UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Simulátor dynamických soustav na AMiNi4DS

Tomáš Vlk

Bakalářská práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Tomáš Vlk**
Osobní číslo:	**I10483**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Řízení procesů**
Název tématu:	**Simulátor dynamických systémů na AMiNI4DS**
Zadávající katedra:	**Katedra řízení procesů**

Zásady pro vypracování:

Vytvořit sadu programů pro malý řídicí systém AMiNi4DS pro simulaci různých dynamických systémů do rozměrů 4 vstupy a 4 výstupy. Vstupy a výstupy matematického modelu budou připojeny na napěťové vstupy a výstupy řídicího systému. Matematický model bude realizován jako diskrétní stavový s minimální dosažitelnou periodou vzorkování.
Teoretická část:
a) nastudovat popis lineárních spojitých vícerozměrových dynamických systémů ve formě matice přenosů
b) nastudovat zápis přenosové matice v MATLABu a její převod na diskrétní stavový model
c) seznámit se s řídicím systémem AMiNi4DS a vývojovým prostředím DetStudio pro tvorbu programového vybavení
Praktická část:
a) v prostředí MATLABu vytvořit skript, který pro zadanou přenosovou matici a interval vzorkování vytvoří matice parametrů odpovídajícího diskrétního stavového modelu
b) vytvořit a odladit sadu programů (ST strukturovaný text) využívajících standardní bloky, které realizují různé diskrétní stavové modely
c) experimentálně určit nejkratší možnou periodu vzorkování procesu realizujícího výpočet stavového modelu
d) všechny programy budou umožňovat zobrazení aktuálních napětí vstupů a výstupů na vestavěném LCD zobrazovači

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

DUŠEK, František. Matlab a Simulink: úvod do používání. Vyd. 1. Pardubice: Univerzita Pardubice, 2005, 172 s. ISBN 80-719-4776-8.
BALÁTĚ, Jaroslav. Automatické řízení. 2., přeprac. vyd. Praha: BEN, 2004, 663 s. ISBN 80-730-0148-9.
on line dokumentace k programu MATLAB
dokumentace k malému řídicímu systému AMiNi4DS dostupná na AMiT - řídicí systémy a elektronika pro průmyslovou automatizaci [online]. 2012 [cit. 2012-10-29]. Dostupné z: http://www.amit.cz/inet_dir/cz/currently.htm
a) návod na obsluhu http://www.amit.cz/docs/cz/amini/amini4ds_g_cz_100.pdf
b) popis programu http://www.amit.cz/docs/cz/sw/detstudio_g_cz_101.pdf
c) download DetStudio
http://www.amit.cz/inet_dir/cz/sw/paramsw.htmdetstudio

Vedoucí bakalářské práce: **doc. Ing. František Dušek, CSc.**
Katedra řízení procesů

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

Univerzita Pardubice -8- Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Daniel Honc, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Břehách dne 19. 5. 2013 Tomáš Vlk

Poděkování

Tímto bych rád poděkoval vedoucímu mé bakalářské práce doc. Ing. Františku Duškovi, CSc., za maximální vstřícnost a ochotu při řešení jakéhokoliv problému či nesrovnalosti, který nastal při řešení bakalářské práce. Dále bych rád poděkoval Ing. Liboru Havlíčkovi, Ph.D. za vstřícnost během celého bakalářského studia a hlavně bych rád poděkoval své rodině za podporu během celého mého studia.

Anotace

Tato práce se zabývá vytvořením sady programů pro řídicí systém AMiNi4DS pro simulaci různých dynamických SISO systémů. Vstupy a výstupy matematického modelu systému budou připojeny na napěťové vstupy a výstupy řídicího systému. Matematický model bude realizován jako diskrétní stavový s minimální dosažitelnou periodou vzorkování. Programy pro AMiNi4DS budou umožňovat zobrazení aktuálních napětí vstupů a výstupů na vestavěném LCD zobrazovači. Sada programů bude vytvořena v prostředí programu DetStudio. Zadávání konkrétních systémů s volbou řádu bude realizováno pomocí GUI v programu MATLAB. Dynamické systémy se v MATLABu budou vkládat formou spojitého popisu systému.

Klíčová slova

AMiNi4DS, MATLAB, Guide, GUI, SISO, DetStudio, simulace, dynamický systém

Title

Simulator of dynamical systems on the AMiNi4DS

Annotation

The aim of this work is to develop a set of programs for the control system AMiNi4DS to simulate different dynamic SISO systems. Inputs and outputs of a mathematical model will be connected to the voltage inputs and outputs of the control system. The mathematical model is implemented as a discrete state with the minimum achievable sampling period. Programs for AMiNi4DS will allow the display of actual voltage inputs and outputs on the LCD display. A set of programs will be created in program DetStudio. Entering specific systems with a choice of order will be realized using the GUI in MATLAB. Dynamic systems in MATLAB will be insterted in the form of a continuous description.

.

Keywords

AMiNi4DS, MATLAB, GUIDE, GUI, SISO, DetStudio, simulation, dynamical system

Obsah

Seznam zkratek

SISO	Single Input Single Output
GUI	Graphical User Interface
XML	Extensible Markup Language

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době se takřka v každém odvětví vývoje či testování nahrazují reálné objekty takzvanými hardwarovými simulátory. Simulátor napodobuje dynamické chování reálného objektu na úrovni signálů. Nahrazení reálného objektu se realizuje z důvodu lepší manipulovatelnosti s objektem a například u simulace nárazu automobilu se šetří i finanční prostředky. Pro potřeby školní výuky se zavádějí takzvané laboratorní modely, které simulují chování reálných soustav. Studenti si mohou vyzkoušet chování reálných systémů v laboratorních podmínkách.

V této práci bude vytvořen hardwarový simulátor pro potřebu výuky, a pro jeho vytvoření se použije malý kompaktní řídicí systém AMiNi4DS, který obsahuje vestavěný černobílý LCD zobrazovač a je vyráběn českou firmou Amit, spol s.r.o.. Prozatím postačí vědět, že obsahuje 8 tlačítek pro ovládání zobrazovače, 8 analogových galvanicky oddělených vstupů, 4 analogové výstupy a 8 číslicových galvanicky oddělených vstupů i výstupu.

Tento řídicí systém je použit jako hardwarový simulátor dynamických SISO systémů s online výběrem ze sady předpřipravených modelů. Možnost vytvořit nový model bude k dispozici v simulačním programu MATLAB s GUI rozhraním. MATLAB je interaktivní prostředí pro numerické výpočty, vizualizaci a programování.

Cílem bakalářské práce Simulátor dynamických soustav na AMiNi4DS je vytvoření programové sady, pro řídicí systém AMiNi4DS, která realizuje hardwarový simulátor 4 paralelně běžících dynamických SISO soustav s možností on-line výběru z 10 předpřipravených modelů.

Dalším cílem je vytvoření programu v MATLABu, umožňující volbu parametrů soustav a zápis parametrů do programové sady pro AMiNi4DS.

V první části této práce jsou ukázány možné matematické popisy dynamických soustav, popis návrhového prostředí DetStudio a popis programu MATLAB včetně tvorby GUI.

V praktické části pro AMiNi4DS je popsáno vytvoření programové sady pro řídicí systém a popis její funkce. Dále je ukázáno odhadnutí minimálního intervalu vzorkování pro simulaci chování 4 oddělených systémů současně.

V Praktické části pro MATLAB je popsáno grafické uživatelské rozhraní a je vysvětlena jeho funkce a všechny možnosti, které nabízí.

V příloze jsou grafy zobrazující chování vybraného matematického modelu podle softwaru SIMULINK a chování stejného modelu vytvořeného v AMiNi4DS. Oba modely reagují na stejný vstupní signál.

1 Definice a popis systému

Systémem rozumíme obecně soubor prvků, mezi nimiž existují vzájemné vazby a jako celek má určité vztahy ke svému okolí. Každý systém je charakterizován dvěma základními vlastnostmi:

- **chováním systému**, charakterizujícím jeho vnější vztahy k okolí
- a **strukturou systému**, charakterizující jeho vnitřní funkční vztahy.

Obě tyto vlastnosti systému jsou ve velmi úzkém vztahu, který lze charakterizovat jednak, že určité struktuře odpovídá jednoznačně určité chování a naopak, že určitému chování odpovídá třída struktur, definovaná tímto chováním.

Podle toho, zda nastává interakce s okolím, dělíme systémy na:

- **otevřené**
- a **uzavřené**.

Dále systémy dělíme podle toho, zda si pamatují předchozí vnitřní stav na:

- **statické**
- a **dynamické**.

A další dělení podle spojitosti signálu v čase na systémy:

- **spojité**
- a **diskrétní**.

1.1 Statický systém

Statický systém je definován tak, že jeho výstup je jednoznačně definován jeho vstupem a nezávisí na předešlé historii vstupů a výstupů.

1.2 Dynamický systém

Hodnoty výstupních veličin závisí nejenom na aktuálních hodnotách vstupů, ale i na předešlé historii vstupů a výstupů. Mezi parametry dynamických systémů patří vždy čas. Dynamický systém lze popsat vnitřním a vnějším popisem.

1.3 Vnější popis systému

Vnější popis vyjadřuje dynamické vlastnosti dějů mezi vstupem a výstupem daného systému. Systém si lze představit jako černou skříňku a jeho vlastnosti se určují pomocí vztahů mezi vstupními a výstupními signály. Děje, které probíhají uvnitř, nás nezajímají. Vnější popis lze realizovat buď v časové, nebo frekvenční oblasti. Pro vysvětlení budou uvažovány jen systémy s jedním vstupem a jedním výstupem, takzvané SISO (Single Input Single Output) systémy. Každý systém je možné popsat několika způsoby vnějšího popisu [1] [2].

1.3.1 Lineární diferenciální rovnice

Lineární diferenciální rovnice je nejobecnější vnější popis lineárního spojitého systému. Pro jednorozměrný systém vypadá lineární diferenciální rovnice následovně:

$$\begin{aligned} &a_n\, y^{(n)}(t) + a_{n-1}\, y^{(n-1)}(t) + \ldots + a_1\, y'(t) + a_0\, y(t) = \\ &= b_m\, u^{(m)}(t) + b_{m-1}\, u^{(m-1)}(t) + \ldots + b_1\, u'(t) + b_0\, u(t) \end{aligned} \quad . \tag{1.1}$$

Kde $u(t)$ je vstup systému, $y(t)$ výstup systému a a_i a b_i jsou reálné konstantní koeficienty. Uvedená rovnice lze zapsat i takto:

$$\sum_{i=0}^{n} a_i\, y^{(i)}(t) = \sum_{j=0}^{m} b_j\, u^{(j)}(t) \tag{1.2}$$

Systémy, které jsou popsány diferenciální rovnicí s vyšším stupněm m derivace vstupního signálu, než je stupeň derivace n výstupního signálu jsou fyzikálně nerealizovatelné. Proto musí platit podmínka realizovatelnosti, že $m \leq n$. Pro řešení diferenciální rovnice je nutné znát průběh vstupní veličiny a počáteční podmínky $y(0)$ až $y^{(n-1)}(0)$. Rovnice lze řešit, buď standardní metodou, nebo s využitím Laplaceovy transformace [1] [2].

1.3.2 Operátorový přenos systému

Operátorový přenos se získá pomocí Laplaceovy transformace z diferenciální rovnice při nulových počátečních podmínkách. Pro spojitý systém a čas $t \geq 0$, je definován, jako poměr Laplaceova obrazu výstupní veličiny k Laplaceově obrazu vstupní veličiny. Přenosová funkce ve tvaru racionální lomené funkce vypadá následovně:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m\, s^m + b_{m-1}\, s^{m-1} + \ldots + b_1\, s + b_0}{a_n\, s^n + a_{n-1}\, s^{n-1} + \ldots + a_1\, s + a_0}. \tag{1.3}$$

Kde s je Laplaceův operátor, a i zde musí platit podmínka realizovatelnosti. Oba polynomy přenosové funkce v čitateli i jmenovateli lze vyjádřit ve tvaru součinu kořenových činitelů jako:

$$A(s) = a_n\, (s - p_1)(s - p_2)\ldots(s - p_n) \tag{1.4}$$

$$B(s) = a_m\, (s - p_1)(s - p_2)\ldots(s - p_m) \tag{1.5}$$

Kde p_i se nazývají póly přenosu a n_j nuly přenosu. Polynom $A(s)$ se nazývá charakteristický polynom a má rozhodující vliv na stabilitu systému [1] [2].

1.3.3 Nuly a póly přenosu systému

Nuly i póly mohou být reálné, komplexně sdružené nebo ryze imaginární. Nuly charakterizují spojení systému s okolím a jsou to komplexní frekvence, pro které je přenos mezi vstupem a výstupem blokován. Nuly mění odezvu systému a tím komplikují jeho návrh a řízení.

Póly charakterizují vnitřní dynamiku systému a jsou to komplexní frekvence, které je systém sám schopen generovat, aniž by tyto frekvence obsahoval vstupní signál.

Pokud se přechodová charakteristika ustálí na nějaké ustálené hodnotě zesílení *Z*, systém je stabilní, to znamená, že všechny póly mají zápornou reálnou část. V opačném případě je systém, buď na hraně stability, nebo nestabilní a některé póly mají nulovou nebo kladnou reálnou část.

Pokud se přechodová charakteristika ustálí na nulové hodnotě, znamená to, že systém obsahuje v čitateli alespoň jednu nulu, soustava má derivační vlastnosti.

Pokud přechodová charakteristika kmitá, systém obsahuje i komplexně sdružené póly. Opačné tvrzení však nemusí platit [3] [4].

$$G_{5s}(s) = \frac{Z(s+z_1)(s+z_2)(s+z_3)(s+z_4)}{(s+p_1)(s+p_2)(s+p_3)(s+p_4)(s+p_5)} \frac{p_1 p_2 p_3 p_4 p_5}{z_1 z_2 z_3 z_4}$$

Obrázek 1.1 – Popis systému 5. řádu pomocí nul, pólu a zesílení přenosu

1.3.4 Přechodová funkce a charakteristika

Přechodová funkce systému se definuje jako odezva systému na jednotkový skok při nulových počátečních podmínkách a přechodová charakteristika systému je grafické znázornění přechodové funkce. Přechodová funkce se značí jako *h*(*t*).

Jednotkový skok nebo jinak jednotková změna vstupní veličiny je definována následovně:

$$\begin{aligned} u_0(t) &= 0 \text{ pro } t < 0 \\ u_0(t) &= 1 \text{ pro } t \geq 0 \end{aligned} . \tag{1.6}$$

Fyzikální realizovatelnost skokové změny, kterou lze přepočítat na jednotkový skok, je pro elektrické systémy pouze vypnutí a zapnutí systému. U mechanických nebo hydraulických systému je nutná určitá aproximace [1].

Obraz jednotkového skoku vypočítaný pomocí Laplaceovy transformace:

$$U_0(s) = L\{u_0(t)\} = \frac{1}{s}. \tag{1.7}$$

Obraz přechodové funkce:

$$H(s) = L\{h(t)\} = G(s)U_0(s) = \frac{G(s)}{s}. \tag{1.8}$$

1.3.5 Impulsní funkce a charakteristika

Impulsní funkce systému je definována jako odezva systému na vstupní signál tvaru Diracova impulsu při nulových počátečních podmínkách a impulsní charakteristika je znázornění impulsní funkce. Impulsní funkce systému se značí $g(t)$ a Diracův impuls jako $\delta(t)$. Diracův impuls je definován následovně:

$$\delta(t) = 0 \text{ pro } t \neq 0; \quad \int_{-\infty}^{+\infty} \delta(t)\, dt = 1. \tag{1.9}$$

Fyzikální realizovatelnost Diracova impulsu není možná, ale používá se její aproximace, kde doba trvání impulsu je zanedbatelná vůči časovým konstantám uvažovaného systému. Reálnou aproximací Diracova impulsu je použití impulsu o šířce h a výšce $\frac{1}{h}$, kde šířka impulsu $h \to 0$ [1].

Laplaceův obraz Diracova impulsu:

$$\delta(s) = L\{\delta(t)\} = 1. \tag{1.10}$$

Obraz impulsní funkce:

$$G(s) = L\{g(t)\} = G(s)\,\delta(s) = G(s). \tag{1.11}$$

1.3.6 Frekvenční přenosová funkce

Frekvenční přenosová funkce se získá jako podíl Fourierových obrazů výstupního a vstupního signálu při nulových počátečních podmínkách. Předpokladem pro stanovení frekvenční přenosové funkce je lineární systém a harmonický vstupní signál. Lze ji získat i dosazením $j\omega$ za s do operátorového přenosu:

$$G(j\omega) = \frac{Y(j\omega)}{U(j\omega)} = \frac{b_m (j\omega)^m + b_{m-1} (j\omega)^{m-1} + \ldots + b_1 j\omega + b_0}{a_n (j\omega)^n + a_{n-1} (j\omega)^{n-1} + \ldots + a_1 j\omega + a_0}. \tag{1.12}$$

Dalším vyjádření frekvenční přenosové funkce je jako komplexní číslo s reálnou a imaginární složkou:

$$G(j\omega) = \frac{Y(j\omega)}{U(j\omega)} = \mathrm{Re}\{G(j\omega)\} + j\,\mathrm{Im}\{G(j\omega)\}. \tag{1.13}$$

Nebo v polárních souřadnicích se složkami jako je amplitudové zesílení a fázové natočení procházejícího signálu [1].

$$G(j\omega) = \frac{Y(j\omega)}{U(j\omega)} = |G(j\omega)| \cdot e^{j\varphi(\omega)}. \tag{1.14}$$

1.3.7 Frekvenční charakteristika

Frekvenční charakteristika je grafické znázornění frekvenčního přenosu v závislosti na frekvenci a popisuje dynamické vlastnosti systému ve frekvenční oblasti. Lze vyjádřit

- v **komplexní rovině** jako amplitudo-fázová, respektive Nyquistova charakteristika
- a v **logaritmických souřadnicích** jako takzvaná Bodeho charakteristika [5].

1.4 Vnitřní popis systému

Vnitřní popis vyjadřuje dynamické vlastnosti systému v časové oblasti a vede na takzvaný stavový popis systému. Nejdříve uvedení rozdílu mezi vnějším a stavovým popisem.

- Stavový popis vyjadřuje dynamiku změn stavových proměnných v podobě soustavy n obyčejných diferenciálních rovnic prvního řádu.
- Vnější popis systému definuje závislost výstupu na vstupu v podobě obyčejné diferenciální rovnice n-tého řádu.

1.4.1 Úvod do stavového popisu

Nejprve budou uvedeny základní vlastnosti:

- **Stav systému** je popsán soustavou proměnných, nazývaných stavovými proměnnými. Známe-li hodnoty všech stavových proměnných v čase a hodnoty vstupu v čase, můžeme určit chování soustavy.
- **Stavový vektor** lze považovat za n stavových proměnných.
- **Stavovým prostorem** nazýváme n-rozměrný prostor, jehož souřadnicemi jsou stavové proměnné. Každý stav je pak představován jedním bodem stavového prostoru [6].

1.4.2 Stavová rovnice

Mějme lineární diferenciální rovnici n-tého řádu popisující obecný systém, která je uvedena v rovnici 1.1.

Zavedou-li se stavové proměnné $x_1(t) = y(t)$, $x_2(t) = y'(t)$ až $x_n(t) = y^{(n)}(t)$, může se tato rovnice vyjádřit ve formě soustavy *n* lineárních diferenciálních rovnic prvního řádu, které se nazvou stavovými rovnicemi:

$$
\begin{aligned}
x_1'(t) &= x_2(t) \\
x_2'(t) &= x_3(t) \\
&\cdot \\
&\cdot \\
x_{n-1}'(t) &= x_n(t) \\
x_n'(t) &= \frac{1}{a_n}u(t) - \frac{a_0}{a_n}x_1(t) - \frac{a_1}{a_n}x_2(t) - \ldots - \frac{a_{n-1}}{a_n}x_n(t)
\end{aligned}
\qquad (1.16)
$$

Výstupní rovnice soustavy se získá dosazením stavových proměnných do diferenciální rovnice:

$$y(t) = b_0\, x_1(t) + b_1\, x_2(t) + ... + b_m\, x_{(m+1)}(t)\,. \tag{1.17}$$

Soustava diferenciálních rovnic prvního řádu se může vyjádřit i maticově:

$$\begin{aligned} \boldsymbol{x}' &= \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{B}\,u \\ y &= \boldsymbol{C}\,\boldsymbol{x} \end{aligned} \tag{1.18}$$

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ x_n \end{pmatrix} \quad \boldsymbol{A} = \begin{pmatrix} 0 & 1 & 0 & \cdot & 0 \\ 0 & 0 & 1 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & 1 \\ -\frac{a_0}{a_n} & -\frac{a_1}{a_n} & -\frac{a_2}{a_n} & \cdot & -\frac{a_{n-1}}{a_n} \end{pmatrix}, \tag{1.19}$$

$$\boldsymbol{B} = \begin{pmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \frac{1}{a_n} \end{pmatrix} \quad \boldsymbol{C} = \begin{pmatrix} 1 & 0 & 0 & \cdot & 0 \end{pmatrix}$$

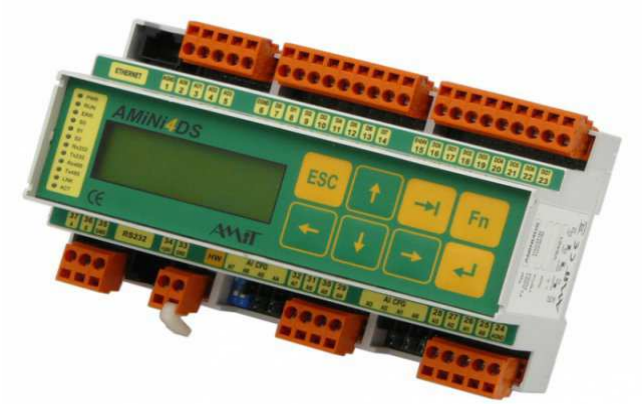
kde $\boldsymbol{x}$ je stavový vektor, $\boldsymbol{A}$ je matice soustavy, $\boldsymbol{B}$ vstupní matice, $\boldsymbol{C}$ je matice výstupu, u je vstup a y výstup soustavy [6] [7].

2 AMiNi4DS

AMiNi4DS je malý kompaktní řídicí systém používaný pro simulaci a řízení, který má následující vlastnosti:

- 8 tlačítek,
- 8 galvanicky oddělených číslicových vstupů,
- 8 galvanicky oddělených číslicových výstupů,
- 8 analogových vstupů,
- 4 analogové výstupy,
- grafický LCD 122 x 32 bodů,
- sériové rozhraní RS232,
- sériové rozhraní RS485 s galvanickým oddělením,
- rozhraní Ethernet 10 Mbps,
- montáž na lištu DIN 35 mm,
- paměť FLASH o velikosti 256 + 1024 kB,
- paměť RAM o velikosti 1024 kB a
- paměť EEPROM o velikosti 2 kB.

Pro bližší popis vlastností a možností je nutno navštívit manuálové stránky AMIT z kterých bylo čerpáno pro celou druhou kapitolu [7] [8].



Obrázek 2.1 – Řídící systém AMiNi4DS [8]

2.1 Operační systém NOS

NOS je operační systém, který umožňuje práci v AMiNi4DS. Konkrétně spuštění aplikací vytvořených v programu DetStudio. Komunikaci s protokolem DB-Net pomocí RS232 i RS485 pro až třicet dva stanic, dále komunikaci po DB-Net/IP pomocí Ethernetu pro neomezené množství stanic a další rozšíření práce se vstupy a výstupy zařízení, které pro tuto práci nejsou potřeba.

Operační systém není standardně součástí řídicího systému a je nutno jej do něj nahrát. Nahrání lze uskutečnit pomocí linky RS232. Systémy s označením W mají možnost nahrát systém i pomocí Ethernetu. Samotné nahrání probíhá přes prostředí DetStudia, kde se pomocí hlavního menu, záložky Přenos a výběru Nahrát NOS zobrazí průvodce nahrání operačního systému, kde se dále postupuje dle pokynů.

2.2 Nastavení komunikace

Pro spojení s řídicím systémem AMiNi4DS je nutné nastavit jisté parametry, jak na řídicím systému, tak i na používaném počítači. Konkrétně se jedná o adresu, která musí být jedinečná a rychlost komunikace, která musí být shodná na všech stanicích nacházejících se v síti.

Nastavení na řídicím systému probíhá prostřednictvím označených přepínačů a u některých systémů lze nastavení provést pomocí konfigurační obrazovky. Konkrétní popis nastavení přepínačů lze nalézt v manuálu k programu DetStudio nebo v manuálu k samotnému řídicímu systému.

Nastavení komunikace na počítači se provede v DetStudiu pomocí hlavního menu, záložky Přenos a výběr Nastavení komunikace. Kde je nastavení intuitivní.

Ke kontrole správného nastavení a tedy funkčnosti komunikace slouží výběr Identifikace ze záložky Přenos. Správnost nastavení, tedy je-li řídicí systém připojen správně k počítači a je-li připojen k napájecímu napětí je potvrzena, pokud v Identifikaci je vše vyplněno a vše souhlasí.

2.3 Činnost řídicího systému

Činnost systému probíhá sekvenčně a je rozdělena do takzvaných procesů. Proces je část programu, která pracuje relativně samostatně a nezávisle na ostatních procesech. U jednoduchých řídicích aplikací, se jedním procesem obvykle řídí jeden měřící nebo regulační okruh, čímž je zajištěna správná časová součinnost, správná vazba všech prvků v okruhu a její nezávislost na dalších okruzích. U složitějších aplikací se do jednoho procesu vkládají okruhy se stejným časováním. V programu DetStudio lze definovat:

- Procesy **Interrupt_0** až **Interrupt_15**, které mají v rámci projektu nejvyšší prioritu a možnost přerušovat se navzájem. Interrupt proces s nejvyšší prioritou je ten proces, který má nejnižší číslo.
- Procesy **Hi_0** a **Hi_1** jsou procesy s druhou nejvyšší prioritou a mají také možnost přerušovat ostatní procesy, ovšem jen ty s prioritou nižší než mají oni. Jsou velmi rychlé a periodu mají od 1 do 1667 ms.
- Proces **Quick**, který je považován ještě za rychlý proces a umí opět přerušit procesy s nižší prioritou. Možnost nastavení periody 5, 10, 20, 50 a 100 ms.
- Procesy **Normal_0** až **Normal_15**, které jsou považovány za řádné procesy. Neumí vyvolat přerušení ostatních procesů a mají periodu 0,1 až 1000000 s.
- Proces **IDLE** zvaný jako prázdný. Je spouštěn v okamžiku, kdy si žádný ze zbývajících procesů nenárokuje procesorový čas.
- Proces **INIT** je vykonán pouze jednou, a to jako první po studeném nebo teplém startu řídicího systému.

2.4 Přeložení a přenos projektu do řídicího systému

Pro přeložení projektu se v DetStudiu vybere menu Generace a záložka Generuj vše. Pokud se objeví chyby lze si je prohlédnout v okně Seznam chyb. Pokud je aplikace úspěšně vygenerována, tedy bez chyb, zobrazí se okno, které zobrazí předpokládané obsazení pamětí RAM a FLASH řídicího systému. Pro samotný přenos programu do řídicího systému se využije záložka Přenos a výběr Přenos systému.

2.5 DetStudio

DetStudio je návrhové prostředí pro programování řídicích systémů firmy Amit, pro tento případ systému AMiNi4DS. Založení nového projektu se provede volbou Soubor a Nový, poté se zobrazí okno Vytvoření nového projektu, kde se musí vybrat typ Řídicího systému, v tomto případě AMiNi4DS, terminál interní a dále se vybere název projektu a jeho umístění. Po potvrzení se otevře okno Parametry projektu, kde pod záložkou Různé nastavíme Id1 a Id2, což jsou řetězce, které řídicí systém vrátí jako odpověď na dotaz identifikace. Pod záložkou Protokol, zaškrtneme volbu Pasivní stanice, což znamená, že systém v komunikační síti DB-Net aktivně nekomunikuje s ostatními systémy, ovšem tato volba se netýká komunikace prostřednictvím Ethernetu. V záložce Komunikace se nastaví Adresa stanice na 2, Způsob komunikace na volbu Ethernet a IP Stanice na 192.168.1.102. Samotná tvorba aplikace začíná prací s oknem Projekt, v tomto okně se stromovou strukturou je na výběr z mnoha možností. Začíná se otevřením adresáře Databáze a v něm otevření a vytvoření potřebných proměnných a aliasů. Dalším adresářem je IO konfigurace, která skrývá informace o dostupných kanálech. DetStudio používá dva typy kanálů.

- **Fyzický kanál**, což je skupina signálů, která je fyzicky dostupná na zvoleném řídicím systému a
- **Logický kanál**, kterým definujeme, jakým způsobem se bude přistupovat ke kanálům fyzickým. K jednomu kanálu fyzickému, tak můžeme přistupovat z více kanálu logických. Jako příklad lze uvést analogové vstupy, ke kterým lze přistupovat jako k proudovým a napěťovým nebo jako ke vstupům čidla Ni1000.

Tato filosofie umožňuje efektivně přecházet mezi různými druhy řídicích systémů se stejným aplikačním programem, neboť jsou potlačeny fyzické rozdíly.

Další položkou je adresář Procesy, kde se již po otevření nachází dva základní automaticky vytvořené a to Proc00, což je hlavní proces a ProcIDLE jako obsluha obrazovek. Přidáním a vytvořením dalších procesů se rozšiřuje funkce výsledného programu. Při jejich tvorbě se nastavuje typ programovacího jazyka, pro tento případ jazyk ST jako strukturovaný, typ procesu a perioda v milisekundách. Programování ve strukturovaném jazyce spočívá především ve vkládání funkčních modulů z okna Toolbox a jejich parametrizaci. Posledním potřebným adresářem je položka Obrazovky. Princip tvorby obrazovek spočívá v umisťování statických a ovládacích prvků na plochu představující LCD a v jejich parametrizaci. Existují dva základní typy obrazovek:

- **Global**, která je určena pouze ke vkládání neviditelných prvků, které se vždy vykonají dříve než prvky v aktuálně vybrané standardní obrazovce. Do globální obrazovky se vkládají takové prvky, které musejí být prováděny vždy, tedy nezávisle na aktuálně zobrazované standardní obrazovce. Příkladem může být signalizace blikající diodou nebo reakce na stisk nějaké klávesy v celé aplikaci. A druhou obrazovkou je
- **Standardní**, na kterou lze umístit prvky viditelné i neviditelné. Ty prvky, které jsou vázány na proměnné, musí být periodicky obnovovány. Pro obnovování se požívá klasický mechanismus, který si zajišťuje sama obrazovka, spočívá v překreslování všech prvků na obrazovce se zvolenou periodou. Perioda lze nastavit ve vlastnosti obrazovky RefreshPeriod. Existuje ještě rozšířené obnovování s využitím časovače, u kterého lze nastavit různou periodu obnovování pro každý prvek samostatně. Oba způsoby lze i kombinovat nebo lze nastavit hodnotu RefreshPeriod na nulu a obrazovka se stává statickou.

Při práci s prvky a obrazovkami, jsou pro jejich ovládání využívány klávesy na řídicím systému. Klávesy jsou následující:

- **Enter** provádí potvrzování vybraných položek, spuštění editace prvku a její ukončení.
- **Esc** zaručuje návrat o úroveň zpět a ukončení editace bez její potvrzení.
- **Šipky** zajišťují rolování v nabídkách. Šipky nahoru a dolu editují hodnotu po aktivaci editačního prvku a šipkami doleva a doprava se posunujeme mezi jednotlivými řády.
- **Tab** přepíná mezi jednotlivými editačními prvky, které jsou umístěny na aktuální obrazovce.

Poslední důležitou věcí při práci s obrazovkami je možnost odchodu z aktuální obrazovky na nadřazenou. To se programuje pomocí neviditelného prvku KeyScreen, který má dva důležité parametry:

- **GoToScreen**, kterým vybereme, na kterou obrazovku se má přeskočit a
- **KeyCode**, kterým definujeme klávesu, kterou bude přeskok realizován.

3 Matlab

Matlab je vysokoúrovňový jazyk a interaktivní prostředí pro numerické výpočty, vizualizaci a programování. Používá se pro vědeckotechnické výpočty, simulaci, modelování, analýzu dat, návrhy algoritmů, měření a zpracování dat, zpracování obrazu a videa, ve finančnictví, pro tvorbu aplikací včetně grafického rozhraní a v neposlední řadě pro návrh řídících a komunikačních systémů. Název Matlab vznikl spojením slov matrix a laboratoř, což v překladu znamená laboratoř pro matice. Vlastní programovací jazyk vychází z jazyka C.

Matlab je implementován na platformách Windows, Linux, Mac i Solaris a nabízí rychlé výpočetní jádro, podporu paralelních výpočtů, otevřený a rozšiřitelný systém, 2D i 3D grafiku, množství aplikačních knihoven, objektové programování, integraci s jazykem Java, nástroje pro tvorbu grafického a uživatelského rozhraní GUIDE, komunikaci s externím vybavením pomocí sériové linky, rozhraní GPIB nebo VISA, a mnoho dalších funkcí.

Vlastností, která nejvíce přispěla k celosvětovému rozšíření Matlabu, je jeho otevřená architektura, což znamená, že jeho jazyk je neomezeně rozšiřitelný a nově vytvořené funkce jsou snadno přenositelné mezi různými platformami.

Základním nástrojem výpočetního systému je uživatelské rozhraní, které obsahuje prohlížeč pracovního prostoru zvaný Workspace, prohlížeč souborů a adresářů, okno historie příkazů, editor, debugger, nápovědu, příkazové okno a mnoho dalších užitečných nástrojů. Důležitým pomocníkem je nástroj pro import dat, který umožňuje načítání dat z prakticky jakéhokoliv objektu, jako je text, tabulka, databáze, binární data, obrázky, a další.

Předností programu je těsná integrace s jazykem Java, což znamená, že v Matlabu je možnost vytvářet složitá grafická rozhraní s použitím grafických objektů Javy. Další výhodou spojenou s otevřenou architekturou je vznik knihoven funkcí, nazývaných toolboxy, které rozšiřují použití programu v mnoha vědních a technických oborech. Pro příklad v aplikované matematice, zpracování signálů, zpracování obrazu, výpočetní biologie, automatické řízení a regulace, finanční analýza nebo třeba modelování fyzikálních soustav [10] [11] [12].

3.1 GUIDE

Prostředí Matlab umožňuje vytvořit grafické rozhraní k uživatelskému programu i bez potřeby dodatečných toolboxů. Je možné vytvořit hlavní okno se standardními prvky, které lze programově měnit a události neboli změny těchto prvků, jako je jejich použití či editace lze spojit s vlastní programovou obsluhou. Takováto obsluha se nazývá callback function. Výsledný program s grafickým rozhraním se skládá ze souboru s příponou .m, která má předepsanou strukturu a obsahuje vlastní program, tedy funkce obsluhující jednotlivé události. Jeho součástí je povinná inicializační funkce, obsluha programového vytvoření okna figure a definice prvků uicontrol.

Vytvoření grafického rozhraní pomocí toolboxu GUIDE vytvoří standardní soubor s příponou .m, u kterého se automaticky vygeneruje inicializační funkce a hlavičky callback funkcí použitých ovládacích prvků. A soubor s příponou .fig, který obsahuje popis grafického rozhraní jako jsou vlastnosti okna figure, použité grafické objekty uicontrol a jejich výchozí vlastnosti a umístění.

GUIDE neboli Graphical User Interface Development Environment je nástroj Matlabu pro tvorbu grafického uživatelského prostředí. GUIDE se spustí, pokud se

do příkazové řádky Matlabu vloží příkaz guide. Poté se otevře dialog GUIDE Quick start, který nabídne otevření prázdného projektu, nebo je možnost otevření nějakého již předdefinovaného. Po otevření vybraného projektu se spustí návrhový editor, z kterého se ovládají ostatní nástroje. Návrhový editor umožňuje snadno a rychle vytvářet grafické rozhraní přetahováním součástí zvaných uicontrol jako jsou:

- tlačítka (Push Buttons),
- posuvníky (Sliders),
- výběrová tlačítka (Radio Buttons),
- zaškrtávací pole (Check Box),
- upravitelné texty (Edit Text),
- statické texty (Static Text),
- výběry z vysouvacích nabídek (Pop-up Menu),
- výběry z nabídek (Listbox),
- přepínací tlačítka (Toggle Button),
- tabulky (Table),
- osy (Axes),
- a další

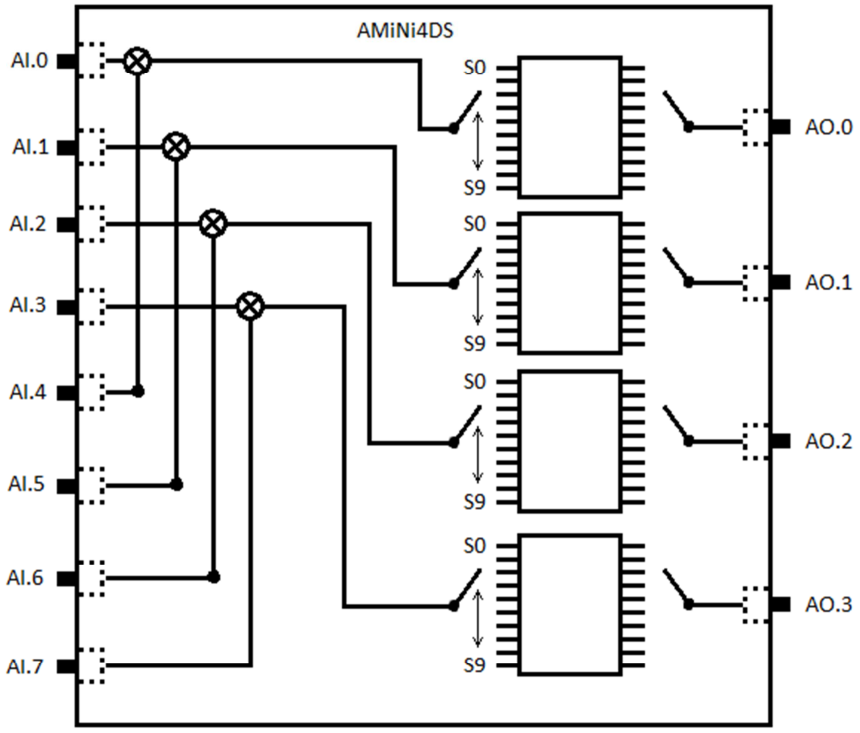
z palety komponent do návrhové plochy. Velikost a umístění jednotlivých uicontrol se upravuje snadnou prací s myší.

Pro další nastavení uicontrol je možnost využít nástroje jako jsou:

- Align Objects – seřadí objekty navzájem mezi sebou
- Menu Editor – vytváří řádkové menu okna a kontextové menu
- Tab Order Editor – mění pořadí prvků, v kterém jsou vybrány tabulátorem
- Toolbar Editor – nastavení menu figure
- Editor – uloží vytvořené rozhraní do souboru .fig
- Property Inspector- nastavuje a dohlíží na vlastnosti objektů
- Object Browser – sleduje seznam ukazatelů objektů.

Návrhová plocha se po aktivaci stává vlastním grafickým uživatelským prostředím. Vytvoření samotného GUI tedy zahrnuje vybrat a uspořádat komponenty v návrhovém editoru GUIDE a naprogramovat callback funkce použitých komponent v předpřipraveném souboru s příponou .m, který se vytvoří po aktivaci návrhové plochy [10] [11] [12].

4 Praktická část pro AMiNi4DS



Obrázek 4.1 – Znázornění vnitřního zapojení vstupů a výstupů AMiNi4DS

4.1 Řešení programové sady v prostředí DetStudio

V periodickém opakování vypočítává výstupy řídicího systému na základě aktuálních hodnot vstupů systému. Výpočet je realizován pomocí diskrétního lineárního stavového modelu. Vstupem $u(k)$ do modelu je hodnota odpovídající součtu dvojic analogových vstupů AI.n a AI.(n + 4) malého řídicího systému AMiNi4DS, kde $n = \{0, 1, 2, 3\}$ je číslo vstupu. Vypočítaný výstup y (k) je poslán na analogový výstup A0.n.

Tímto způsobem jsou realizovány 4 nezávislé soustavy se stejným intervalem vzorkování. Pro každou soustavu se vstupem AI.n + AI.(n + 4) a výstupem AO.n, kde $n = \{0, 1, 2, 3\}$ lze zvolit z 10 předpřipravených modelů soustav. Každý model je určen trojicí matic parametrů $\boldsymbol{A}_x$, $\boldsymbol{b}_x$, $\boldsymbol{c}_x$, kde $x = \{0, 1, 2, ..., 8, 9\}$. Dále má každý model přiřazeny 4 vektorové proměnné $\boldsymbol{x}_{x\_1}$, $\boldsymbol{x}_{x\_2}$, $\boldsymbol{x}_{x\_3}$, $\boldsymbol{x}_{x\_4}$, kde v indexu $x = \{0, 1, 2, ..., 8, 9\}$, reprezentující stavové vektory stavového diskrétního popisu. Každý stavový vektor soustavy je přiřazen přesně k jednomu vstupu/výstupu $0 - 3$. Tedy stavových vektorů je potřeba vytvořit přesně 40. Matice parametrů jsou vytvořeny a vloženy do programové sady pro řídicí systém AMiNi4DS pomocí programu MATLAB.

V periodickém opakování se souběžně s výpočty výstupů systému aktualizuje LCD zobrazovač. Na zobrazovači je možné čtení hodnot analogových vstupů a výstupů ve formě hodnoty elektrického napětí. Je možné čtení číslicových vstupů i výstupů. Dále je možnost zobrazení základních informací o řídicím sytému a hlavně výběr soustav, které

budou vloženy jako modely mezi analogové vstupy a výstupy řídicího systému. Každá soustava má přiřazené číslo od 0 do 9 a každý vstup/výstup má přiřazenou proměnnou, která svoji hodnotou reprezentuje použitou soustavu.

Programová sada vytvořená v DetStudiu obsahuje 4 procesy s rozdílnými prioritami představující části programové sady.

4.1.1 Zjištění minimální periody vzorkování

Použije se proces typu IDLE, který se dostane k procesorovému času pouze v době, kdy si žádný jiný proces procesorový čas nenárokuje. Počet provedení procesu IDLE na určitém časovém intervalu je úměrný zatížení systému, čím vícekrát se provede, tím méně je systém zatížen.

Pro zjištění minimální periody vzorkování pro simulaci chování vybraných soustav se v procesu IDLE vytvoří proměnná, která se inkrementuje při každém spuštění tohoto procesu. Dále se vytvoří proces HI_0 s nejvyšší prioritou a periodou vzorkování 1 s. V tomto procesu se inkrementovaná proměnná nastaví na hodnotu nula, ale ještě předtím se uloží do jiné proměnné, která se zobrazí v ladícím nástroji DetStudia. Čím menší je hodnota této proměnné, tím méněkrát se provedl proces IDLE, který se spouští jen v okamžiku, když si žádný proces nenárokuju procesorový čas. Testováním bylo odzkoušena minimální perioda vzorkování a tento test je znázorněn v Tab. 4.1.. Hodnoty v tabulce vyjadřují proměnnou určující počet inkrementací proměnné v procesu IDLE.

Minimální změřená perioda při simulaci 4 modelů 5. řádu, při které je zařízení funkční, je 14 ms. Ovšem pro manipulaci s řídicím systémem, jako je zobrazování vstupů, či změna simulovaných modelů, je systém více zatížen a z tohoto důvodu jsem zvolil minimální periodu větší. Zvolil jsem 20 ms.

Tabulka 4.1 Testování minimální vzorkovací periody

Vzorkovací perioda	4 x SISO 5. řádu	4 x SISO 4. řádu	4 x SISO 1. řádu
20 ms	301	408	688
18 ms	222	360	665
16 ms	140	298	640
15 ms	91	260	625
14 ms	36	217	617
13 ms	0	165	600

4.1.2 Proces ProcHI01

Periodický proces ProcHI01 je typu Hi_1 s periodou 20 ms a simuluje časové chování soustav popsaných stavovým modelem. Proces začíná přečtením hodnot na analogových vstupech pomocí logického kanálu a uložením těchto hodnot do proměnné ***AI***. Rozsah měřeného napětí je nastaven na 0 až 10 V. Poté proběhnou čtyři komplexní výpočty simulující časové chování soustav na vstupech/výstupech 0 až 3. O použitých soustavách 0 až 9 na jednotlivých vstupech/výstupech rozhodují proměnné *Ukaz0* až *Ukaz3*, které se editují na vestavěném LCD. Z každého ze čtyř výpočtů získáme výstupní napětí a stavový vektor.Vektor se použije při následných výpočtech výstupního napětí

periodického procesu a výstupní napětí čtyř soustav se uloží do proměnné ***AO***. Dále už následuje pouze nastavení hodnot na analogových výstupech pomocí logického kanálu a proměnné ***AO*** a proces běží za 20 ms znovu. Rozsah analogových výstupů je nastaven opět na 0 až 10 V. Rovnice 2.1 ukazuje teoretický výpočet stavového vektoru ***x*** a rovnice 2.2 ukazuje teoretický výpočet výstupu *y*.

$$\boldsymbol{x}_3(k+1) = \boldsymbol{A}_3\, \boldsymbol{x}_3(k) + \boldsymbol{B}_3\, u(k) \tag{2.1}$$

Kde $\boldsymbol{x}_3$ je stavový vektor soustavy 3, $\boldsymbol{A}_3$ je matice soustavy 3 a $\boldsymbol{B}_3$ je vstupní matice soustavy 3.

$$y(k) = \boldsymbol{C}_3\, \boldsymbol{x}_3(k) \tag{2.2}$$

Kde *y* je výstup soustavy 3, $\boldsymbol{C}_3$ je matice výstupů soustavy 3 a $\boldsymbol{x}_3$ je stavový vektor soustavy 3.

Na obr. 4.2 je vidět praktický výpočet stavového vektoru a výstupu soustav v programu DetDtudio.

```
case 3
     Let u2[0,0]=AI[2,0]+AI[6,0]
     MtxMul y2, c3, x3_3
     Let AO[2,0]=y2[0,0]
     MtxMul x3a, b3, u2
     MtxMul x3b, A3, x3_3
     MtxAdd x3_3, x3a, x3b
endcase
```

Obrázek 4.2 – Výpočet simulující dynamické chování soustavy 3

4.1.3 Proces Proc00

Periodický proces Proc00 je typu Normal_0 s periodou 100 ms. Proces čte napětí baterie a napájecí napětí řídicího systému z logických kanálů a zapisuje stavy do proměnných *Pwr* a *Bat*. Dále pomocí logického kanálu čte digitální vstupy a jejich hodnoty zapisuje do proměnné ***DI***. Poslední funkcí procesu je zápis digitálních výstupů z proměnné ***DO*** a opět použitím logického kanálu. Tento proces je vytvořen pouze pro zobrazení určitých parametrů řídicího systému. Pro simulaci není využit.

4.1.4 Proces ProcIDLE

Proces ProcIDLE je spouštěn v okamžiku, kdy si žádný ze zbývajících procesů nenárokuje procesorový čas.

4.1.5 Proces ProcINIT

Jednorázový proces ProcINIT vykonaný při startu řídicího systému, který zjistí název a IP-konfiguraci stanice.

4.1.6 Obrazovka Menu

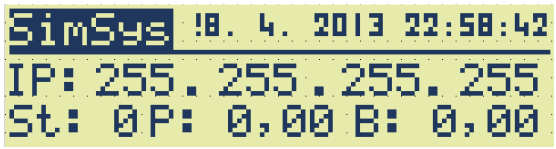
Obrazovka Menu je úvodní obrazovka, která umožňuje přístup ke všem dalším volbám řídicího systému. Na obrazovce je zobrazen řádkový seznam možností, které lze zobrazit vybráním a stiskem klávesy ENTER. Pro návrat z další volby zpět na hlavní menu slouží klávesa ESC. Volby na obrazovce Menu:

- Informace o systému-obrazovka System
- Analogové v/v-obrazovka Analog
- Digitální v/v-obrazovka Digital
- Výběr soustavy-obrazovka Selection of System

Obrázek 4.3 – Obrazovka Menu

4.1.7 Obrazovka System

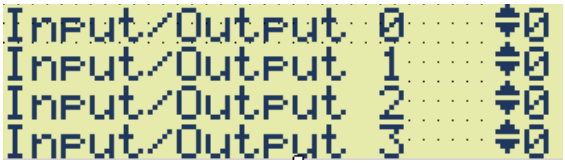
Obrazovka System zobrazí aktuální datum a čas v době otevření obrazovky. Dále zobrazí název a IP adresu stanice v závislosti na přečtených údajích v procesu ProcINIT.



Obrázek 4.4 – Obrazovka System

4.1.8 Obrazovka Selection of System

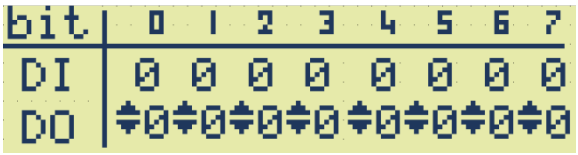
Obrazovka Selection of System umožní vybrat aktivní soustavu 0 – 9 na vstup/výstup 0 – 3 nastavením proměnných *Ukaz0* – *Ukaz3*. Je zde možnost vybrat stejnou soustavu pro více vstupů/výstupů.



Obrázek 4.5 – Obrazovka Selection of System

4.1.9 Obrazovka Digital

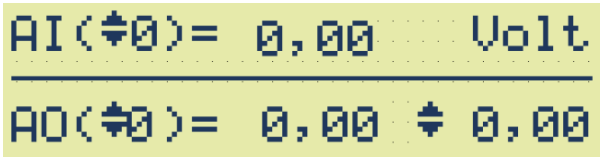
Obrazovka Digital zobrazuje hodnoty proměnné **DI**, které reprezentují logický kanál digitálního výstupu a umožňuje nastavit hodnoty proměnné ***DO*** reprezentující logický kanál digitálního výstupu.



Obrázek 4.6 – Obrazovka Digital

4.1.10 Obrazovka Analog

Obrazovka Analog umožňuje výběr a pomocí logického kanálu i zobrazení libovolného analogového vstupu ***AI*** nebo libovolného analogového výstupu ***AO*** ve voltech. Dále umožňuje i nastavit výstupní napětí na jednotlivých výstupech.



Obrázek 4.7 – Obrazovka Analog

5 Praktická část pro MATLAB

5.1 Řešení GUI v MATLABu

Program umožní vytvoření nových modelů dynamických soustav, popsaných trojicí matic parametrů $\boldsymbol{A}_x$, $\boldsymbol{b}_x$, $\boldsymbol{c}_x$, kde $x = \{0, 1, 2, ..., 8, 9\}$. A jejich vložení do programové sady řídicího systému ve formě XML souboru.

Matice jsou vytvořeny na základě volby parametrů spojitých přenosů soustav maximálně 5. stupně. Parametry spojitého přenosu soustavy jsou do programu zadávány přes GUI a jsou ve formě nul, pólů a zesílení soustavy. Program přepočte spojitý popis soustavy na diskrétní stavový a výsledkem jsou matice parametrů $\boldsymbol{A}_x$, $\boldsymbol{b}_x$, $\boldsymbol{c}_x$ se zadaným interval vzorkování, kde $x = \{0, 1, 2, ..., 8, 9\}$, reprezentující nový model.

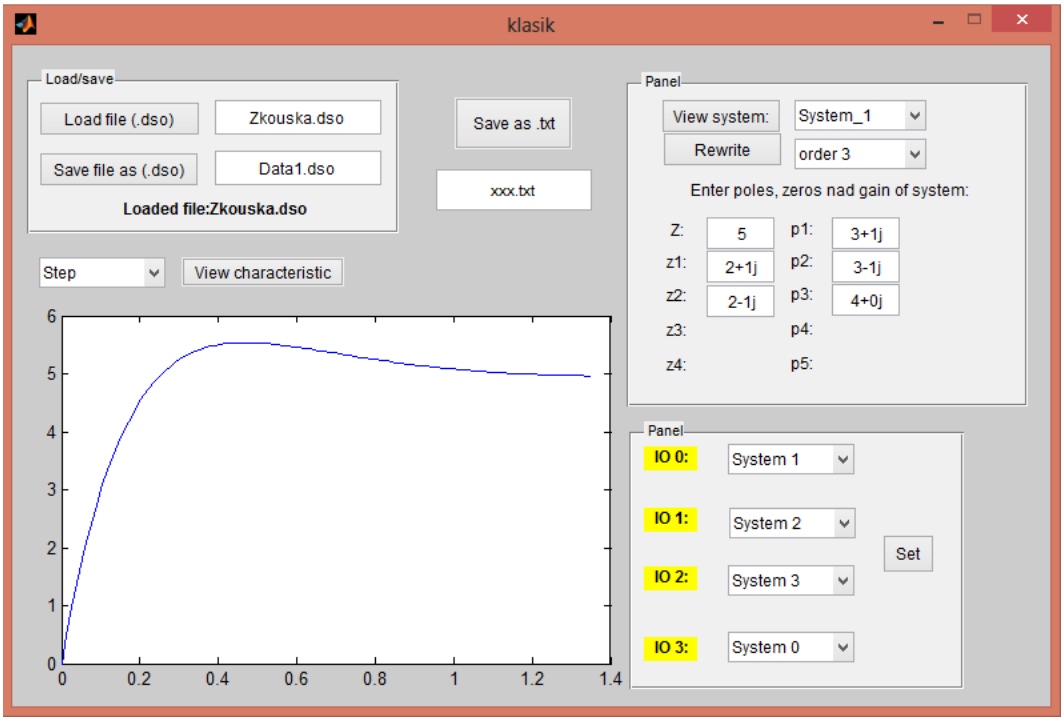
Pro vložení se nejprve načte zdrojový soubor ve formátu XML, reprezentující programovou sadu pro AMiNi4DS a do ní se bude vkládat nově vytvořený model ve formě řetězce znaků. Soubor ve formátu XML je načten jako řetězec. Samotné vložení proběhne přepsáním matic parametrů reprezentující model soustavy. S vložením matic přepíše i rozměry stavových vektorů $\boldsymbol{x}_{x\_1}$, $\boldsymbol{x}_{x\_2}$, $\boldsymbol{x}_{x\_3}$, $\boldsymbol{x}_{x\_4}$, kde v indexu $x = \{0, 1, 2, ..., 8, 9\}$, na základě rozměrů matic parametrů, které souvisí s řádem soustavy.

Po vytvoření nového modelu a jeho vložení do XML souboru, kdy se upravují i rozměry 4 stavových proměnných nabízí GUI možnost uložení XML souboru.

Program dále umožní pomocí GUI zobrazení modelů uložených v nahraném XML souboru. Zobrazení modelů je formou různých typů grafů a pomocí spojitého popisu soustavy. Model je přečten z nahraného XML souboru a následně přepočítán na spojitý popis ve formě kořenů a zesílení soustavy, který lze zobrazit v GUI.

Program umožňuje i přiřazení určitého modelu soustavy, určitému vstupu/výstupu 0 – 3. Jeden model může být i na všech vstupech/výstupech. To že jsou nezávislé, zajistí čtveřice stavových vektorů $\boldsymbol{x}_{x\_1}$, $\boldsymbol{x}_{x\_2}$, $\boldsymbol{x}_{x\_3}$, $\boldsymbol{x}_{x\_4}$, kde v indexu $x = \{0, 1, 2, ..., 8, 9\}$. Přiřazení probíhá přepsáním hodnot proměnných *Ukaz0* až *Ukaz3* v nahraném XML souboru. Další možností programu je uložení všech 10 modelů do textového souboru ve formě kořenů a zesílení soustavy.

Program se stará i o zadávání čísel do editovatelných polí grafického prostředí, tedy kontroluje, zda se do polí zadávají čísla a ne znaky. Hodnoty kořenů mohou být i komplexní, zesílení nikoliv.



Obrázek 5.1 – Grafické uživatelské prostředí vytvořené v Matlab Guide

5.1.1 Tlačítko Load file

Po stisku tlačítka Load file se do proměnné *nazev* uloží řetězec, který je napsán v upravitelném textu edit_Load. Poté se pomocí funkce try() vyzkouší, zda je možné daný soubor nahrát, a to pomocí funkce fileread(*nazev*). Pokud se soubor nepodařilo nahrát, vypíše se do statického textu text_Load chybová hláška.

Pokud bude nahrání souboru úspěšné, uloží se nahraný soubor do uživatelské paměti jako proměnná *data*. Dále se použije čtyřikrát funkce PrectiUkazatel(), která z nahraného souboru, tedy proměnné *data* přečte inicializační hodnoty proměnných *Ukaz0* až *Ukaz3*, které signalizují aktuální nastavení aktivních soustav. Po přečtení se hodnoty uloží do uživatelské paměti a podle nich se nastaví výběry z vysouvacích nabídek popupmenu_IO0 až popupmenu_IO3.

Dále se několikrát použije funkce CtiHodnotu(), která načte z proměnné *data* inicializační hodnoty pro proměnné ***A0*** až ***A9***, ***b0*** až ***b9*** a ***c0*** až ***c9***. Tyto hodnoty se uloží do uživatelské paměti a postupně se pomocí funkce size() s argumentem v podobě matice ***A0*** až ***A9*** zjistí, o jaké řády soustav se jedná.

Poté se v závislosti na řádu pomocí funkce Precti() spočítají kořeny a zesílení soustav 0 až 9 a hodnoty se uloží do uživatelské paměti. Posledními kroky je zapsání hlášky o správném načtení do statického textu edit_Load a aktualizace uživatelských dat.

- **Funkce PrectiUkazatel(*x.data,‘Ukaz0‘*)**

Vstupními parametry funkce je nahraný soubor ve formě proměnné *data* a řetězec definující název proměnné, u které se bude hledat inicializační hodnota, tedy proměnná definující, jaká soustava je v době nahrání souboru aktivní. Výstupním parametrem funkce PrectiUkazatel() je hodnota této proměnné.

Funkce nejdříve vytvoří řetězec *string* s názvem naší proměnné a to spojením řetězců <Variable Name=“, vstupního řetězce samotné funkce a znaku uvozovky. Spojení těchto dvou řetězců a znaku je realizováno funkcí strcat(). Dále použije funkci strfind() pro nalezení začátku proměnné *string* v proměnné *data* reprezentující načtený soubor a načtenou pozici uloží do proměnné *index*. Poté se vytvoří řetězec znaků pro hledání výstupního parametru funkce, jinak řečeno inicializační hodnoty ukazatele na aktivní soustavu, který se nazve *s1*. Vytvoření řetězce *s1* se realizuje použitím proměnné *data*, která se použije pouze od znaku reprezentující proměnnou *index* do konce načteného souboru, což zjednoduší vyhledávání.

Jelikož program DetStudio má rozdílný zápis proměnné inicializovanou na nulu a na ostatní čísla, konkrétně pokud je proměnná inicializována na nulu, tváří se jako proměnná pouze deklarovaná, je nutné v tomto případě rozlišit, zda je aktivní soustava nula či ostatní. Pro toto rozlišení se použije rozdíl v počtu znaků mezi řetězci </Variable> a </InitialValueFormat>. Rozdíl v zápise je vidět na obr. 5.2, kdy proměnná *Ukaz2* je inicializována na hodnotu tři a proměnná *Ukaz3* je inicializována na hodnotu nula. K tomuto rozlišení je potřeba znát polohu již zmíněných řetězců a ta se zjistí dvojnásobným využitím funkce strfind(). Jednou uloží do proměnné ***v*** nalezené pozice řetězce </Variable> a podruhé uloží do proměnné ***t*** nalezené pozice řetězce </InitialValueFormat>. Obě hledání funkce strfind() samozřejmě probíhají v řetězci *s1*.

```
<Variable Name="Ukaz2" Class="Amit.Psp4.Shared.Psp4VariableBase":
  <Wid>2052</Wid>
  <Station>2</Station>
  <Type>dbtInt</Type>
  <Comment>Ukazatel na IO2</Comment>
  <WarmInit>False</WarmInit>
  <InitialValueFormat>0</InitialValueFormat>
  <InitialValue>3</InitialValue>
</Variable>
<Variable Name="Ukaz3" Class="Amit.Psp4.Shared.Psp4VariableBase":
  <Wid>2055</Wid>
  <Station>2</Station>
  <Type>dbtInt</Type>
  <Comment>Ukazatel na IO3</Comment>
  <WarmInit>False</WarmInit>
  <InitialValueFormat>0</InitialValueFormat>
</Variable>
```

Obrázek 5.2 – Ukázka rozdílného zápisu proměnné v programu Detstudio

Nyní funkce vytvoří proměnnou *zdroj*, která je rovna rozdílu proměnných *v*(1) a *t*(1), tedy rozdílu prvního výskytu řetězce </Variable> a prvního výskytu řetězce </InitialValueFormat>. Tento rozdíl funkce použije k rozhodnutí, zda je hledaná

inicializační hodnota ukazatele rovna nule, nebo ostatním číslům reprezentujícím označení soustav. K rozhodnutí je použita funkce if().

Jestliže je hodnota proměnné *zdroj* větší než padesát znaků, znamená to, že ukazatel je inicializován na jiné číslo než je nula. Pokud je počet znaků menší, značí to, že ukazatel je inicializován na nulu.

Funkce PrectiUkazatel() by tedy měla vrátit jako výstupní hodnotu nulu, tedy soustava 0 je aktivní, pokud je proměnná *zdroj* menší než padesát. Ovšem funkce vrátí hodnotu jedna, protože narozdíl od DetStudia se v této práci v Matlabu pracuje se soustavami 1 až 10, z důvodu používání uicontrol popupmenu v GUIDE.

Jestliže je proměnná větší, vytvoří se další proměnná s názvem ***f***, do které se načtou hodnoty výskytu řetězce <InitialValue> v řetězci *s1*. Poté se vytvoří další proměnná, která bude udávat znak reprezentující samotnou inicializační hodnotou, proměnná se bude jmenovat *u*. Tato proměnná je rovna znaku na pozici hodnoty prvního výskytu řetězce <InitialValue> navýšená o hodnotu čtrnáct. Samozřejmě navýšena je pozice, nikoliv znak. Poté už se jenom hodnota znaku *u* převede pomocí funkce str2double() na číslo, navýší se o hodnotu jedna a použije se jako výstup funkce PrectiUkazatel(). Navýšení je ze stejného důvodu jako u prvního případu.

- **Funkce CtiHodnotu(*x.data,'A0'*)**

Vstupními parametry funkce je nahraný soubor ve formě proměnné *data* a řetězec definující název proměnné, u které se bude hledat hodnota nebo více hodnot matice dané názvem. Tato matice reprezentuje buď, matici soustavy ***A***, matici vstupní ***B***, nebo matici výstupů ***C***, které reprezentují soustavy 0 až 9 ve stavovém popisu. Výstupním parametrem funkce CtiHodnotu() je tato matice.

Funkce nejdříve vytvoří proměnné *string*, *index* a *s1*, které se vytvoří totožně jako u funkce PrectiUkazatel(). Poté funkce potřebuje přečíst rozměry zadané matice, tedy počet řádků a sloupců. Nejprve funkce načte počet řádků, vytvoří tedy proměnnou ***k***, do které se uloží pozice výskytů řetězce <Row> pomocí funkce strfind(). Všechna hledání probíhají samozřejmě v řetězci *s1*. Pak se vytvoří další proměnná, která bude udávat znak reprezentující samotnou inicializační hodnotou, reprezentující počet řádků matice, proměnná se bude jmenovat *row*. Tato proměnná je rovna znaku na pozici hodnoty prvního výskytu řetězce <Row> navýšená o hodnotu pět. Samozřejmě navýšena je pozice, nikoliv znak. Poté se hodnota znaku *row* převede pomocí funkce str2double() na číslo a uloží se do proměnné *r*. Stejným způsobem, jen za využití jiných proměnných a řetězce <Col> namísto <Row>, se přečte počet sloupců zadané matice a uloží se do proměnné *c*.

Dalším krokem funkce CtiHodnotu() je rozhodnutí, zda matice kterou bude číst je inicializovaná na nuly, případně nulu nebo na jiná čísla. Proč se toto řeší je vysvětleno v popisu funkce PrectiUkazatel() a je to částečně ukázáno na obr. 5.2. Rozhodnutí se řeší obdobně jako u funkce PrectiUkazatel(), vytvoří se proměnná *zdroj*, ale pro rozdíl pozic řetězců jsou použity řetězce </Variable> a <Col> a rozhodnutí pomocí funkce if() je také jiné. Pokud je hodnota proměnné *zdroj* vyšší než číslo osmnáct, není matice nulová a musí se přečíst.

Pokud je ale menší než osmnáct, vytvoří se nulová matice o počtu řádků dle proměnné *r* a počtu sloupců dle proměnné *c* a pošle se na výstup funkce CtiHodnotu(). Vytvoření nulové matice zajistí funkce zeros().

Jestliže se nejedná o nulovou matici, najde se pomocí funkce strfind() začátek řetězce <InitialValue> a přičte se k němu hodnota čtrnáct reprezentující délku tohoto řetězce. Tím se získá ukazatel na začátek řetězce s hodnotou nebo hodnotami matice a pojmenuje se *zac*. Ještě se musí najít konec tohoto řetězce, ten se nejde opět pomocí

funkce strfind(), kdy hledaným řetězcem je </InitialValue> a od nalezené hodnoty se odečte hodnota jedna. Tato pozice se nazve *kon*. Nyní se vytvoří řetězec reprezentující hodnotu nebo hodnoty matice, a to použitím řetězce *s1* v rozmezí hodnot *zac* a *kon* a pojmenuje se *hodnoty*. Dále je potřeba nalézt oddělovač jednotlivých hodnot, v tomto případě je to středník, což se provede funkcí strfind() v řetězci *hodnoty* a nalezené pozice se uloží do proměnné *n*. Dále definujeme pomocné proměnné *index1* a *index2* a inicializujeme je na nulu.

```
<Variable Name="A2" Class="Amit.Psp4.Shared.Psp4VariableBase">
  <Wid>2038</Wid>
  <Station>2</Station>
  <Type>dbtFloatMtx</Type>
  <Comment>matice A soustavy 2 (20 ms)</Comment>
  <WarmInit>False</WarmInit>
  <InitialValueFormat>0</InitialValueFormat>
  <Row>3</Row>
  <Col>3</Col>
  <InitialValue>2.882753;-1.384859;0.4434602;2;0;0;0;1;0;</InitialValue>
</Variable>
```

Obrázek 5.3 – Ukázka uložení proměnné ***A2*** ve zdrojovém souboru

V programu DetStudio se hodnoty matic zapisují podle řádků matice, tedy nejdříve se uvedou hodnoty prvního řádku, poté dalšího a tak dále. Samotné čtení hodnot matice probíhá pomocí dvou cyklů s opakováním for. První cyklus se opakuje od jedné do počtu řádků matice, což reprezentuje proměnná *r*. A druhý cyklus vložený do prvního se opakuje od jedné, do počtu sloupců matice, což reprezentuje proměnná *c*. Cyklus postupně vyhledá všechny čísla oddělená středníkem a zapíše je do výstupní matice. Cykl je zobrazen na obr. 5.4. Výstupem funkce CtiHodnotu() je tedy matice hodnot o počtu řádků *r* a počtu sloupců *c*.

```
index1=0;
index2=0;

for rr=1:r,
    for cc=1:c,
        index1=index1+1;
        y=n(index1);
        m=s1(zac+index2:zac+y-2);
        index2=index2+length(m)+1;
        mm=str2double(m);
        A(rr,cc)=mm;
    end
end
```

Obrázek 5.4 – Ukázka řešení for cyklů ve funkci CtiHodnotu()

- **Funkce Precti(*A*,*B*,*C*,*rad*)**

Vstupními parametry funkce jsou matice ***A***, ***B*** a ***C*** reprezentující stavový popis dané soustavy a řád daného systému. Výstupními parametry funkce Precti() je zesílení a kořeny dané soustavy, které se vypočtou z vstupních matic funkce.

Nejprve se inicializuje matice ***D*** na hodnotu nula a použije se funkce ss() s periodou vzorkování 0,02 s, tím se vytvoří objekt reprezentující diskrétní stavový popis soustavy. Následně se tento stavový popis převede funkcí tf() na popis vnější diskrétní. A poté se vnější diskrétní popis převede funkcí d2c() na vnější spojitý popis.

Následně proběhne pomocí funkce if() s argumentem *rad* rozhodnutí, kolik kořenů se bude počítat a jakým způsobem. Použije se funkce zpkdata() s vstupními argumenty v podobě spojité soustavy a řetězce *v*, který říká, že se jedná o SISO soustavu. Výstupní parametry funkce jsou tři, a to dva vektory reprezentující nuly a póly a hodnota zesílení *K*. Z vektorů reprezentující nuly a póly se v závislosti na řádu systému uloží daný počet kořenů do uživatelské paměti, ale je potřeba změnit jejich znaménko na kladné. Zesílení je nutno přepočítat, a to úpravou přenosu zapsaného pomocí nul, pólů a zesílení systému.

Na rovnici 3.1 je zobrazen přenos systému druhého řádu zapsaný pomocí nul, pólů a zesílení systému, pokud se použije funkce zpkdata() je nutné zesílení dopočítat, a to podle rovnice 3.2.

$$G(s) = \frac{K(s+z_1)}{(s+p_1)(s+p_2)} \frac{p_1\, p_2}{z_1} \tag{3.1}$$

$$Z = K \frac{z_1}{p_1\, p_2} \tag{3.2}$$

Po výpočtu zesílení *Z* a daného počtu kořenů se zbývající nuly a póly položí rovny hodnotě 0 a pomocí funkce sprintf() s argumenty označující šířku pole znaků a přesnost se hodnota zesílení uloží do výstupní proměnné. Stejným způsobem, jen za pomoci dalších funkcí jako jsou real() a imag() se uloží do výstupních proměnných i hodnoty kořenů. Zápis je ukázán na obr. 5.5.

```
elseif rad == 2
[zz1,P,K]=zpkdata(S,'v');
z1=-zz1;p2=-P(1);p1=-P(2);
Z=(K*z1)/(p1*p2);
p3=0;p4=0;p5=0;z2=0;z3=0;z4=0;
p1=sprintf('%2.8g%+2.8gj',real(p1),imag(p1));
p2=sprintf('%2.8g%+2.8gj',real(p2),imag(p2));
z1=sprintf('%2.8g',z1);
Z=sprintf('%2.8g',Z);
```

Obrázek 5.5 – Výpis kořenů a zesílení soustavy 2. řádu do editovatelných polí GUI

5.1.2 Tlačítko View System

Po stisku tlačítka View se do proměnné *systém* uloží hodnota z popupmenu_Edit, reprezentující vybranou soustavu 0 – 9 a pomocí funkce switch s argumentem *system* se rozhoduje, o jakou soustavu se jedná. Dále se pomocí funkce size() s argumentem matice ***A*** zjistí řád systému a uloží se do proměnné *r*. Následuje použití funkce Zobraz(), která v závislosti na řádu systému zviditelní daný počet editovatelných polí a nastaví hodnotu řádu na vysouvací nabídce popupmenu_Rad dle proměnné *rad*. Posledními kroky je vypsání hodnot kořenů a zesílení z uživatelské paměti do editovatelných polí.

- **Funkce Zobraz(*rad, handles.edit_*p_2**…**)**

Vstupními parametry funkce je řád systému a proměnné odkazující se na jednotlivá editovatelná pole. Pole pro zesílení *Z* a pro pól p_1 jsou zobrazeny stále, takže se jich tato funkce netýká. Výstupní parametry funkce nemá.

Funkce Zobraz() nejdříve vyhodnotí o jaký řád systému se jedná pomocí funkce switch s argumentem *rad*, a poté zviditelní daný počet editovatelných polí v závislosti na řádu.

5.1.3 Tlačítko Rewrite

Po stisku tlačítka Rewrite se nejdříve přečte řád systému z vysouvací nabídky popupmenu_Rad a o jakou soustavu se jedná z vysouvací nabídky popup_Edit. Dalším krokem je přečtení řetězců zapsaných v editovatelných polích pro zadávání zesílení a kořenů soustavy a jejich převedení na čísla pomocí funkce str2double(). Poté se podle soustavy 0 až 9 vypočtou matice ***A***, ***B*** a ***C*** stavového popisu pomocí funkce SISO() a uloží se do uživatelské paměti.

Následuje trojnásobné použití funkce ZapisMatici(), které přepíše hodnoty matic ***A***, ***B*** a ***C*** v načteném souboru *data* našimi maticemi vypočtenými a šestinásobné použití funkce ZapisRozmery(), která upraví rozměry čtyřem stavovým a dvěma pomocným stavovým vektorům. Jeden stavový vektor pro použití na jednom vstupu/výstupu 0 až 3.

- **Funkce [*A,B,C,D*]=SISO(*Z,*p_1*,*p_2*,* p_3*,*p_4*,* p_5*,*z_1*,*z_2*,* z_3*,*z_4*,rad*)**

Vstupními parametry funkce je řád, zesílení a kořeny systému. Výstupními parametry jsou matice ***A***, ***B***, ***C*** a ***D*** reprezentující stavový popis systému.

Nejprve funkce SISO() rozhodne pomocí funkce if() s parametrem *rad*, o jaký řád systému se jedná, a podle toho pomocí funkce tf() vytvoří objekt reprezentující vnější spojitý popis systému. Vytvoření tohoto systému odpovídá rovnicím 3.3 až 3.7, kde dolní index názvu soustavy odpovídá řádu systému.

$$G_1(s) = \frac{Z\, p_1}{s + p_1} \tag{3.3}$$

$$G_2(s) = \frac{Z\,(s + z_1)}{(s + p_1)(s + p_2)} \frac{p_1\, p_2}{z_1} \tag{3.4}$$

$$G_3(s) = \frac{Z\,(s + z_1)(s + z_2)}{(s + p_1)(s + p_2)(s + p_3)} \frac{p_1\, p_2\, p_3}{z_1\, z_2} \tag{3.5}$$

$$G_4(s) = \frac{Z(s+z_1)(s+z_2)(s+z_3)}{(s+p_1)(s+p_2)(s+p_3)(s+p_4)} \frac{p_1\, p_2\, p_3\, p_4}{z_1\, z_2\, z_3} \tag{3.6}$$

$$G_5 s) = \frac{Z(s+z_1)(s+z_2)(s+z_3)(s+z_4)}{(s+p_1)(s+p_2)(s+p_3)(s+p_4)(s+p_5)} \frac{p_1\, p_2\, p_3\, p_4\, p_5}{z_1\, z_2\, z_3\, z_4} \tag{3.7}$$

Následně pomocí funkce c2d() s parametrem 0,02 s, reprezentujícím vzorkovací frekvenci,přepočítá vnější popis spojitý na vnější popis diskrétní. Dále pomocí funkce ss() přepočte vnější diskrétní popis na vnitřní diskrétní ve formě stavového popisu. Tento popis vrátí pomocí funkce ssdata() matice ***A***, ***B***, ***C*** a ***D*** a ty se stanou výstupními parametry funkce SISO().

- **Funkce ZapisMatici(*x.data*,'A3',*A3*)**

Vstupními parametry funkce je nahraný soubor ve formě proměnné *data*, řetězec definující název proměnné, u které se budou přepisovat inicializační hodnoty a samotná matice s hodnotami, kterou se bude přepisovat původní matici. Tato matice reprezentuje buď, matici soustavy ***A***, matici vstupní ***B***, nebo matici výstupů ***C***, které reprezentují soustavy 0 až 9 ve stavovém popisu. Matice ***D*** bude vždy nulová, takže se o ni funkce nestará. Výstupním parametrem funkce ZapisMatici() je upravený řetězec *data* reprezentující nahraný soubor.

Funkce nejdříve přečte pomocí funkce [*r*,*s*]=size() rozměry vstupní matice, kterou bude zapisovat. Do proměnné *r* uloží počet řádků a do proměnné *s* uloží počet sloupců. Poté je potřeba, proměnné *r* a *s* převést na znaky. Do proměnných *row* a *col* uloží funkce ZapisMatici() pomocí funkce sprintf() počet řádků a sloupců jako řetězec, v tomto případě jenom jako jeden znak. Následně použije dvakrát funkci s opakováním for(), ve které do řetězce *hodnoty* uloží všechny hodnoty nové matice oddělené středníkem, přesně tak, jak to vyžaduje program DetStudio, což je vidět na obr 5.6. Nyní funkce disponuje rozměry i hodnoty nové matice v příslušném stavu.

```
hodnoty='';
for radky=1:r,
    for sloupce=1:s,
        hodnota=sprintf('%8.15f;',matice(radky,sloupce));
        hodnoty=strcat(hodnoty,hodnota);
    end
end
```

Obrázek 5.6 – Uložení hodnot nové matice, ve formě řetězce, do proměnné *hodnoty*

Následně funkce vytvoří proměnné *string*, *index* a *s1*, které se vytvoří totožně jako u funkce PrectiUkazatel() nebo CtiHodnotu(). Navíc se vytvoří řetězec *s2*, jehož vytvoření se realizuje použitím proměnné *data*, která se použije pouze od prvního znaku po znak reprezentující proměnnou *index*. Řetězec *s2* zjednoduší vyhledávání a bude použit při skládání výstupního řetězce funkce ZapisMatici().

Poté funkce potřebuje změnit rozměry zadané matice, tedy počet řádků a sloupců, na nové. Nejprve funkce musí najít pozici znaku reprezentující počet řádků matice.

Nalezení znaku proběhne vytvořením proměnné ***k***, do které se uloží pozice výskytů řetězce <Row>, pomocí funkce strfind(). Pak se najde pozice znaku, reprezentující samotnou inicializační hodnotou počtu řádků. Tato pozice je rovna pozici znaku prvního výskytu řetězce <Row> navýšená o hodnotu pět. Samozřejmě navýšena je pozice, nikoliv znak. Poté se přepíše znak na nalezené pozici číslem respektive znakem uloženým v proměnné *row*. Stejným způsobem, jen za využití jiných proměnných a řetězce <Col> namísto <Row>, se přepíše i počet sloupců zadané matice. Všechna hledání probíhají samozřejmě v řetězci *s1*.

Nyní je potřeba vyřešit zápis řetězce, reprezentující nové hodnoty matice, na správné místo v řetězci *s1*. Opět je zde problém v rozdílném zápisu matice inicializované na nuly, případně nulu nebo na jiná čísla. Proč se toto řeší je vysvětleno v popisu funkce PrectiUkazatel() a je to částečně ukázáno na obr. 5.2. Rozhodnutí se řeší obdobně jako u funkce PrectiUkazatel(), vytvoří se proměnná *zdroj*, ale pro rozdíl pozic řetězců jsou použity řetězce </Variable> a <Col> a rozhodnutí pomocí funkce if() je také jiné.

Pokud je hodnota proměnné *zdroj* vyšší než číslo osmnáct, není matice nulová a je nutno řetězec reprezentující nové hodnoty zapsat mezi již existující řetězce <InitialValue> a </InitialValue>. Pro tento zápis je nutné řetězec *s1* rozdělit na řetězec se znaky před původními inicializačními hodnotami a řetězec za znaky s původními inicializačními hodnotami. Tedy najít pozici posledního znaku řetězce <InitialValue> a pozici prvního znaku řetězce </InitialValue>. Nejprve se tedy najdou pomocí funkce strfind() všechny pozice řetězců <InitialValue> a </InitialValue> a uloží se ve stejném pořadí do proměnných ***u*** a ***t***. Pozice posledního znaku <InitialValue> se vypočte přičtením hodnoty třináct k prvnímu výskytu řetězce <InitialValue> a pozice prvního znaku řetězce </InitialValue> se získá z indexu prvního výskytu tohoto řetězce, tedy z proměnné ***t***.

Následně proběhne spojení výstupního řetězce, kdy se pomocí funkce strcat() spojí řetězec *s2* s řetězcem *s1* použitým od jeho začátku po hodnotu reprezentující poslední znak řetězce<InitialValue>. Dále se tento řetězec spojí s řetězcem *hodnoty* reprezentující nové hodnoty matice a za něj se připojí řetězec *s2* použitý od znaku reprezentující začátek řetězce </InitialValue> až po konec řetězce *s2*. Tímto proběhne přepsání hodnot matice a přepsaný řetězec *data* je poslán na výstup funkce ZapisMatici().

Pokud je ale proměnná *zdroj* menší než osmnáct, tak v deklaraci matice v nahraném souboru *data* chybí řádek s řetězci <InitialValue> a </InitialValue>, což je vidět na obr. 5.2. Je tedy nutné tyto řetězce vytvořit a do nahraného souboru *data* vložit na správné místo. K vložení řetězců je potřeba rozdělit řetězec *s1* na část před obvyklým výskytem řetězce <InitialValue> a část po obvyklém výskytu řetězce </InitialValue>.

První část bude končit na pozici reprezentující znak prvního výskytu řetězce </Col> posunutého o jistou hodnotu. Posunutí proběhne tak, že se k této pozici přičte hodnota osmnáct a výsledek se uloží do proměnné *zac*. Toto posunutí způsobí, že proměnná *zac* ukazuje na znak, který se nachází na prvním místě řádku pod řetězcem

Druhá část bude začínat na pozici reprezentující znak prvního výskytu řetězce </Col> posunutého o jistou hodnotu, tedy stejně jako první část, ale tato bude posunuta o hodnotu jinou. Posunutí se uskuteční přičtením hodnoty šest a výsledná pozice se uloží do proměnné *kon*. Proměnná *kon* ukazuje na pozici reprezentující znak hned za řetězcem </Col>, respektive na prázdné místo za řetězcem.

Nyní proběhne spojení výstupního řetězce funkce ZapisMatici(). Pro spojení se použije funkce strcat() a postupně se spojí řetězec *s2* s řetězcem *s1* použitým od jeho začátku po proměnnou *zac*, reprezentující předtím popsanou pozici v řetězci. Tento řetězec se dále spojí s řetězcem <InitialValue>, který se zapíše na první pozici řádku pod řetězcem

</Col> a k němu se připojí řetězec *hodnoty* reprezentující nové hodnoty matice. Za tyto nové hodnoty se připojí řetězec </InitialValue> a na nový řádek řetězec *s1* použitý od proměnné *kon*, reprezentující předtím vysvětlenou pozici znaku, po konec řetězce. Tímto spojením jsme vytvořili řetězec reprezentující řetězec *data* s přidanými novými hodnotami matice a tento řetězec se stává výstupním argumentem funkce ZapisMatici().

- **Funkce ZapisRozmery(*x.data*,'x9_3',*rad*)**

Vstupními parametry funkce je nahraný soubor ve formě proměnné *data*, řád systému a řetězec definující název proměnné, u které se budou přepisovat rozměry, tedy počet řádků a sloupců. Přepisují se rozměry stavovým nebo pomocným stavovým vektorům. Výstupem funkce je upravený řetězec *data* reprezentující nahraný soubor.

Funkce ZapisRozmery() nejdříve vytvoří proměnné *string, index,s1* a *s2*, které se vytvoří totožně jako u funkce ZapisMatici(). Navíc se vytvoří znak *stupen* vytvořený použitím funkce num2str() s argumentem *rad*, která reprezentuje nový počet řádků stavového nebo pomocného stavového vektoru. Počet sloupců těchto stavových vektorů je u SISO systémů vždy roven hodnotě jedna, což je vidět na rovnici 1.19.

Nyní se pomocí funkce strfind() v řetězci *s1* naleznou pozice znaků reprezentujících počet řádku a sloupců daného vektoru a přepíší se. Znak udávající počet řádků se přepíše znakem *stupen* a znak reprezentující počet sloupců se přepíše znakem čísla jedna. Způsob práce s funkcí strfind() a čtení pozice počtu řádků a sloupců je pospán u funkce CtiHodnotu(). Následuje už pouze vytvoření výstupního řetězce. Funkcí strcat() se spojí řetězec *s2* s upraveným řetězcem *s1* a výsledný řetězec je výstupním argumentem funkce ZapisRozmery().

5.1.4 Tlačítko Set

Po stisku tlačítka Set se nejdříve přečtou vybrané volby na vysouvacích nabídkách popupmenu_IO0 – popupmenu_IO3, tedy nové nastavení aktivních soustav na vstupech/výstupech 0 – 3 a uloží se do proměnných *IO1 – IO3*. Poté se čtyřikrát použije funkce ZapisUkazatel(), která zapíše nový stav aktivních soustav do načteného souboru *data*.

- **Funkce ZapisUkazatel(*x.data*,'Ukaz3',*IO3*-1)**

Vstupními parametry funkce je nahraný soubor ve formě řetězce *data*, dekrementovaná proměnná reprezentující číslo nové aktivní soustavy a řetězec definující název proměnné, u které se bude přepisovat její inicializační hodnota.Výstupem funkce je upravený řetězec *data*, reprezentující nahraný soubor. Vstupní proměnná se dekrementuje z důvodu rozdílného značení soustav v programu Matlab a v grafickém prvku uicontrol popupmenu.

Funkce ZapisUkazatel() nejdříve vytvoří proměnné *string, index,s1* a *s2*, které se vytvoří totožně jako u funkce ZapisMatici(). Navíc se vytvoří proměnná jménem *soustava*, která dekrementovanou vstupní proměnnou převede na znak funkcí num2str(). Následuje podobný postup jako v případě předešlých funkcí. Pro odlišení soustavy 0 a soustav 1 – 9 se využije vzdálenost mezi řetězci <InitialValueFormat> a </Variable>.

V případě soustavy 0 je nutné vytvořit řetězce <InitialValue> a </InitialValue>a vložit je spolu s novou inicializační hodnotou do řetězce s nahraným souborem.

Pokud byl ukazatel inicializován na soustavy 1 – 9, přepíše se inicializační hodnota ukazatele na hodnotu proměnné *soustava*.

5.1.5 Tlačítko Characteristic

Po stisku tlačítka button_Characteristic se přečte, jaký systém se má zobrazit na grafu axes z popupmenu popup_Edit a o jaký typ grafu se jedná, což se přečte z popupmenu popup_Char. Dále se vytvoří systém popsaný stavovým popisem, a to využitím matic $\boldsymbol{A}$, $\boldsymbol{B}$, a $\boldsymbol{C}$ daného systému přečtených pomocí tlačítka Load nebo vytvořených pomocí tlačítka Rewrite. K vytvoření stavového popisu se použije funkce ss() u které se ještě inicializuje matice $\boldsymbol{D}$ na nulu a perioda vzorkování na 0,02 s.

Vytvořený stavový popis soustavy se převede pomocí funkce tf() na popis vnější diskrétní a pomocí funkce d2c() na popis vnější spojitý. Dále se pomocí funkce switch() rozhodne, jaký graf se má zobrazit a pomocí funkcí step() a plot(), nyquist(), bode() nebo pzmap() se graf zobrazí na grafu axes.

5.1.6 Tlačítko Save file as

Po stisku tlačítka button_Save se do proměnné *nazev* uloží řetězec, který je napsán v upravitelném textu edit_Save. Proměnná *nazev* reprezentuje název souboru typu dso, do kterého se uloží upravený řetězec *data*. Poté se pomocí funkce fopen() vytvoří nebo otevře tento soubor jehož jméno je dáno touto proměnnou. A následně se do tohoto čistého souboru zapíše pomocí funkce fprintf() upravený řetězec *data* a soubor se zavře pomocí funkce fclose().

5.1.7 Tlačítko Save as txt

Po stisku tlačítka Save as txt se do proměnné *nazev* uloží řetězec, který je napsán v upravitelném textu edit_TXT. Následně se vytvoří nebo otevře soubor s názvem daným proměnnou *nazev* pomocí funkce fopen(). A pomocí funkce fprintf() vypíšeme do textového souboru spojité popisy všech 10 soustav uložených v nahraném XML souboru.

5.1.8 Editovatelná pole pro zadávání zesílení a kořenů přenosů

Pole jsou nastavena tak, že při změně řádu na popupmenu_Rad se automaticky aktualizuje počet zobrazených polí.

Dále u pole reprezentující zesílení přenosu se řetězec obsažený v poli převede pomocí funkce str2double na číslo a pomocí funkce if() se testuje zda je zadané číslo reálné. Pokud reálné není, vloží se do pole hodnota nula. Pomocí funkce if() se testuje, i zda není pole prázdné, pokud je, vloží se do něj opět hodnota nula. A posledním testem s využitím funkce if() se testuje, zda je zadaný řetězec číslo, pokud není, opět se nastaví do pole hodnota nula.

Pro zadávání kořenů se testují stejné podmínky, jen kořeny můžou být i komplexní.

Závěr

Cílem bylo vytvoření hardwarového simulátoru 4 paralelně běžících nezávislých dynamických SISO systémů na AMiNi4DS s online výběrem modelů systému přes program MATLAB s rozhraním GUI.

Byla vytvořena sada programů pro řídicí systém AMiNi4DS, která v periodickém opakování vypočítává výstupy řídícího systému v závislosti na aktuálních hodnotách vstupů systému a vložených modelech soustav. Dále realizuje zobrazení vybraných údajů na LCD zobrazovači.

Výpočet se realizuje pomocí diskrétního lineárního stavového modelu. Pro naprogramování výpočtu je zapotřebí návrhové prostředí, které podporuje maticový počet a typ strukturovaného programovacího jazyka. Pro výpočet modelů byly vytvořeny maticové proměnné s domluvenými názvy a nulovými inicializačními hodnotami. Dále byly vytvořeny 4 proměnné, jejichž hodnota reprezentuje vybraný model na daném vstupu/výstupu 0 – 3. Tyto ukazatele jsou inicializovány na libovolné číslo 0 – 9. Plnění matic novými soustavami probíhá přes MATLAB.

Na LCD zobrazovači se při startu systému objeví obrazovka Menu, a z ní je možnost pomocí kláves řídicího systému výběr dalších podobrazovek. Lze zobrazit informace o řídicím systému, hodnoty analogových a číslicových vstupů i výstupů a obrazovku s volbou výběru použitých modelů na vstupech/výstupech 0 – 3. Pro obrazovky byly tedy vytvořeny proměnné používající se při obsluze obrazovek.

Plnění stavových matic jednotlivých modelů v programové sadě AMiNi4DS probíhá přes grafické uživatelské prostředí programu MATLAB. Kromě vytvoření a zápisu nových matic lze v GUI zobrazit i modely již vytvořené a uložit je do textového souboru.

Řešil jsem možnost, zvolit si jeden, ze tří typů, programovacího jazyka v prostředí DetStudio, prvním je strukturovaný text, druhým je jazyk podobný assembleru, který pracuje s vrcholem zásobníku a posledním je jazyk reléových schémat. Pro tento typ úlohy lze využít pouze jazyka strukturovaného.

Počet dynamických systémů v programové sadě jsem zvolil deset, s názvem 0 – 9, z důvodu přehlednosti a ulehčení práce s řetězci. K popisu chování dynamických soustav má sloužit diskrétní stavový popis systému s minimální dosažitelnou periodou vzorkování. Z čehož vyplývá, že procesy budou obsahovat maticové proměnné soustav 0 – 9 s jednoznačnými názvy a nulovými inicializačními hodnotami, které budou reprezentovat matice diskrétních stavových popisů.

Minimální bezpečnou vzorkovací periodu jsem odzkoušel při simulaci chování 4 soustav 5. řádu , tedy při nejsložitějším výpočtu pro řídicí systém. Minimální vzorkovací periodu jsem zvolil 20 ms.

Přes LCD zobrazovač jsem zvolil možnost zobrazit údaje o samotném řídícím systému, vstupech, výstupech a volbu výběru modelu použitého na vstupu/výstupu 0 – 3.

Zadávat nové soustavy lze mnoha druhy popisu, jelikož nejvíce se dají soustavy odhadnout z přenosu zapsaného ve tvaru nul, pólů a zesílení využil jsem tento způsob.

Pomocí programu s uživatelským rozhraním je tedy nutné načíst XML soubor s uloženými výchozími hodnotami a hodnoty uložit ho do uživatelské paměti. Načtený soubor bude k dispozici ve formě řetězce. Nyní bude zapotřebí umožnit přes grafické rozhraní zadání nové soustavy ve formě nul, pólů a zesílení systému, maximální řád soustav jsem zvolil pátý. Po zadání je nutné tento popis přepočítat na stavový a přepsat inicializační hodnoty zadané soustavy v načteném řetězci. Ovšem je nutné změnit i rozměry všech stavových matic a všech jejich přidružených stavových vektorů, dle řádu

systému. Poté je zapotřebí upravený řetězec uložit do souboru a pomocí prostředí DetStudio nahrát do řídicího systému Takto by bylo možné zadávání nových soustav.

Ovšem pro uživatelské pohodlí by mělo grafické rozhraní umožnit prohlížení všech soustav v nahraném souboru, ať už původních nebo nově vložených. Dále by měla být k dispozici možnost nastavení libovolných modelů soustav na vstupy/výstupy 0 – 3 a samozřejmě přehled o zvolených soustavách. Určitě by neměla chybět možnost zobrazit nejpoužívanější popisové charakteristiky systému v grafu a uložení popisu všech soustav do textového souboru pro potřebu uživatele. Všechny tyto vlastnosti moje grafické prostředí nabízí.

Do budoucna by bylo vyhovující doplnit do GUI podrobnější a přehlednější výpis soustav do textového souboru. Dále přidat textová pole informující o stavu prováděné operace a v programové sadě pro řídicí systém vytvořit více pomocných podobrazovek. Zapotřebí je i určité doplnění programových výjimek v programové sadě MATLABu.

Hardwarový simulátor AMiNi4DS simuluje chování reálných soustav takovým způsobem, že od teoretického chování se liší jen minimálně, což je vidět v příloze.

Literatura

[1] SROVNAL, Vilém. *Kybernetika.* [online]. Ostrava: Vysoká škola Báňská, 2008, 250 s. [cit. 5. 5. 2013]. Dostupné z: http://skola.spectator.cz/5_SEMESTR/Kybernetika/Kyb%E8a.pdf.

[2] HUBÁČEK, Jiří. Analýza. In: *Vnější popis.* [online]. Zlín: Univerzita Tomáše Bati, 30. 11. 2007. [cit. 5. 5. 2013]. Dostupné z: http://195.178.89.122/CAAC_PHP/CAAC/cesky/analyza/s_vnejpop/s_vnejpop.php.

[3] ROUBAL, Jiří. *Teorie dynamických systémů: Nuly a póly systému.* [online]. Praha: ČVUT. Fakulta elektrotechniky, 2006, 8 s., 31.01.2007. [cit. 5. 5. 2013]. Dostupné z: https://support.dce.felk.cvut.cz/pub/roubalj/teaching/TDS/seminars/TDS_cv7_npss.pdf.

[4] ČIRKA, L'uboš. *Póly a nuly prenosových funkcií systémov.* [online]. Bratislava: Slovenská technická univerzita. Fakulta chemickej a potravinárskej technológie, 2006, 13 s., 26.03.2012. [cit. 5. 5. 2013]. Dostupné z: http://www.kirp.chtf.stuba.sk/~cirka/vyuka/lcza/pdf//chappn.pdf

[5] KUPKA, Libor. *Frekvenční charakteristiky.* [online]. Liberec: Technická univerzita. Fakulta mechatroniky, informatiky a mezioborovýh studií, 2008, 18 s., 29. 1. 2009. [cit. 5. 5. 2013]. Dostupné z: http://www.fm.tul.cz/~libor.kupka/Frekvencni_charakteristiky.pdf.

[6] SKALICKÝ, Jiří. *Teorie řízení.* [online]. 2. vyd. Brno: Vysoké učení technické. Fakulta elektrotechniky a komunikačních technologií, 2002, 98 s. [cit. 5. 5. 2013]. Dostupné z: http://www.vutbr.cz/www_base/priloha.php?dpid=66661.

[7] NAVRÁTIL, Pavel. Analýza. In: *Vnitřní popis*. [online]. Zlín: Univerzita Tomáše Bati, 6. 8. 2008. [cit. 5. 5. 2013]. Dostupné z: http://195.178.89.122/CAAC_PHP/CAAC/cesky/analyza/s_vnitpop/s_vnitpop.php.

[8] AMIT, spol. s r. o. *AMiNi4DS: Kompaktní řídicí systém s rozhraním Ethernet: Návod na obsluhu.* [online]. Praha, 2009, 34 s. [cit. 3. 5. 2013]. Dostupné z: http://www.amit.cz/docs/cz/amini/amini4ds_g_cz_100.pdf.

[9] AMIT, spol. s r. o. *DetStudio: Průvodce první aplikací: Návod na obsluhu.* [online]. Praha, 2011, 70 s. [cit. 5.5. 2013]. Dostupné z: http://www.amit.cz/docs/cz/sw/detstudio_g_cz_104.pdf.

[10] HUMUSOFT s.r.o.. *MATLAB*: *Jazyk pro technické výpočty.* [online]. 1991-2013 [cit. 5. 5. 2013]. Dostupné z: http://www.humusoft.cz/produkty/matlab/matlab/.

[11] BLAŠKA, Jan, Michal KRUMPHOLC a Miloš SEDLÁČEK. MATLAB A GUI. In: *Využití grafického uživatelského rozhraní MATLABu ve výzkumu a výuce měření.* [online]. Praha, 2003. [cit. 2013-04-29]. Dostupné z: http://dsp.vscht.cz/konference_matlab/matlab03/blaska.pdf.

[12] DUŠEK, František. *Tvorba grafického prostředí v MATLABu.* [online]. Pardubice: Univerzita Pardubice. Fakulta elektrotechniky a informatiky, 2011, 13 s. [cit. 5. 5. 2013]. Dostupné z: http://fei-learn.upceucebny.cz/mod/resource/view.php?id=3832.

Příloha A – Zobrazení chování vybraných soustav

Bylo simulováno chování dvou reálných modelů soustav. V grafech je porovnáno teoretické a praktické chování modelu. Teoretické chování bylo nasimulováno v SIMULINKU a praktické chování bylo naměřeno na fyzických výstupech simulátoru AMiNi4DS.

Model 0:

$$G_S = \frac{Z\,(s+z_1)}{(s+p_1)(s+p_2)}\,\frac{p_1\ p_2}{z_1}, \quad \text{kde } Z=4,\ p_1=0.6,\ p_2=8,\ z_1=2$$

$$G_S = \frac{4\,(s+2)}{(s+0.6)(s+8)}\,\frac{0.6\cdot 8}{2}$$

$$G_S = \frac{9.6\,s}{s^2+8.6\,s+4.8}$$

Model 1:

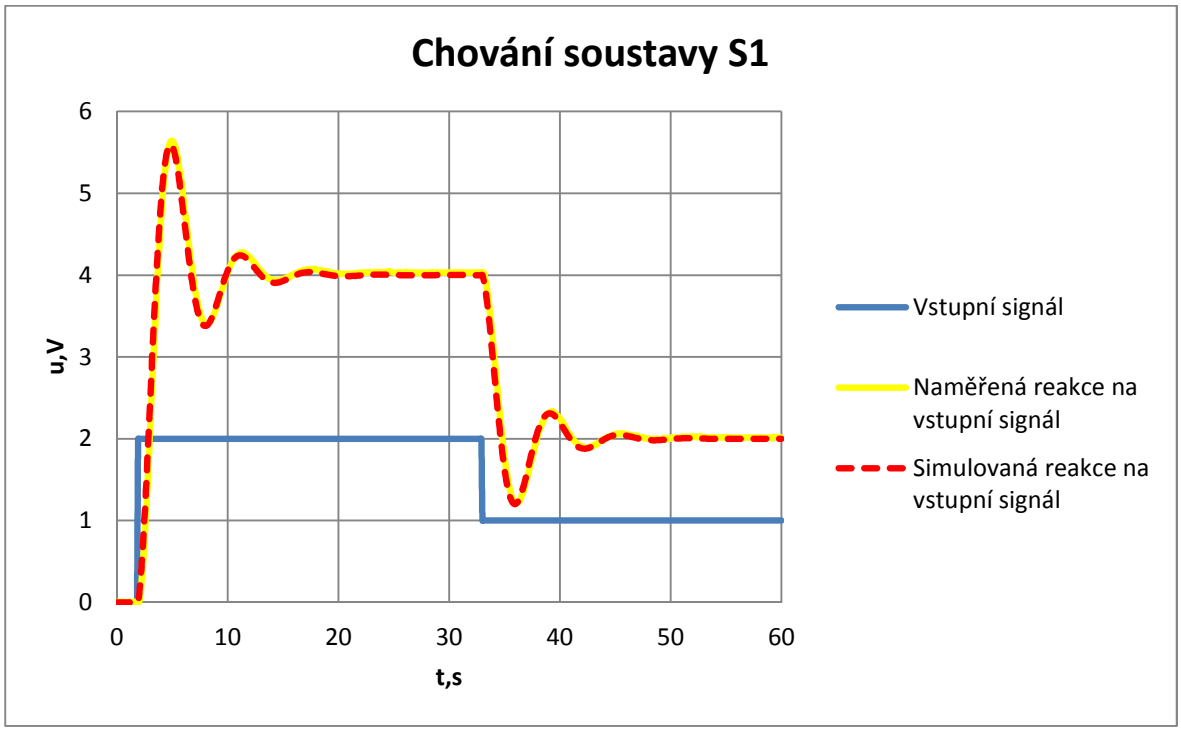
$$G_S = \frac{Z\,(s+z_1)(s+z_2)}{(s+p_1)(s+p_2)(s+p_3)}\,\frac{p_1\ p_2\ p_3}{z_1\ z_2},$$

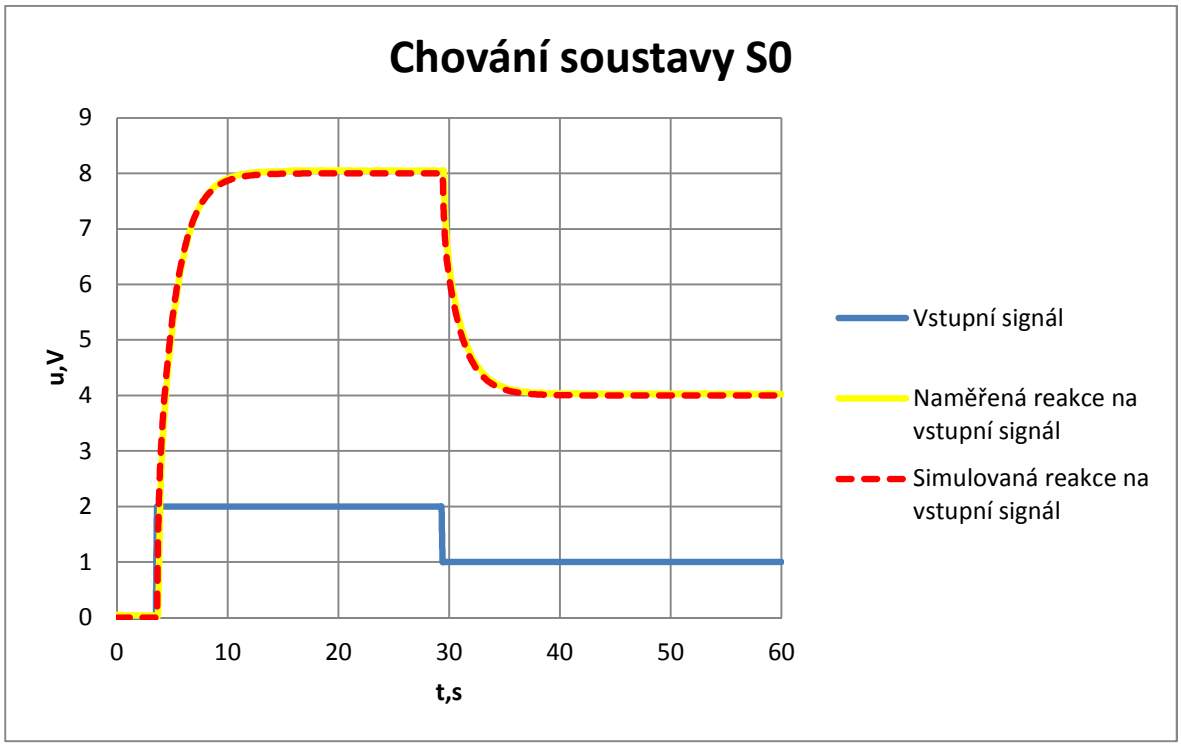
$$\text{kde } Z=2,\ p_1=0.3-j,\ p_2=0.3+j,\ p_3=100\ z_1=5,\ z_2=70$$

$$G_S = \frac{2\,(s+5)(s+70)}{[s+(0.3+j)][s+(0.3-j)](s+100)}\,\frac{(0.3+j)(0.3-j)(100)}{5\cdot 70}$$

$$G_S = \frac{2\cdot\left(s^2+75s+350\right)}{(s^2+0.6s+1.09)\,(s+100)}\,\frac{(0.09+1)(100)}{350}$$

$$G_S = \frac{0.6229s^2+47.71s+218}{s^3+100.6s^2+61.09s+109}$$


Chování soustavy S1
u,V
t,s
0
1
2
3
4
5
6
0
10
20
30
40
50
60
Vstupní signál
Naměřená reakce na vstupní signál
Simulovaná reakce na vstupní signál


Chování soustavy S0
u,V
t,s
0
1
2
3
4
5
6
7
8
9
0
10
20
30
40
50
60
Vstupní signál
Naměřená reakce na vstupní signál
Simulovaná reakce na vstupní signál