České vysoké učení technické v Praze
Fakulta elektrotechnická

BAKALÁŘSKÁ PRÁCE

Knihovna funkčních bloků pro analýzu tvaru a predikci průběhu číslicových signálů v PLC

Praha, 2008 **Autor: Martin Cicvárek**

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Martin Cicvárek**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Knihovna funkčních bloků pro analýzu tvaru a predikci průběhu číslicových signálů v PLC.**

Pokyny pro vypracování:

Cílem je vytvořit soubor funkčních bloků pro programování vyspělých regulačních a diagnostických algoritmů, podle zásad normy IEC EN 61131 - 3 v PLC.

- seznamce s PLC systémy Tecomat a s jejich vývojovým systémem Mosaic,
- seznamte se se zásadami programování a vytváření FB podle normy IEC EN 61131 - 3,
- seznamte se s obvyklými algoritmy predikce a analýzy tvaru signálů,
- navrhněte detailní zadání knihovny funkčních bloků,
- realizujte programy funkčních bloků, ověřte je a zpracujte metodickou dokumentaci.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Martin Hlinovský, Ph.D.

Platnost zadání: do konce zimního semestru 2008/2009

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

v.z. M. Demlová
doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 25. 2. 2008

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V praze dne 15.5.2008 ..... Civánek Martin .....

podpis

Anotace

Predikce. Podle slovníku z Wikipedie predikce označuje obecně předpověď, předvídání, předpoklad budoucího průběhu či stavu, například odhad budoucích hodnot průběhu nějakého jevu, procesu či děje.

Znalost pravděpodobné hodnoty sledovaného signálu v některém z příštích výpočetních kroků může být velice užitečná. Přesto predikce nepatří mezi metody, které by byly dnes často využívány v aplikacích programovatelných automatů PLC. Věřím, že vytvoření knihovny s funkčním blokem, který by predikci ovládal, by tento stav mohlo změnit.

Jako metodu predikce jsem zvolil algoritmus diferenční polynomiální predikce, protože z výsledků diplomové práce pana Kozderky [3] vykazuje výborné vlastnosti, hlavně jednoduchost výpočtu.

Annotation

The prediction. By dictionary of Wikipedia, the prediction is prognosis, prevision, presumption of future development or state, for example judgement future values of course for any phenomenon, process or event.

It can be very useful to know probable values of the watched signal in some of the future steps. In spite of the prediction does not pertain to the methods, which are often used in applications of programmable logic controllers (PLC) today. I believe that creating a library with function block managing the prediction should change this state.

I chosen differential polynomial prediction as a algorithm for the prediction, because it showed very good properties in graduation theses of mister Kozderka [3], mainly for easy calculation.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Teoretická část

1.1 Algoritmus diferenční polynomiální predikce

1.1.1 Diferenční polynomiální predikce

Algoritmus diferenční polynomiální predikce lze považovat za zobecnění algoritmu lineární diferenční predikce dle [2], kdy se předpovídá hodnota veličiny pro budoucí krok jako lineární extrapolace kroku současného a minulého (první diference). U diferenční polynomiální predikce se v každém predikčním kroku z hodnoty diference nejvyššího řádu spočte hodnota diference o řád nižší, z té opět hodnota diference o další řád nižší, až je po několika krocích z první diference vypočtena predikovaná hodnota zpracovávané veličiny. Výhodou tohoto algoritmu je, že ho lze naprogramovat pomocí jednoduchých operací sčítání a odčítání. Podívejme se na tuto metodu podrobněji.

Podle [2] pro obyčejné diference platí vztahy:

$$\begin{aligned}
\nabla y(k) &= y(k) - y(k-1) \\
\nabla^2 y(k) &= \nabla y(k) - \nabla y(k-1) \\
\nabla^3 y(k) &= \nabla^2 y(k) - \nabla^2 y(k-1) \\
&\vdots \\
\nabla^n y(k) &= \nabla^{n-1} y(k) - \nabla^{n-1} y(k-1)
\end{aligned} \tag{1.1}$$

,kde znak ∇ označuje zpětnou diferenci.

Vztahy můžeme dále upravit i tak, aby obsahovaly jen hodnoty vzorků veličiny různého stáří:

$$\begin{aligned}
\nabla y(k) &= y(k) - y(k-1) \\
\nabla^2 y(k) &= \nabla y(k) - \nabla y(k-1) = y(k) - 2y(k-1) + y(k-2) \\
\nabla^3 y(k) &= \nabla^2 y(k) - \nabla^2 y(k-1) = y(k) - 3y(k-1) + 3y(k-2) - y(k-3) \\
\ldots\ldots
\end{aligned} \tag{1.2}$$

Koeficienty jsou binomickými čísly, podobně jako při rozvoji mocnin. Dále můžeme upravit diference tak, aby se dal vyjádřit budoucí krok:

$$\begin{aligned}
\nabla^{n-1} y(k+1) &= \nabla^n y(k+1) + \nabla^{n-1} y(k) \\
\nabla^{n-2} y(k+1) &= \nabla^{n-1} y(k+1) + \nabla^{n-2} y(k) \\
&\vdots \\
\nabla^2 y(k+1) &= \nabla^3 y(k+1) + \nabla^2 y(k) \\
\nabla y(k+1) &= \nabla^2 y(k+1) + \nabla y(k) \\
y(k+1) &= \nabla y(k+1) + y(k)
\end{aligned} \tag{1.3}$$

Ze vztahů (1.3) je vidět, že pokud vyjádříme nejvyšší řád diference ∇^ny(k+1), lze predikovat budoucí vývoj veličiny. Dále si ukážeme, že je opodstatnělé nahradit člen ∇^ny(k+1) členem ∇^ny(k).

1.1.2 Polynomiální funkce

Nechť fyzikální veličina $y(t)$ je polynomiální funkcí spojitého času t:

$$y(t) = a_n t^n + a_{n-1} t^{n-1} + a_{n-2} t^{n-2} + \ldots + a_2 t^2 + a_1 t + a_0 \tag{1.4}$$

Dále předpokládejme, že hodnoty veličiny budeme měřit v diskrétních časových okamžicích $t = k\Delta t$ s ekvidistantním intervalem $t = T_{vz}$, kde k=0,1,2,... je pořadovým číslem vzorku hodnoty y v diskrétním čase. Dosazením do definičního polynomu získáme podobný polynom pro diskrétní proměnnou k stejného řádu, ovšem s jinými koeficienty:

$$y(k) = b_n k^n + b_{n-1} k^{n-1} + b_{n-2} k^{n-2} + \ldots + b_2 k^2 + b_1 k + b_0 \tag{1.5}$$

,kde $b_n = a_n / T_{vz}$.

Jeho první a postupně vyšší diference budou polynomy s postupně klesajícím řádem:

$$\begin{aligned}
\Delta y(k) &= c_{n-1}k^{n-1} + c_{n-2}k^{n-2} + \ldots + c_2k^2 + c_1k + c_0 \\
\Delta^2 y(k) &= d_{n-2}k^{n-2} + d_{n-3}k^{n-3} + \ldots + d_2k^2 + d_1k + d_0 \\
&\vdots \\
\Delta^{n-2} y(k) &= pk^2 + qk + r \\
\Delta^{n-1} y(k) &= 2pk - p + q \\
\Delta^{n} y(k) &= 2p \\
\Delta^{n+1} y(k) &= 0 \\
\Delta^{n+2} y(k) &= 0
\end{aligned} \tag{1.6}$$

,kde znak Δ označuje dopřednou diferenci.

Vidíme, že diference polynomiální funkce má v určitém řádu konstantní hodnotu a v řádech vyšších jsou diference nulové. Tato zákonitost platí obecně pro jakékoli polynomiální funkce.

1.1.3 Transcendentní funkce

Pro technickou praxi jsou důležité i různé transcendentní funkce, které nejsou polynomiální, například sinus, cosinus, exponenciála, jejich kombinace, součiny a konvoluce, odezvy soustav různých řádů na skokové změny, atd. Pro trigonomické a exponenciální funkce jsou v matematických tabulkách publikovány standardní mocninné řady, které vycházejí z Taylerova či McLaurantova rozvoje odpovídající funkce. Tabulka 1.1 podává přehled některých z nich.

Podle vztahů z tabulky 1.1 lze vypočítat diference i pro transcendentní funkce. Teoreticky bychom počítali diference do nekonečna, ale pro praktické výpočty je třeba použít jen konečný počet členů. Při náhradě transcendentní funkce mocninou řadou s konečným počtem členů (polynomem) řádu n, bude diference řádu n konstantní a vyšší řády budou nulové. Lze postupovat i opačně. Pokud vypočteme podle vztahů (1.1) diference funkce do řádu n a vyšší řády budeme ignorovat (považovat za nulové), můžeme získat polynomiální vyjádření nahrazující ideální funkci.

Tabulka přepisu funkcí na mocninné řady		
sin x	$\frac{x}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots+(-1)^n\frac{x^{2n+1}}{(2n+1)!}\pm\ldots$	pro $\lvert x\rvert<\infty$
cos x	$1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots+(-1)^n\frac{x^{2n}}{(2n)!}\pm\ldots$	pro $\lvert x\rvert<\infty$
tg x	$x+\frac{1}{3}x^3+\frac{2}{15}x^5+\frac{17}{315}x^7+\frac{62}{2835}x^9+..$	pro $\lvert x\rvert<\frac{\pi}{2}$
e^x	$1+\frac{x}{1!}+\frac{x^2}{2!}-\frac{x^3}{3!}+\ldots+\frac{x^n}{n!}+\ldots$	pro $\lvert x\rvert<\infty$
ln x	$\frac{x-1}{1}-\frac{(x-1)^2}{2}+\frac{(x-1)^3}{3}-\ldots+(-1)^{n+1}\frac{(x-1)^n}{n}\pm..$	pro $0<x\leq 2$

Tabulka 1.1: Tabulka přepisu funkcí na mocninné řady

Pokud bude hodnota zanedbávaných diferencí (stupně n a vyšší) dostatečně malá, bude i náhrada ideální funkce polynomem generovaným z vypočítaných diferencí podle vztahů 1.3 „dostatečně kvalitní“ pro „odpovídající počet kroků“ generování (predikce).

1.2 PLC Tecomat

Obrázek 1.1: Tecomat 700

Programovatelné automaty Tecomat jsou určeny pro řízení nejrůznějších technologií, strojů a linek ve všech průmyslových oborech, pro nasazení v energetice, dopravě apod. Provedení a programové možnosti řadí systémy Tecomat mezi dobrý standart světových PLC. Svým rozsahem pokrývají aplikace od jednotek vstupů a výstupů (malé kompaktní systémy TC400) přes větší kompaktní systémy TC500, TC600 a TC650, až po velké aplikace, pro které jsou určeny modulární systémy NS950 a TC700. Tabulka 1.2 podává základní přehled o řadách výrobků Tecomat.

Typová řada	Binární I/O	Analogové I/O	Komunikační kanály	Displej/ klávesnice	Paměť programu
TC400	6/4	2/-	2 serial	externí	32 kB
TC500	20/20	4/4	2 serial	integrovaný	32 kB
TC600	48/44	24/8	3 serial	externí	32 kB
TC650	48/44	24/8	3 serial, 1 Ethernet	externí	64 + 64 kB
Foxtrot	až 128 DI/DO	až 80AI / 80AO	2× serial, 1×CIB, 1×Ethernet	externí	192 + 64 kB
TC700	přes 6500	až 3300/700	až 10 serial, 2 Ethernet, 1 USB	externí	192 + 64 kB

Tabulka 1.2: Řady výrobků Tecomat

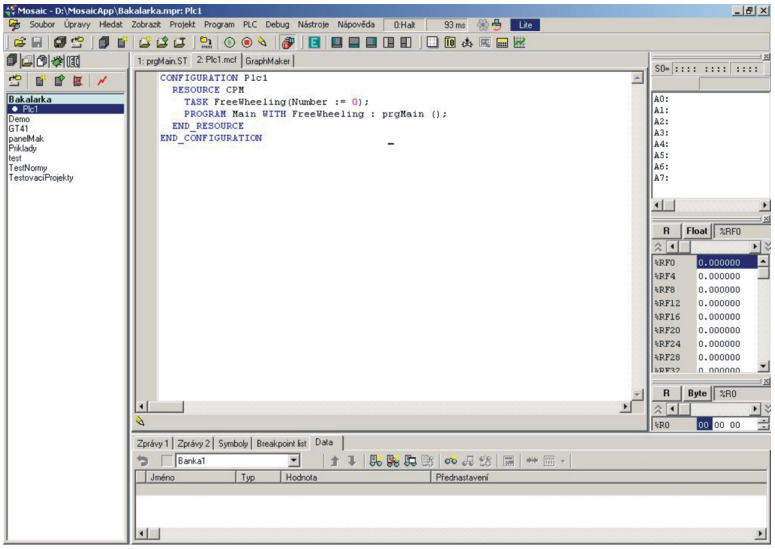
Za nadstandardní v programovacích možnostech lze považovat zejména:

- soubor aritmetických instrukcí v pevné i pohyblivé řádové čárce
- instrukce pro práci s tabulkami a datovými strukturami
- prostředky pro vytváření zásobníkových struktur a zaznamníků dat a událostí s časovými značkami
- instrukce regulátoru PID a nově i regulátoru PID s momentovým autotunerem (pouze pro TC700)
- uživatelské instrukce, podprogramy a komponenty pro fuzzy logiku
- uživatelské instrukce pro interpolaci tabelárně zadných funkcí jedné a dvou proměnných

Veškeré systémy TECOMAT lze programovat z vývojového prostředí Mosaic.

Podrobnosti a veškerá dokumentace o programovatelných automatech z produkce Teco a.s. Kolín jsou uvedeny v [3] a [4].

1.3 Vývojové prostředí Mosaic



Obrázek 1.2: Vývojové prostředí Mosaic

K programování systémů Tecomat použijeme vývojový systém Mosaic. Mosaic je integrované vývojové prostředí, které umožňuje vytvářet aplikační programy pro PLC Tecomat. Systém je uživatelsky přívětivý a poskytuje značný komfort, a to nejen při samotné tvorbě uživatelských programů pro PLC, ale i při jejich ladění, dokumentování, úpravách a diagnostice řízených procesů. Ojedinělou a velice důležitou vlastností je možnost pracovat s "virtuálním PLC" a program ladit na počítači bez přítomnosti fyzického programovatelného automatu.

Mosaic umožňuje komunikovat s řídicím systémem přes sériovou linku, Ethernet či USB. V prostředí je zahrnuta i podpora pro vytáčené připojení přes telefonní síť nebo GSM modem a v poslední době oblíbené spojení přes Wi-Fi, které umožňuje dálkovou správu.

Vývojové prostředí Mosaic navíc obsahuje užitečné nástroje, například PIDMaker pro vytváření a nastavování PID regulátorů, PanelMaker pro tvorbu dialogů uživatelského panelu a PanelSim pro jejich simulaci a GraphMaker pro podporu ladění a diagnostiku řízeného systému. GraphMaker umožňuje zobrazení průběhů vybraných proměnných, funkci digitálního osciloskopu až pro 16 kanálů a funkci logického analyzátoru. Má dva sledovací kurzory, nastavitelnou periodu vzorkování a umožňuje ukládat data na disk i jejich export do databázových programů.

Programovacím jazykem, který je společný pro všechny systémy Tecomat, je firemní mnemokód (jazyk typu IL), který zajišťuje kompatibilitu programování a přenositelnost. Ovšem u nových typů PLC Tecomat, založených na 32-bitových procesorech, je možné psát uživatelský program i v jazyce strukturovaného textu (ST), jazyce instrukcí (IL), jazyce reléových schémat (LD) a jazyce funkčních bloků (FBD), vše podle mezinárodní normy IEC EN 61131-3, což přináší řadu výhod - mj. zpřehledňuje a zefektivňuje programování a zároveň je možné využít podpůrné nástroje, které nejsou u starších typů CPU podporovány.

Podrobnosti a veškerá dokumentace o integrovaném vývojovém prostředí Mosaic jsou uvedeny v [3] a [4].

1.4 Mezinárodní norma IEC EN 61 131-3

1.4.1 Základní myšlenky normy IEC 61 131-3

Norma IEC 61 131 pro programovatelné řídící systémy má pět základních částí a představuje souhrn požadavků na moderní řídící systémy. Je nezávislá na konkrétní organizaci či firmě a má širokou mezinárodní podporu. Jednotlivé části normy jsou věnovány jak technickému tak programovému vybavení těchto systémů.

Na normu 61 131-3 je možné pohlížet z různých hledisek, např. tak, že je to výsledek náročné práce sedmi mezinárodních společností, které do vypracování normy vložily svoji desetiletou zkušenost na poli průmyslové automatizace. Výsledkem je specifikace syntaxe a sémantiky unifikovaného souboru programovacích jazyků, včetně obecného softwarového modelu a strukturujícího jazyka. Tato norma byla přijata jako směrnice u většiny významných výrobců PLC.

Podrobné informace o mezinárodní normě IEC EN 61131-3 jsou uvedeny v [4]. Uvedu zde pouze základní myšlenky a vlastnosti normy.

1.4.1.1 Programovací jazyky

V rámci standardu jsou definovány čtyři programovací jazyky. Jejich sémantika i syntaxe je přesně definována a neponechává žádný prostor pro nepřesné vyjadřování. Zvládnutím těchto jazyků se tak otevírá cesta k používání široké škály řídicích systémů, které jsou na tomto standardu založeny.

Programovací jazyky se dělí do dvou základních kategorií:

Textové jazyky	IL	Instruction List	jazyk seznamu instrukcí
	ST	Structured Text	jazyk strukturovaného textu
Grafické jazyky	LD	Ladder Diagram	jazyk příčkového diagramu
	FBD	Function Block Diagram	jazyk funkčního blokového schématu

Volba programovacího jazyka je závislá na zkušenostech programátora, na typu řešeného problému, na úrovni popisu problému, na struktuře řídicího systému a na řadě dalších okolností, jako jsou např. typ odvětví průmyslu, zvyklosti firmy implementující řídicí systém, zkušenosti spolupracovníků v týmu apod. Všechny čtyři základní jazyky (IL, ST, LD a FBD) jsou vzájemně provázány.

Jazyk strukturovaného textu je podobný vyšším programovacím jazykům. Syntaxí se nejvíce blíží k Pascalu. Norma IEC EN 61 131 zaručuje použití základních programových příkazů, známých z vyšších jazyků, jako jsou **IF, CASE, RETURN, EXIT** , cykly **FOR, WHILE, REPEAT**, atd. a tím poskytují programátorovi značné pohodlí.

1.4.1.2 Společné prvky

Typy dat

V rámci společných prvků jsou definovány typy dat. Běžné datové typy jsou **BOOL, BYTE, WORD, INT** (Integer) **,REAL, DATE, TIME, STRING** atd. Z těchto základních datových typů je pak možné odvozovat vlastní uživatelské datové typy, tzv. odvozené datové typy.

Proměnné

Oblast působnosti proměnných je běžně omezena pouze na tu programovou organizační jednotku (POU - viz níže), ve které byly deklarovány (proměnné jsou v ní lokální). To znamená, že jejich jména mohou být používána v jiných částech bez omezení. Pokud mají mít proměnné globální působnost, např. v rámci celého projektu, pak musí být jako globální deklarovány (v bloku **VAR_GLOBAL**). Aby bylo možné správně nastavit počáteční stav procesu nebo stroje, může být parametrům přiřazena počáteční hodnota při startu nebo studeném restartu.

1.4.2 Programové organizační jednotky

Funkce, funkční bloky a programy jsou v rámci normy IEC 61 131 nazývány společně *programové organizační jednotky* (Program Organization Units). Jak vyplývá z názvu, POU je nejmenší nezávislá část uživatelského programu. POU mohou být dodávány od výrobce řídícího systému nebo je může napsat uživatel. Každá POU může volat další POU a při tomto volání může volitelně předávat volané POU jeden nebo více parametrů.

1.4.2.1 Funkce

Nejjednodušší POU je funkce, jejíž hlavní charakteristikou je to, že pokud je volána se stejnými vstupními parametry, musí produkovat stejný výsledek (funkční hodnotu). Funkce může vracet pouze jeden výsledek. IEC 61 131-3 definuje standardní funkce a uživatelem definované funkce. Standardní funkce jsou např. ADD, ABS, SQRT, atd. Jakmile jsou jednou definovány nové uživatelské funkce, mohou být používány opakovaně.

1.4.2.2 Funkční bloky

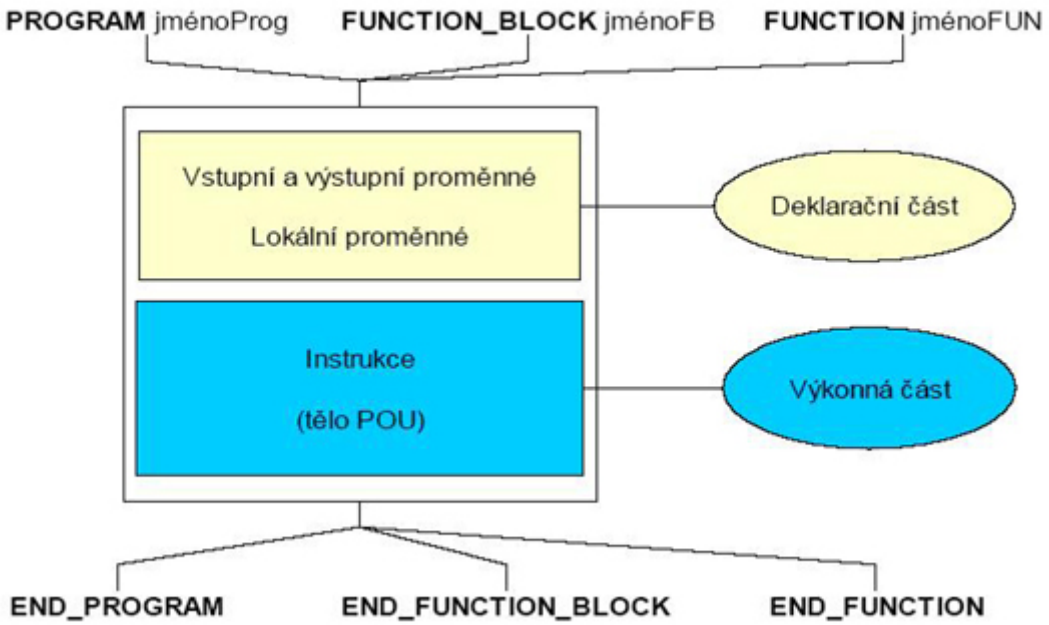
Funkční blok si na rozdíl od funkce, může pamatovat některé hodnoty z předchozího volání (např. stavové informace). Ty pak mohou ovlivňovat výsledek. Hlavním rozdílem mezi funkcí a funkčním blokem je tedy schopnost funkčního bloku vlastnit paměť pro zapamatování hodnot některých proměnných. Funkční blok může také (na rozdíl od funkce) vracet více než jeden výsledek. Jakmile je jednou funkční blok definován, může být používán opakovaně v daném programu, nebo v jiném programu, nebo dokonce i v jiném projektu. Je tedy univerzální a mnohonásobně použitelný. Funkční bloky mohou být zapsány v libovolném z jazyků definovaném v normě.

1.4.2.3 Programy

Program představuje vrcholovou programovou jednotku v uživatelském programu. Centrální jednotka PLC může zpracovávat více programů a programovací jazyk ST obsahuje prostředky pro definice spouštění programů (v jaké periodě vykonávat program, s jakou prioritou, apod).

1.5 Struktura programové organizační jednotky

Každá POU se skládá ze dvou základních částí : **deklarační** a **výkonné**, jak je vidět na obrázku (1.3). V deklarační části POU se definují proměnné potřebné pro činnost POU. Výkonná část pak obsahuje vlastní příkazy pro realizaci požadovaného algoritmu.



Obrázek 1.3: Struktura programové organizační jednotky

1.5.1 Deklarační část

Deklarační část POU obsahuje definice proměnných potřebných pro činnost POU. Každá proměnná je definována jménem proměnné a datovým typem. Proměnné se deklarují ve čtyřech základních deklaračních blocích.

VAR_INPUT	- pro deklaraci vstupních proměnných
VAR_OUTPUT	- pro deklaraci výstupních proměnných
VAR	- pro deklaraci lokálních proměnných, jejichž hodnota se uchovává mezi jednotlivými voláními POU (nelze použít pro funkce)
VAR_TEMP	- pro deklaraci lokálních proměnných, které platí jen v rámci jednoho volání POU

Všechny tyto bloky jsou ukončeny klíčovým slovem END_VAR. Obrázek (1.4) ukazuje příklad deklarací.

```
FUNCTION_BLOCK Priklad
VAR_INPUT
    vstup            :        real;  //vstupní parametr
  END_VAR
  VAR_OUTPUT
   vystup            :        real;  //vystupní proměnná
  END_VAR
  VAR
  Pocitadlo          :        int;   //vnitřní proměnná čítače volání FB
  END_VAR
  VAR_TEMP
  i                  :        sint;  //proměnná např. pro for cyklus
  END_VAR
```

Obrázek 1.4: Příklad deklarační části

1.5.2 Výkonová část POU

Výkonová část POU následuje za částí deklarační a obsahuje příkazy a instrukce, které jsou zpracovávány centrální jednotkou PLC. Ve výjimečných případech nemusí definice POU obsahovat žádnou deklarační část a potom je výkonná část uvedena bezprostředně za definicí začátku POU. Výkonová část POU může obsahovat volání dalších POU. Při volání mohou být předávány parametry pro volané funkce, resp. funkční bloky. Obrázek (1.5) ukazuje příklad výkonové části s voláním funkčního bloku.

```
PROGRAM prgMain
  VAR
  a   :   real;
  b   :   real;
  fb  :   Priklad;
  END_VAR

  a:=3.0+8.3;
  fb(vstup:=a,vystup=>b);  //Zavolání instance funkčního bloku typu Priklad
                           //a předání parametrů a-pro vstup a b-pro výstup

END_PROGRAM
```

Obrázek 1.5: Příklad výkonové části

Kapitola 2

Realizace bloků

Knihovnu predikce jsem vypracoval ve vývojovém prostředí Mosaic v jazyce ST (structual text). Abych mohl výsledky své práce ladit, testovat a posléze prezentovat, navrhl jsem další funkční bloky, generující signály sinus, exponenciálu, odezvu soustavy 1.řádu a odezvu soustavy 2.řádu. Tyto funkční bloky jsem také zahrnul do své knihovny.

2.1 Generátory

V této kapitole vysvětlím svůj postup návrhu jednotlivých funkčních bloků generujících signály. Podrobné informace jsou uvedeny v Příloze Manuál knihovny Predikt_Lib_20080406.

2.1.1 Generátor sinusového průběhu

Základní myšlenkou byla vize, aby vstupem funkčního bloku byla perioda signálu a výstup generovaný průběh. Pro generování pravidelného sinusového průběhu je potřeba mít informaci o čase. Jelikož nelze uvnitř funkčního bloku volat stavové registry systému, aby si funkční blok sám zjistil potřebný časový údaj, bylo potřeba přidat vstup s informací o čase nebo hodinovým signálem.

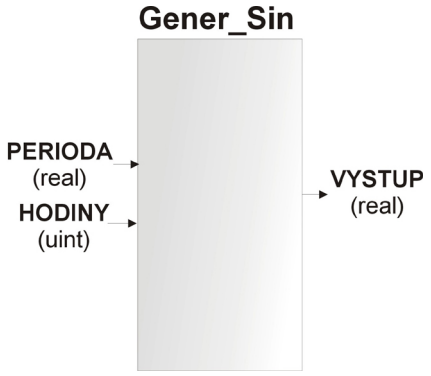
Rozhodl jsem se naprogramovat vstup jako požadavek na čítač desetin sekund. Tento způsob řešení je sice složitější než použití hodinového signálu, ale zaručí nám správný tvar a pravidelnou periodu i v případě, že bychom měli problémy s dodržením přesných časových intervalů. Pro PLC Tecomat vyhovuje tomuto požadavku systémový registr

SW14.

Pro generování sinusového průběhu na výstupu bloku jsem využil knihovní funkci sin() - viz [3]. Funkce sinus vyžaduje jako parametr úhel v radiánech a vrací hodnotu sinu.

Největším problémem bylo, jak správně spočítávat měnící se úhel v čase. Kdybych použil přímo registr čítače desetin sekund ze vstupu, nastal by při přetečení tohoto čítače skok z momentální hodnoty funkce na počátek, který by se pravidelně opakoval. Takováto chyba je u generátoru sinusovky nepřípustná. Proto jsem zvolil metodu, kdy vyhodnotím časový rozdíl od posledního volání funkčního bloku, který přičtu k vnitřnímu čítači času. Pokud tento vnitřní čítač překročí hodnotu periody, dobu periody od něj odečtu.

Vstup s dobou periody se zadává v sekundách. Blok vyhodnotí časový rozdíl, který uběhl od minulého volání a vypočte správný úhel pro současnou chvíli. Tento úhel se poté zadá funkci sin() a výsledek je nasměrován na výstup funkčního bloku.



Obrázek 2.1: Generátor sinusového průběhu

Závěrem musím upozornit na skok, který nastává při změně požadavku na dobu periody. Funkční blok Gener_Sin pracuje správně pro periody v rozmezí od 1 sekundy do 6553 sekund (109 minut). V případě potřeby lze generovat i pomalejší či rychlejší průběhy přivedením na vstup registr počítající jiné časové jednotky. V tom případě je ovšem nutné pozměnit i hodnotu periody. Více informací v příloze Manuál Predikt_Lib_20080406.

2.1.2 Generátor exponenciálního průběhu

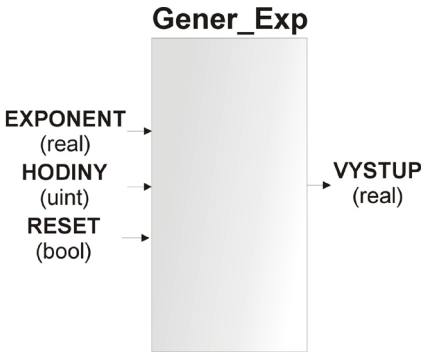
Prvotní představa o funkčním bloku, který bude na výstupu generovat exponenciální funkci, byla se dvěmi vstupy, jedním udávající hodnotu exponentu a druhým bitovým pro nulování čítače času. Ze stejného důvodu, jako pro funkční blok generující sinusový průběh, jsem byl nucen přidat na vstup ještě požadavek na čítač desetin sekund.

Pro generování exponenciálního průběhu na výstupu bloku jsem využil knihovní funkci exp() - viz [3]. Funkce exp() vyžaduje jako svůj parametr exponent. Tento exponent se spočítá vynásobením požadovaného exponentu ze vstupu a času.

Pro výpočet časového rozdílu od minulého volání funkčního bloku jsem vyřešil stejným algoritmem, jaký je popsán v kapitole 2.1.1 Generátor sinusového průběhu.

Jelikož funkce exp() vrací hodnotu typu real, je nutné ohlídat, zda parametr funkce (exponent) není větší než jaký výsledek se vejde do tohoto datového typu. Datový typ real má maximální hodnotu $3,4e38$ [3], proto by exponent neměl být vyšší než $ln(3,4e38)$, což je přibližně hodnota $88,7$. Pokud funkční blok vypočte exponent větší než 88, průběh dále nevypočítává.

Čítač času funkčního bloku se vynuluje ve dvou případech. Pokud se na vstupu RESET objeví logická jednička nebo pokud se změní hodnota na vstupu EXPONENT. Průběh tedy vždy začíná na hodnotě $1,0$.



Obrázek 2.2: Generátor exponenciálního průběhu

Funkční blok Gener_Exp pracuje správně, pokud je volán alespoň jednou za $6553,5$ sekundy, z důvodu správného rozpoznání časového rozdílu mezi dvěmi voláními. Podrobnější informace příloze Manuál Predikt_Lib_20080406.

2.1.3 Generátor odezvy soustavy 1.řádu

Základní představa funkčního bloku, generujícího na výstupu odezvu soustavy 1.řádu, byla se dvěmi vstupy, první požadované ustálené hodnoty a druhý požadované časové konstanty τ. Ze stále stejných důvodů jako v předchozích případech jsem byl nucen přidat další vstup. Rozhodl jsem se tentokrát požadovat na vstupu hodinové pulzy, protože jsem úlohu řešil pomocí diferencí.

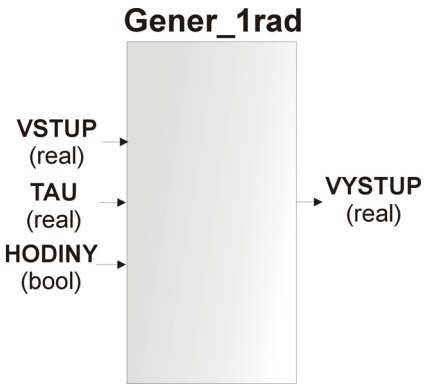
Při řešení jsem vycházel ze vztahu (2.1) dle [5] :

$$\frac{Y(s)}{U(s)} = \frac{a}{s+a} \tag{2.1}$$

,kde $a = 1/\tau$.

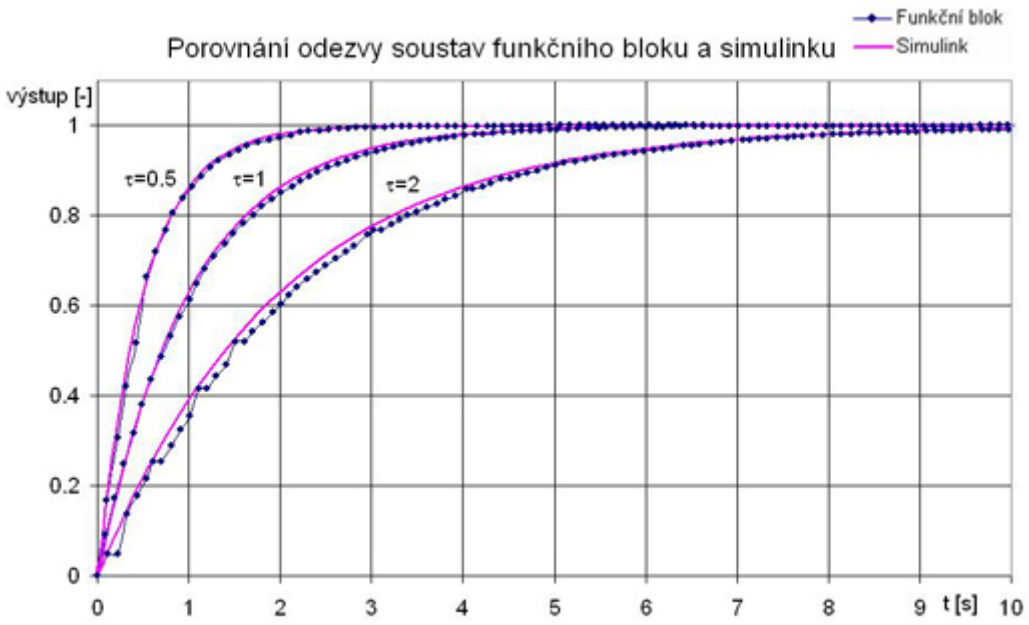
Operátor s znázorňuje Laplacův obraz derivace. V našem diskrétním případě jsem ovšem místo derivací použil diference (a to zpětné). Vztah jsem tedy upravil takto:

$$\begin{aligned} &(s+a)Y(s) = aU(s) \\ &\nabla y(k) + ay(k) = au(k) \\ &y(k) - y(k-1) + ay(k) = au(k) \\ &y(k) = \tfrac{au(k)+y(k-1)}{1+a} \end{aligned} \tag{2.2}$$



Obrázek 2.3: Generátor odezvy soustavy 1.řádu

Funkční blok generátoru odezvy soustavy 1.řádu vyžaduje na svém vstupu HODINY signál s periodou $0,1$ sekundy. Pro PLC Tecomat tomu odpovídá signál S13.0. Blok reaguje na vzestupnou hranu signálu. Pokud se zadá jiná frekvence, je nutné patřičně poupravit i vstup TAU (τ). Porovnání generovaného výstupu funkčního bloku a odezvy soustavy z programu Simulink je na obrázku 2.4.



Obrázek 2.4: Porovnání odezvy soustav 1.řádu funkčního bloku a Simulinku

2.1.4 Generátor odezvy soustavy 2.řádu

Při návrhu funkčního bloku generujícího na výstupu odezvu soustavy 2.řádu jsem vycházel ze vztahu (2.3) dle [5]:

$$\frac{Y(s)}{U(s)} = \frac{\omega_n^2}{s^2 + 2\varsigma\omega_n s + \omega_n^2} \tag{2.3}$$

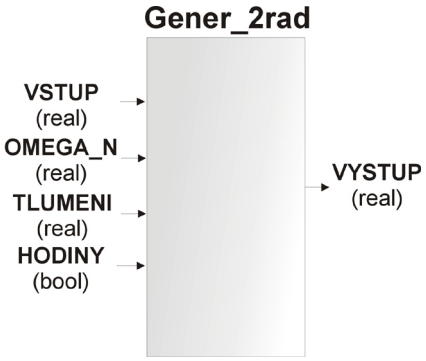
,kde ω_n je vlastní frekvence a ς představuje tlumení soustavy.

Operátor s znázorňuje Laplacův obraz derivace. V našem diskrétním případě jsem ovšem místo derivací použil diference (a to zpětné). Vztah jsem tedy upravil takto:

$$\begin{aligned}
&(s^2 + 2\varsigma\omega_n s + \omega_n^2)Y(s) = \omega_n^2 U(s) \\
&\nabla^2 y(k) + 2\varsigma\omega_n \nabla y(k) + \omega_n^2 y(k) = \omega_n^2 u(k) \\
&y(k) - 2y(k-1) + y(k-2) + 2\varsigma\omega_n y(k) - 2\varsigma\omega_n y(k-1) + \omega_n^2 y(k) = \omega_n^2 u(k) \\
&(1 + 2\varsigma\omega_n + \omega_n^2)y(k) = \omega_n^2 u(k) + (2 + 2\varsigma\omega_n)y(k-1) - y(k-2) \\
&y(k) = \tfrac{\omega_n^2 u(k) + (2+2\varsigma\omega_n)y(k-1) - y(k-2)}{1+2\varsigma\omega_n+\omega_n^2}
\end{aligned} \tag{2.4}$$

Funkční blok má tedy na vstup přivedenu hodnotu žádané vlastní frekvence soustavy a vstup s požadovanou hodnotou tlumení.

Ze stejného důvodu jako u generátoru odezvy soustavy 1.řádu jsem na vstup přidal požadavek na hodinové pulzy.

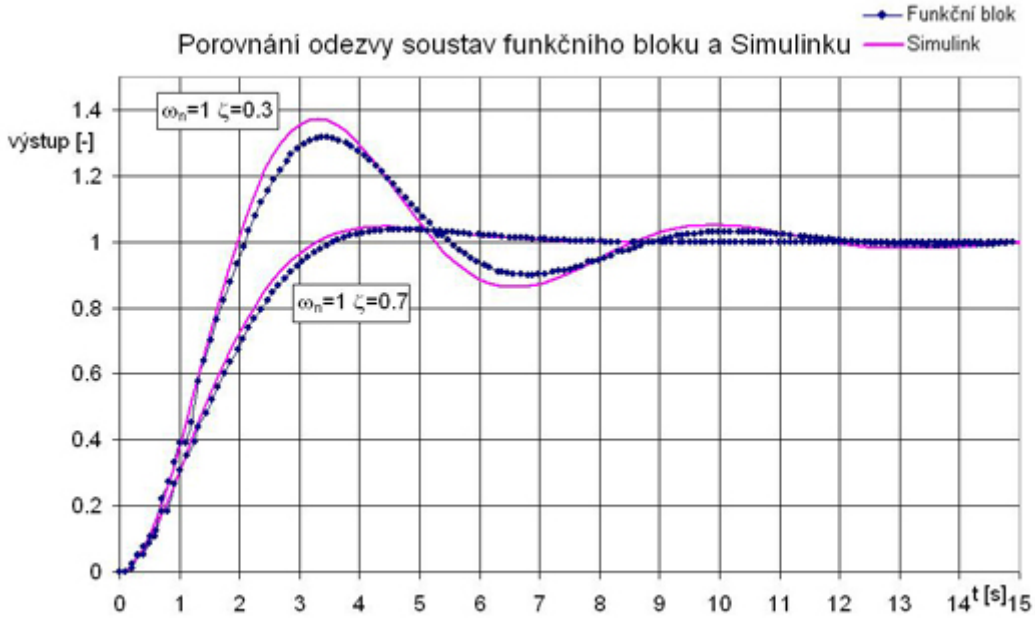


Obrázek 2.5: Generátor odezvy soustavy 2.řádu

Funkční blok generátoru odezvy soustavy 2.řádu vyžaduje na svém vstupu HODINY signál s periodou $0,1$ sekundy. Pro PLC Tecomat tomu odpovídá signál S13.0. Blok reaguje na vzestupnou hranu signálu. Pokud se zadá jiná frekvence, je nutné patřičně poupravit i vstup OMEGA_N (ω_n).

Na obrázku 2.6 je porovnání generovaného výstupu funkčního bloku a odezvy soustavy z programu Simulink. Z grafu se dá tvrdit, že funkční blok generuje průběh odezvy soustavy 2.řádu se správnou vlastní frekvencí, ale více tlumený než je požadavek. Domnívám se, že chyba je v zaokrouhlování při počítání průběhů. Tato chyba je pro analýzu funkčního bloku nepodstatná. Pro analýzu funkčního bloku predikce tedy tento funkční blok mohu použít, ovšem pro přesnou simulaci se nehodí.

Dále je třeba upozornit na skutečnost, že funkčním blokem nelze generovat odezvu soustavy s nulovým tlumením. Odezva bude vždy tlumená. Problém s největší pravděpodobností spočívá v nepřesnosti nuly pro datový typ real. Více informací o tomto datovém typu najdete v [6].



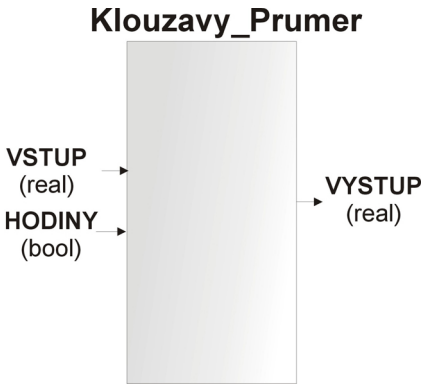
Obrázek 2.6: Porovnání odezvy soustav 2.řádu funkčního bloku a Simulinku

2.2 Filtry

2.2.1 Filtr typu klouzavý průměr

Pro pozdější testování algoritmu predikce pro reálné signály jsem navrhl i funkční blok filtru typu klouzavý průměr.

Funkční blok má vstup pro signál, který má být filtrován a vstup s hodinovým signálem, určujícím diskrétní čas. Výstupem je filtrovaný signál.

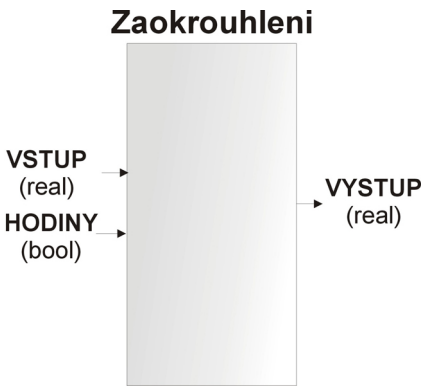


Obrázek 2.7: Filtr typu klouzavý průměr

Funkční blok je naprogramován pro počítání průměrné velikosti signálu z posledních deseti hodnot. Z těchto deseti hodnot se vyloučí nejvyšší a nejnižší hodnota a ze zbylých osmi hodnot se spočítá průměr.

2.2.2 Funkční blok pro zaokrouhlení

Pro další pozdější testování algoritmu predikce jsem také navrhl funkční blok pro zaokrouhlování. Funkční blok má na výstupu hodnotu signálu na vstupu zaokrouhlenou na jedno desetinné místo.



Obrázek 2.8: Funkční blok pro zaokrouhlení

2.3 Funkční blok predikce

2.3.1 Adaptační část

Algoritmus adaptační části, kdy se spočítají patřičné diference pro daný čas, vychází z rovnic (1.1). Podle výsledků práce pana Kozderky [1] vychází výpočet sedmi diferencí jako dostatečný, k čemuž je potřeba osmi kroků adaptace. Po osmi krocích jsou diference správně spočteny a predikční část z nich dokáže vypočítat odpovídající hodnoty. Na obrázku 2.9 je zobrazen algoritmus adaptace. Celý zdrojový text je v příloze Zdrojový kód Predikt_Lib_20080406.

```
//adaptacni cast
//vypocet novych hodnot pro predikci
dif_predik0:=vstup;
dif_predik1:=dif_predik0-dif_adapt0;
dif_predik2:=dif_predik1-dif_adapt1;
dif_predik3:=dif_predik2-dif_adapt2;
dif_predik4:=dif_predik3-dif_adapt3;
dif_predik5:=dif_predik4-dif_adapt4;
dif_predik6:=dif_predik5-dif_adapt5;
dif_predik7:=dif_predik6-dif_adapt6;

//kopie novych diferenci do pole pro adaptaci
dif_adapt0:=dif_predik0;
dif_adapt1:=dif_predik1;
dif_adapt2:=dif_predik2;
dif_adapt3:=dif_predik3;
dif_adapt4:=dif_predik4;
dif_adapt5:=dif_predik5;
dif_adapt6:=dif_predik6;
dif_adapt7:=dif_predik7;
```

Obrázek 2.9: Algoritmus adaptační části

2.3.2 Predikční část

Algoritmus predikční části, kdy se ze spočtených diferencí predikuje budoucí vývoj signálu, vychází z rovnic (1.3). Podle diferencí spočtených pro aktuální čas se predikuje postupně první krok předpokládaného budoucího vývoje signálu, posléze druhý, třetí ... až osmý. Na obrázku 2.10 je zobrazen algoritmus výpočtu. Celý zdrojový text je v příloze Zdrojový kód Predikt_Lib_20080406.

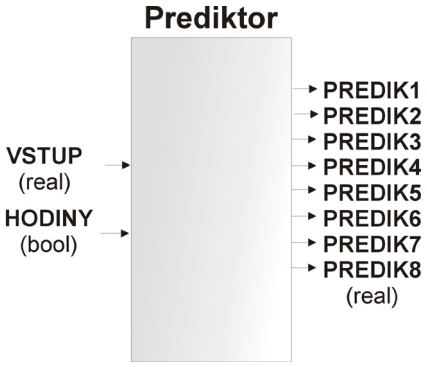
```
//Cyklus predikcni casti
FOR i:=1 TO 8 DO
 dif_predik6:=dif_predik7+dif_predik6;
 dif_predik5:=dif_predik6+dif_predik5;
 dif_predik4:=dif_predik5+dif_predik4;
 dif_predik3:=dif_predik4+dif_predik3;
 dif_predik2:=dif_predik3+dif_predik2;
 dif_predik1:=dif_predik2+dif_predik1;
 dif_predik0:=dif_predik1+dif_predik0;

CASE i OF
 1:  predik1:=dif_predik0;
 2:  predik2:=dif_predik0;
 3:  predik3:=dif_predik0;
 4:  predik4:=dif_predik0;
 5:  predik5:=dif_predik0;
 6:  predik6:=dif_predik0;
 7:  predik7:=dif_predik0;
 8:  predik8:=dif_predik0;
END_CASE;
END_FOR;
```

Obrázek 2.10: Algoritmus predikční části

2.3.3 Realizace funkčního bloku predikce

Základní představou o funkčním bloku zvládajícím predikci je, aby na vstupu byl signál, jehož další průběh se má předpovídat, a na výstupu několik hodnot předpovídající hodnotu signálu pro příštích n kroků. Jelikož se v algoritmu používají diference, je potřeba na vstup přivést signál, který bude určovat diskrétní čas.



Obrázek 2.11: Funkční blok predikce

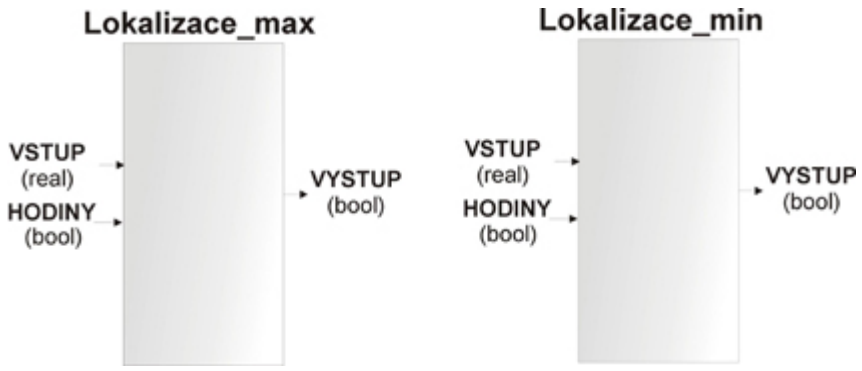
Funkční blok predikce se dá rozdělit na dvě části, které jsou také naprogramovány zvlášť do dvou samostatných funkčních bloků. První z nich je Blok_diference, kde se spočítají patřičné diference pro daný čas, a druhý je Blok_predikce, kdy se ze vstupních diferencí predikuje budoucí vývoj signálu. Kompletní popis obou funkčních bloků lze nalézt v příloze Manuál knihovny Predikt_Lib_20080406.

2.4 Funkční bloky pro lokalizace extrémů a ustálení

Znalost budoucího vývoje signálu vybízí k naprogramování fukčních bloků, které budou umět podle predikovaných hodnot detekovat lokální extrémy nebo ustálení signálu. Při analýze signálu může být tato znalost velice užitečná. A tak jsem do své knihovny zahrnul i funkční bloky pro rozpoznání těchto stavů.

2.4.1 Funkční bloky pro lokalizaci maxima a minima

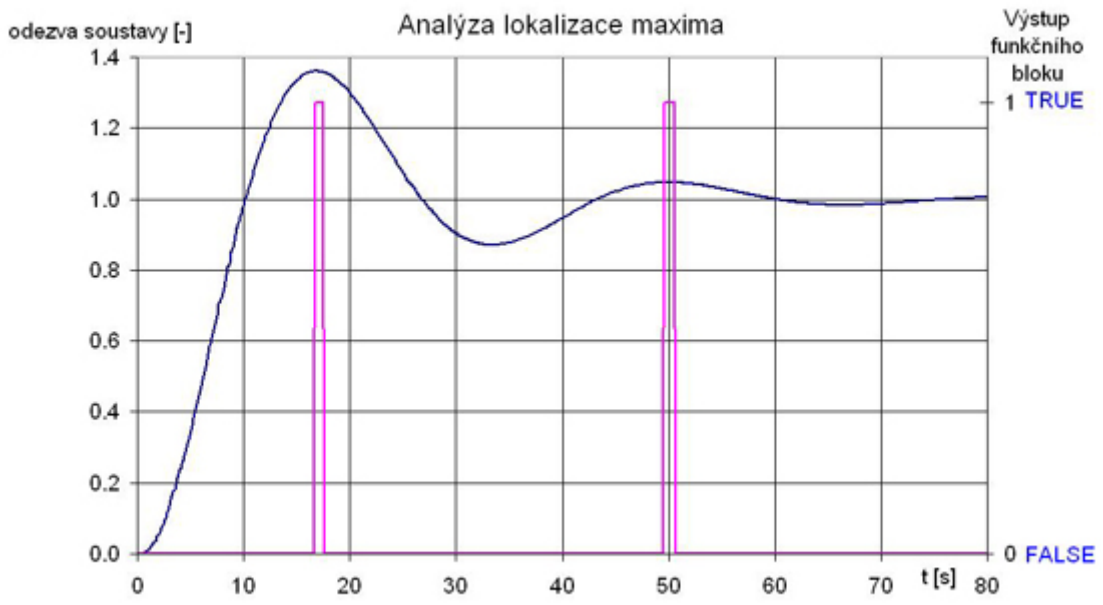
Tyto bloky se liší jen ve vnitřní podmínce, proto se v této podkapitole zaměřím pouze na funkční blok lokalizace maxima. Vše, co platí pro něj, platí i pro funkční blok lokalizace minima.



Obrázek 2.12: Funkční bloky pro lokalizaci maxima a minima

Tento funkční blok využívá algoritmus predikce stejný jako predikátor. Ze vstupního signálu vypočítává budoucí hodnoty vývoje pro čtyři následující kroky, které se ukázali jako dostatečné. Pokud byl minulý signál menší (resp. větší) než současný a zároveň také predikované hodnoty jsou menší (resp. větší), pak se jedná o lokální maximum (resp. minimum) a na výstupu funkčního bloku se objeví logická jednička.

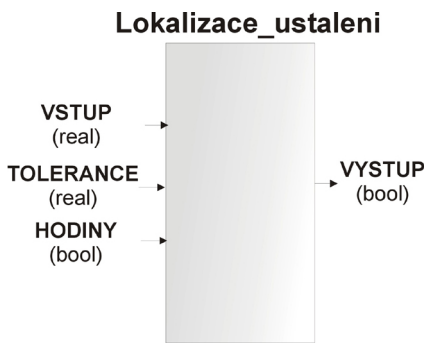
Na obrázku 2.13 je analýza detekce lokálního maxima pro odezvu soustavy druhého řádu ($\varsigma = 0,3$ a $\omega_n = 0,2$). Algoritmus tedy funguje.



Obrázek 2.13: Příklad detekce maxima

2.4.2 Funkční blok pro lokalizaci ustálení

Funkční blok pro lokalizaci ustálení bude na vstupu požadovat zkoumaný signál, hodiny a navíc hodnotu tolerančního pásma, ve které se může zkoumaný signál vlnit.

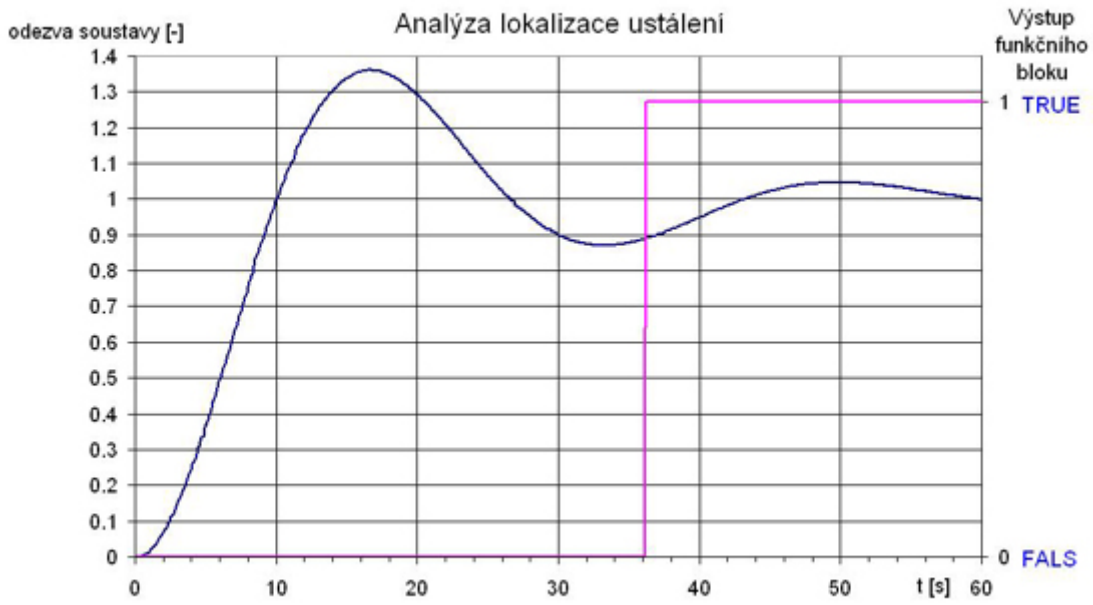


Obrázek 2.14: Funkční blok pro lokalizaci ustálení

Pokoušel jsem se pro detekci ustálení použít také algoritmus predikce, ten ovšem v tomto případě selhal. Při ustáleném stavu, kdy systém měnil svůj výstup jen v rámci své tolerance, se algoritmus predikce často „rozkmital“ a začal predikovat nesmyslné průběhy.

Proto jsem v tomto případě použil jednodušší algoritmus. Funkční blok si pamatuje

posledních 10 hodnot vstupního signálu. Z těchto hodnot spočítá průměr, ke kterému spočte hranice tolerančního pásu. Poté postupně porovnává hodnoty od nejvzdálenějšího kroku k současnosti. Pokud je pokaždé signál uvnitř tolerančního pásu, pak se na výstupu funkčního bloku objeví logická jednička.



Obrázek 2.15: Příklad detekce ustálení

Na obrázku 2.15 je znázorněna detekce ustálení pro odezvu soustavy druhého řádu ($\varsigma = 0,3$ a $\omega_n = 0,2$), kde je na vstup tolerance přivedena hodnota $0,1$.

Algoritmus detekce ustálení funguje, ale pro správnou funkčnost je potřeba volit vhodnou vzorkovací periodu, aby funkční blok nereagoval také na lokální extrémy signálu.

Kapitola 3

Analýza funkčního bloku predikce

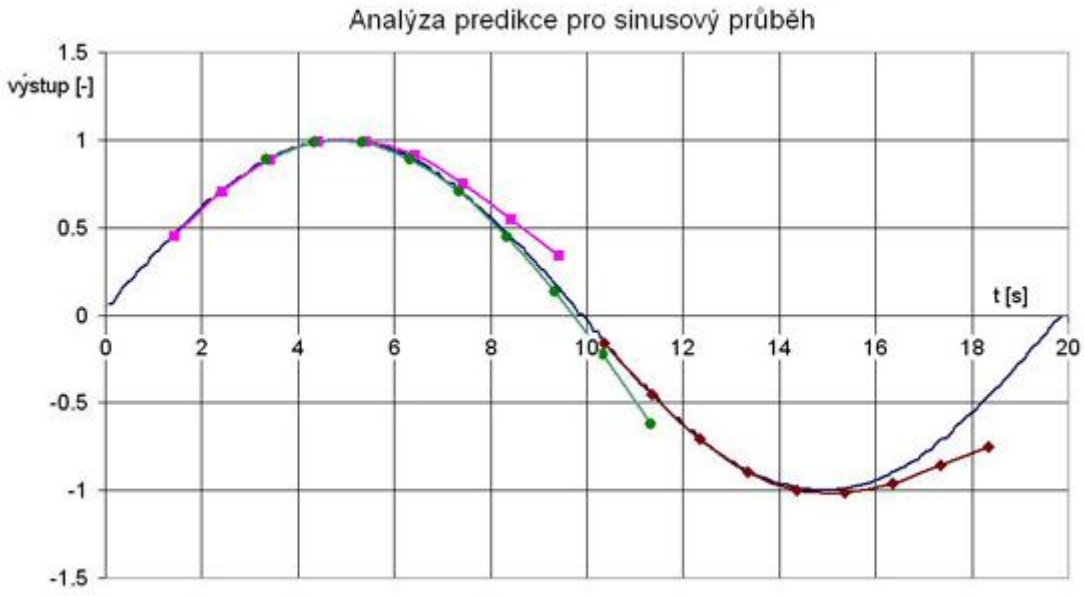
Použitelnost funkčního bloku predikce budeme zkoušet na čtyřech generovaných signálech, a to sinusovém, exponenciálním, odezvě soustavy 1.řádu a odezvě soustavy 2.řádu.

3.1 Sinusový průběh

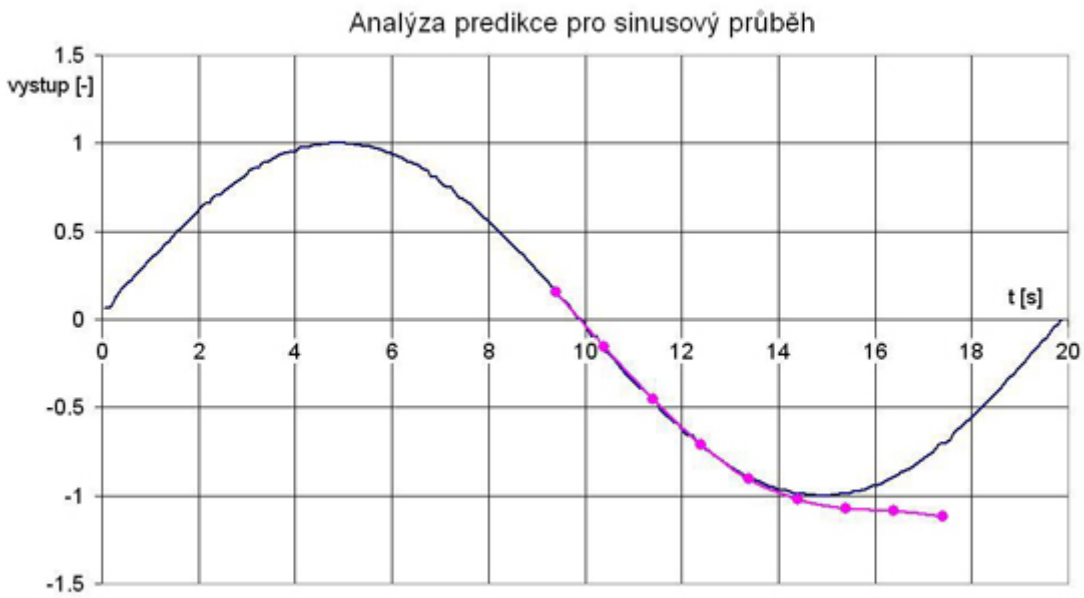
První signál, na kterém algoritmus predikce vyzkoušíme, bude sinusový. Zaměříme se na schopnost předpovědět vrchol a na chování predikovaného průběhu před inflexním bodem signálu. Použijeme dobu periody 20 sekund a diskrétní čas pro blok predikce 1 sekundu.

Z obrázku 3.1 je patrné, že predikátor dokáže předpovědět vrchol signálu a jeho následné klesání. Předpověď je přesnější, čím je okamžik predikce k vrcholu blíže. Na druhém obrázku 3.2 je zajímavá skutečnost, že funkční blok dokáže předpovědět změnu konvexnosti a konkávnosti před inflexním bodem.

Z výsledků lze říci, že pro čtyři nejbližší kroky je signál přesný a posléze se od průběhu odchyluje.



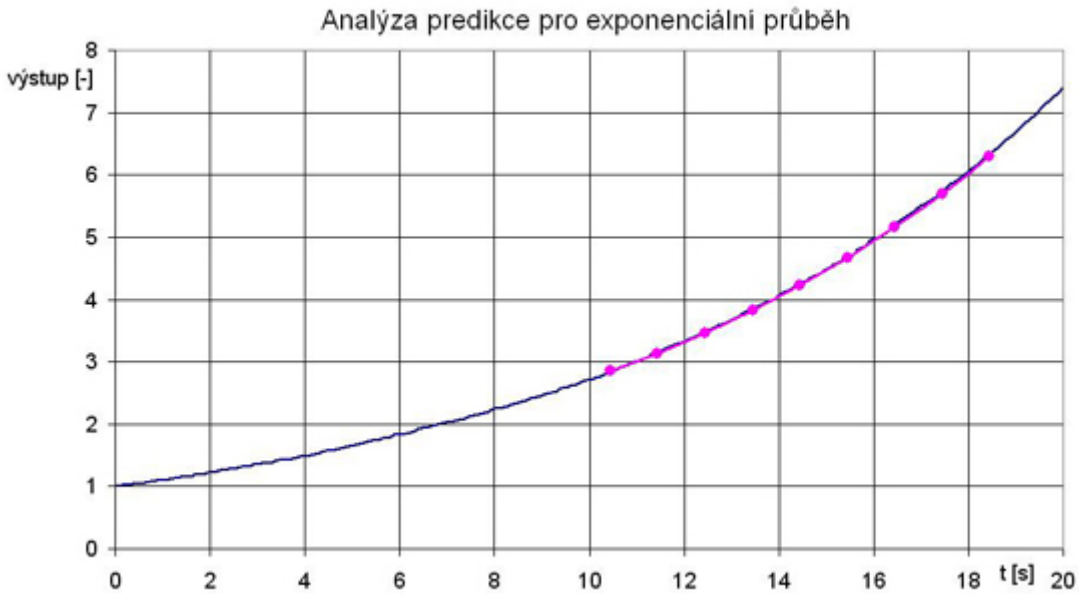
Obrázek 3.1: Analýza predikce pro sinusový průběh



Obrázek 3.2: Analýza predikce pro sinusový průběh

3.2 Exponenciální průběh

Další signál, na kterém vyzkoušíme predikci průběhu, bude exponenciální funkce. Jako exponent jsem zvolil dostatečně malé číslo, aby se funkční blok stihl adaptovat a predikovat v přiměřeném intervalu hodnot pro zobrazení. Vzhledem k charakteru funkce lze tvrdit, že okamžik predikce nebude mít vliv na vlastnosti predikovaného průběhu. Jako exponent jsem zvolil hodnotu $0,5$ a jako diskrétní čas pro blok predikce opět 1 sekundu.

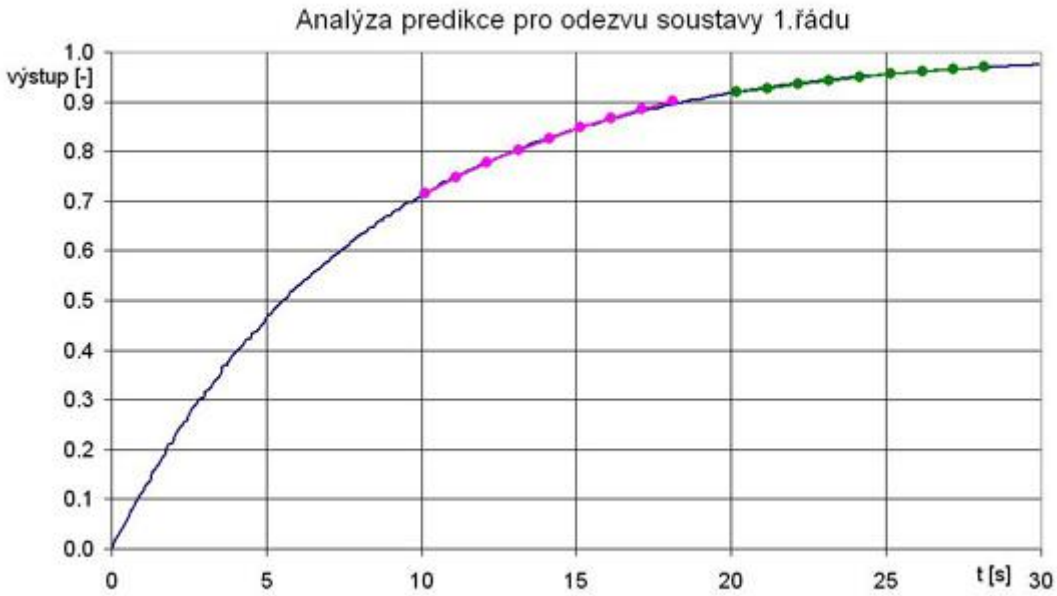


Obrázek 3.3: Analýza predikce pro exponenciální průběh

Z obrázku 3.3 lze usoudit, že funkční blok umí výborně predikovat exponenciální průběh.

3.3 Odezva soustavy 1.řádu

Další důležitou vlastností funkčního bloku bude, jak umí predikovat odezvu soustavy 1.řádu. Jako hodnotu τ použiji 8 sekund a vstupem bude skok o jednotkové velikosti. Jako diskrétní čas pro blok predikce použiji opět 1 sekundu.

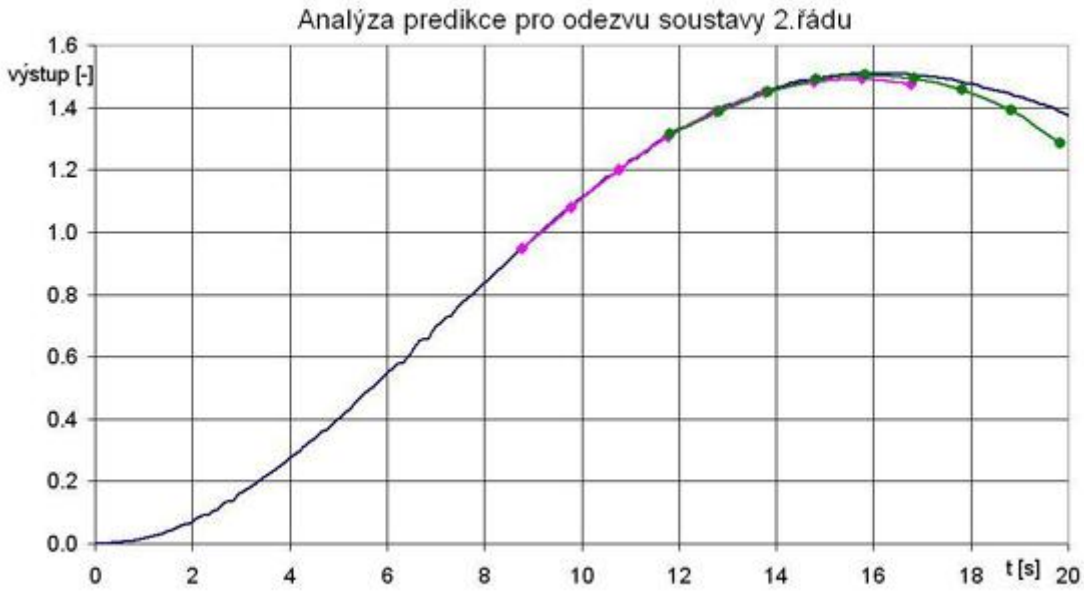


Obrázek 3.4: Analýza predikce pro odezvu soustavy 1.řádu

Z grafu 3.4 lze tvrdit, že funkční blok zvládá predikovat průběh odezvy soustavy 1.řádu výborně. Hodnoty pro sedm budoucích kroků vycházejí velmi dobře, hodnota osmého kroku se již mírně odchyluje.

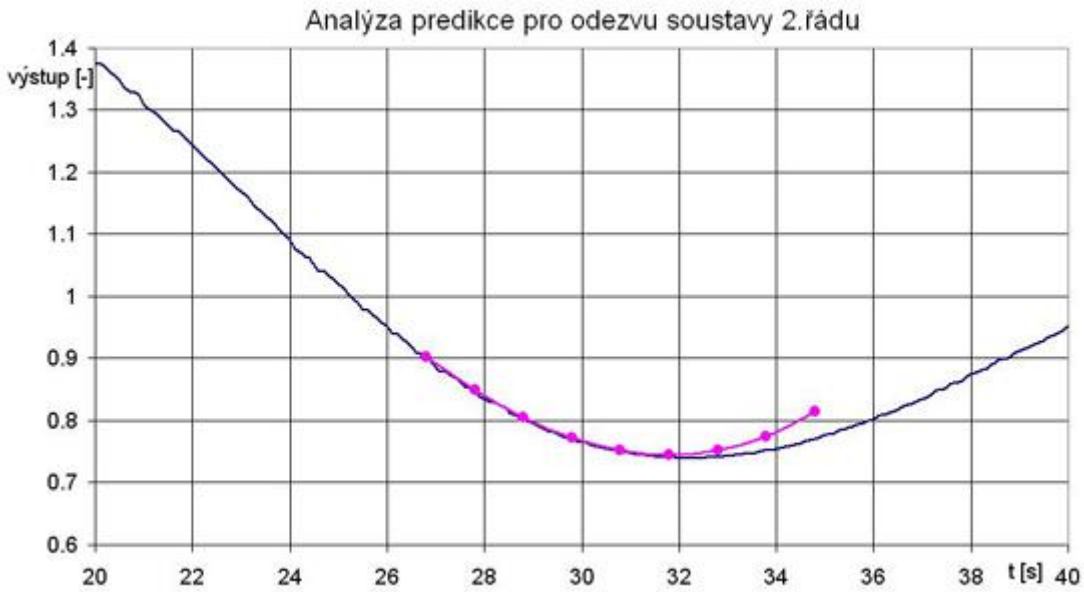
3.4 Odezva soustavy 2.řádu

Jako poslední test vyzkoušíme predikování odezvy soustavy 2.řádu. Zvolil jsem hodnoty tlumení a vlastní frekvence tak, abychom mohli vyšetřit předpověď prvého maxima. Poté také vyšetříme předpověď průběhu predikovaného signálu při kmitání soustavy. Za hodnotu tlumení jsem zvolil $\varsigma = 0.3$ a vlastní frekvenci nastavil na hodnotu $\omega_n = 0.2$ rad/s. Jako diskrétní čas pro blok predikce použiji opět 1 sekundu.



Obrázek 3.5: Analýza predikce pro odezvu soustavy 1.řádu

Z výsledků je patrné, že funkční blok predikce předpoví správně hodnoty průběhu pro budoucích pět kroků, zbývající tři se však již odchylují. Funkční blok predikuje za vrcholem strmější průběh než jak soustava reaguje ve skutečnosti, přesto lze říci, že pozná blížící se vrchol.



Obrázek 3.6: Analýza predikce pro odezvu soustavy 1.řádu

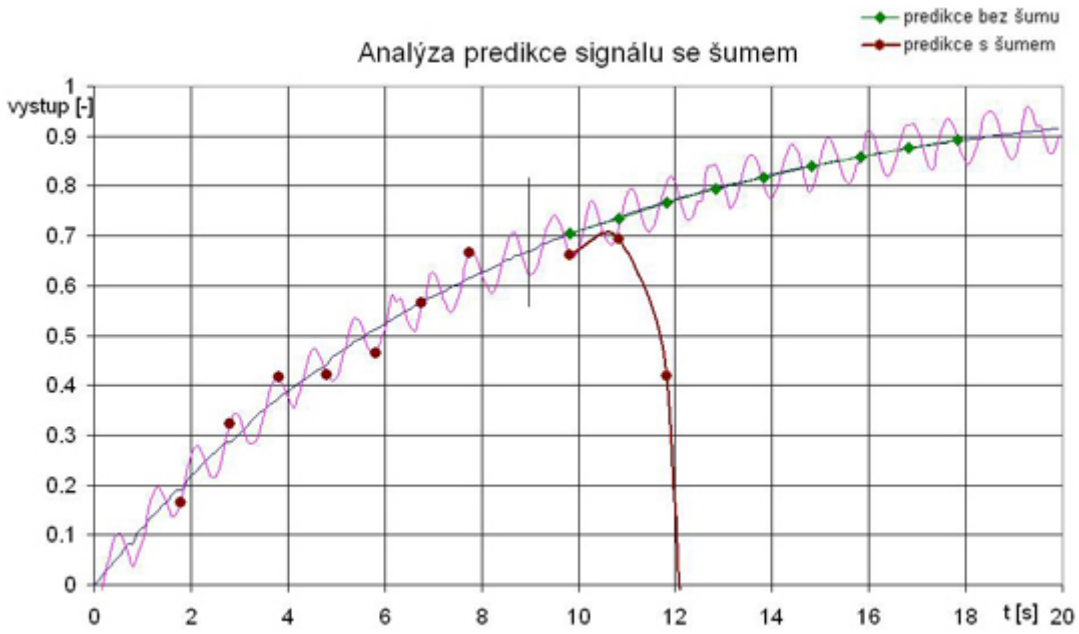
3.5 Zašuměný signál

Doposud jsme v analýze funkčního bloku vycházeli z dokonalého, čistého signálu. Nyní vyzkoušíme algoritmus predikce pro zašumělý signál. Šum bude představovat malý sinusový signál necelistvého násobku vzorkovací periody pro predikátor.

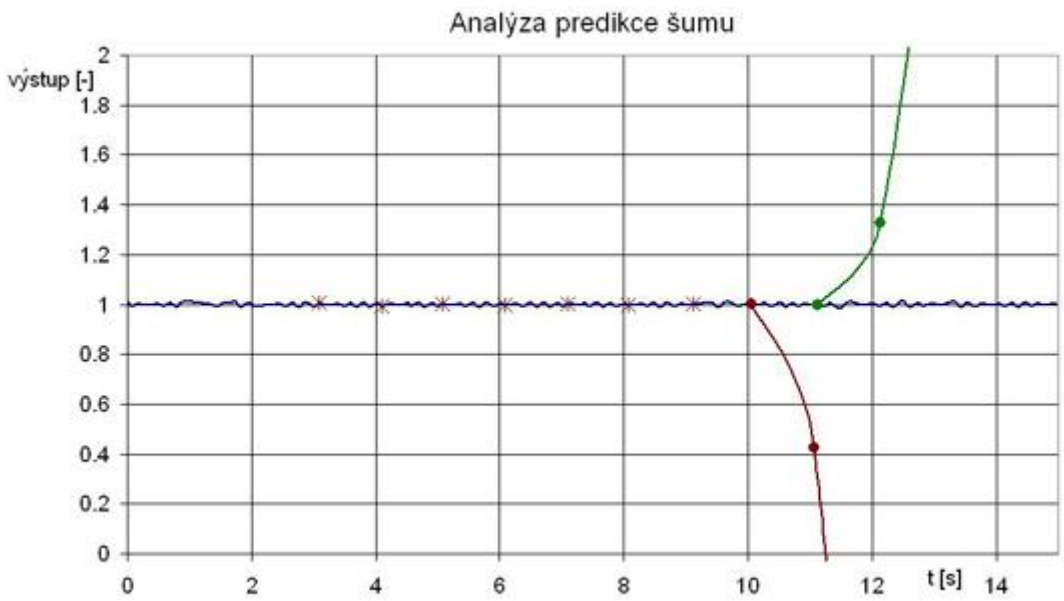
Pro první pokus přičteme šumový signál k průběhu generátoru odezvy soustavy 1. řádu, kde funkční blok vykazoval výborné výsledky. Jako u analýzy dokonalého signálu jsem nastavil hodnotu τ rovnu 8 sekundám, dobu periody šumového signálu na 0.821 sekund a amplitudu rovnu 0.05. Také diskrétní čas pro blok predikce jsem použil opět 1 sekundu.

Výsledky jsou velice znepokojivé. Šum interpolovaný na ideálním vstupu způsobí, že algoritmus predikce je absolutně nepoužitelný. Červené body před okamžikem predikce značí body adaptace. Graf obsahuje pro srovnání predikovaný průběh z ideálního průběhu. Obrázek 3.7 nám tedy ukazuje na nutnost použití filtrů.

Nyní zkusíme testovat, jak malý šum je ještě nežádoucí. Podíváme se, co způsobí šum o velikosti jednoho procenta signálu. Testovací signál bude konstantní hodnoty rovné jedné s přičteným sinusovým průběhem s amplitudou rovné 0.01 a době periodě rovné 0.223 sekund.



Obrázek 3.7: Analýza predikce signálu se šumem



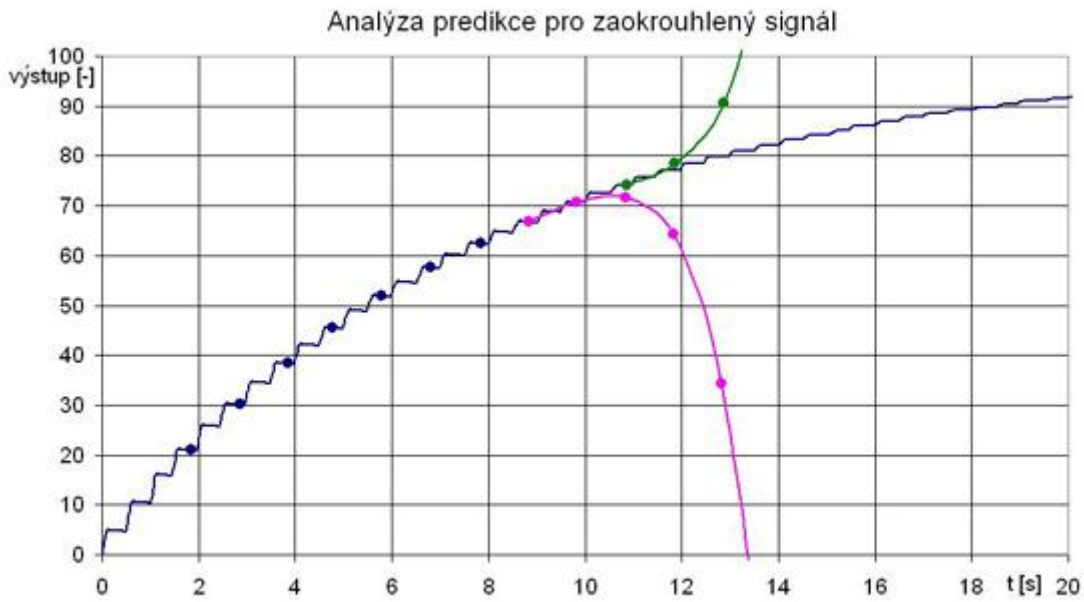
Obrázek 3.8: Analýza predikce šumu

I tak malý šum dokáže zcela znehodnotit algoritmus predikce. Funkční blok nedokáže predikovat jak budoucí hodnoty průběhu, tak ani směr vývoje průběhu. V každém

okamžiku predikce se předpovědi zcela rozcházejí.

3.6 Zaokrouhlení hodnot

Vyzkoušíme odstranit šum zaokrouhlením výsledku. Budeme testovat průběh odezvy soustavy 1.řádu na skokovou změnu žádané hodnoty, kde výsledky zaokrouhlíme na jedno desetinné místo. Žádanou hodnotu jsem nastavil na 100 a budeme sledovat predikovaný vývoj signálu. Hodnota τ pro odezvu soustavy 1.řádu je opět 8 a diskrétní čas pro blok predikce opět 1 sekund1.



Obrázek 3.9: Analýza predikce pro zaokrouhlený signál

Z grafu 3.9 lze konstatovat, že i když funkční blok získává vzorky signálu pro adaptaci s rostoucí množinou, predikovaný vývoj signálu se zcela rozchází se skutečností a navíc v každém okamžiku se mezi sebou rozchází i předpovědi vývoje.

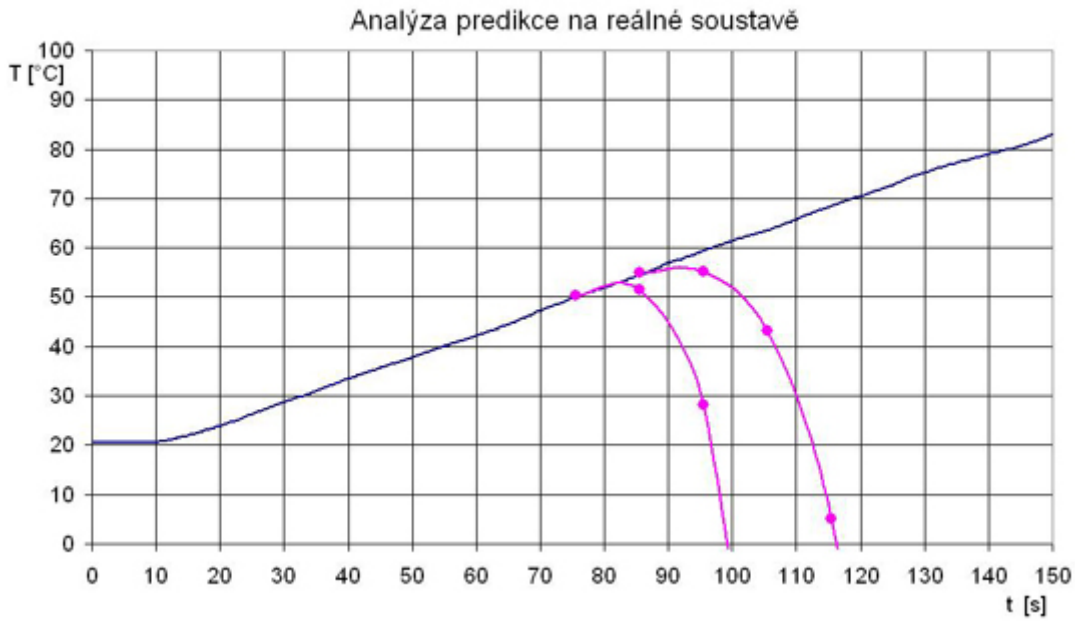
Podle výsledků simulací pro zašumělý signál si myslím, že algoritmus predikce funguje, ale je velice citlivý na přesné hodnoty signálu. Předpovídá správně pouze ty průběhy, jež se dají vyjádřit pomocí polynomiální diferenční rovnice maximálně osmého řádu přesně nebo zbytek jest zanedbatelný.

3.7 Reálný průběh

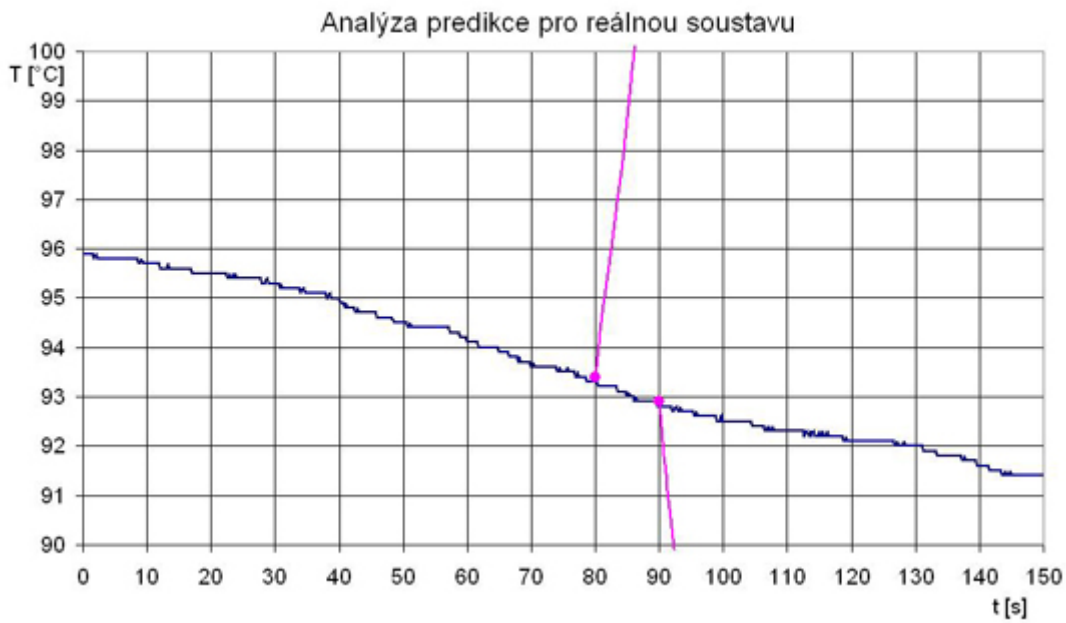
Získal jsem k dispozici data z reálného měření tepelné soustavy. Soustava byla složena z varné konvice a teplotního čidla Pt 1000. Vyzkoušíme tedy funkční blok predikátoru na těchto reálných datech.

Reálná data jsem do prostředí Mosaic dostal tak, že jsem vytvořil datové pole naplněné hodnotami z měření po desetinách sekund. V simulaci jsem využil nultý bit časového registru S13, který obsahuje pulzy s periodou jedné desetiny sekundy. Při každém pulzu jsem inkrementoval ukazatele do pole dat.

Zkusil jsem predikovat vývoj z průběhu pro ohřev (obrázek 3.10) a poté pro chladnutí vody (obrázek 3.11). Vzorkovací frekvenci pro predikátor jsem zvolil 10 sekund.



Obrázek 3.10: Analýza predikce pro reálnou soustavu



Obrázek 3.11: Analýza predikce pro reálnou soustavu

Z výsledků vyplývá, že funkční blok pro predikování průběhu reálné tepelné soustavy naprosto selhal. Jeho předpovědi absolutně nesouhlasí se skutečností.

Kapitola 4

Závěr

Algoritmus diferenční polynomiální predikce se zdál být velice slibný. Navrhl jsem funkční blok predikce, který implementoval tento algoritmus a s dalšími funkčními bloky, které buď generovali požadované průběhy, filtrovali nebo analyzovaly svůj vstupní signál, jsem je uzavřel do knihovny Predikt_Lib_20080406. Tato knihovna je navržena podle normy IEC 61 113.

Výsledky analýzy na ideálních generovaných signálech dopadly velice slibně. Ukázalo se, že po osmi dobách adaptace, je funkční blok schopen předpovídat budoucí vývoj signálu. Nejlépe dokázal předpovídat průběh exponenciály a odezvy soustavy 1.řádu. U odezvy soustavy 2.řádu a sinusovky dokázal odhadovat budoucí vývoj, i když vzdálenější kroky predikce se od skutečného průběhu již odchylovali.

Bohužel při testování signálu s nasimulovaným šumem se odhalila veliká citlivost funkčního bloku na šum. I malé kolísání kolem požadované hodnoty způsobilo naprosté znehodnocení algoritmu predikce. I po značné snaze algoritmus vylepšit a několika marných pokusech, se mi nepodařilo problém odstranit. Funkční blok predikce selhal i při použití dat z reálného měření tepelné soustavy, kde šum byl potlačen.

Znalost budoucího vývoje veličiny by mohla být velice užitečná, například pro soustavy s dopravním zpožděním, kde není možnost měřit nezpožděný signál. Muselo by se ovšem použít velice dobrých filtrů, neboť i malé zakolísání způsobí velké chyby predikce. Je otázka, zda by tak kvalitní filtry nezpůsobily značné zpoždění signálu, že by blok predikce ztrácel smysl.

Funkční blok predikce by tedy mohl být teoreticky použit pro předpovídání budoucího vývoje signálu lineárních systémů, ale prakticky se ovšem jeví jako zcela nepoužitelný.

Mnou navržená funkční knihovna Predikt_Lib_20080406 je připravena k použití, ale jsem si vědom, že k bezproblémovému nasazení v praxi je potřeba vyřešit problém s citlivostí funkčního bloku predikce. Tato úloha mi může být v budoucnu zadána k dopracování nebo může být svěřena druhé osobě. Proto je v příloze mé bakalářské práce přiložen manuál knihovny, zdrojový kód a CD obsahující potřebné materiály, včetně naprogramované knihovny Predikt_Lib_20080406.

Literatura

[1] KOZDERKA, Jan. *Prediktivní algoritmus pro PLC Tecomat a aplikace v praxi* (diplomová práce), 2004. 50 s. Liberec: Technická univerzita v Liberci.

[2] BUBENÍK, František, PULTAR, Milan, PULTAROVÁ, Ivana. *Matematické vzorce a metody.* 1. vyd. Praha : ČVUT, 1994. 282 s. ISBN 80-01-01164-X.

[3] ZOUBEK, Martin. *Řídící systémy pro stroje, procesy a budovy* [online]. 2005-2008 , 13.4.2008 [cit. 2008-03-01]. Dostupný z: *www.tecomat.cz*

[4] *Teco DVD INFO 10/2007*, Teco a. s., Kolín, 2007

[5] FRANKLIN, Gene F., POWELL, J. David, EMAMI-NAEINI, Abbas. *Feedback control of dynamic systems.* 5th edition. New Jersey : PEARSON Prentice Hall, 2006. 900 s. ISBN 0-13-149930-0.

[6] BAYER, Jiří, ŠEBEK, Zdeněk, PÍŠA, Pavel. *Počítače pro řízení.* 1. vyd. Praha : ČVUT, 2002. 271 s.

Přílohy

A - Manuál knihovny Predikt_Lib_20080406

B - Zdrojový kód knihovny Predikt_Lib_20080406

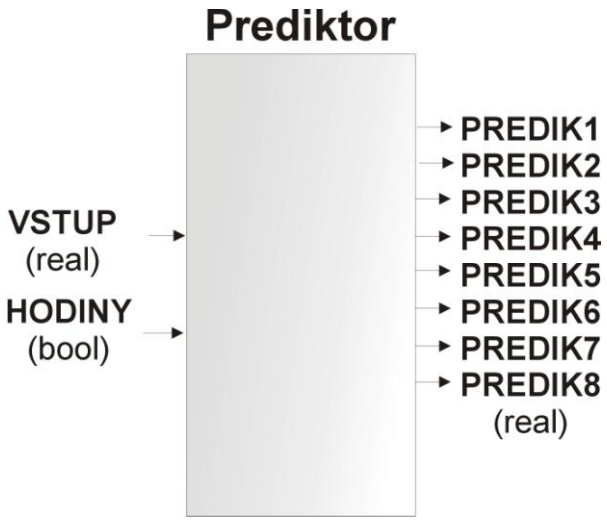
C - CD s elektronickými dokumenty a knihovnou Predikt_Lib_20080406

Manuál knihovny Predikt_Lib_20080406

Obsah:

1. Prediktor

Funkční blok Prediktor předpovídá budoucích osm kroků vývoje signálu, jež má přiveden na vstup. Budoucí kroky počítá pomocí algoritmu polynomiální diferenční predikce. K výpočtu používá hodnotu signálu v současném kroku a sedmi minulých.



Obr. 1 - Funkční blok Prediktor

Vstupem bloku je signál, jehož budoucí vývoj se má předvídat.

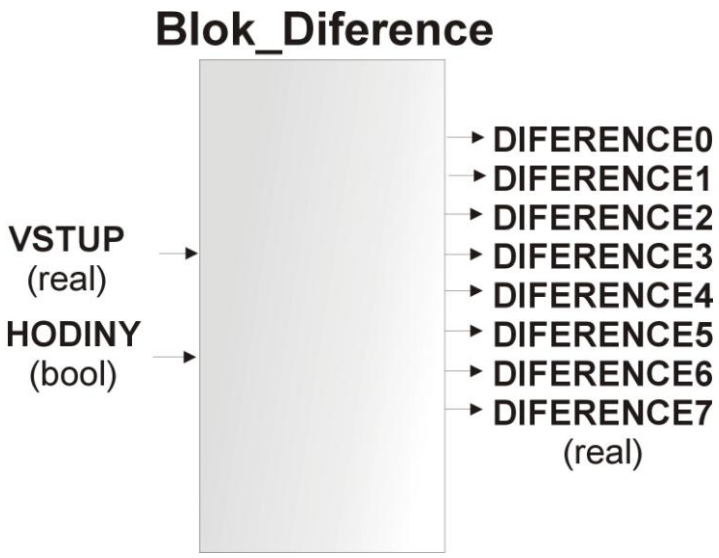
Dalším vstupem bloku je hodinový signál, který určuje vzorkovací okamžiky. Funkční blok reaguje na náběžnou hranu tohoto hodinového signálu.

Výstupem je osm predikovaných budoucích hodnot.

Upozornění: Funkční blok dokáže predikovat správný průběh až po sedmi krocích adaptace!

2. Blok_Diference

Funkční blok Blok_Diference je vlastně první polovina funkčního bloku Prediktoru. Funkční blok z hodnot vstupního signálu ze současného a sedmi minulých kroků spočte diference do sedmého řádu, které se objeví na výstupu. Funkční blok je možno přímo spojit s funkčním blokem Blok_Predikce.



Obr. 2 – Funkční blok Blok_Diference

Vstupem bloku je signál, jehož diference se mají spočítat.

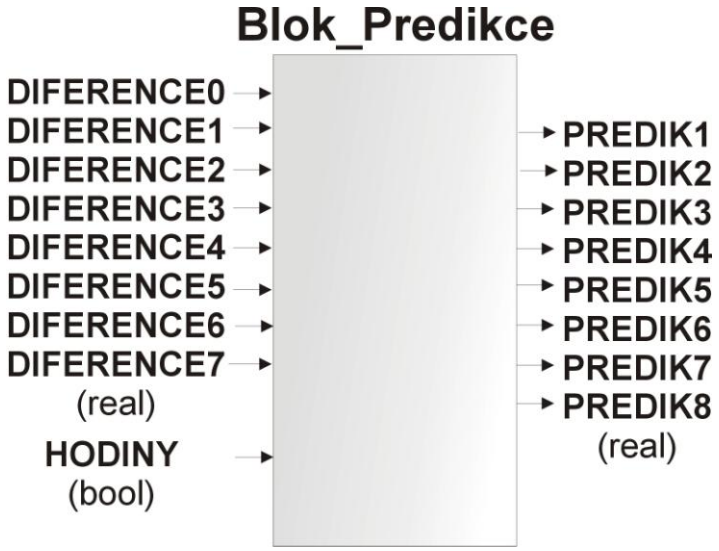
Dalším vstupem je hodinový signál, který určuje vzorkovací okamžiky. Funkční blok reaguje na náběžnou hranu tohoto hodinového signálu.

Výstupem je osm hodnot, které představují hodnoty spočtených diferencí Výstup DIFERENCE0 je hodnota současného vstupu.

Upozornění: Funkční blok dokáže spočítat správnou hodnotu n-té diference až po n krocích adaptace. Tedy správné hodnoty až do sedmé diference jsou správně spočteny nejdříve až v osmém kroce!

3. Blok_Predikce

Funkční blok Blok_Predikce je vlastně druhou polovinou funkčního bloku prediktoru. Funkční blok z hodnot diferencí, přivedeným na vstup, predikuje osm hodnot v budoucích osmi výpočetních krocích. Funkčním blok je možno přímo navázat na funkční blok Blok_Diference.



Obr. 3 – Funkční blok Blok_Predikce

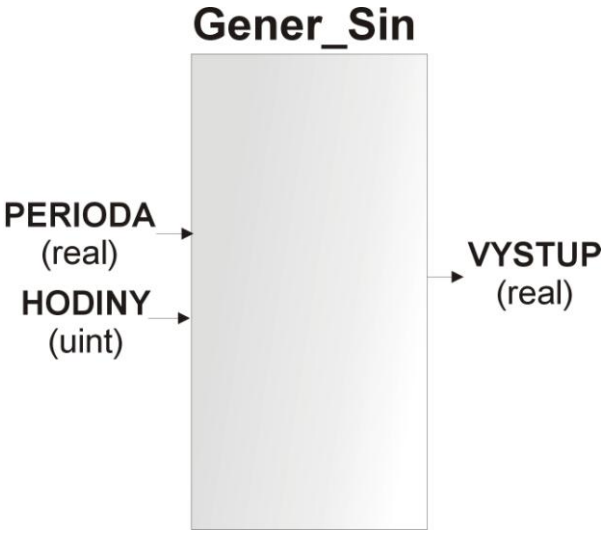
Vstupem bloku je tedy osm hodnot diferencí. Vstup DIFERENCE0 očekává hodnotu signálu v současném kroce.

Dalším vstupem je hodinový signál, který startuje výpočet. Funkční blok reaguje na náběžnou hranu tohoto hodinového signálu.

Výstupem je osm predikovaných budoucích hodnot.

4. Gener_Sin

Funkční blok Gener_Sin na svém výstupu generuje sinusový průběh s periodou žádanou na vstupu. Funkce sinus je počítána pomocí funkce sin( ) z knihovny StdLib_V18_20060404.



Obr.4 – Funkční blok Gener_Sin

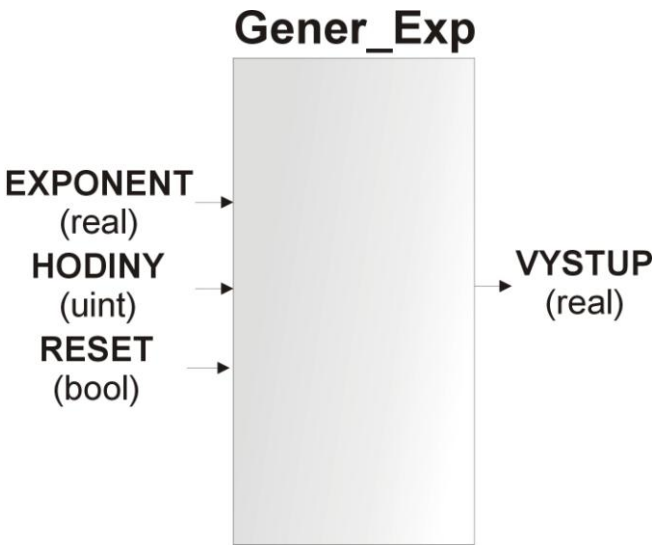
Vstup PERIODA představuje periodu signálu v sekundách. Perioda signálu se může pohybovat v rozmezí 0,1 sekundy do 6553,5 sekundy, což je přes 109 minut.

Vstupem je i hodinový registr, ze kterého se určují přesné časové okamžiky. Tento 16-ti bitový registr má být čítač desetin sekund a pomocí jeho se počítají časové rozdíly mezi jednotlivými otočkami programu automatu. Pro PLC Tecomat doporučuji registr SW14. Pokud budeme požadovat pomalejší sinusový signál, je možno na vstup HODINY přivést i čítač čítající jiné časové jednotky.

Výstupem je generovaný sinusový signál.

5. Gener_Exp

Funkční blok Gener_Exp na svém výstupu generuje exponenciální průběh s exponentem zadaným na vstupu. Exponenciální funkce je počítána pomocí funkce exp( ) z knihovny StdLib_V18_20060404.



Obr. 5 – Funkční blok Gener_Exp

Vstupem funkčního bloku EXPONENT se zadává požadovaný exponent pro časovou jednotku sekund.

Vstupem je i hodinový registr HODINY, ze kterého se určují přesné časové okamžiky. Tento 16-ti bitový registr má být čítač desetin sekund a pomocí jeho se počítají časové rozdíly mezi jednotlivými otočkami programu automatu. Pro PLC Tecomat doporučuji registr SW14. Pokud budeme požadovat pomalejší exponenciální průběh, je možno na vstup HODINY přivést i čítač čítající jiné časové jednotky.

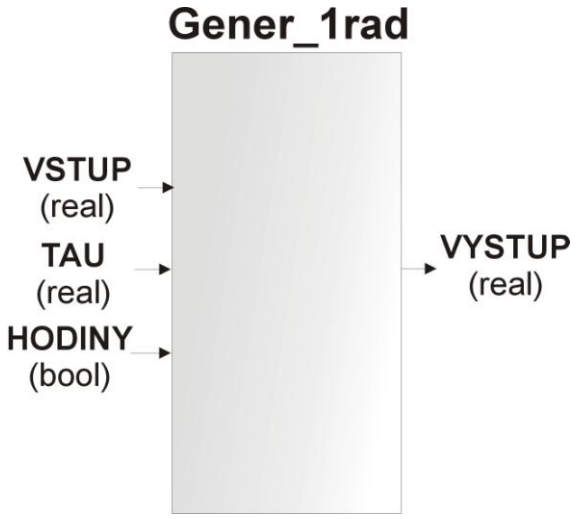
Dalším vstupem logického typu je RESET, který při hodnotě logické jedničky nastaví výstup do výchozí hodnoty rovné 1.

Výstupem je generovaný exponenciální průběh.

Upozornění: Funkční blok generuje exponencielu jen do okamžiku, kdy exponent násobený časem dosáhne hodnoty vyšší než 88, neboť poté by došlo k přetečení datového typu real.

6. Gener_1rad

Funkční blok Gener_1rad na svém výstupu generuje odezvu soustavy prvního řádu.



Obr. 6 – Funkční blok Gener_1rad

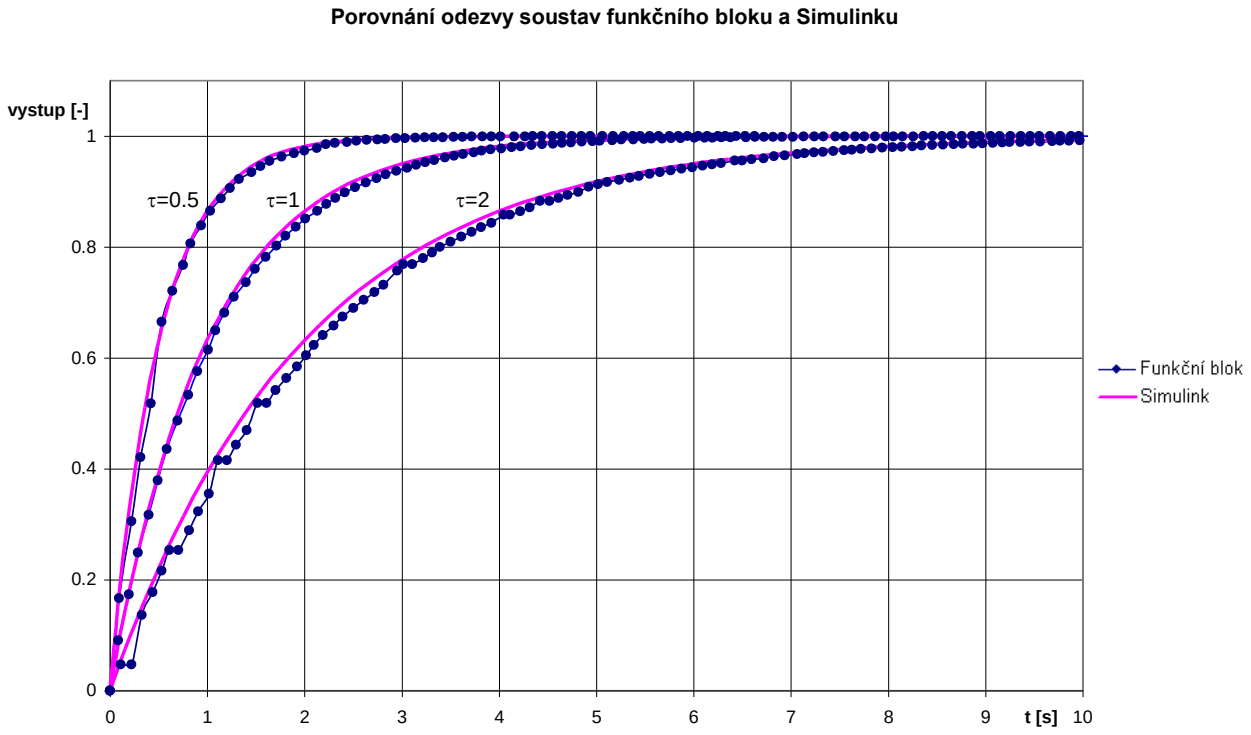
Na vstup VSTUP se přivádí požadovaná hodnota soustavy a na vstup TAU časová hodnota τ. Výpočet odezvy se počítá ze vztahu:

$$P(S) = \frac{Y(s)}{U(s)} = \frac{1}{\tau s + 1}$$

, kde Y(s) značí obraz výstupu a U(s) obraz vstupu a derivaci nahradí diference.

Dalším vstupem je hodinový signál, který určuje vzorkovací okamžiky. Funkční blok reaguje na náběžnou hranu tohoto hodinového signálu.

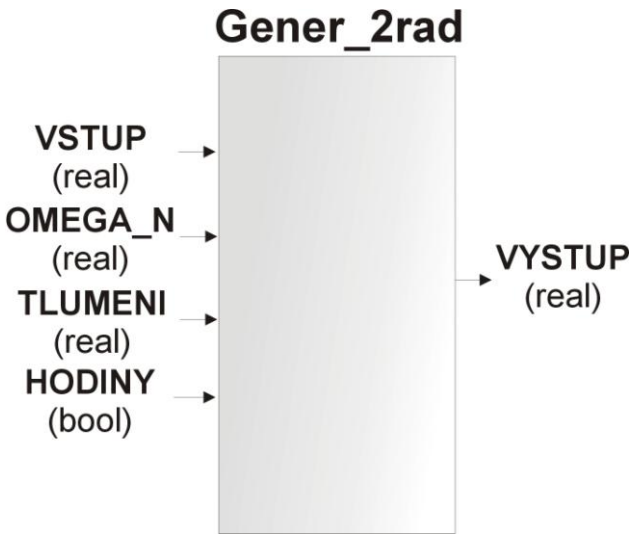
Výstupem je generovaná odezva soustavy prvního řádu. Obrázek číslo 7 ukazuje porovnání generované odezvy funkčním blokem a generované pomocí programu Matlab.



Obr.7 – Porovnání odezvy soustav funkčního bloku a Simulinku

7. Gener_2rad

Funkční blok Gener_2rad na svém výstupu generuje odezvu soustavy druhého řádu.



Obr. 8 – Funkční blok Gener_2rad

Na vstup VSTUP se přivádí požadovaná hodnota soustavy, na vstup OMEGA_N požadovaná vlastní frekvence soustavy a na vstup TLUMENI se přivádí požadované tlumení soustavy.

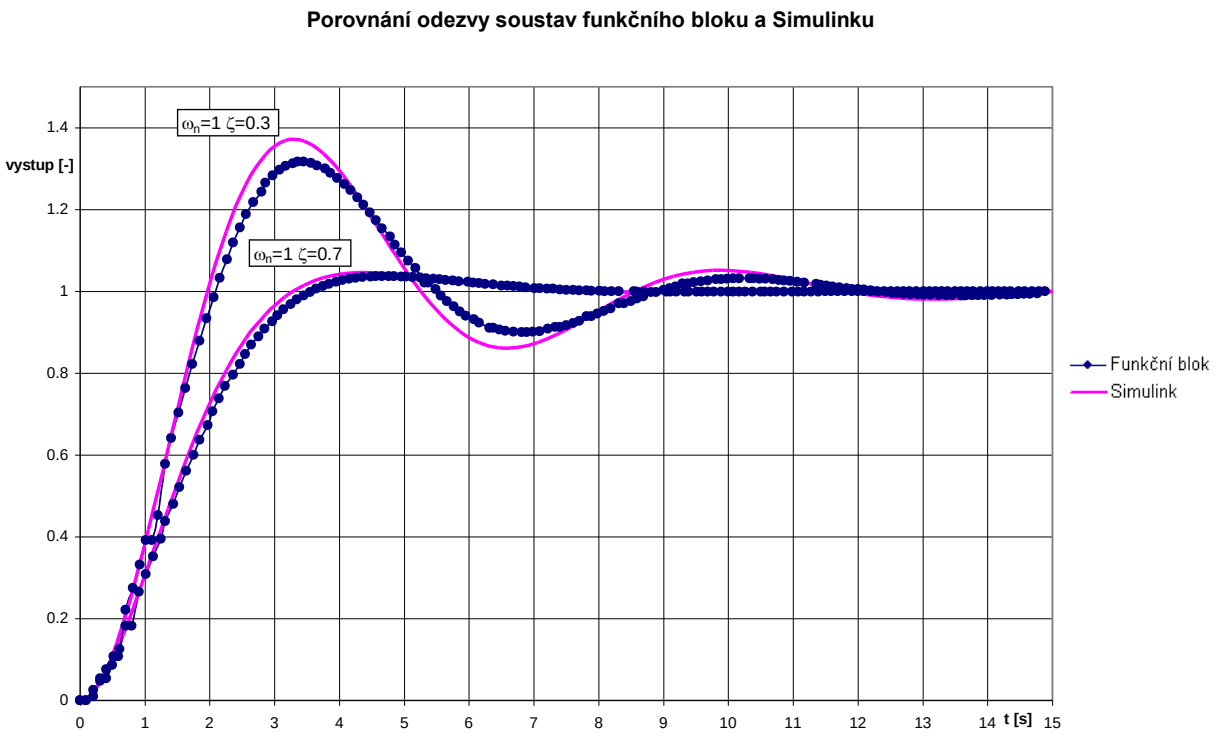
Výpočet odezvy se počítá ze vztahu:

$$P(S) = \frac{Y(s)}{U(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

, kde Y(s) značí obraz výstupu, U(s) obraz vstupu, ω_n vlatní frekvenci a ζ tlumení. Derivace nahradí diference.

Dalším vstupem je hodinový signál, který určuje vzorkovací okamžiky. Funkční blok reaguje na náběžnou hranu tohoto hodinového signálu.

Výstupem je generovaná odezva soustavy druhého řádu. Obrázek číslo 9 ukazuje porovnání generované odezvy funkčním blokem a generované pomocí programu Matlab.

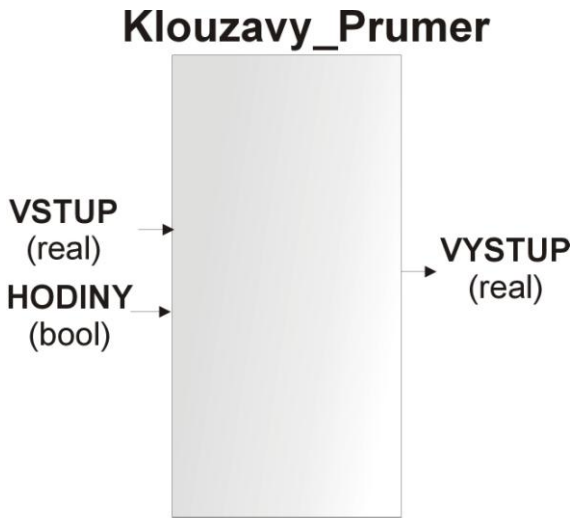


Obr.9 – Porovnání odezvy soustav funkčního bloku a Simulinku

Upozornění: Funkční blok generuje průběh odezvy soustavy druhého řádu se správnou vlastní frekvencí, ale více tlumený než je požadavek. Navíc funkčním blokem nelze generovat odezvu soustavy s nulovým tlumením. Odezva bude vždy tlumená.

8. Klouzavy_Prumer

Funkční blok Klouzavy_Prumer na svém výstupu hodnotu průměrné hodnoty posledních deseti vzorků ze signálu přivedeného na vstup. Při výpočtu průměru se ovšem vynechává nejmenší a největší hodnota.



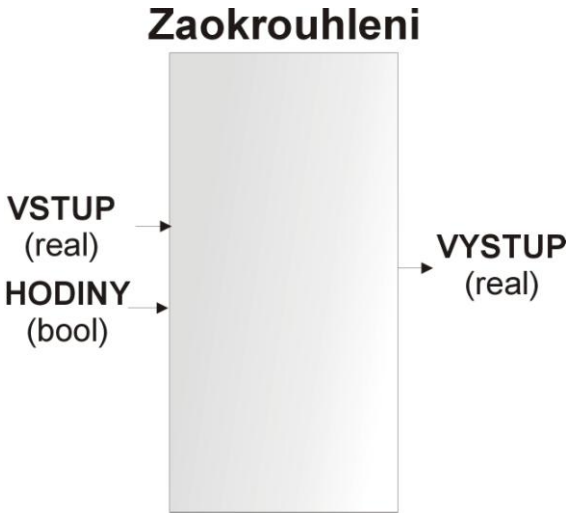
Obr.10 – Funkční blok Klouzavy_Prumer

Vstupem bloku je tedy signál, který chceme filtrovat a výstupem je jeho klouzavý průměr.

Dalším vstupem je hodinový signál, který určuje vzorkovací okamžiky. Funkční blok reaguje na náběžnou hranu tohoto hodinového signálu.

9. Zaokrouhleni

Funkční blok Zaokrouhlení má na svém výstupu zaokrouhlenou hodnotu vstupního signálu na jedno desetinné místo.

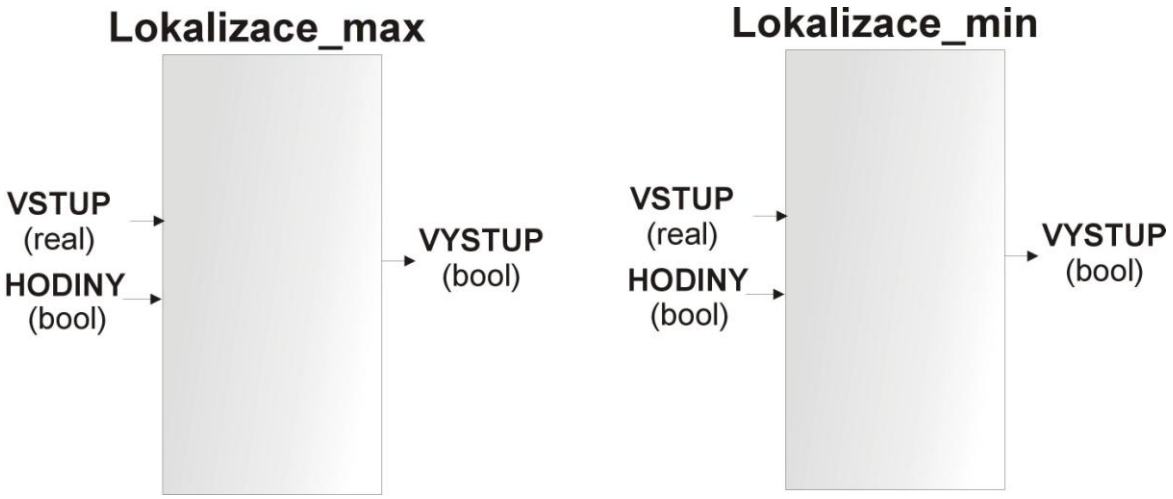


Obr.11 – Funkční blok Zaokrouhleni

Dalším vstupem je hodinový signál, který určuje vzorkovací okamžiky. Funkční blok reaguje na náběžnou hranu tohoto hodinového signálu.

10. Lokalizace_max a Lokalizace_min

Funkční blok Lokalizace_max (resp. Lokalizace_min) analyzuje signál přivedený na vstup a pomocí algoritmu polynomiální diferenční predikce detekuje lokální maxima (resp. minima) signálu.



Obr. 12 – Funkční bloky Lokalizace_max a Lokalizace_min

Algoritmus detekce je řešen pomocí predikce vývoje pro čtyři budoucí kroky a porovnání, zda hodnoty vývoje budou menší (resp. větší) než současná hodnota a zároveň zda i hodnota z minulého kroku je menší (resp. větší). Na vstup je tedy přiveden signál, který chceme analyzovat.

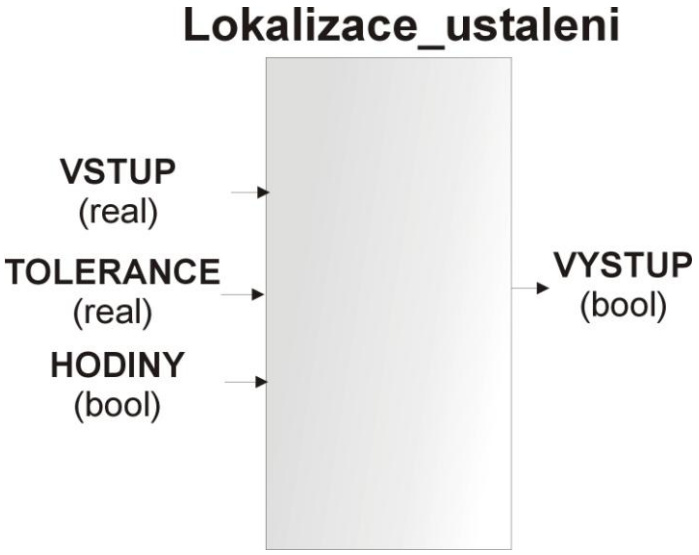
Dalším vstupem je hodinový signál, který určuje vzorkovací okamžiky. Funkční blok reaguje na náběžnou hranu tohoto hodinového signálu.

Výstupem je logická proměnná, na které se objeví logická jednička, pokud funkční blok detekuje lokální maximum (resp. minimum).

Upozornění: Protože funkční blok využívá algoritmus polynomiální diferenční predikce pro budoucí čtyři kroky, dokáže správný vývoj signálu předpovídat až po sedmi krocích adaptace!

11. Lokalizace_ustaleni

Funkční blok Lokalizace_ustaleni analyzuje signál přivedený na vstup a detekuje lokální ustálení signálu.



Obr. 13 – Funkční blok Lokalizace_ustaleni

Algoritmus detekce je řešen tak, zda posledních deset hodnot nepřekročilo toleranční pásmo, jehož střed je spočten jako průměrná hodnota signálu z těchto deseti hodnot. Hranice se spočítají jako střed plus/minus hodnota přivedená na vstup TOLERANCE.

Dalším vstupem je hodinový signál, který určuje vzorkovací okamžiky. Funkční blok reaguje na náběžnou hranu tohoto hodinového signálu.

Výstupem je logická proměnná, na které se objeví logická jednička, pokud funkční blok detekuje lokální ustálení v daném tolerančním pásu.

Upozornění: Je důležité zvolit hodinový signál uváženě tak, aby funkční blok nedetekoval i kolena signálu.

Zdrojový kód knihovny Predikt_Lib_20080406

```
//Funkcni blok prediktor predpovida budoucich osm kroku ze soucasneho a
//7 minulych kroku. Algoritmus je zalozen na polynomialni diferencni predikci.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu. Vystupem je osm predikovanych
//budoucich hodnot.
FUNCTION_BLOCK Prediktor
VAR_INPUT
 //vstupni signal
 vstup          :    real;
 //hodinovy vstup - nabezna hrana
 hodiny         :    bool;
 END_VAR
 VAR
 //promenna hodiny z predchoziho cyklu
 minule_hodiny  :    bool;

 //diference pro adaptaci
 dif_adapt0     :    real;
 dif_adapt1     :    real;
 dif_adapt2     :    real;
 dif_adapt3     :    real;
 dif_adapt4     :    real;
 dif_adapt5     :    real;
 dif_adapt6     :    real;
 dif_adapt7     :    real;

 //diference pro predikci
 dif_predik0    :    real;
 dif_predik1    :    real;
 dif_predik2    :    real;
 dif_predik3    :    real;
 dif_predik4    :    real;
 dif_predik5    :    real;
 dif_predik6    :    real;
 dif_predik7    :    real;
 END_VAR
 VAR_OUTPUT
 //vystupem jsou predikovane hodnoty pro budoucich 8 kroku
 predik1        :    real;
 predik2        :    real;
 predik3        :    real;
 predik4        :    real;
 predik5        :    real;
 predik6        :    real;
 predik7        :    real;
 predik8        :    real;
 END_VAR
```

```
VAR_TEMP
//promenna pro FOR cyklus
i                    :        sint;
END_VAR



//detekce hrany, pri ktere se provede predikce
 IF hodiny=TRUE AND minule_hodiny=FALSE THEN

//adaptacni cast
//vypocet novych hodnot pro predikci
dif_predik0:=vstup;
dif_predik1:=dif_predik0-dif_adapt0;
dif_predik2:=dif_predik1-dif_adapt1;
dif_predik3:=dif_predik2-dif_adapt2;
dif_predik4:=dif_predik3-dif_adapt3;
dif_predik5:=dif_predik4-dif_adapt4;
dif_predik6:=dif_predik5-dif_adapt5;
dif_predik7:=dif_predik6-dif_adapt6;

//kopie novych diferenci do pole pro adaptaci
dif_adapt0:=dif_predik0;
dif_adapt1:=dif_predik1;
dif_adapt2:=dif_predik2;
dif_adapt3:=dif_predik3;
dif_adapt4:=dif_predik4;
dif_adapt5:=dif_predik5;
dif_adapt6:=dif_predik6;
dif_adapt7:=dif_predik7;

//Cyklus predikcni casti
FOR i:=1 TO 8 DO
 dif_predik6:=dif_predik7+dif_predik6;
 dif_predik5:=dif_predik6+dif_predik5;
 dif_predik4:=dif_predik5+dif_predik4;
 dif_predik3:=dif_predik4+dif_predik3;
 dif_predik2:=dif_predik3+dif_predik2;
 dif_predik1:=dif_predik2+dif_predik1;
 dif_predik0:=dif_predik1+dif_predik0;

CASE i OF
 1:  predik1:=dif_predik0;
 2:  predik2:=dif_predik0;
 3:  predik3:=dif_predik0;
 4:  predik4:=dif_predik0;
 5:  predik5:=dif_predik0;
 6:  predik6:=dif_predik0;
 7:  predik7:=dif_predik0;
 8:  predik8:=dif_predik0;
```

```
  END_CASE;
  END_FOR;

  END_IF;
  //ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
  minule_hodiny:=hodiny;

END_FUNCTION_BLOCK

//Funkcni blok Blok_Diference je vlastne prvni polovina funkcniho bloku
//prediktoru. Funkcni blok ze soucasneho a ze 7 minulych kroku spocte diference
//do sedmeho radu, ktere se objevi na vystupu.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu.
FUNCTION_BLOCK Blok_Diference
VAR_INPUT
  //vstupni signal
  Vstup              :      real;
  //hodinovy vstup - nabezna hrana
  hodiny             :      bool;
  END_VAR
  VAR
  //promenna hodiny z predchoziho cyklu
  minule_hodiny      :      bool;

  //diference pro adaptaci
  dif_adapt0         :      real;
  dif_adapt1         :      real;
  dif_adapt2         :      real;
  dif_adapt3         :      real;
  dif_adapt4         :      real;
  dif_adapt5         :      real;
  dif_adapt6         :      real;
  dif_adapt7         :      real;
  END_VAR
  VAR_OUTPUT
  //vystupem jsou spoctene diference do sedmeho radu
  diference0         :      real;
  diference1         :      real;
  diference2         :      real;
  diference3         :      real;
  diference4         :      real;
  diference5         :      real;
  diference6         :      real;
  diference7         :      real;
  END_VAR
  VAR_TEMP
  END_VAR
```

```
 //detekce hrany, pri ktere se provede predikce
  IF hodiny=TRUE AND minule_hodiny=FALSE THEN

 //adaptacni cast
 //vypocet novych hodnot diferenci
 diference0:=vstup;
 diference1:=diference0-dif_adapt0;
 diference2:=diference1-dif_adapt1;
 diference3:=diference2-dif_adapt2;
 diference4:=diference3-dif_adapt3;
 diference5:=diference4-dif_adapt4;
 diference6:=diference5-dif_adapt5;
 diference7:=diference6-dif_adapt6;

 //kopie novych diferenci do pole pro adaptaci
 dif_adapt0:=vstup;
 dif_adapt1:=diference1;
 dif_adapt2:=diference2;
 dif_adapt3:=diference3;
 dif_adapt4:=diference4;
 dif_adapt5:=diference5;
 dif_adapt6:=diference6;
 dif_adapt7:=diference7;

 END_IF;
 //ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
 minule_hodiny:=hodiny;

END_FUNCTION_BLOCK

//Funkcni blok Blok_predikce je vlastne druha polovina funkcniho bloku
//prediktoru. Funkcni blok z diferenci do sedmeho radu privedenych na vstup
//predikuje hodnoty pro budoucich osm kroku, ktere se pote objevi na vystupu.
//Algoritmus je zalozen na polynomialni diferencni predikci.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu.
FUNCTION_BLOCK Blok_predikce
VAR_INPUT
 //vstupem jsou diference do sedmeho radu
 diference0        :    real;
 diference1        :    real;
 diference2        :    real;
 diference3        :    real;
 diference4        :    real;
 diference5        :    real;
 diference6        :    real;
 diference7        :    real;
 //hodinovy vstup - nabezna hrana
 hodiny            :    bool;
 END_VAR
```

```
VAR
//promenna hodiny z predchoziho cyklu
minule_hodiny        :        bool;

//diference pro predikci
dif_predik0          :        real;
dif_predik1          :        real;
dif_predik2          :        real;
dif_predik3          :        real;
dif_predik4          :        real;
dif_predik5          :        real;
dif_predik6          :        real;
dif_predik7          :        real;
END_VAR
VAR_OUTPUT
//vystupem jsou predikovane hodnoty pro budoucich 8 kroku
predik1              :        real;
predik2              :        real;
predik3              :        real;
predik4              :        real;
predik5              :        real;
predik6              :        real;
predik7              :        real;
predik8              :        real;
END_VAR
VAR_TEMP
//promenna pro FOR cyklus
i                    :        sint;
END_VAR


//detekce hrany, pri ktere se provede predikce
 IF hodiny=TRUE AND minule_hodiny=FALSE THEN

 //prirazeni vstupu k promennym pouzitych v algoritmu
 dif_predik0:=diference0;
 dif_predik1:=diference1;
 dif_predik2:=diference2;
 dif_predik3:=diference3;
 dif_predik4:=diference4;
 dif_predik5:=diference5;
 dif_predik6:=diference6;
 dif_predik7:=diference7;

//Cyklus predikcni casti
FOR i:=1 TO 8 DO
 dif_predik6:=dif_predik7+dif_predik6;
 dif_predik5:=dif_predik6+dif_predik5;
 dif_predik4:=dif_predik5+dif_predik4;
```

```
 dif_predik3:=dif_predik4+dif_predik3;
 dif_predik2:=dif_predik3+dif_predik2;
 dif_predik1:=dif_predik2+dif_predik1;
 dif_predik0:=dif_predik1+dif_predik0;

CASE i OF
 1:  predik1:=dif_predik0;
 2:  predik2:=dif_predik0;
 3:  predik3:=dif_predik0;
 4:  predik4:=dif_predik0;
 5:  predik5:=dif_predik0;
 6:  predik6:=dif_predik0;
 7:  predik7:=dif_predik0;
 8:  predik8:=dif_predik0;
END_CASE;
END_FOR;

END_IF;
//ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
minule_hodiny:=hodiny;

END_FUNCTION_BLOCK

//Funkcni blok Gener_Sin na svem vystupu generuje sinusovy prubeh s periodou
//zadanou na vstupu. Perioda se udava v sekundach.
//Vstupem bloku je i hodinovy registr, ktery urcuje presne casove okamziky.
//Tento registr je citac desetin sekund a pomoci jeho se pocitaji casove
//rozdily mezi jednotlivymi otockami automatu. Pokud se dopocita do periody,
//vnitrni citac se vynuluje. Perioda signalu tedy muze byt teoreticky od 0.1
//sekundy do 6553.5 sekundy.
FUNCTION_BLOCK Gener_Sin
VAR_INPUT
  //perioda udana v sekundach
  Perioda          :      real;
  //registr s citacem desetin sekund %SW14!!!
  hodiny           :      uint;
 END_VAR
 VAR
  //Promenna cas si stale pocita cas od 0 do periody
  cas              :      real;
  //Promenna pro ulozeni hodnoty citace desetin sekund pro dalsi cykl
  minule_hodiny    :      dint;
 END_VAR
 VAR_OUTPUT
  //vystupem je generovany sinusovy signal
  vystup           :      real;
 END_VAR
```

```
 VAR_TEMP
  //Promenna rozdil_hodin je znamenkova z duvodu identifikace zaporneho
  //rozdilu, ktery ukazuje preteceni vnejsiho citace desetin sekund
  rozdil_hodin          :          dint;
  //promenna hodiny_VDint pro konverzi typu
  hodiny_VDint          :          dint;
 END_VAR

  //Ziskani rozdilneho casu mezi dvemi volanimi, pokud citac desetin sekund
  //pretekl, rozdil_hodin bude zaporny a k soucasnym hodinam se pricte cas,
  //kolik zbyval do konce
  hodiny_VDint:=uint_to_dint(hodiny);
  rozdil_hodin:=hodiny_VDint-minule_hodiny;
  IF rozdil_hodin<0 THEN
   rozdil_hodin:=(65535-minule_hodiny)+hodiny_VDint;
  END_IF;

  //K hodnote promenne cas pricteme rozdil hodin vydelen deseti
  //z duvodu poctu v sekundach
  cas:=cas+((dint_to_real(rozdil_hodin))/10.0);
  //Cas musi byt v rozmezi 0 az perioda
  IF (cas>perioda) THEN
     cas:=cas-perioda;
  END_IF;
  //Podminka z duvodu nebezpeci deleni nulou
  IF perioda>0.0 THEN
   vystup:=sin((6.283185307/perioda)*cas);  //(2PI/T)*t
  END_IF;

  //ulozeni hodnoty hodin z minuleho cyklu kvuli vyhodnoceni rozdilu casu
  minule_hodiny:=hodiny_VDint;
END_FUNCTION_BLOCK

//Funkcni blok Gener_Exp na svem vystupu generuje exponencialni prubeh s
//exponentem zadanym na vstupu. Exponent se udava pro casovou jednotku sekund.
//Vstupem bloku je i hodinovy registr, ktery urcuje presne casove okamziky.
//Tento registr je citac desetin sekund a pomoci jeho se pocitaji casove
//rozdily mezi jednotlivymi otockami automatu. Funkcni blok generuje
//exponencielu maximalne do exponentu rovneho 88, nebot pote by presahl rozsah
//datoveho typu real.
FUNCTION_BLOCK Gener_Exp
VAR_INPUT
   //exponent
   Exponent              :          real;
   //registr s citacem desetin sekund %SW14!!!
   hodiny                :          uint;
   //Promenna reset slouzi k vyresetovani promenne citac, ze
   //ktere se pocita exponenciala
   reset                 :          bool;
 END_VAR
```

```
VAR
//Promenna cas si stale pocita cas do vynulovani
cas                 :       real;
//Promenna pro ulozeni hodnoty citace desetin sekund pro dalsi cykl
minule_hodiny       :       dint;
//Promenna pro ulozeni
minuly_exponent     :       real;
END_VAR
VAR_OUTPUT
//vystupem je generovany exponencialni prubeh
vystup              :       real;
END_VAR
VAR_TEMP
 //Promenna exponent_novy slouzi k vypoctu exponentu nasobeneho casem
 exponent_novy      :       real;
 //Promenna rozdil_hodin je znamenkova z duvodu identifikace zaporneho rozdilu
 rozdil_hodin       :       dint;
 //promenna hodiny_VDint pro konverzi typu
 hodiny_VDint       :       dint;
END_VAR

 //z duvodu aby se pri kazdem cyklu mohla ulozit promenna minule_hodiny
 hodiny_VDint:=uint_to_dint(hodiny);

 IF minuly_exponent<>exponent OR reset=TRUE THEN
   cas:=0.0;
   vystup:=1.0;
 ELSE
 //Ziskani rozdilneho casu mezi dvemi volanimi, pokud citac desetin sekund
 //pretekl, rozdil_hodin bude zaporny a k soucasnym hodinam se pricte cas,
 //kolik zbyval do konce
 rozdil_hodin:=hodiny_VDint-minule_hodiny;
 IF rozdil_hodin<0 THEN
  rozdil_hodin:=(65535-minule_hodiny)+hodiny_VDint;
 END_IF;

 //K hodnote promenne cas pricteme rozdil hodin vydelen deseti
 //z duvodu poctu v sekundach
 cas:=cas+((dint_to_real(rozdil_hodin))/10.0);

 //Aby funkce neprekrocila rozsah typu real, musi byt exponent
 //mensi nez 88.
  exponent_novy:=exponent*cas;
  IF exponent_novy<88.0 THEN
  vystup:=exp(exponent_novy);
  END_IF;
 END_IF;
```

```
  //ulozeni hodnoty hodin z minuleho cyklu kvuli vyhodnoceni rozdilu casu
  minule_hodiny:=hodiny_VDint;
  minuly_exponent:=exponent;

END_FUNCTION_BLOCK

//Funkcni blok Gener_1rad na svem vystupu generuje odezvu soustavy prvniho
//radu na zadanou hodnotu privedenou na vstup. Vstupni promenna tau predstavuje
//casovou konstantu pro vypocet odezvy. Odezva se pocita ze vztahu
//P(s)=a/s+a, a=1/tau, kde se derivace nahradi diferenci.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu.
FUNCTION_BLOCK Gener_1rad
  VAR_INPUT
  //pozadovana hodnota
  vstup                 :       real;
  //Casova konstanta tau
  tau                   :       real;
  //hodinovy vstup - nabezna hrana
  hodiny                :       bool;
  END_VAR
  VAR
  //hodnota minuleho vystupu
  minuly_vystup         :       real;
  //promenna hodiny z predchoziho cyklu
  minule_hodiny         :       bool;
  END_VAR
  VAR_OUTPUT
  //vystupem je generovana odezva soustavy prvniho radu
  vystup                :       real;
  END_VAR
  VAR_TEMP
  //promenna a pro vyraz P(s)=a/s+a, a=1/tau.
   a                    :       real;
  END_VAR

   //detekce hrany, pri ktere se provede vypocet odezvy
   IF minule_hodiny=FALSE AND hodiny=TRUE AND tau>0.0 THEN
    //Protoze pouzivame krok po 0.1s, delime a deseti
    a:=0.1/(tau);
    vystup:=((a*vstup)+minuly_vystup)/(1.0+a);  //y(k)=(a*u(k)+y(k-1))/(1+a)
    minuly_vystup:=vystup;
   END_IF;

   //ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
   minule_hodiny:=hodiny;

END_FUNCTION_BLOCK
```

```
//Funkcni blok Gener_2rad na svem vystupu generuje odezvu soustavy druheho
//radu na zadanou hodnotu privedenou na vstup. Vstupni promenne tlumeni a
//omega_n predstavuji parametry pro vypocet odezvy. Odezva se pocita ze vztahu
//P(s)=omega_n^2/(s^2+2*tlumeni*omega_n*s+omega_n^2), kde se derivace nahradi
//diferencemi.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu.
FUNCTION_BLOCK Gener_2rad
  VAR_INPUT
  //pozadovana hodnota
  vstup                 :       real;
  //omega_n je vlastni frekvence
  omega_n               :       real;
  //tlumeni systemu
  tlumeni               :       real;
  //hodinovy vstup - nabezna hrana
  hodiny                :       bool;
  END_VAR
  VAR
  //hodnota minuleho a predminuleho vystupu
  minuly_vystup         :       real;
  predminuly_vystup     :       real;
  //promenna hodiny z predchoziho cyklu
  minule_hodiny         :       bool;
  END_VAR
  VAR_OUTPUT
  //vystupem je odezva soustavy druheho radu
  //pro vyraz P(s)=omega_n^2/(s^2+2*tlumeni*omega_n*s+omega_n^2).
  vystup                :       real;
  END_VAR
  VAR_TEMP
  //promenna jmenovatel pro vyraz P(s)=omega_n^2+2*tlumeni*omega_n+1.
   jmenovatel           :       real;
   //Protoze pouzivame krok po 0.1s, delime omega_n deseti
   male_omega           :       real;
  END_VAR

   //detekce hrany, pri ktere se provede vypocet odezvy
   IF hodiny=TRUE AND minule_hodiny=FALSE THEN
    //Protoze pouzivame krok po 0.1s, delime omega_n deseti
    male_omega:=omega_n/10.0;
    //Spocteni jmenovatele, abychom ho nepocitali vicekrat
    jmenovatel:=(male_omega*male_omega)+(2.0*tlumeni*male_omega)+1.0;
    //y(k)=(omega_n^2/jmenovatel)*u(k)+((2+2*tlumeni*omega_n)/jmenovatel)*y(k-1)
    //-(1/jmenovatel)*y(k-2)
    vystup:=((male_omega*male_omega)/jmenovatel)*vstup+
       ((2.0+2.0*tlumeni*male_omega)/jmenovatel)*minuly_vystup-
       (1.0/jmenovatel)*predminuly_vystup;
```

```
  predminuly_vystup:=minuly_vystup;
  minuly_vystup:=vystup;
  END_IF;

  //ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
  minule_hodiny:=hodiny;

END_FUNCTION_BLOCK

//Funkcni blok Klouzavy_prumer ma na svem vystupu hodnotu prumeru poslednich
//deseti hodnot s vynechanim nejvetsiho a nejmensiho udaje.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu.
FUNCTION_BLOCK Klouzavy_Prumer
 VAR_INPUT
 //vstup je signal, ktery chceme filtrovat
 vstup                  :        real;
  //hodinovy vstup - nabezna hrana
 hodiny                 :        bool;
 END_VAR
 VAR
 //hodnota hodin v minulem cyklu
 minule_hodiny          :        bool;
 //pole pro ulozeni poslednich 9 kroku
 minule_vstupy          : ARRAY [0..9] OF real;
 END_VAR
 VAR_OUTPUT
 //Vystupem je prumer z poslednich 10 hodnot (vcetne soucasne)
 //S VYNECHANIM NEJMENSI A NEJVETSI HODNOTY!
  vystup                :        real;
 END_VAR
 VAR_TEMP
 //pole pro bublinove trideni podle velikosti
 udaje                  : ARRAY [0..9] OF real;
 //pomocna_promenna pro docasne presunuti hodnot pri prohazovani
 //obsahu bunek pole a posleze pro spocteni prumeru
 pomocna_promenna :      real;
 //Promenne i,j pro repeat cykly bublinoveho trideni
 i                      :        int;
 j                      :        int;
 END_VAR

  //detekce hrany, pri ktere se provede vypocet prumeru
  IF hodiny=TRUE AND minule_hodiny=FALSE THEN

  //presunuti minulych vstupu do pole pro setrideni
   minule_vstupy[0]:=vstup;
   FOR i:=0 TO 9 DO
    udaje[i]:=minule_vstupy[i];
   END_FOR;
```

```
  //cyklus bublinoveho trideni
   i:=9;
   j:=0;
  WHILE i>0 DO
    WHILE  j<i DO
      IF udaje[j] > udaje[j+1] THEN
       pomocna_promenna:=udaje[j+1];
       udaje[j+1]:=udaje[j];
       udaje[j]:=pomocna_promenna;
      END_IF;
      j:=j+1;
    END_WHILE;
   i:=i-1;
   j:=0;
  END_WHILE;

  //spocteni prumeru S VYNECHANIM NEJMENSI A NEJVETSI HODNOTY!
   pomocna_promenna:=0.0;
  FOR i:=1 TO 8 DO
   pomocna_promenna:=pomocna_promenna+udaje[i];
  END_FOR;
   vystup:=pomocna_promenna/8.0;

  //posunuti minulych vstupu o jedno dal
  FOR i:=0 TO 8  DO
   minule_vstupy[9 - i]:=minule_vstupy[8 - i];
  END_FOR;
  END_IF;

  //ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
  minule_hodiny:=hodiny;

END_FUNCTION_BLOCK

//Funkcni blok Zaokrouhleni ma na svem vystupu zaokrouhlenou hodnotu vstupniho
//signalu na jedno desetinne misto.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu.
FUNCTION_BLOCK Zaokrouhleni
  VAR_INPUT
  //vstupem je signal, ktery chceme zaokrouhlit
  vstup                 :        real;
  //hodinovy vstup - nabezna hrana
  hodiny                :        bool;
  END_VAR
  VAR
  //hodnota hodin v minulem cyklu
  minule_hodiny         :        bool;
  END_VAR
```

```
 VAR_OUTPUT
 //vystup je zaokrouhleny vstup na jedno desetinne misto
  vystup                 :          real;
 END_VAR
 VAR_TEMP
  //promenna pro ulozeni desetinneho mista
  desetiny               :          real;
  //operator mod neumi s typem real, proto pouzijeme pomocnou promennou
  //typu dint
  pomocna_promenna :          dint;
 END_VAR

  //detekce hrany, pri ktere se provede zaokrouhleni
  IF hodiny=TRUE AND minule_hodiny=FALSE THEN

    //Potrebujeme posledni dve desetinna mista. Operator mod neumi s typem real
    pomocna_promenna:=real_to_dint((vstup*100.0)/1.0);
    //Zaokrouhleni na jedno desetinne misto
     IF ((pomocna_promenna mod 10)>=5) THEN
       pomocna_promenna:=pomocna_promenna+10;
     END_IF;
     //zbavime se druheho desetinneho mista
     pomocna_promenna:=pomocna_promenna/10;
     //Prevod na typ real
     desetiny:=dint_to_real(pomocna_promenna mod 10)*0.1;
     vystup:=dint_to_real((pomocna_promenna/10))+desetiny;
  END_IF;

  //ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
  minule_hodiny:=hodiny;

END_FUNCTION_BLOCK

//Funkcni blok Lokalizace_max generuje na svem vystupu logickou jednicku pokud
//se na analyzovanem signalu vyskytne lokalni maximum. Algoritmus je resen
//pomoci algoritmu predikce. Funkcni blok porovna, zda minula hodnota vstupu
//byla mensi nez soucasna a zaroven zda i predikovany vyvoj je mensi. Predikuji
//se budouci ctyri kroky, ktere se ukazaly jako dostatecne.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu.
FUNCTION_BLOCK Lokalizace_max
 VAR_INPUT
 //vstupem je signal, ktery chceme analyzovat
 vstup                   :          real;
 //hodinovy vstup - nabezna hrana
 hodiny                  :          bool;
 END_VAR
```

```
VAR
//hodnota hodin v minulem cyklu
minule_hodiny        :        bool;

//diference pro adaptaci
dif_adapt0           :        real;
dif_adapt1           :        real;
dif_adapt2           :        real;
dif_adapt3           :        real;
dif_adapt4           :        real;
dif_adapt5           :        real;
dif_adapt6           :        real;
dif_adapt7           :        real;

//hodnota minuleho vstupu
minuly_vstup         :        real;
END_VAR
VAR_OUTPUT
//vystupem je logicka promenna detekujici lokalni maximum
 vystup              :        bool;
END_VAR
VAR_TEMP
//promenne pro predikovane hodnoty
 Fbpredik1           :        real;
 FBpredik2           :        real;
 FBpredik3           :        real;
 FBpredik4           :        real;

//diference pro predikci
dif_predik0          :        real;
dif_predik1          :        real;
dif_predik2          :        real;
dif_predik3          :        real;
dif_predik4          :        real;
dif_predik5          :        real;
dif_predik6          :        real;
dif_predik7          :        real;

//promenna pro FOR cyklus
i                    :        sint;
END_VAR

 //detekce hrany, pri ktere se provede detekce maxima
IF hodiny=TRUE AND minule_hodiny=FALSE THEN
//adaptacni cast
//vypocet novych hodnot pro predikci
dif_predik0:=vstup;
dif_predik1:=dif_predik0-dif_adapt0;
dif_predik2:=dif_predik1-dif_adapt1;
dif_predik3:=dif_predik2-dif_adapt2;
```

```
 dif_predik4:=dif_predik3-dif_adapt3;
 dif_predik5:=dif_predik4-dif_adapt4;
 dif_predik6:=dif_predik5-dif_adapt5;
 dif_predik7:=dif_predik6-dif_adapt6;

 //kopie novych diferenci do pole pro adaptaci
 dif_adapt0:=dif_predik0;
 dif_adapt1:=dif_predik1;
 dif_adapt2:=dif_predik2;
 dif_adapt3:=dif_predik3;
 dif_adapt4:=dif_predik4;
 dif_adapt5:=dif_predik5;
 dif_adapt6:=dif_predik6;
 dif_adapt7:=dif_predik7;

 //Cyklus predikcni casti (staci 4 kroky)
 FOR i:=1 TO 4 DO
  dif_predik6:=dif_predik7+dif_predik6;
  dif_predik5:=dif_predik6+dif_predik5;
  dif_predik4:=dif_predik5+dif_predik4;
  dif_predik3:=dif_predik4+dif_predik3;
  dif_predik2:=dif_predik3+dif_predik2;
  dif_predik1:=dif_predik2+dif_predik1;
  dif_predik0:=dif_predik1+dif_predik0;

 CASE i OF
  1:  FBpredik1:=dif_predik0;
  2:  FBpredik2:=dif_predik0;
  3:  FBpredik3:=dif_predik0;
  4:  FBpredik4:=dif_predik0;
 END_CASE;
 END_FOR;

//vyhodnoceni, zda predesly a nasledujici kroky (staci 4) budou mensi nez
//soucasny krok
    IF (minuly_vstup<vstup) AND (FBpredik1<vstup) AND (FBpredik2<vstup) AND
      (FBpredik3<vstup) AND (FBpredik4<vstup) THEN
     vystup:=TRUE;
    ELSE
     vystup:=FALSE;
    END_IF;

 minuly_vstup:=vstup;
  END_IF;

  //ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
  minule_hodiny:=hodiny;

END_FUNCTION_BLOCK
```

```
//Funkcni blok Lokalizace_min generuje na svem vystupu logickou jednicku pokud
//se na analyzovanem signalu vyskytne lokalni minimum. Algoritmus je resen
//pomoci algoritmu predikce. Funkcni blok porovna, zda minula hodnota vstupu
//byla vetsi nez soucasna a zaroven zda i predikovany vyvoj je vetsi. Predikuji
//se budouci ctyri kroky, ktere se ukazaly jako dostatecne.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu.
FUNCTION_BLOCK Lokalizace_min
  VAR_INPUT
  //vstupem je signal, ktery chceme analyzovat
  vstup              :      real;
  //hodinovy vstup - nabezna hrana
  hodiny             :      bool;
  END_VAR
  VAR
  //hodnota hodin v minulem cyklu
  minule_hodiny      :      bool;

  //diference pro adaptaci
  dif_adapt0         :      real;
  dif_adapt1         :      real;
  dif_adapt2         :      real;
  dif_adapt3         :      real;
  dif_adapt4         :      real;
  dif_adapt5         :      real;
  dif_adapt6         :      real;
  dif_adapt7         :      real;

  //hodnota minuleho vstupu
  minuly_vstup       :      real;
  END_VAR
  VAR_OUTPUT
  //Vystupem je logicka promenna detekujici lokalni minimum
   vystup            :      bool;
  END_VAR
  VAR_TEMP
  //promenne pro predikovane hodnoty
   Fbpredik1         :      real;
   FBpredik2         :      real;
   FBpredik3         :      real;
   FBpredik4         :      real;

  //diference pro predikci
   dif_predik0       :      real;
   dif_predik1       :      real;
   dif_predik2       :      real;
   dif_predik3       :      real;
   dif_predik4       :      real;
   dif_predik5       :      real;
   dif_predik6       :      real;
```

```
 dif_predik7          :        real;
 //promenna pro FOR cyklus
 i                    :        sint;
END_VAR

//detekce hrany, pri ktere se provede detekce minima
IF hodiny=TRUE AND minule_hodiny=FALSE THEN
//adaptacni cast
//vypocet novych hodnot pro predikci
dif_predik0:=vstup;
dif_predik1:=dif_predik0-dif_adapt0;
dif_predik2:=dif_predik1-dif_adapt1;
dif_predik3:=dif_predik2-dif_adapt2;
dif_predik4:=dif_predik3-dif_adapt3;
dif_predik5:=dif_predik4-dif_adapt4;
dif_predik6:=dif_predik5-dif_adapt5;
dif_predik7:=dif_predik6-dif_adapt6;

//kopie novych diferenci do pole pro adaptaci
dif_adapt0:=dif_predik0;
dif_adapt1:=dif_predik1;
dif_adapt2:=dif_predik2;
dif_adapt3:=dif_predik3;
dif_adapt4:=dif_predik4;
dif_adapt5:=dif_predik5;
dif_adapt6:=dif_predik6;
dif_adapt7:=dif_predik7;

//Cyklus predikcni casti (staci 4 kroky)
FOR i:=1 TO 4 DO
 dif_predik6:=dif_predik7+dif_predik6;
 dif_predik5:=dif_predik6+dif_predik5;
 dif_predik4:=dif_predik5+dif_predik4;
 dif_predik3:=dif_predik4+dif_predik3;
 dif_predik2:=dif_predik3+dif_predik2;
 dif_predik1:=dif_predik2+dif_predik1;
 dif_predik0:=dif_predik1+dif_predik0;

CASE i OF
 1:  FBpredik1:=dif_predik0;
 2:  FBpredik2:=dif_predik0;
 3:  FBpredik3:=dif_predik0;
 4:  FBpredik4:=dif_predik0;
END_CASE;
END_FOR;

//vyhodnoceni, zda predesly a nasledujici kroky (staci 4) budou mensi nez
//soucasny krok
   IF (minuly_vstup>vstup) AND (FBpredik1>vstup) AND (FBpredik2>vstup) AND
     (FBpredik3>vstup) AND (FBpredik4>vstup) THEN
```

```
    vystup:=TRUE;
   ELSE
    vystup:=FALSE;
   END_IF;

  minuly_vstup:=vstup;
  END_IF;

  //ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
  minule_hodiny:=hodiny;

END_FUNCTION_BLOCK

//Funkcni blok Lokalizace_ustaleni generuje na svem vystupu logickou jednicku
//pokud se na analyzovanem signalu vyskytne ustaleni. Algoritmus predikce selhal
//pri kolisani signalu. Algoritmus je tedy resen, zda poslednich 10 hodnot
//neprekrocilo tolerancni pasmo vypoctene z prumeru poslednich deseti hodnot
//plus/minus zadana tolerance.
//Vstupem bloku je i hodinovy signal, ktery urcuje vzorkovaci okamziky.
//Hodinovy vstup reaguje na nabeznou hranu.
FUNCTION_BLOCK Lokalizace_ustaleni
 VAR_INPUT
 //vstupem je signal, ktery chceme analyzovat
 vstup                  :       real;
 //tolerancni pasmo, ve kterem budeme predpokladat ustaleny stav
 tolerance              :       real;
 //hodinovy vstup - nabezna hrana
 hodiny                 :       bool;
 END_VAR
 VAR
 //hodnota hodin v minulem cyklu
 minule_hodiny          :       bool;
 //pole pro ulozeni poslednich deseti hodnot vstupu
 minule_vstupy          : ARRAY [0..9] OF real;
 END_VAR
 VAR_OUTPUT
 //vystupem je logicka promenna detekujici lokalni ustaleni
  vystup                :       bool;
 END_VAR
 VAR_TEMP
 //promenne pro ulozeni spoctenych hranic pasma
  hranice_plus          :       real;
  hranice_minus         :       real;
  //hodnota prumeru z poslednich deseti kroku
  prumer                :       real;
  //pomocna promenna pro vypocet prumeru
  pomocna_promenna :    real;
  //promenna i pro FOR cyklus
  i                     :       int;
 END_VAR
```

```
//detekce hrany, pri ktere se provede detekce ustaleni
 IF hodiny=TRUE AND minule_hodiny=FALSE THEN
  //posun poslednich 10 hodnot
  FOR i:=0 TO 8  DO
   minule_vstupy[9 - i]:=minule_vstupy[8 - i];
  END_FOR;
  minule_vstupy[0]:=vstup;

 //Vypocet prumeru z poslednich 10 hodnot
  pomocna_promenna:=0.0;
 FOR i:=0 TO 9 DO
  pomocna_promenna:=pomocna_promenna+minule_vstupy[i];
 END_FOR;
  prumer:=pomocna_promenna/10.0;

//vypocet hranic pasma
hranice_plus:=prumer+tolerance;
hranice_minus:=prumer-tolerance;
//Testovani neprekroceni pasma v poslednich 10 hodnotach
  vystup:=TRUE;
  FOR i:=0 TO 9 DO
   IF (minule_vstupy[i]<hranice_minus) OR (minule_vstupy[i]>hranice_plus) THEN
   vystup:=FALSE;
   END_IF;
  END_FOR;

 END_IF; //konec podminky hodin

 //ulozeni hodnoty hodin z minuleho cyklu kvuli detekci hrany
 minule_hodiny:=hodiny;

END_FUNCTION_BLOCK
```