľový avšak líšia sa číslami liniek.

Ku tabuľke *T* priradíme do posledného stĺpca typ bloku, ktorý využijeme pri filtrovaní jednotlivých úrovní maskovania. Na obr. 2.6 je zložený cyklus ktorý slúži na priradenie typu bloku k portu. Matica *XX* je iba pomocná matica pre lepšiu názornosť. Cyklus ide po celej dĺžke matice *bloky*.



Obr. 2.6: Priradenie typu bloku k jeho číslu portu a názvu

Príkaz `BlockParam=get(bloky(b))` vyberie parametre každého bloku z matice blokov s názvom *bloky*. Z všetkých parametrov tohto bloku pre zistenie čísel jednotlivých portov použijeme položku:

```
PortHandles: [1x1 struct]
```

Pre názornosť parametre bloku *demux* vyberieme nasledovne:

```
>> BlockParam=get(bloky(1))
>> PortHandler=BlockParam.PortHandles
```

```
PortHandler =

      Inport: 102.0021
     Outport: [160.0012  161.0010  162.0010]
      Enable: []
     Trigger: []
       State: []
       LConn: []
       RConn: []
    Ifaction: []
```

Následne sa už pýtame iba na konkrétne porty (*Inport* alebo *Outport*), porovnáme ich s číslom portu z premennej *porty* a v prípade rovnosti ich priradíme ku číslu portu v matici *XX*. Nie je možné priradenie urobiť v jednom cykle pretože výstupných portov môže byť viac ako vstupných.

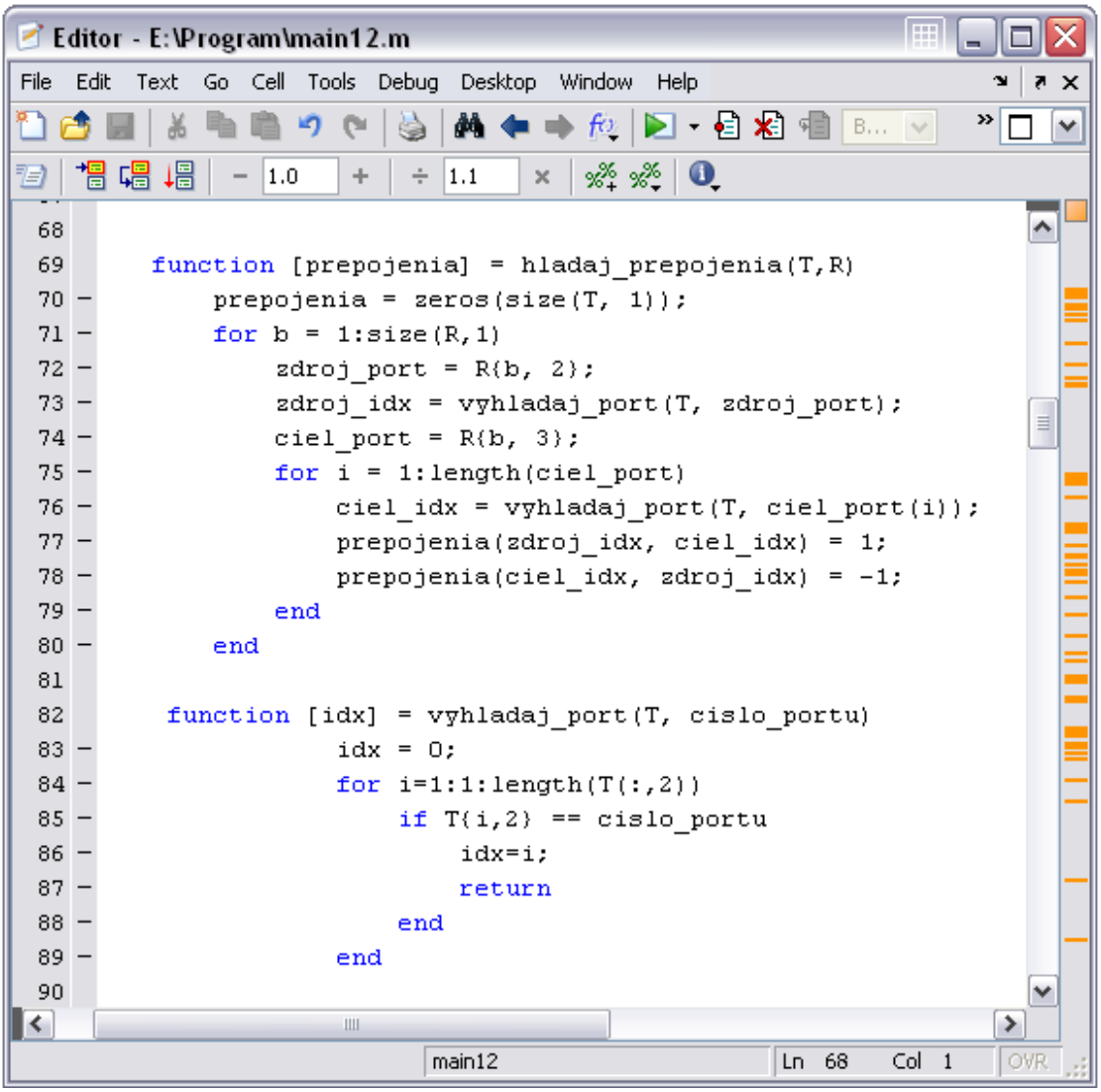
2. 4 Prepojenia portov a liniek

V predošlých dvoch kapitolách (kapitola 2.2 a kapitola 2.3) sme vytvorili dve tabuľky, tabuľku *T* portov a tabuľku *R* liniek. Na obr. 2.7 je funkcia ktorá vytvára vzájomné prepojenia portov, ktorej výstupom je booleovská matica rozmeru N x N, kde N je počet prvkov tabuľky *T*.

V prvom kroku si vytvoríme prázdnu maticu *prepojenia* danej veľkosti ktorú budeme napĺňať. Z tabuľky *R* linek voláme čísla portov a priraďujeme ich do premenných. Do premennej *zdroj_port* vkladáme čísla portov zdrojových a do premennej *ciel_port* čísla portov cieľových. Následne voláme funkciu *vyhladaj_port* ktorá hľadá indexy portov. Tieto indexy sú poradovými číslami v tabuľke *T* a určujú polohu v matici *prepojenia*. Funkcia je v tvare:

```
zdroj_idx = vyhladaj_port(T, zdroj_port)
```

kde indexy sú poradové čísla portov v matici *T*.



Obr. 2.7: Vytvorenie prepojení medzi portami

Funkcia obsahuje cyklus, ktorý prejde po všetkých číslach portov v tabuľke *T* a porovná ich. V prípade zhody vráti číslo zdrojového indexu *zdroj_idx*. To znamená že porty sú prepojené linkou z tabuľky *R* s poradovým číslom *zdroj_idx*. Táto linka vychádza z portu *zdroj_port* s týmto poradovým číslom. Vo funkcii *hladaj_prepojenia* je cyklus ktorý prejde po celej dĺžke stĺpca cieľových portov tabuľky *R* a zavolá znovu funkciu *vyhladaj_port* v tvare:

```
ciel_idx = vyhladaj_port(T, ciel_port(i))
```

V prípade zhody funkcia vráti index cieľového portu *ciel_idx z tabuľky T* pre danú linku.

Tieto indexy určujú polohu v matici *prepojenia*. Pre vysvetlenie si označíme *zdroj_idx* ako *„a“, ciel_idx* ako *„b“*. Prázdnu maticu *prepojenia* napĺňame nasledovne:

- ak *zdroj_idx* je v riadku a *ciel_idx* je v stĺpci, tak smer prepojenia a matica bude vyzerať:
  a→b *prepojenia*(a,b) = 1

- ak *zdroj_idx* je v stĺpci a *ciel_idx* je v riadku, tak smer prepojenia a matica bude vyzerať:
  a←b *prepojenia*(b,a) = -1

Na obr. 2.7 *zdroj_idx* a *ciel_idx* napĺňanie matice *prepojenia* je reprezentované riadkami:

```
prepojenia(zdroj_idx, ciel_idx) = 1;
prepojenia(ciel_idx, zdroj_idx) = -1;
```

V prvom prípade vkladá v „1“ ak poradové číslo zdrojového portu z tabuľky *T* (riadky matice *prepojenia*) je vstupom do cieľových portov z tabuľky *T* (stĺpce matice *prepojenia*). V druhom prípade vkladá „-1“ ak do poradového čísla zdrojového portu z tabuľky *T* (riadky matice *prepojenia*) vstupujú linky z cieľových portov z tabuľky *T* (stĺpce matice *prepojenia*).

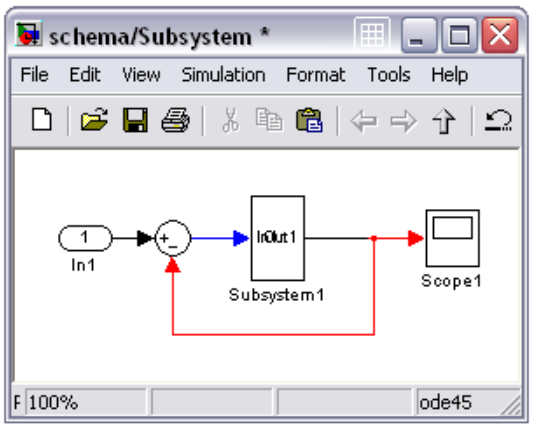
Na obr. 2.8 sú schematicky zobrazené prepojenia portov pre blok *Subsystem* zo schémy (obr. 2.1). Maticu *prepojenia* čítame po riadkoch.

Modrým je zobrazený vstup do bloku *Subsystem1* (poradové číslo 3) z bloku *Sum* (poradové číslo 7) a jemu priradená hodnota „1“. Červeným je zobrazený výstup z bloku *Subsystem1* (poradové číslo 4) do blokov *Sum* (poradové číslo 6) a *Scope* (poradové číslo 2). Na schéme sú tieto vstupy a výstupy zobrazené rovnakou farbou. V tabuľke *R* je však ešte jeden vstup z bloku *Subsystem1*. Tento vstup vznikol rozdelením čiary výstupu. SIMULINK si pri každom vetvení čiary túto čiaru rozdelí na niekoľko ďalších. Táto výstupná čiara je v schéme čiernou farbou a vstupuje do uzla, ktorý sa delí na vstup do *Out1* a *Sum2*. Nemá vplyv na prepojenia portov.

Rovnako je možné čítať maticu *prepojenia* aj po stĺpcoch a rozdiel bude iba v znamienku pri „1“.

Obr. 2.8 je vytvorený pre konkrétny podsystém systému *schema* no v kapitole 2.2 a 2.3 máme vytvorené tabuľky všetkých portov a liniek bez rozdelenia na podsystémy. Takéto triedenie na jednotlivé úrovne je v kapitole 2.5.

```
TRes =

    [1]    [2.2070e+003]    [1x20 char]    'outport'    'Inport'
    [2]    [2.2080e+003]    [1x23 char]    'inport'     'Scope'
    [3]    [2.2090e+003]    [1x27 char]    'inport'     'SubSystem'
    [4]    [2.2100e+003]    [1x27 char]    'outport'    'SubSystem'
    [5]    [2.2230e+003]    [1x21 char]    'inport'     'Sum'
    [6]    [2.2240e+003]    [1x21 char]    'inport'     'Sum'
    [7]    [2.2250e+003]    [1x21 char]    'outport'    'Sum'

RRes =

    [1]    [2.2070e+003]    [2.2230e+003]    'schema/Subsystem'
    [2]    [2.2100e+003]    [2.2080e+003]    'schema/Subsystem'
    [3]    [2.2100e+003]    [2.2240e+003]    'schema/Subsystem'
    [4]    [2.2100e+003]     [2x1 double]    'schema/Subsystem'
    [5]    [2.2250e+003]    [2.2090e+003]    'schema/Subsystem'

prepojenia =

     0     0     0     0     1     0     0
     0     0     0    -1     0     0     0
     0     0     0     0     0     0    -1
     0     1     0     0     0     1     0
    -1     0     0     0     0     0     0
     0     0     0    -1     0     0     0
     0     0     1     0     0     0     0
```

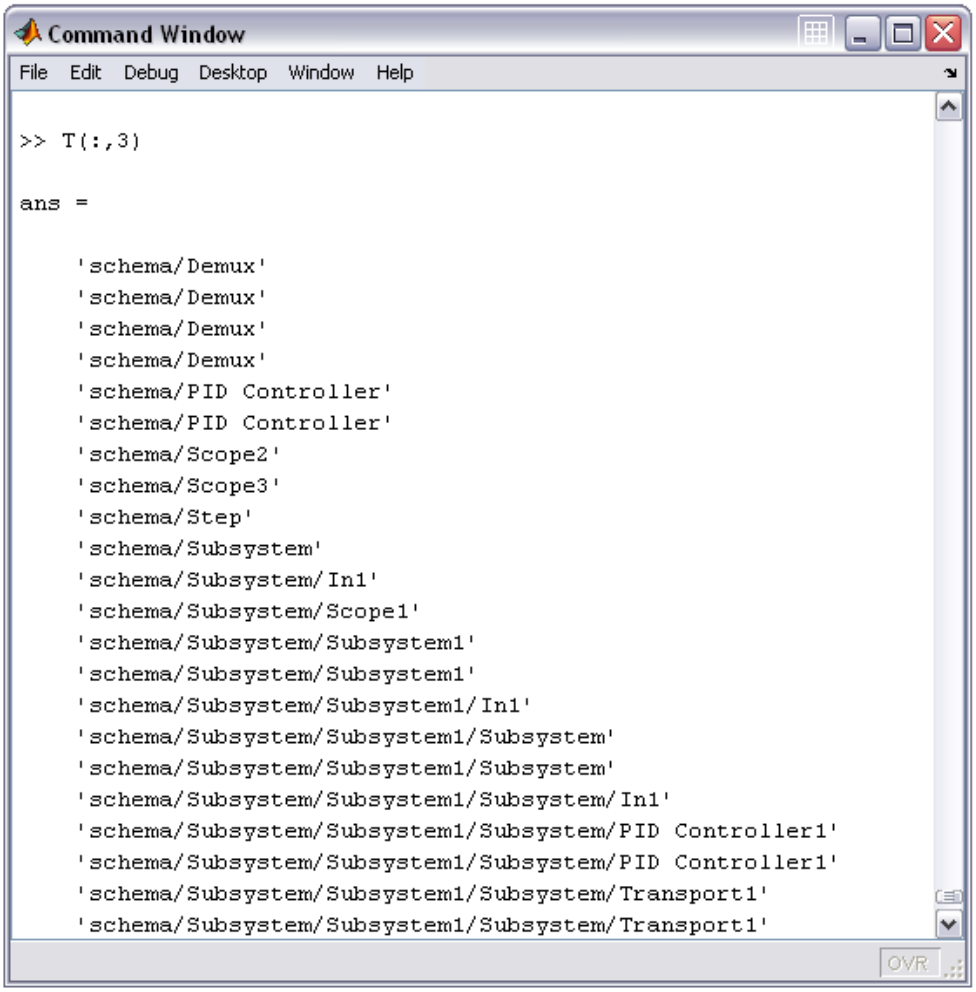


Obr. 2.8: Prepojenia medzi portami

2. 5 Filtrovanie jednotlivých úrovní maskovania

Aby sme zistili aké bloky sa nachádzajú v hlavnom systéme (obr. 2.1) a v jednotlivých podsystémoch (obr. 2.2 a obr. 2.3) je nutné odfiltrovať tie bloky ktoré sa nenachádzajú na danej úrovni. Takto zistíme nové tabuľky *T* a *R* pre prepojenia portov iba na danej úrovni. Úrovňou rozumieme buď hlavný systém alebo každý samostatný podsystém.

Na obr. 2.9 je časť tretieho stĺpca tabuľky portov *T*. Sú to názvy jednotlivých blokov, kde každý podsystém je oddelený od systému lomítkom. Z obrázka vidno, že na prvej úrovni sa nachádzajú iba všetky bloky ktorých názov obsahuje 'schema/' ale iba s jedným lomítkom. Toto bude prvé kritérium filtrovania tabuľky *T*.



Obr. 2.9: Názvy blokov v tabuľke *T*

Aby sme mohli z tabuľky *T* odfiltrovať jednotlivé úrovne najprv musíme nájsť všetky bloky ktorých názvov (T(:,3)) je v tvare ’nazov/’. Pre prvú úroveň (obr. 2.1) nájdeme bloky ktorých názov v tabuľke T je práve ’schema/’. Funkcia `strcat({nazov},{'/'})` prilepí na koniec názvu *nazov* volaného z príkazového riadka lomítko. Nasledovne nájdeme všetky lomítka v tomto názve cez príkaz `strfind(nazov_n0,'/')` a rovnako nájdeme všetky lomítka v názvoch blokov z tretieho stĺpca tabuľky *T*. Príkazom `strfind(T{i, 3},nazov_n1)` nájdeme v tabuľke iba tie názvy ktoré obsahujú ’schema/’.

Ak premenná *fo1* do ktorej je vkladaná pozícia lomítok nie je prázdna a dĺžka názvu je práve rovná ’schema/’ vráti nám iba tabuľku *TRes* ktorá je tabuľkou danej úrovne.

Na obr. 2.10 je podrobný kód tohto filtrovania. Posledný cyklus cez *TRes* slúži iba na prečíslovanie poradových čísel.

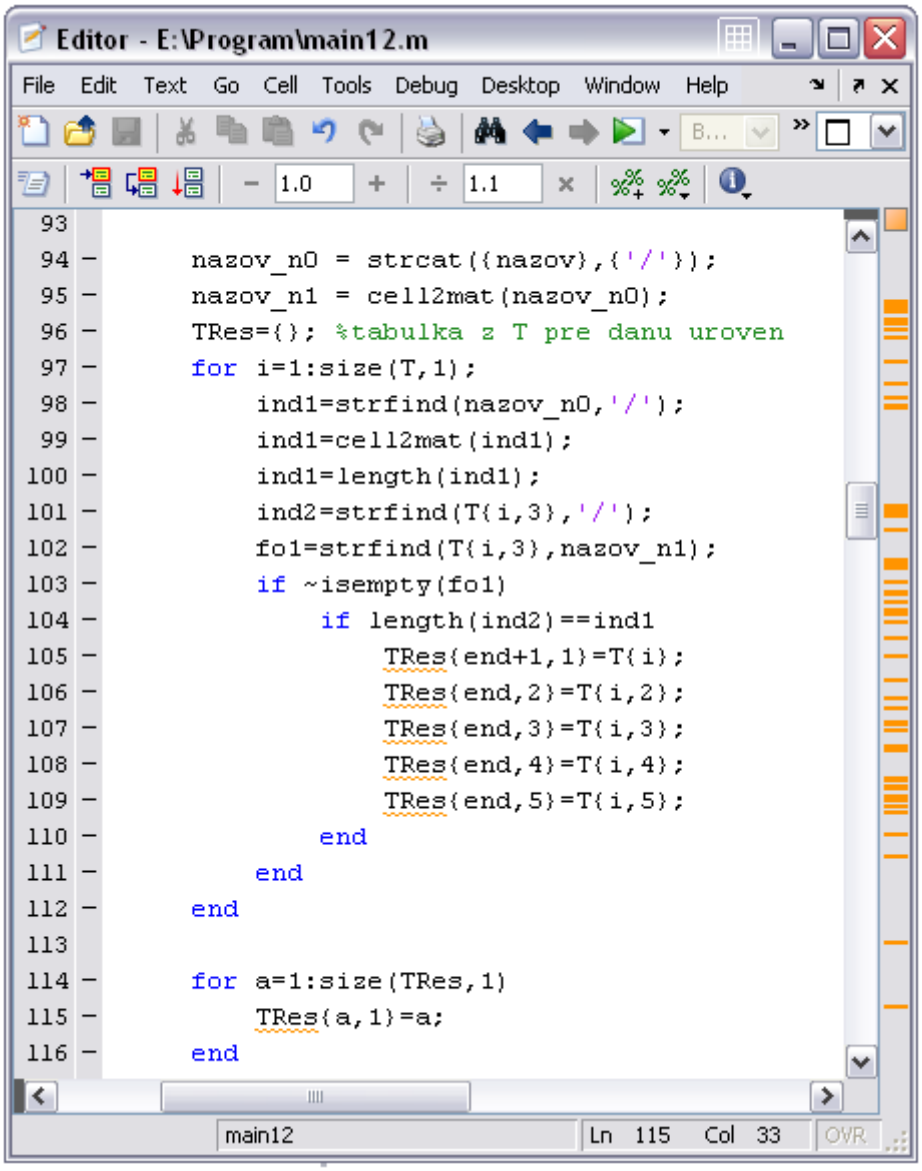
Ak by sme chceli odfiltrovať tabuľku *T* pre jeden zo subsystémov v schéme na obr. 2.1 zadáme názov v tvare:

```
>> nazov='schema/Subsystem'
```

a po filtrovaní získame tabuľku *TRes* pre danú úroveň čiže *Subsystem* obr. 2.2. Bez blokov maskovaných vo vnútri tohto subsystému.

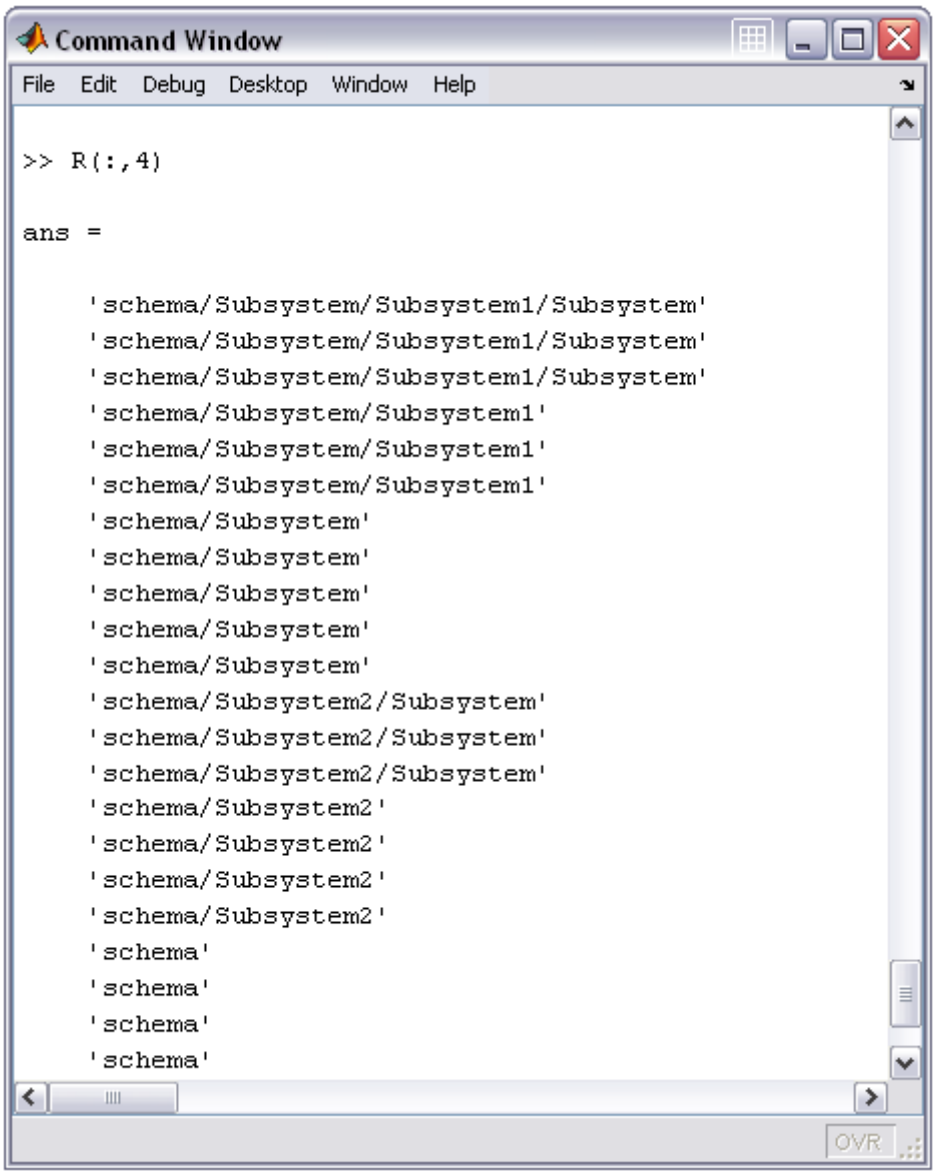
```
TRes =

    [1]    [163.0010]    [1x20 char]    'outport'    'Inport'
    [2]    [164.0010]    [1x23 char]    'inport'     'Scope'
    [3]    [165.0010]    [1x27 char]    'inport'     'SubSystem'
    [4]    [166.0010]    [1x27 char]    'outport'    'SubSystem'
    [5]    [179.0010]    [1x21 char]    'inport'     'Sum'
    [6]    [180.0010]    [1x21 char]    'inport'     'Sum'
    [7]    [181.0010]    [1x21 char]    'outport'    'Sum'
```

```
>> TRes(:,3)

ans =

    'schema/Subsystem/In1'
    'schema/Subsystem/Scope1'
    'schema/Subsystem/Subsystem1'
    'schema/Subsystem/Subsystem1'
    'schema/Subsystem/Sum1'
    'schema/Subsystem/Sum1'
    'schema/Subsystem/Sum1'
```



Obr. 2.10: Filtrovanie úrovne v tabuľke *T*

Podobným spôsobom odfiltrujeme tabuľku *R* avšak kritérium bude iné. Ako vidno z obr. 2.11 v tabuľke R sa na prvej úrovni nachádzajú iba bloky ktorých názov je 'schema'.

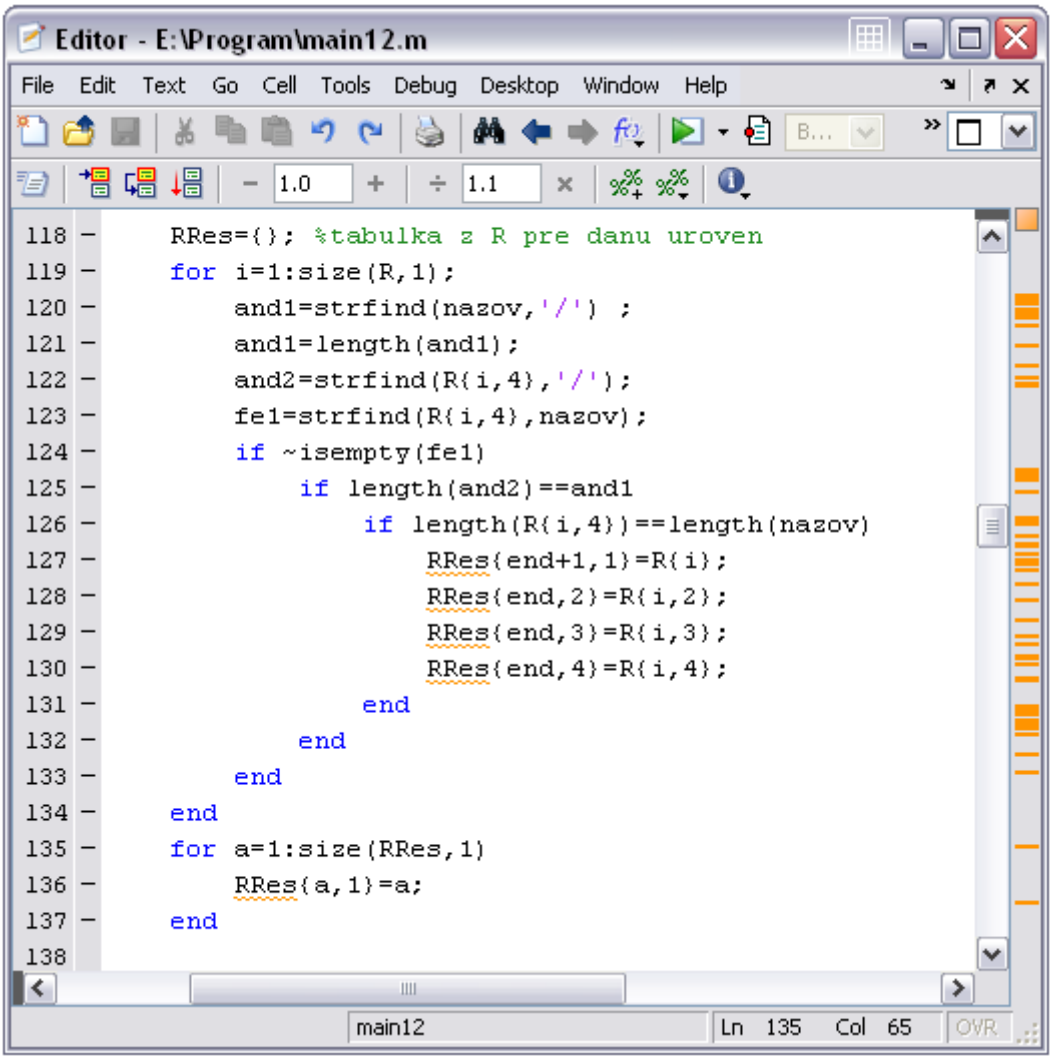


Obr. 2.11: Názvy blokov v tabuľke *R*

Podobne ako pre predchádzajúcu tabuľku *T* nájdeme všetky položky z tabuľky *R* v ktorých sa nachádza názov v tvare zadanom z príkazového riadka (v našom prípade *nazov*). Ďalej nájdeme všetky lomítka v reťazci *nazov* a porovnáme ich s počtom lomítok v tabuľke *R* (štvrtý stĺpec tabuľky *R*). V prípade zhody nasleduje podmienka aby dĺžka názvu v tabuľke bola práve rovná dĺžke reťazca *nazov*. *RRes* nám vráti iba tabuľku danej úrovne.

Pre *Subsystem* (obr. 2.2) bude tabuľka *RRes* vyzerať nasledovne:

```
RRes =

    [1]    [163.0010]    [  179.0010]    'schema/Subsystem'
    [2]    [166.0010]    [  180.0010]    'schema/Subsystem'
    [3]    [166.0010]    [  164.0010]    'schema/Subsystem'
    [4]    [166.0010]    [2x1 double]    'schema/Subsystem'
    [5]    [181.0010]    [  165.0010]    'schema/Subsystem'
```



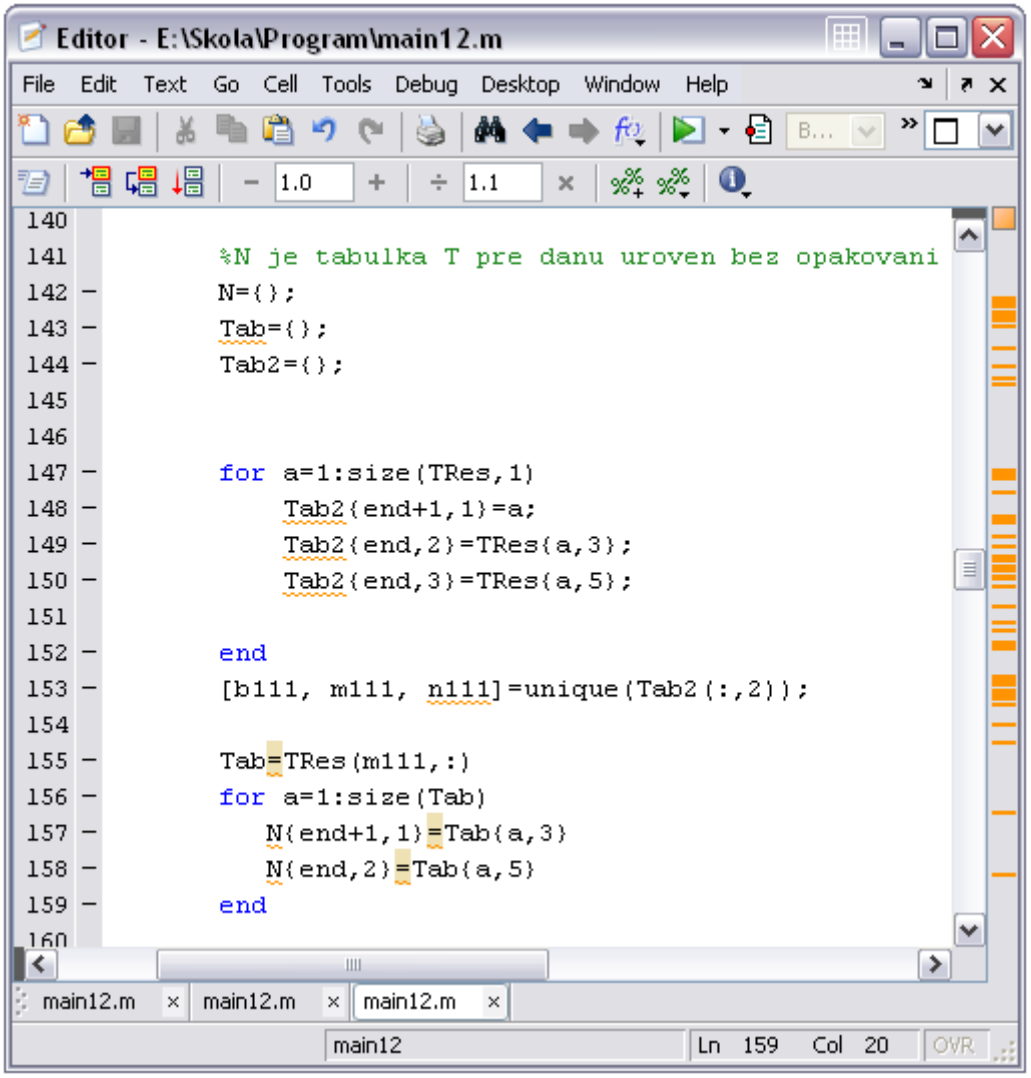
Obr. 2.12: Filtrovanie úrovne v tabuľke *R*

Získali sme dve tabuľky ktoré nám vracajú iba bloky na danej úrovni, či už to je hlavný systém alebo akýkoľvek podsystém avšak s opakovaniami sa názvov.

Pre každý podsystém potrebujeme vybrať iba názvy blokov z tabuľky *T* bez opakovaní. Pre toto filtrovanie nám môže poslúžiť funkcia *unique* v tvare:

```
[b111, m111, n111]=unique(Tab2(:,2))
```

Funkcia *unique* vracia rovnaké hodnoty z druhého stĺpca tabuľky *Tab2* bez opakovaní. Do premennej *b111* vkladá názvy jednotlivých položiek, do *m111* polohu unikátnej jednotky (ak ich je viac rovnakých tak polohu poslednej) a v premennej *n111* je pôvodný nefiltrovaný stĺpec (obr. 2.13). Do tabuľky *Tab* vložíme všetky stĺpce z tabuľky *TRes* ktorých čísla riadkov sú *m111*. Cyklom vytvoríme tabuľku *N* bez opakovaní sa názvov.



Obr. 2.13: Filtrovanie unikátnych názvov z tabuľky *T*

Aby sme oddelili obyčajné masky podsystémov (*subystem*) od masiek s nastaviteľnými parametrami (napr. *PID Controller*), zistili sme si parametre subsystému ako kritérium pre filtrovanie.

Na obr. 2.14 sú parametre subsystému a v modrom rámiku je kritérium pre odfiltrovanie subsystémov.

```
cc3 =                                                    '

    'ShowPortLabels'
    'BlockChoice'
    'TemplateBlock'
    'MemberBlocks'
    'Permissions'
    'ErrorFcn'
    'PermitHierarchicalResolution'
    'TreatAsAtomicUnit'
    'MinAlgLoopOccurrences'
    'PropExecContextOutsideSubsystem'
    'CheckFcnCallInpInsideContextMsg'
    'SystemSampleTime'
```

2.14: DialogParameters subystému

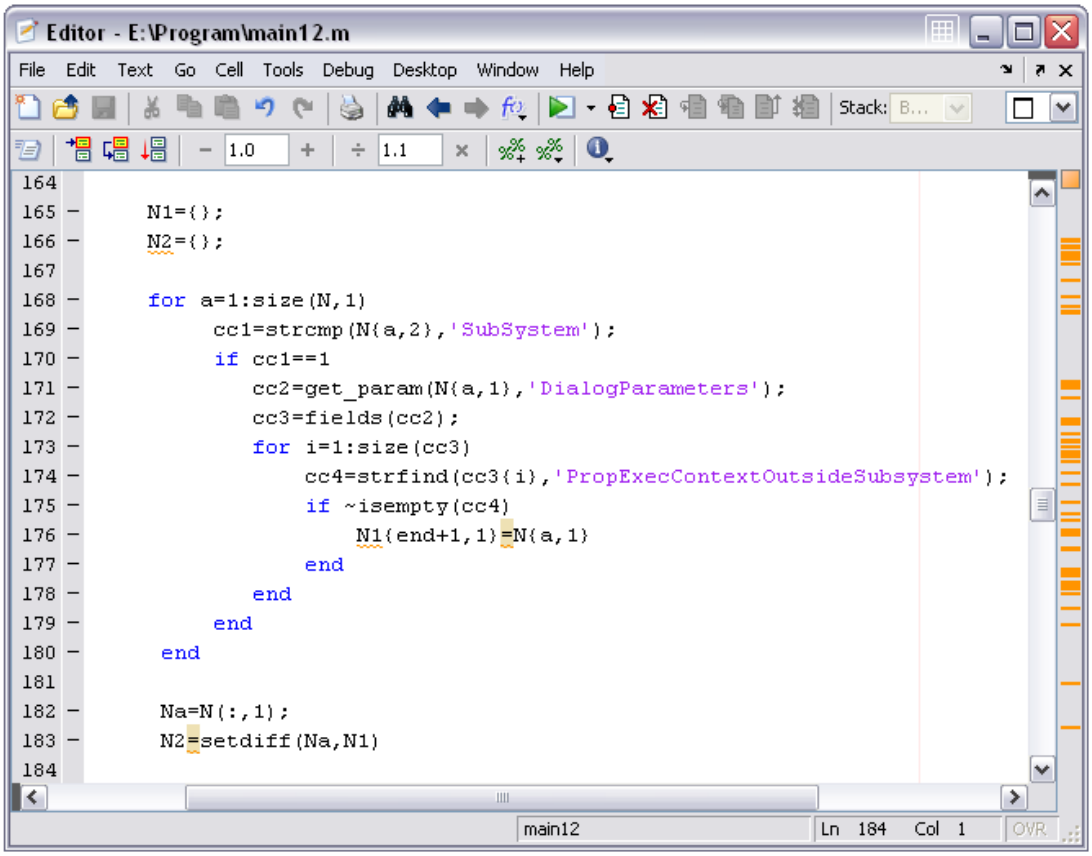
Vonkajší cyklus (obr. 2.15) zbehne po celej tabuľke blokov v danej úrovni a nájde bloky ktorých typ bloku je *Subsystem*. Tu však môžu patriť aj bloky s nastaviteľnými parametrami ako je napr. *PID Controller*. Pomocou príkazu *get_param* nájdeme všetky parametre daného bloku a príkaz *fields* nám vráti názvy jednotlivých parametrov v *cell* formáte. Napríklad keby sme chceli zistiť parametre bloku *PID Controller* v schéme obr. 2.1 by príkazy vyzerali nasledovne:

```
>> cc2=get_param('schema/PID Controller','DialogParameters')

cc2 =

    P: [1x1 struct]
    I: [1x1 struct]
    D: [1x1 struct]
```

```
>> cc3=fields(cc2)

cc3 =

    'P'
    'I'
    'D'
```

Vnútorný cyklus v obr. 2.15 nájde iba tie subsystémy ktoré spĺňajú vyššie uvedené kritérium z obr. 2.14.

Vytvorili sme *N1* tabuľku iba podsystémov a doplnkom pomocou funkcie *setdiff* tabuľku ostatných blokov *N2* vrátane bloku *PID Controller*. Tento algoritmus rozlíši akýkoľvek vlastný subsystém ktorý používa vkladané parametre alebo parametre volané z MATLABu.

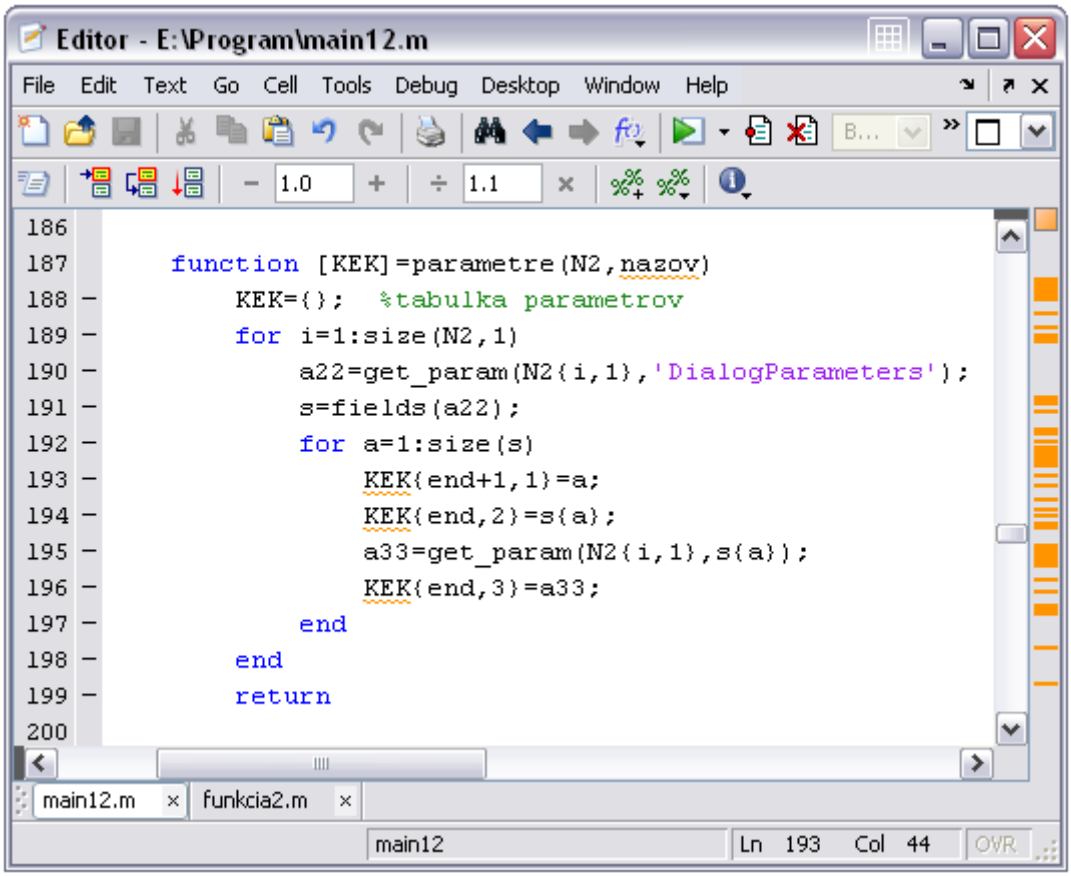


2.15: Oddelenie masiek subsystémov od ostatných blokov

2. 6 Parametre blokov

Parametre blokov v danej úrovni zistíme podobne ako v kapitole 2.5 obr. 2.15.

Na obr. 2. 16 je funkcia ktorá zisťuje parametre blokov. Tieto parametre sú už parametrami nastaviteľných blokov. Funkcia *parametre* priradí parametre blokov do tabuľky *KEK*. Vo vzniknutej tabuľke *KEK* sú už iba parametre nastaviteľných blokov (napr. *Step* alebo *PID Controller*).



2.16: Vytvorenie tabuľky *KEK* parametrov blokov

2. 7 Vytvorenie štruktúry a jej naplnenie

Aby sme mohli napĺňať struktúru premennými alebo tabuľkami je potrebné si štruktúru najprv vytvoriť. Štruktúry vytvárame pomocou príkazu *struct*.

Štruktúry obsahujú súbor údajov uložených do pomenovaných položiek. Štruktúry poskytujú mechanizmus pre hierarchické ukladanie do rôznorodých údajov. Odlišujú sa hlavne spôsobom ukladania dát. Údaje v štruktúre sprístupňujeme pomocou pomenovaných polí. Tab. 2.1 obsahuje základné funkcie MATLABu pre prácu so štruktúrami**[3]**.

Tab. 2.1 Prehľad funkcii na prácu so štruktúrami

Funkcia	**Popis**
fieldnames	zistí názvy položiek štruktúry
getfield	Zistí obsah položiek štruktúry
isfield	vráti hodnotu pravda ak sa položka nachýdza v poli štruktúr
isstruct	vráti pravda, ak je vstupný parameter štruktúra
rmfield	odstráni položku zo štruktúry
setfield	nastaví obsah položky štruktúry
struct	vytvorí alebo skonvertuje vstup na štruktúru
struct2cell	skonvertuje pole štruktúr na bunkové pole

Štruktúry sú v MATLABe polia s pomenovanými prvkami nazývanými položky. Položky v štruktúre môžu obsahovať ľubovoľný druh údajov. Jedna položka môže napríklad obsahovať textový reťazec reprezentujúci názov, druhá položka maticu prepojení a napríklad číslo.

Tak ako štandardné polia, sú aj štruktúry orientované na polia. Jeden záznam štruktúry tvorí pole 1x1 podobne, ako je hodnota 5 číselné pole 1x1.

Štruktúru môžeme vytvoriť dvojakým spôsobom**[3]**:

- použitím príkazu priradenia
- pomocou funkcie struct

Štruktúru pomocou priradenia vytvoríme nasledovne:

```
>> Tabulka_celkom1.cesta='nazov';
>> Tabulka_celkom1.T=T;
>> Tabulka_celkom1.R=R;
>> Tabulka_celkom1

Tabulka_celkom1 =

    cesta: 'nazov'
        T: {17x5 cell}
        R: {10x4 cell}
```

Premenná *Tabulka_celkom1* je pole obsahujúce štruktúru s tromi položkami. Na rozšírenie poľa štruktúr pridáme za meno štruktúry index:

```
>> Tabulka_celkom1(2).cesta='cesta';
>> Tabulka_celkom1

Tabulka_celkom1 =

1x2 struct array with fields:
    cesta
    T
    R
```

Štruktúra *Tabulka_celkom1* má teraz rozmer [1 2]. Akonáhle má pole štruktúr viac ako jeden záznam, MATLAB už nevypisuje jednotlivé položky po zadaní mena poľa. Namiesto toho vypíše súhrnnú informáciu o obsahu jeho štruktúry.

Na získanie informácie o názve jednotlivých polí môžeme použiť funkciu *fieldnames***[2]**:

```
>> fieldnames(Tabulka_celkom1)

ans =

    'cesta'
    'T'
    'R'
```

Veľkosti položiek nemusia byť zhodné pre všetky záznamy v poli. Pole štruktúr môžeme predalokovať funkciou *struct*. Jej základný tvar je:

```
str_array=struct('polozka1', hodnota1, 'polozka2', hodnota2,...)
```

ktorá má vždy dvojice parametrov názov položky a jej hodnotu. Hodnota položky môže byť jednoduchá hodnota v podobe ľubovoľnej dátovej konštrukcie MATLABu alebo bunkové pole hodnôt. Všetky hodnoty položiek v zozname parametrov musia byť rovnakého typu (jednoduchá hodnota alebo bunkové pole)**[3]**.

Aby sme vytvorili štruktúru pomocou funkcie *struct* musíme mať pevný rozmer jednotlivých tabuliek vkladaných do štruktúry. Stĺpce tabuliek budú názvy jednotlivých položiek štruktúry. Pre tabuľku *T* (obr.2.17) sú položky štruktúry modrou farbou.

T =

por_cislo	cislo_portu	nazov	typ	typ_bloku
[1]	[168.0010]	[1x20 char]	'outport'	'Inport'
[2]	[169.0010]	[1x23 char]	'inport'	'Scope'
[3]	[170.0010]	[1x27 char]	'inport'	'SubSystem'
[4]	[171.0010]	[1x27 char]	'outport'	'SubSystem'
[5]	[188.0010]	[1x21 char]	'inport'	'Sum'
[6]	[189.0010]	[1x21 char]	'inport'	'Sum'
[7]	[190.0010]	[1x21 char]	'outport'	'Sum'

Obr. 2.17: Pomenovanie položiek štruktúry

V celkovej štruktúre vytvoríme pole s názvom T a jemu priradíme štruktúru:

```
Tabulka_celkom=struct;
```

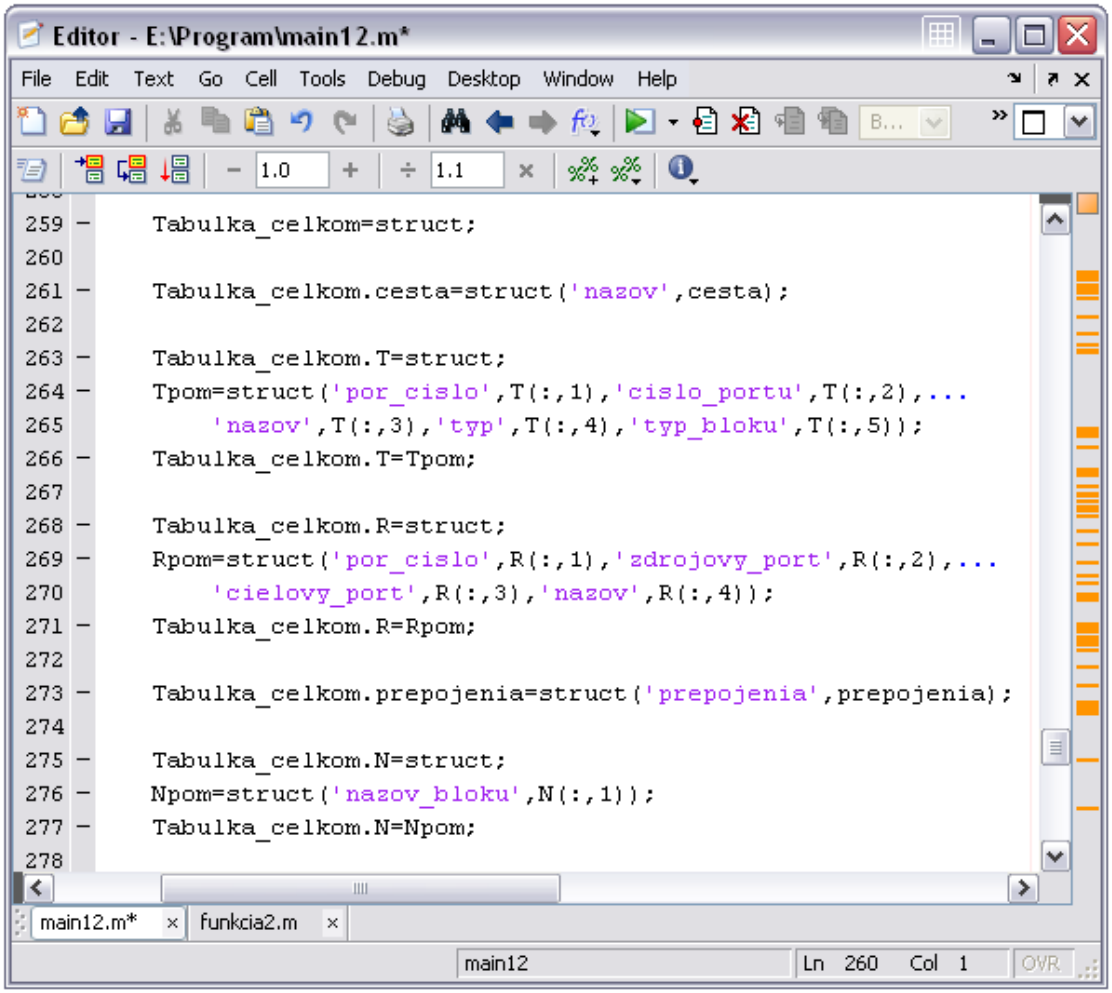
Vytvoríme si štruktúru podľa všeobecného tvaru uvedeného v predchádzajúcom texte kde vložíme názov položky a priradíme jej hodnotu. Pre tabuľku *T* budú hodnotou jej jednotlivé stĺpce:

```
Tpom=struct('por_cislo',T(:,1),'cislo_portu',T(:,2),...
            'nazov',T(:,3),'typ',T(:,4),'typ_bloku',T(:,5));
```

Následne túto štruktúru vložíme do poľa s názvom *T*:

```
Tabulka_celkom.T=Tpom;
```

Podobným spôsobom vkladáme ostatné položky do celkovej štruktúry. Vytvorenie konečnej štruktúry je na obr. 2. 18.



Obr. 2.18: Vytvorenie celkovej štruktúry

Aby sme naplnili štruktúru údajmi a ďalšími vnorenými štruktúrami použijeme príkazy *eval* a *setfield*.

Funkciou *eval* umožňuje MATLAB vykonávanie výrazu pomocou reťazca. Ak by sme chceli pomocou evalu vyriešiť jednoduchý výraz pomocou definovaných premenných vyzeral by nasledovne**[8]**:

```
>> a = 2; b = 1;
>> c = '1/(a+b+7)';
>> eval(c)
ans =
  0.1000
```

Na obr. 2.19 je priradenie tabuliek parametrov do štruktúry. V prvom kroku cyklu nájdeme všetky lomítka v názve bloku a nahradíme ich na podtržníkom (“_”) a v druhom kroku nájdeme všetky prázdne znaky a vymažeme ich. Pomocou funkcie *setfield* priradíme priamo do štruktúry jednotlivé názvy blokov z tabuľky *N2*.
Funkcia *setfield* výzerá vo všeobecnom tvare nasledovne:

```
s = setfield(s, 'field', v)
```

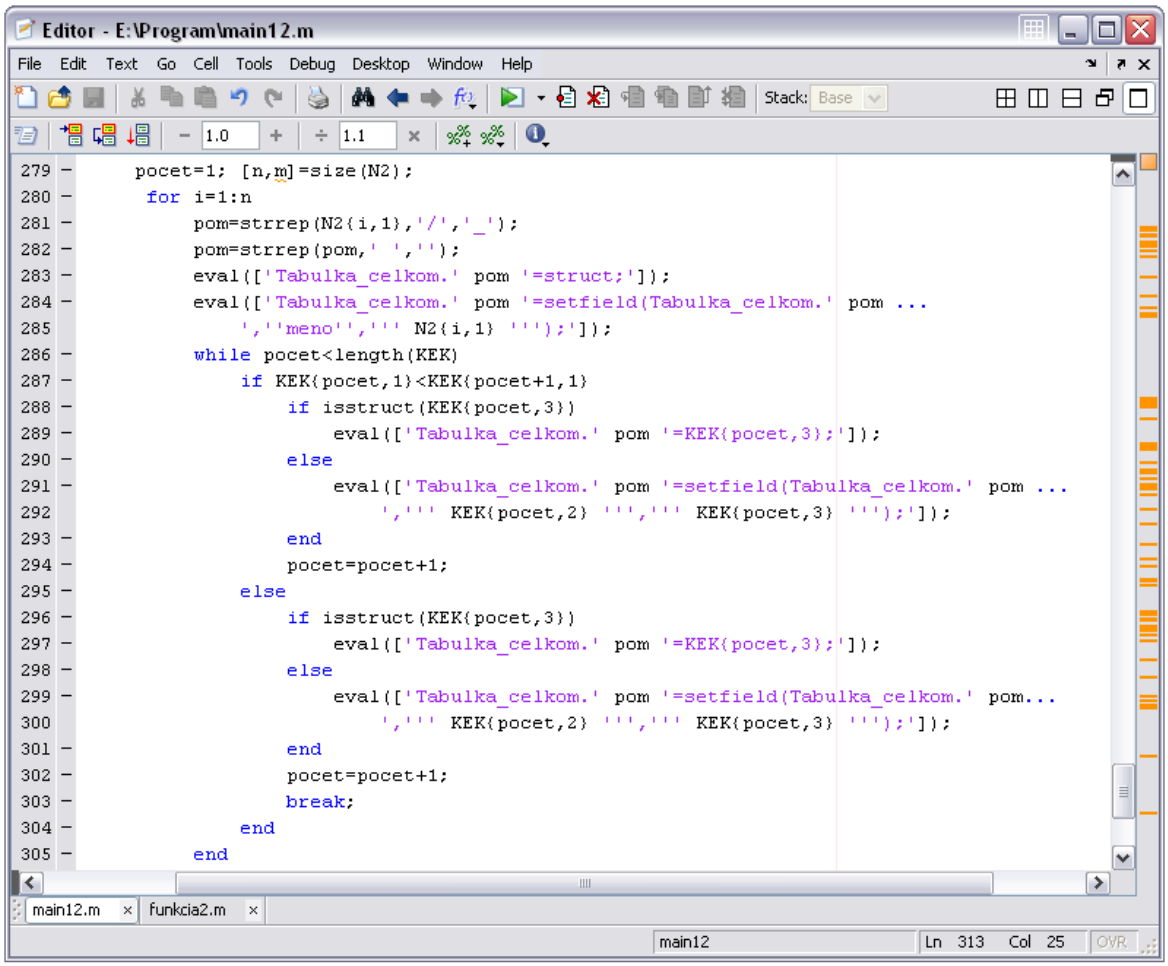
kde *s* je struktura 1x1, *field* je názov poľa a *v* je hodnota ktorú k názvu priradí**[9]**.

Funkcia *eval* obr. 2.19 vykoná celý výraz:

```
eval(['Tabulka_celkom.' pom ...
      '=setfield(Tabulka_celkom.' pom ...
      ','''meno''','''' N2{i,1}''');']);
```

Do štruktúru *Tabulka_celkom.* priradí názvy blokov z tabuľky *N2* vo formáte *string* a v tvare *‘string’*. Vo výraze sú 3 apostrofy: prvý vytvorí z názvu formát *string*, druhý vytvorí nad ním apostrof a tretí celý výraz uzavrie.

Podobným spôsobnom do tej istej štruktúry vkladáme aj parametre jednotlivých blokov z tabuľky *KEK*. Podmienka *while* vo vnútri cyklu slúži na oddelenie parametrov jedného bloku z celej tabuľky.



Obr. 2.19: Priradenie tabuliek parametrov do štruktúry

2. 8 Rekurzívne volanie funkcie

Program sa skladá z niekoľkých funkcii vzájomne na seba nadväzujúcich. Hlavná funkcia je funkcia *main12*. V nej je *funkcia1*:

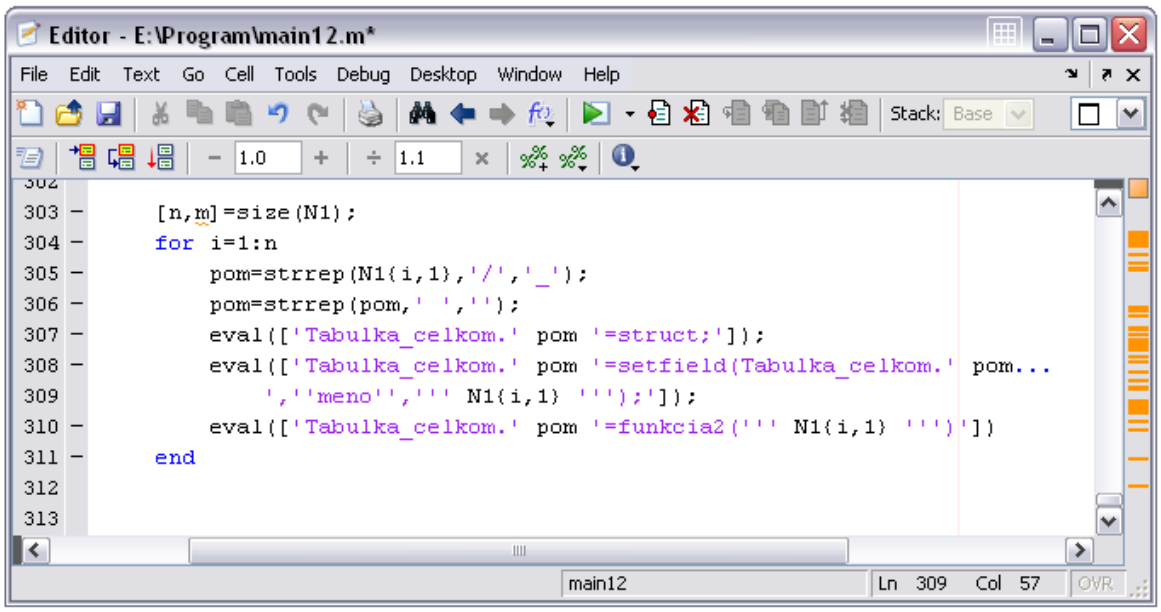
```
[N, N1, N2, TRes, RRes, KEK] = funkcia1(T, R, nazov);
```

Funkcia s názvom *funkcia1* a parametrami *T*, *R* a *nazov* je volaná z hlavnej funkcie *main12* a vracia tabuľky blokov *N*, *N1* a *N2*, *TRes* a *RRes* tabuľky a tabuľku parametrov blokov *KEK*. Posledná dôležitá funkcia je *funkcia2:*

```
[Tabulka_celkom] = funkcia2(cesta);
```

ktorá volá názov v parametri *cesta* a vracia celú štruktúru.

Spustíme funkciu *main12* ktorá postupne volá obe funkcie. Prvú zavolá funkciu *funkcia1* a nasledovne funkciu *funkcia2*. Druhýkrát volá algoritmus znovu funkciu *funkcia2* pričom názov volaný do funkcie už nie je názvov zadaný do príkazového riadka ale reťazec so štruktúry s názvom *cesta*. Toto opakovanie sa ide po celej dĺžky tabuľky subsystémov *N1* (tabuľka subsystémov). Toto umožní volať funkciu až po poslednú úroveň maskovania a priradiť jej rovnakú štruktúru avšak s inými parametrami (obr.2.20).



Obr. 2.20: Napĺňanie masiek subsystémov štruktúrami

3. Použitie a práca s programom

V kapitole 2 sme ukázali, ako bol program vytvorený teraz si ukážeme jednoduchý príklad použitia. Príklad použitia je aplikovaný na schému vytvorenú v kapitole 2 obr. 2.1 aj s jej subsystémami.

Do príkazového riadka v MATLABe zadáme názov schémy v tvare:

```
>> nazov='schema'
```

a zavoláme funkciu *main12*:

```
>> [Tabulka_celkom]=main12(nazov)
```

Funkcia nám vráti celú štruktúru schémy *schema* až do posledného stupňa maskovania aj so všetkými blokmi, tabuľkami a maticou prepojení portov:

```
Tabulka_celkom =

                  cesta: [1x1 struct]
                      T: [17x1 struct]
                      R: [10x1 struct]
             prepojenia: [1x1 struct]
                      N: [9x1 struct]
           schema_Demux: [1x1 struct]
  schema_PIDController: [1x1 struct]
         schema_Scope2: [1x1 struct]
           schema_Step: [1x1 struct]
            schema_Sum: [1x1 struct]
   schema_TransferFcn3: [1x1 struct]
      schema_Transport: [1x1 struct]
      schema_Subsystem: [1x1 struct]
     schema_Subsystem2: [1x1 struct]
```

Do tejto hlavnej štruktúry sú vložené štruktúry v rovnakom tvare pre všetky podsystémy nachádzajúce sa v schéme napr.:

```
>> Tabulka_celkom.schema_Subsystem2

ans =

                              cesta: [1x1 struct]
                                 T: [9x1 struct]
                                 R: [6x1 struct]
                         prepojenia: [1x1 struct]
                                 N: [5x1 struct]
            schema_Subsystem2_In1: [1x1 struct]
    schema_Subsystem2_PIDController1: [1x1 struct]
          schema_Subsystem2_Scope3: [1x1 struct]
            schema_Subsystem2_Sum2: [1x1 struct]
        schema_Subsystem2_Subsystem: [1x1 struct]
```

Prepojenia portov sú v maticovom tvare tak ako sme ich vytvorili v kapitole 2 obr. 2.7.

```
>> Tabulka_celkom.schema_Subsystem2.prepojenia.prepojenia

ans =

     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0    -1
     0     0     0     0     1     0     0     0     0
     0     0     0     0     0    -1     0     0     0
     0     0    -1     0     0     0     0     0     0
     0     0     0     1     0     0     0     1     0
    -1     0     0     0     0     0     0     0     0
     0     0     0     0     0    -1     0     0     0
     0     1     0     0     0     0     0     0     0
```

Parametre všetkých tabuliek sú vložené formou štruktúry s konečným počtom stĺpcov:

```
>> Tabulka_celkom.schema_Subsystem2.T

ans =

9x1 struct array with fields:
    por_cislo
    cislo_portu
    nazov
    typ
    typ_bloku

>> Tabulka_celkom.schema_Subsystem2.T(1,:)

ans =

      por_cislo: 1
    cislo_portu: 192.0010
          nazov: 'schema/Subsystem2/In1'
            typ: 'outport'
      typ_bloku: 'Inport'
```

Napr. každá tabulka *T* v celej štruktúre bude mať rovnaký formát. To isté platí aj pre tabulky *R* a *N2*.

Pre každý blok ktorý má nastaviteľné parametre tieto parametre vieme zistiť:

```
>> Tabulka_celkom.schema_Subsystem2.schema_Subsystem2_PIDController1

ans =

    meno: 'schema/Subsystem2/PID Controller1'
       P: '0.9750'
       I: '0.9750/8'
       D: '0.9750*2'
```

Parametre môžeme vyberať aj jednotlivo a meniť ich bez toho aby sme otvorili SIMULINKovú schému:

```
>> Tabulka_celkom.schema_Subsystem2.schema_Subsystem2_PIDController1.P

ans =

    0.9750

>> Tabulka_celkom.schema_Subsystem2.schema_Subsystem2_PIDController1.P=0.5;
>> Tabulka_celkom.schema_Subsystem2.schema_Subsystem2_PIDController1.P

ans =

    0.5000
```

V prípade že nás nezaujíma štruktúra od najvrchnejšej schémy a vieme cestu napr. ku najspodnejšej vrstve môžeme si zavolať funkciu *main12* iba pre danú vrstvu:

```
>> nazov='schema/Subsystem2/Subsystem'

nazov =

schema/Subsystem2/Subsystem

>> [Tabulka_celkom]=main12(nazov)

Tabulka_celkom =

                                        cesta: [1x1 struct]
                                            T: [6x1 struct]
                                            R: [3x1 struct]
                                    prepojenia: [1x1 struct]
                                            N: [4x1 struct]
                  schema_Subsystem2_Subsystem_In1: [1x1 struct]
                 schema_Subsystem2_Subsystem_Out1: [1x1 struct]
        schema_Subsystem2_Subsystem_TransferFcn1: [1x1 struct]
          schema_Subsystem2_Subsystem_Transport1: [1x1 struct]
```

Týmto spôsobom vieme pracovať s programom vytvoreným v jednej funkcii a síce vo funkcii *main12*. Túto funkciu vieme volať z iných funkcií prípadne iných aplikácii.

Záver

Cieľom tejto diplomovej práce bolo vytvoriť program na spracovanie SIMULINKových schém vrátane schém obsahujúcich masky subsystémov. Spracovaním takéhoto zložitého systému sme získali kompletnú štruktúru tohto systému, parametre jeho blokov a prepojenia portov v systéme. Program bol urobený všeobecne aby bol schopný prečítať akúkoľvek SIMULINKovú schému zostavenú zo štandardných knižníc blokov pre riadenie. Vytvorená konečná štruktúra je vetvená až po najnižšiu masku subsystému v prípade že sú subsystémy viacnásobne maskované.

Program je navrhnutý tak aby sa dal použiť do iných aplikácií a z nich bolo možné jednotlivé parametre v štruktúre čítať a prepisovať. Toto môže byť užitočné pri spracovaní väčšieho množstva schém kde treba nastavovať parametre resp. ich získať.

Diplomová práca je podrobným návodom ku vytvoreniu aplikácie, ktorý uľahčí orientáciu, modifikáciu a prácu s programom.

Literatúra

[1] URL: http://www.humusoft.cz/matlab/matlab.htm

[2] Getting Started with MATLAB® 7, The MathWorks, Inc., 2007

[3] Bartko R., Miller M., MATLAB 1. algoritmizácia a riešenie úloh, Digital Graphic, 2004

[4] URL: http://www.humusoft.cz/matlab/simulink.htm

[5] Dušek F., Matlab a Simulink – úvod do používání, Univerzita Pardubice, 2000

[6] Foltin M., AT&P journal 4/2008

[7] URL: http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/

[8] URL: http://www.kirp.chtf.stuba.sk/~cirka/vyuka/matlab/kap4.php

[9] URL: http://www.mathworks.com/access/helpdesk/help/techdoc/

Prílohy

Vytvorený program v súbore m-file a modelová schéma sa nachádzajú na priloženom CD.