FAKULTA ELEKTROTECHNICKÁ ZÁPADOČESKÉ UNIVERZITY V PLZNI

RICE

Regionální inovační centrum elektrotechniky
Fakulta elektrotechnická
Západočeská univerzita v Plzni

Návrh software a automatické generování kódů v Matlab Simulink

Část 1.

Mikroprocesorové řízení pohonů 2

Jakub Talla

www.rice.zcu.cz

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj pro inovace

OSNOVA PŘEDNÁŠKY č. 3

1) Základní koncepce Simulinku

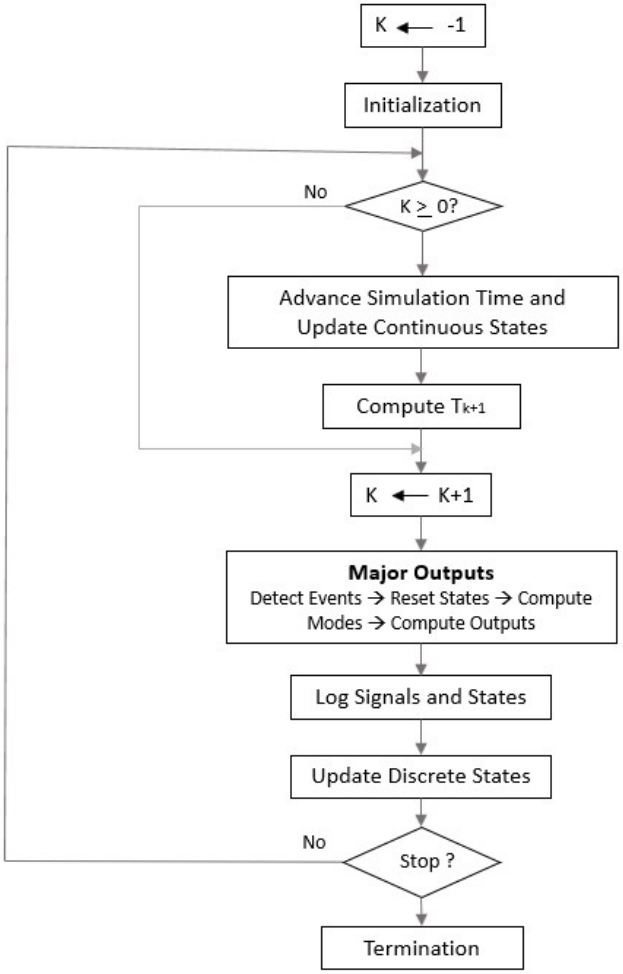
2) Simulink a datové typy

3) Uživatelské funkce v Simulinku

Základní koncepce Simulinku

Průběh simulace v Simulinku

Základní průběh simulace



Spuštění simulace:

Provede se:

1) výpočet výrazů k určení hodnoty parametrů, určení datových typů signálů a jejich dimenze, propagace sample times, optimalizace (redukce bloků), určení pořadí provádění bloků
2) Kompilace modelu do spustitelné formy

Iterační smyčka:

Množství iterací závisí na zvoleném řešiči ODE a celkovém čase simulace (případně zero-crossing)

Inicializace
Na startu dojde k inicializaci počátečních stavů a výstupů systému
Iterační smyčka

1) Výpočet výstupu modelu
2) Výpočet (aktualizace) stavů
3) Kontrola zero-crossing (nespojitostí)
4) Výpočet času dalšího kroku ODE

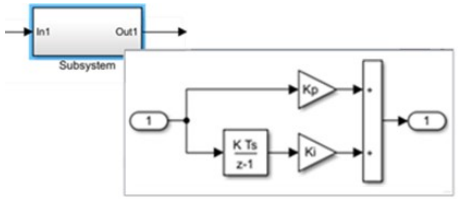
Bloky v Simulinku

Bloky v Simulinku

Klasické bloky

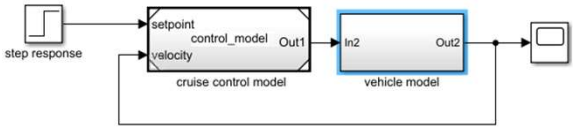
Diskrétní **Spojité (vyžadují solver dif. rov)**

Virtuální bloky (např. subsystém) (pouze pro přehlednost, nebo masku). Nemá vliv na simulaci ani generování

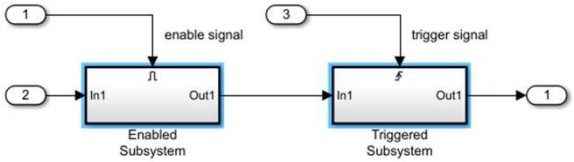


Nevirtuální bloky (nonvirtual blocks) ovlivňují provádění simulace např. pořadí instrukcí i generovaný kód

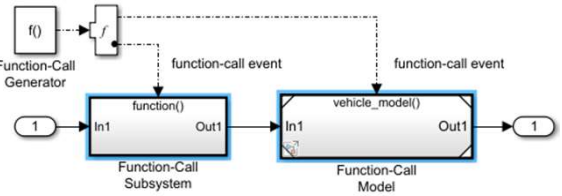
Atomic subsystem



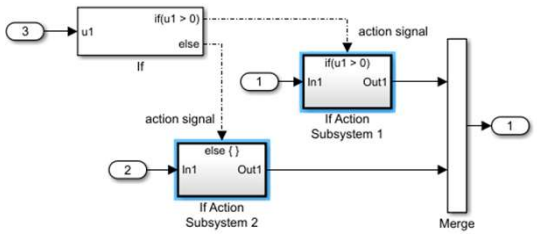
Enabled/trigger subsystem



Function call subsystem



Action subsystem (If, Switch,….)



Čáry (spoje) v Simulinku

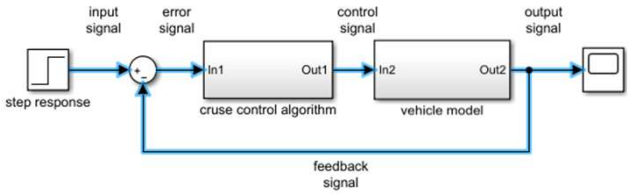
Čáry (spoje) v Simulinku

Čáry spojují bloky (určují datový přenos v modelu) a ovlivňují pořadí provádění instrukcí.

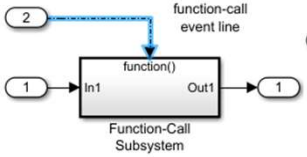
Signální čáry (spojité)– určují přenos dat z bloku do bloku mohou mít datové typy i názvy (které se mohou promítnout do názvu proměnných ve vygenerovaném kódu)

Událostní čáry (čerchované) – Slouží k přenosu událostí např. volání funkce, splnění podmínky apod.

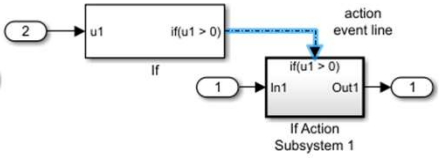
Signál



Událost



Událost



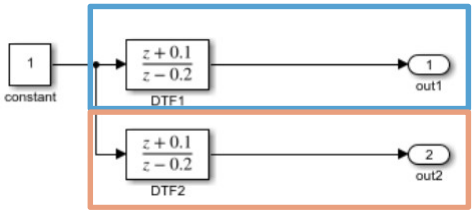
Variable vs. Discrete step solver

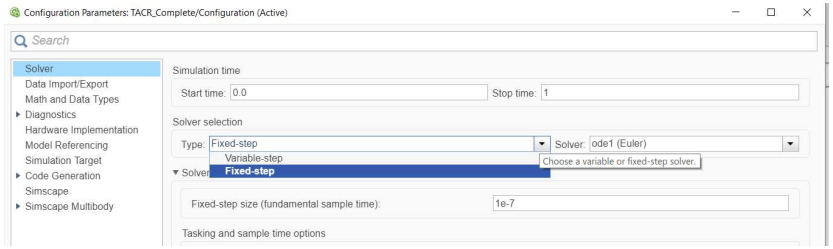
Variable step vs discrete step solver

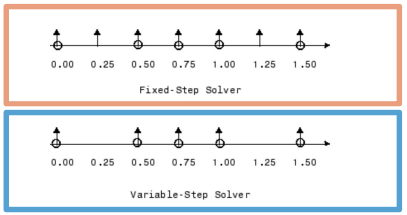
Proměnný vs. pevný krok řešení dif. rovnic

U proměnného kroku dochází k optimalizaci velikosti kroku řešení dif. rovnice ΔT (sample time) v nastavených mezích

+ Lepší optimalizace výpočtu
+ Přesnější zachycení Zero-crossing
- Nutné opatrné nastavení maximálního kroku, může divergovat

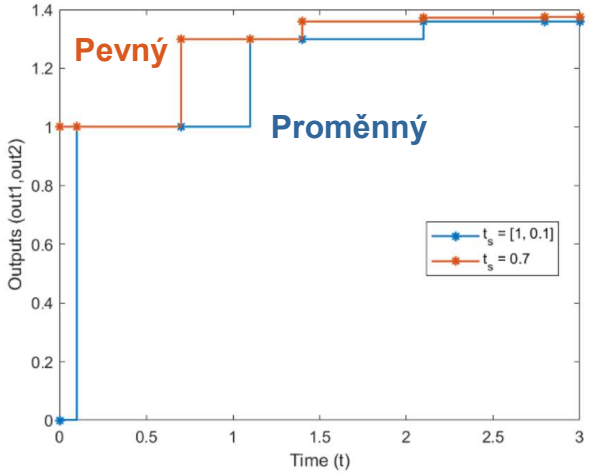






Pevný

Proměnný



MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
EVROPSKÁ UNIE EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ INVESTICE DO VAŠÍ BUDOUCNOSTI
OP Výzkum a vývoj pro inovace

Sample time

Sample time – vzorkovací čas

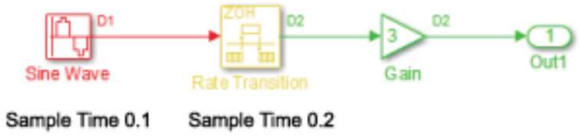
Vzorkovací čas (sample time) definuje s jakou periodicitou je daný blok počítán (tj. krok diferenciální rovnice)

Pro discrete step solver má každý blok přesně danou periodicitu spouštění (kromě událostí asynchronních např. triggerovaný subsystém)

Pro variable step solver může být periodicita spouštění dána metodou řešení diferenciálních rovnic (krok dif. rovnice je obecně proměnný, pokud bloku není konstantní vzorkovací čas vnucen)

V případě, že model obsahuje spojitý i diskrétní (konstantní sample time) systémy jsou časy variable step solveru voleny vždy tak, aby obsahovali diskrétní krok. Tj. sample time je variable step solverem volen i s ohledem na zvolené konstantní sample-times modelu

U bloků u kterých není definovaný sample-time se sample-time přebírá od 1) vstupního bloku, 2) výstupního bloku, 3) nejmenšího sample-time v modelu



Zero-crossing detection

Zero-crossing detection

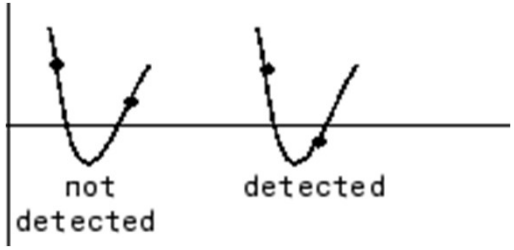
Variable step solver dynamicky mění velikost kroku diferenciální rovnice (sample time). Zjednodušeně řečeno podle rychlosti změny signálů (derivace) dochází k nárůstu nebo poklesu sample time (v reálu je to složitější).

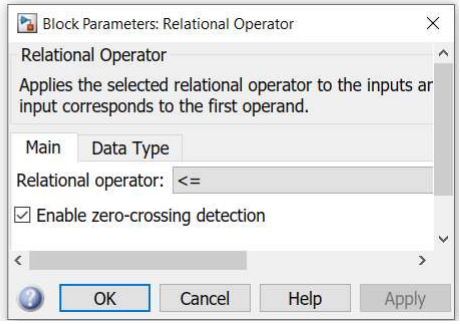
Zero-crossing detection slouží k automatické změně (redukci) sample time pro co nejpřesnější odhad kdy daná proměnná např. mění nějakou podmínku. Simulink zjistí, že se změnil stav podmínky a vrátí se v čase co nejpřesněji do okamžiku ve kterém došlo ke změně podmínky. To vede samozřejmě na nárůst výpočetní náročnosti. Pro nás nutnost při simulaci PWM - koincidence pily s komparačním signálem.

Stále je nutné rozumně nastavit maximální velikost sample time, ani zero-crossing detection nemusí vyhodnotit správně pokud nezaznamená změnu podmínky.

Oba body kladné
ZC nedetekován

Změna polarity
ZC detekován





Algebraic loops

Algebraic loops (algebraické smyčky)

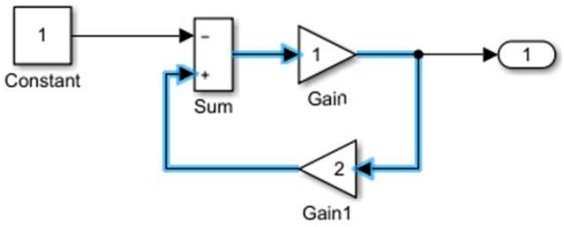
Algebraické smyčky vznikají zapojením zpětnovazebné struktury bloků s přímým průstupem signálu (bez integrace, bez zpoždění).

Mohou výrazně zpomalovat simulace, pro každý krok je nutné vyřešit algebraickou smyčku (a to ještě většinou iterativním způsobem)

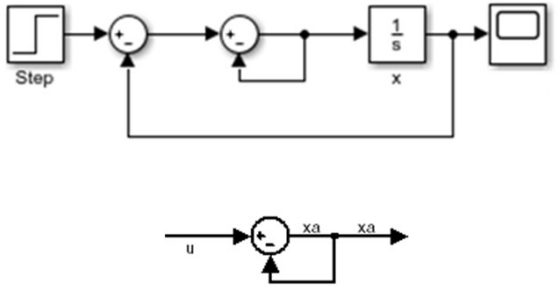
Často vedou na tzv. problém řešení soustavy algebro-diferenciálních rovnic, která je obecně složitě řešitelná.

Generovaný kód nesmí obsahovat algebraické smyčky

Algebraická smyčka



Kombinace ODE a algebraické smyčky

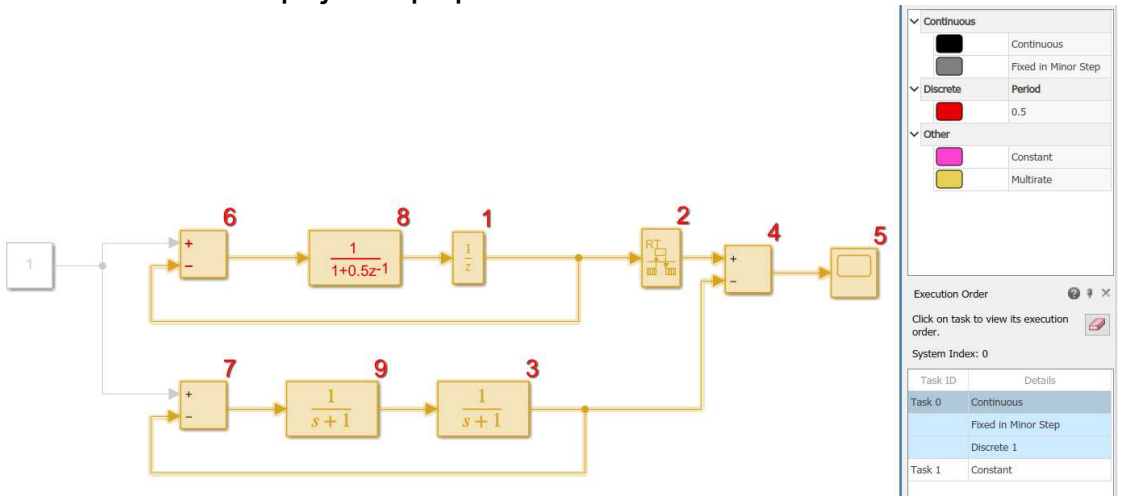


$x_a(t) = u(t) / 2.$

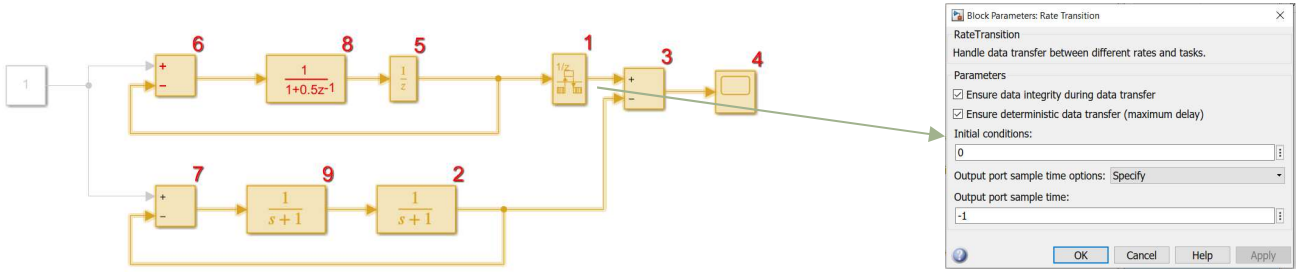
Pořadí provádění bloků

Pořadí provádění bloků

Čáry spojující bloky ovlivňují pořadí provádění instrukcí, ale skutečné pořadí závisí na determinismu operací. Např. pro spočtení výsledku rozdílu 6. bloku je nutné znát výstup bloku 1. Podobně ve spojitém případě bloku 7 a 3.

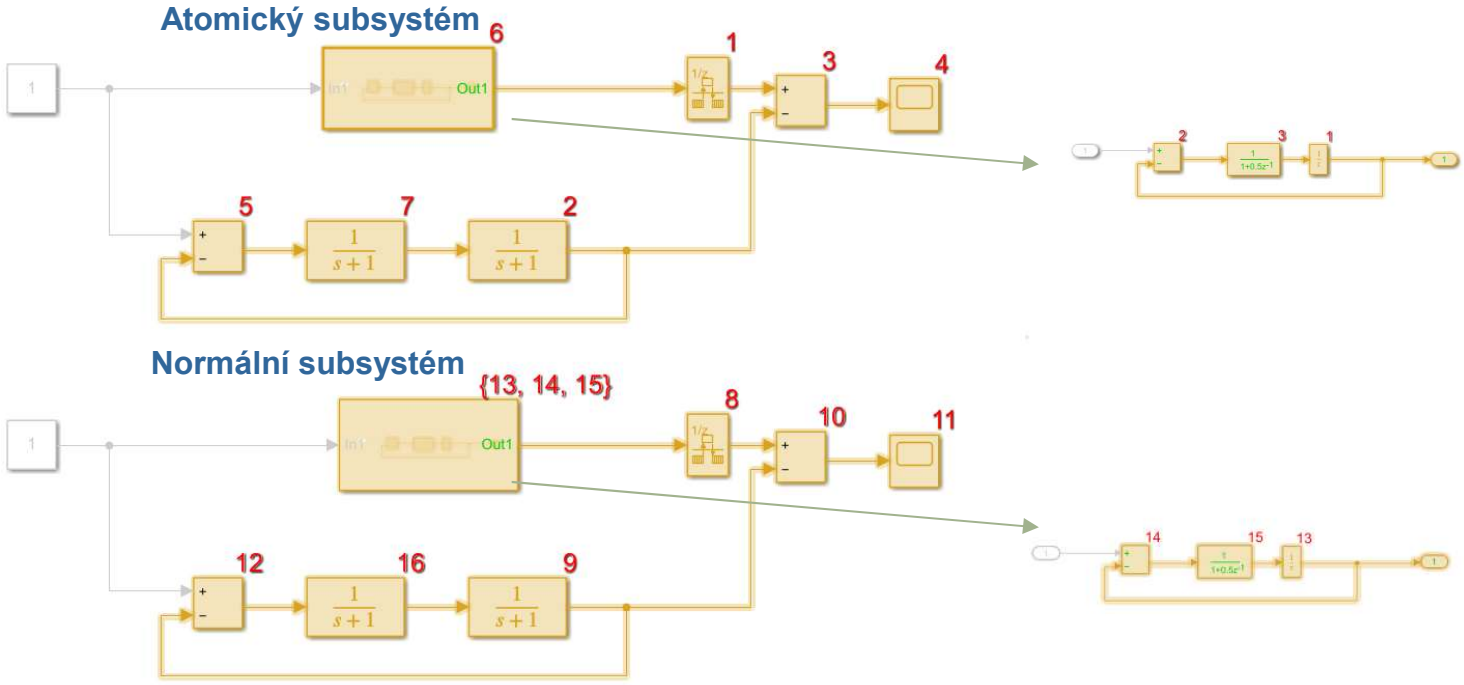


V případě zapnuté datové integrity a deterministického přenosu dat bloku RateTransition



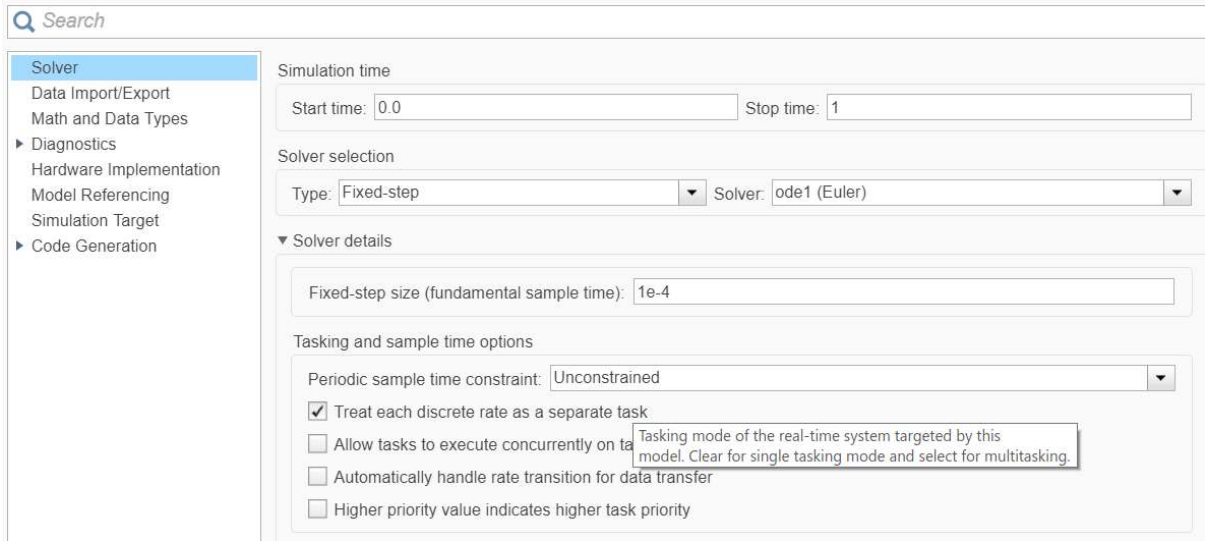
Pořadí provádění bloků

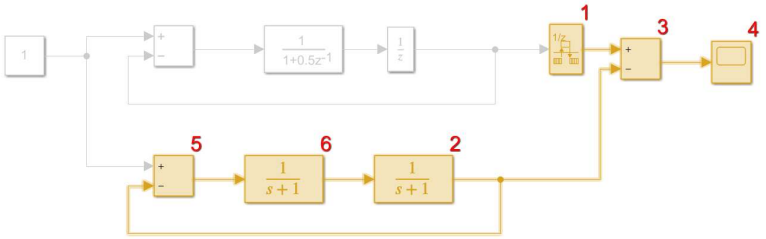
Normální subsystém slouží pouze k přehlednosti modelu apod. a nemá dopady do simulace ani vygenerovaného kódu.
Atomický subsystém je řešen separátně a je nedělitelný. Vygenerovaný kód nemá přes celý model optimalizované pořadí instrukcí.

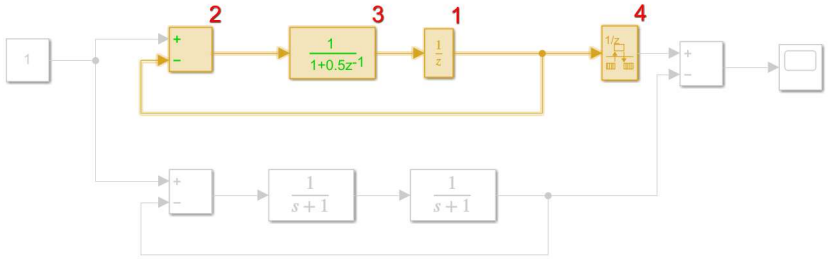


Pořadí provádění bloků

Pro nastavení **Treat each discrete rate as a separate task** jsou jednotlivé časové smyčky prováděné paralelně jako nezávislé úlohy (tasky). Možnost multiprocesorového/vícevláknového zpracování vygenerovaného kódu







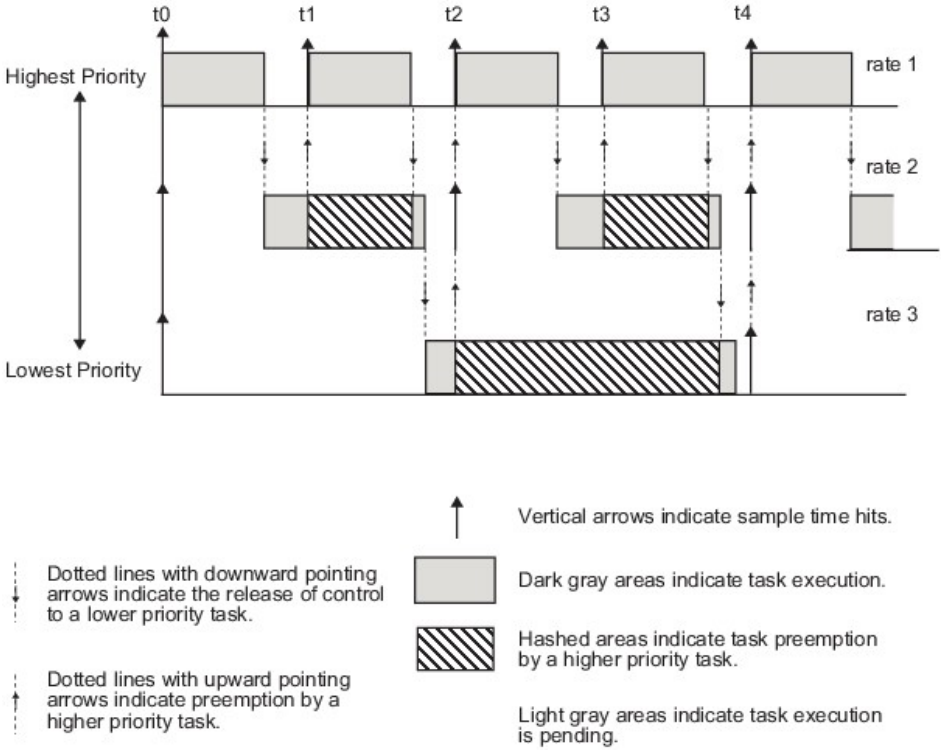
Single-tasking v Simulinku

Single-tasking (jednovláknové) zpracování s více časovými smyčkami (více sample-time). U single-tasking je celá úloha řešená jako jeden proces tj. jednovláknově. V jednotlivé interupty timeru je ale prováděn různý kód podle toho jestli se počítá pouze nejrychlejší smyčka nebo se sejdou dvě či více časových smyček. Pokud se smyčky sejdou, jsou počítané v pořadí od nejmenšího sample-time (s největší prioritou) k největšímu.

3 časové smyčky se sample time T1=1s, T2=2s, T3= 4s

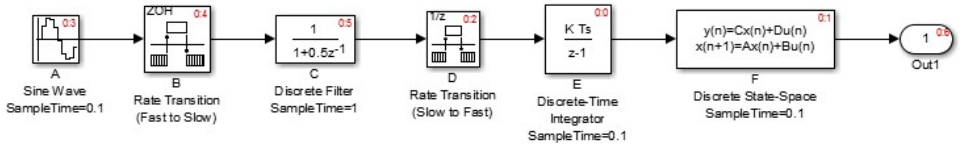
Multi-tasking v Simulinku

Multi-tasking (vícevláknové) zpracování s více časovými smyčkami (více sample-time)
Dochází k přerušení provádění smyček s menšími prioritami (standardně s většími sample time). Dále lze nastavit ještě priority asynchronních událostí (interuptů).



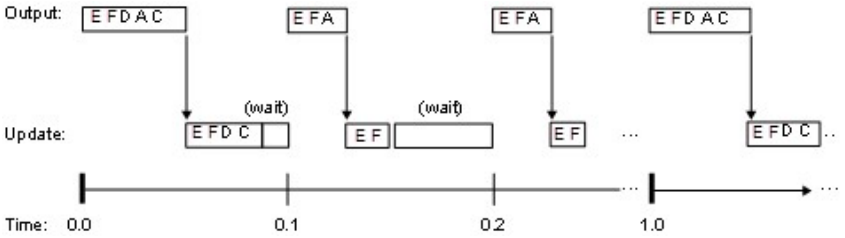
Single-tasking vs. Multi-tasking

Single-tasking vs. Multi-tasking

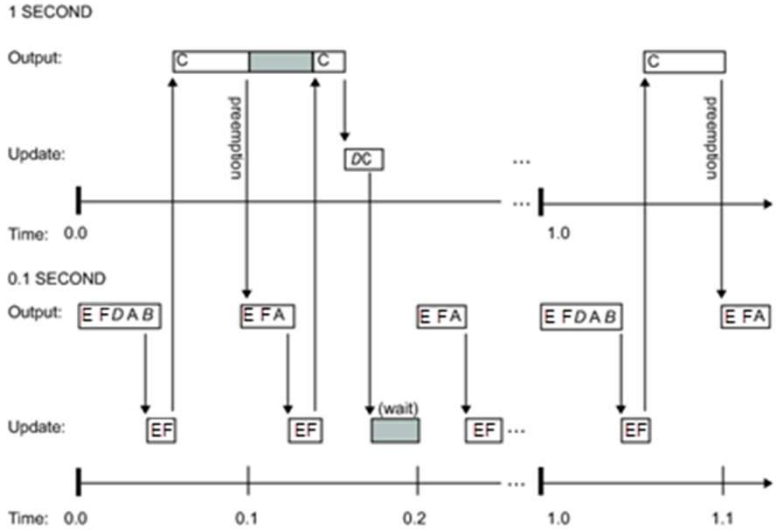


Blocks (in Execution Order)	Sample Time (in Seconds)	Output	Update
E	0.1	Y	Y
F	0.1	Y	Y
D	1	Y	Y
A	0.1	Y	N
C	1	Y	Y

Single-tasking, průběh real-time



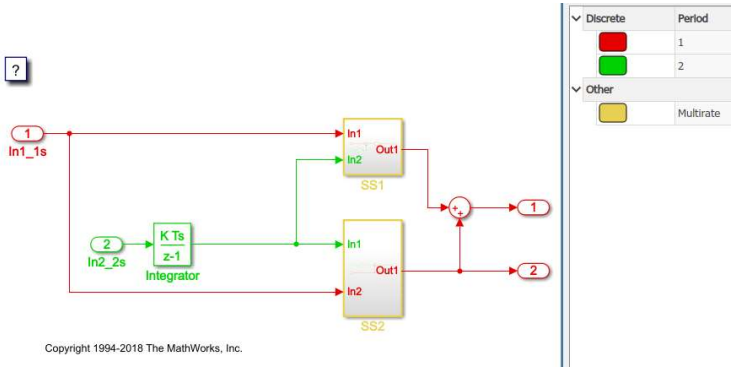
Multi-tasking, průběh real-time



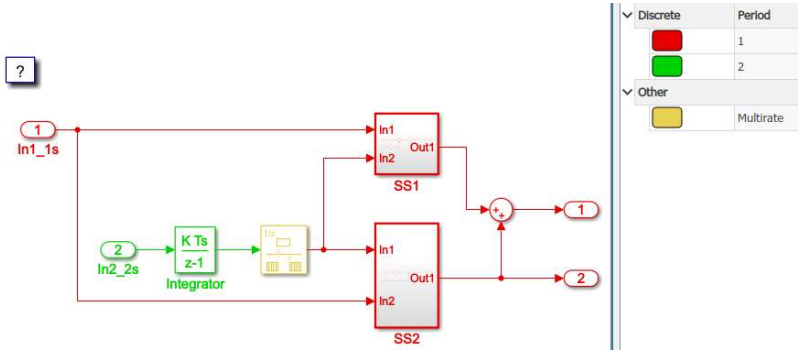
Multi-tasking v Simulinku

Single-tasking (jednovláknové) vs. Multi-tasking model a nastavení

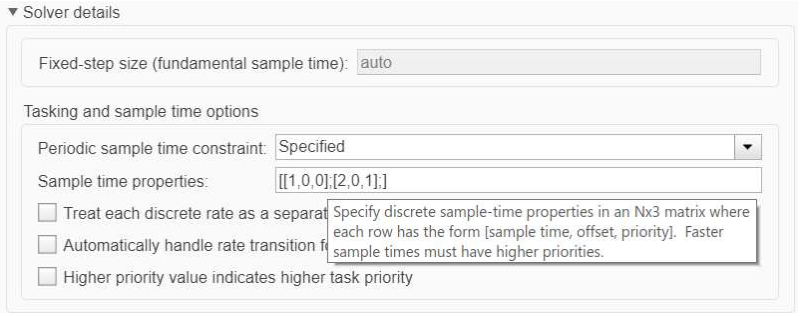
Single-tasking (chybí rate-transition)

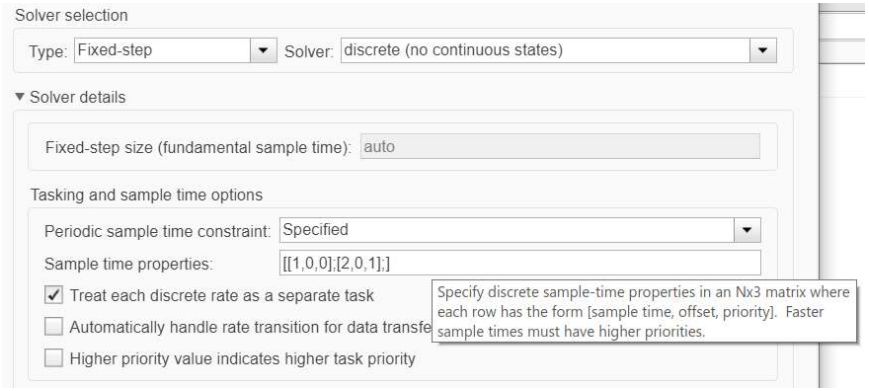


Multi-tasking (má rate-transition)



Nastavení configuration parameters



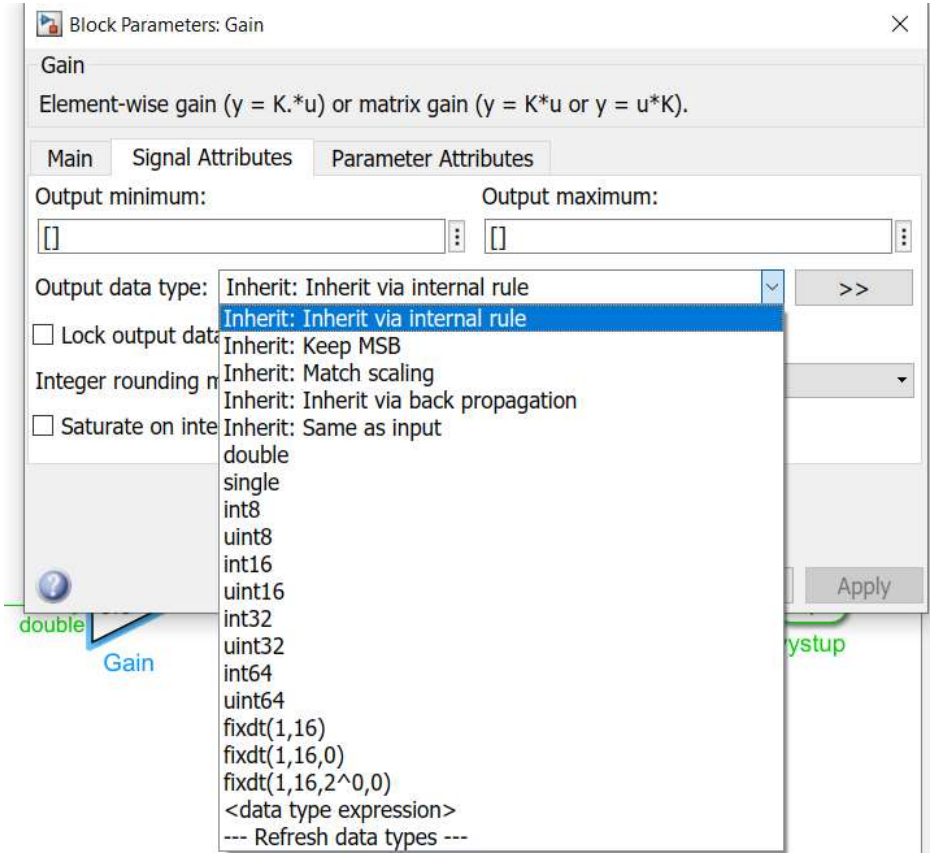


Simulink a datové typy

Simulink a datové typy

Simulink a datové typy

Simulink umožňuje pracovat s velkým množstvím datových typů. Konkrétní realizace datového typu závisí na konkrétním procesoru. Např. int16 bude realizován na mikroprocesoru x86 jako short int.



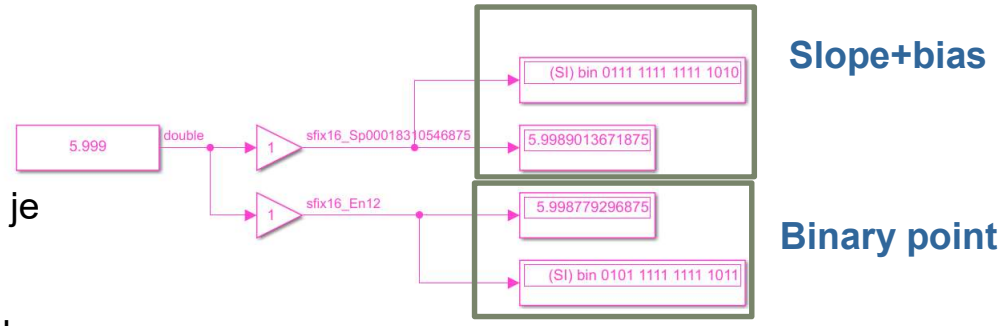
Simulink a datové typy

Simulink a pevná řádová čárka

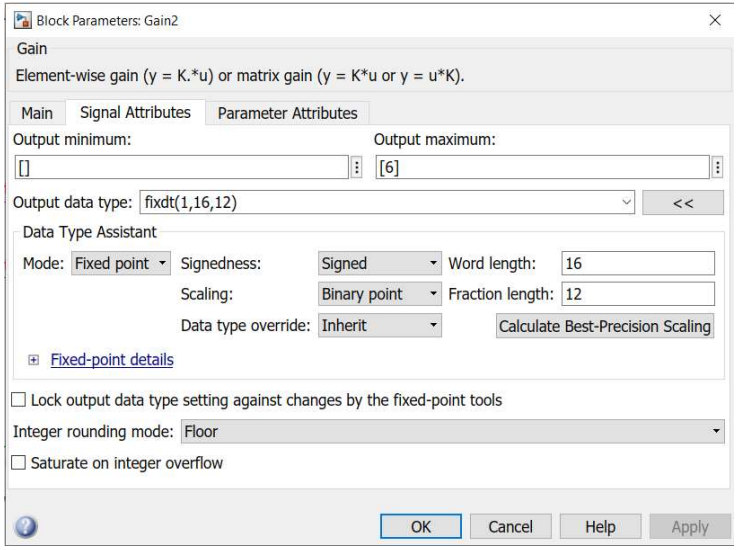
Simulink má dva typy pevné řádové čárky:

Binary point scaling říká za jakým bitem je desetiná čárka Fi=K*Integer, kde K=2^-N

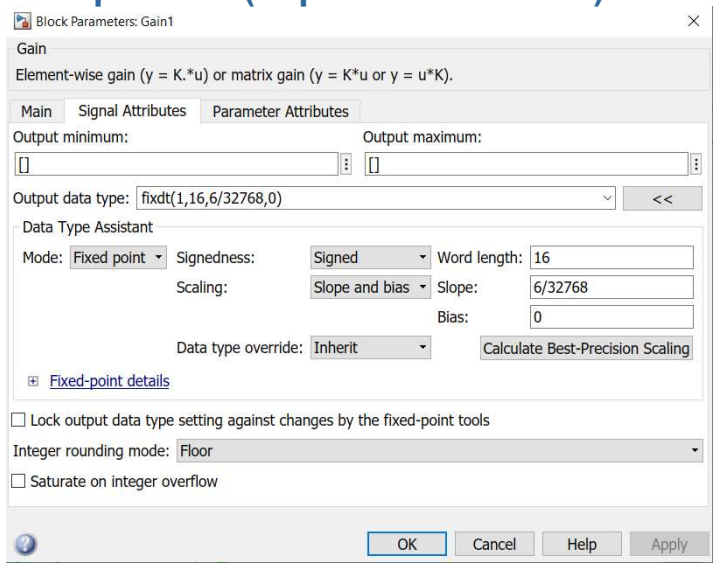
Slope+bias číslo je normované libovolnou hodnotou a může být posunuto o bias Y=K*Integer + Q, kde K i Q jsou libovolná reálná čísla



Binary point scaling



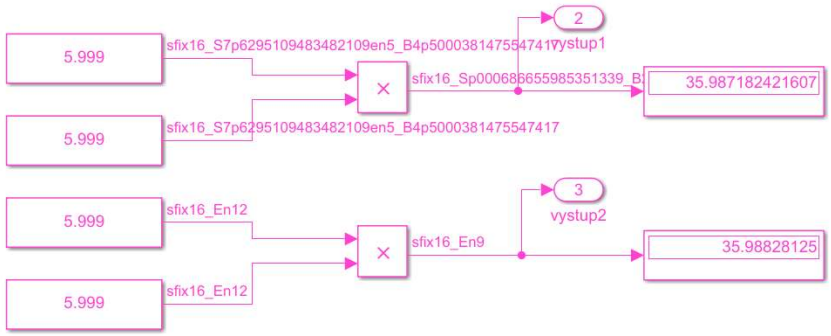
Slope+bias (např. nastavení +-6)



Simulink a datové typy

Simulink a pevná řádová čárka

Slope+bias formát může vést ke zvýšení výpočetní náročnosti (je potřeba provést více aritmetických operací pro např. vynásobení dvou čísel v tomto formátu)



Slope+bias

Binary point

```
Nasobeni_Y.vystup1 = (int16_T)(((int64_T)mul_u64_sr64(mul_u64_loSR((uint64_T)
  (((int64_T)(Nasobeni_P.Constant1_Value * 20480L) << 21U) +
   2533296265560069LL), (uint64_T)(((int64_T)(Nasobeni_P.Constant3_Value *
  20480L) << 21U) + 2533296265560069LL), 50ULL), 64050217356711125ULL) +
  -2589954757973LL) >> 26U);

/* Outport: '<Root>/vystup2' incorporates:
 *  Constant: '<S1>/Constant'
 *  Constant: '<S1>/Constant2'
 *  Product: '<S1>/Product1'
 */
Nasobeni_Y.vystup2 = (int16_T)(((int32_T)Nasobeni_P.Constant2_Value *
  Nasobeni_P.Constant_Value) >> 15U);
```

Slope+bias

Binary point

Simulink a datové typy

Simulink, datové typy a vygenerovaný kód

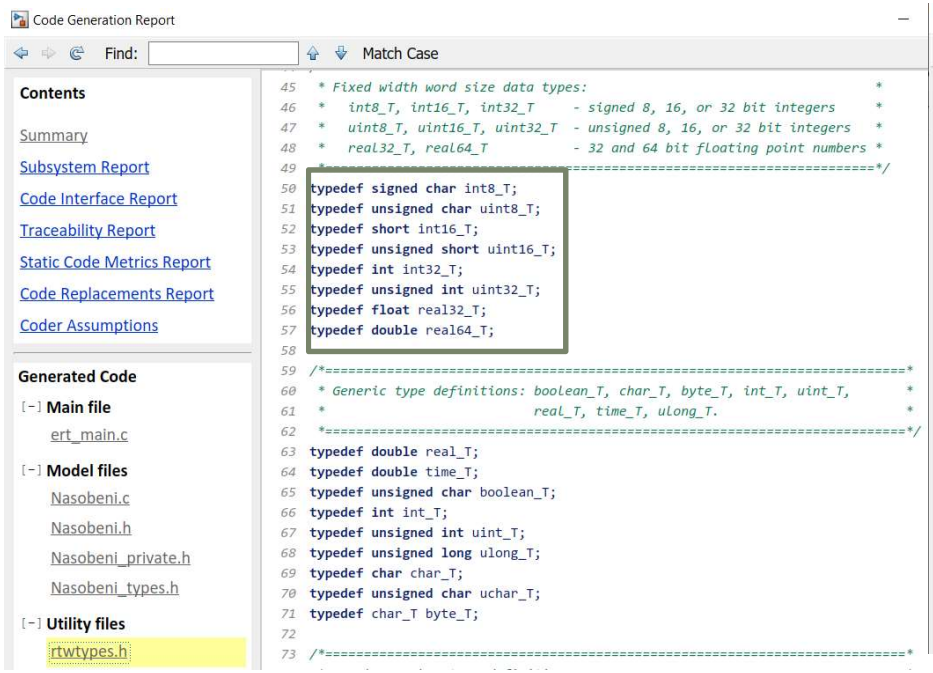
Simulink používá vlastní datové typy. Ty jsou poté definovány v souboru rtwtypes.h pro konkrétní procesory

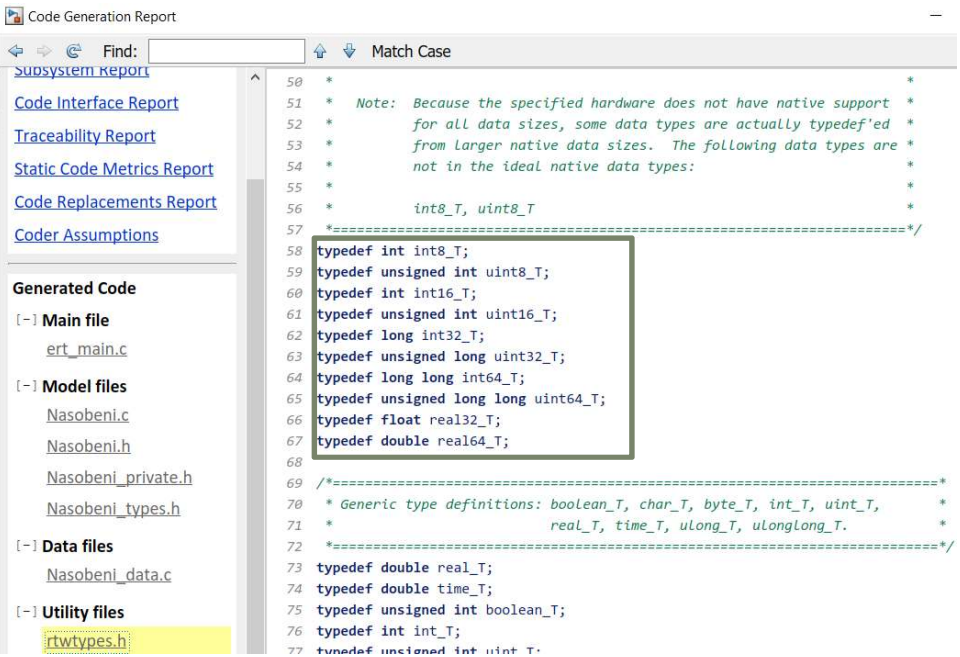
Intel x86 procesor

Nejmenší datový typ 8bitový char, Integer má 32 bitů

Texas Instrument C2000 série (i F28335)

Nejmenší datový typ 16bitový integer, integer má 16 bitů

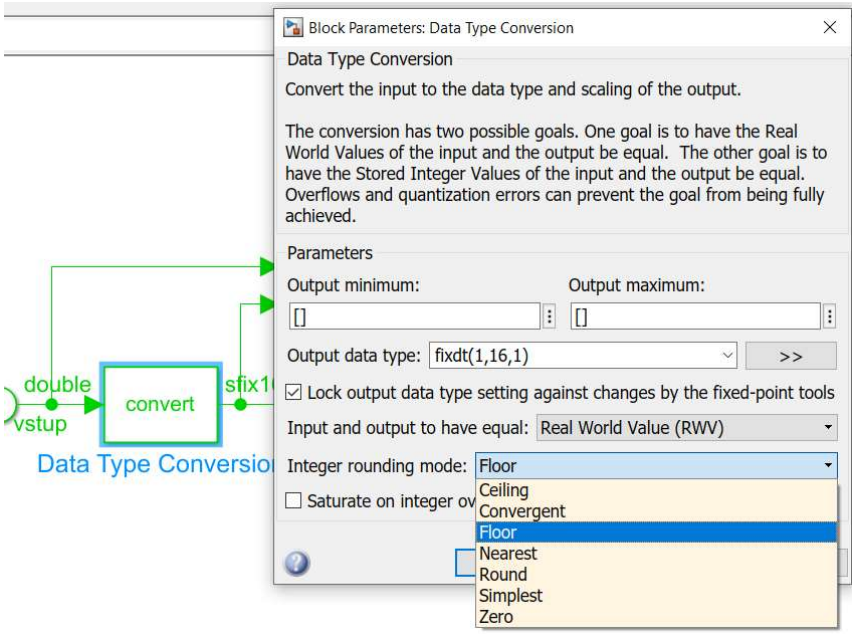




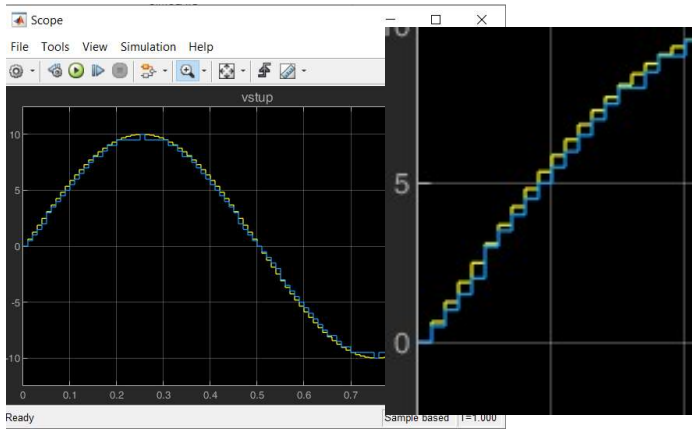
Simulink a datové typy

Převod z plovoucí na pevnou řádovou čárku a „zaokrohlouvací“ metody

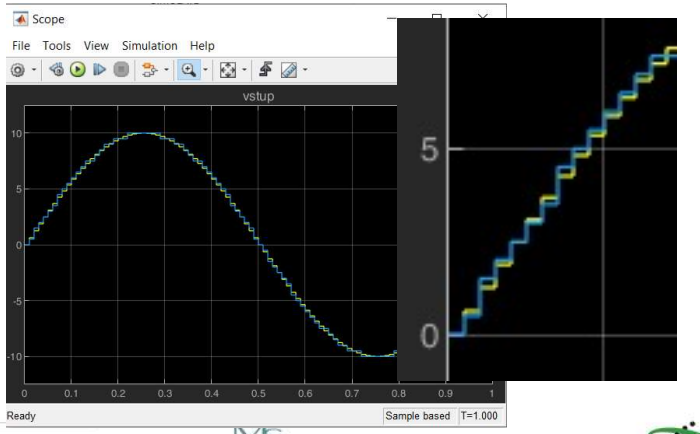
Zaokrouhlovací metody slouží k definování metody převodu čísla v plovoucí řádové čárce do pevné řádové čárky



Simplest



Nearest



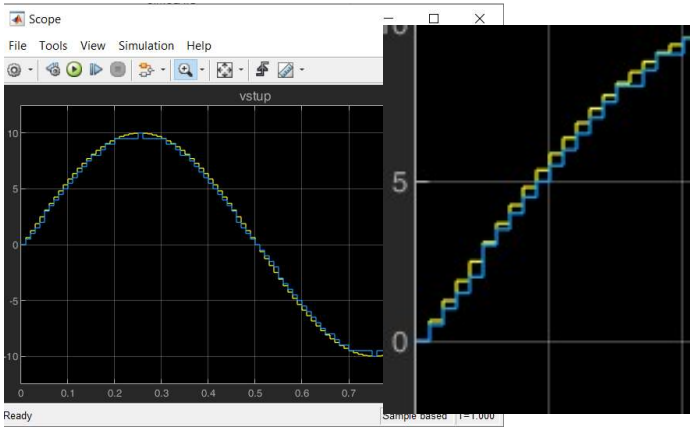
Simulink a datové typy

Převod z plovoucí na pevnou řádovou čárku a „zaokrohlouvací“ metody

Volba zaokrouhlovací metody má dopady do vygenerovaného kódu

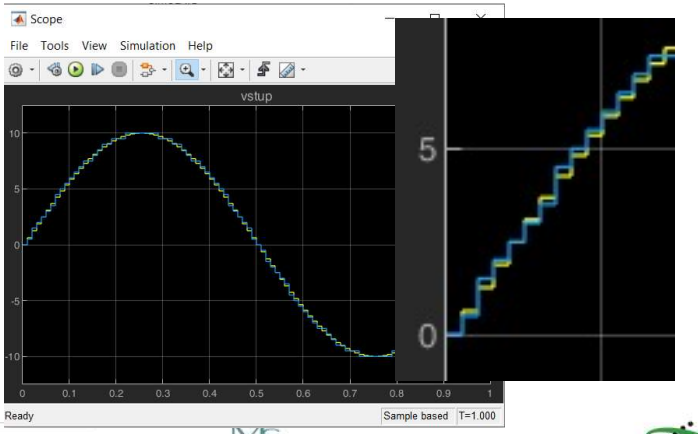
Simplest

```
tmp = trunc(Nasobeni_U.vstup * 2.0);
if (rtIsNaN(tmp) || rtIsInf(tmp)) {
  tmp = 0.0;
} else {
  tmp = fmod(tmp, 65536.0);
}
```



Nearest

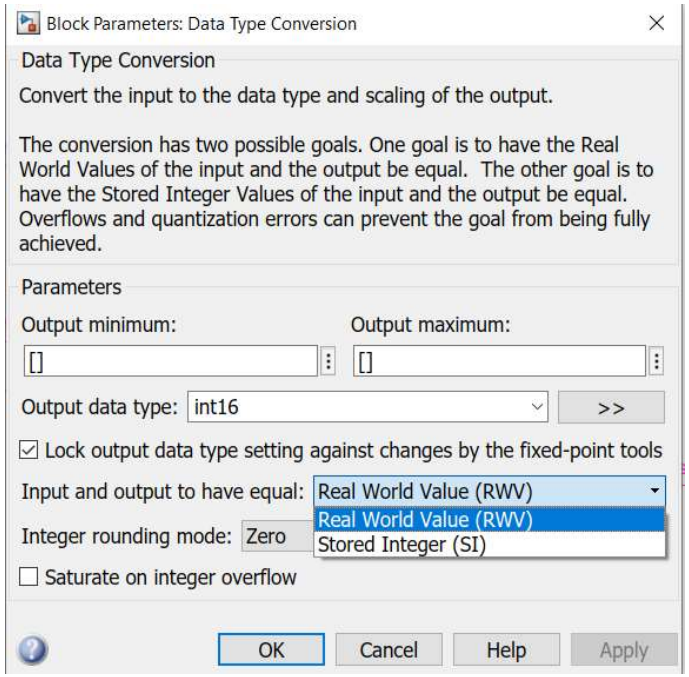
```
u = Nasobeni_U.vstup * 2.0;
v = fabs(u);
if (v < 4.503599627370496E+15) {
  if (v >= 0.5) {
    u = floor(u + 0.5);
  } else {
    u *= 0.0;
  }
}

if (rtIsNaN(u) || rtIsInf(u)) {
  u = 0.0;
} else {
  u = fmod(u, 65536.0);
}
```

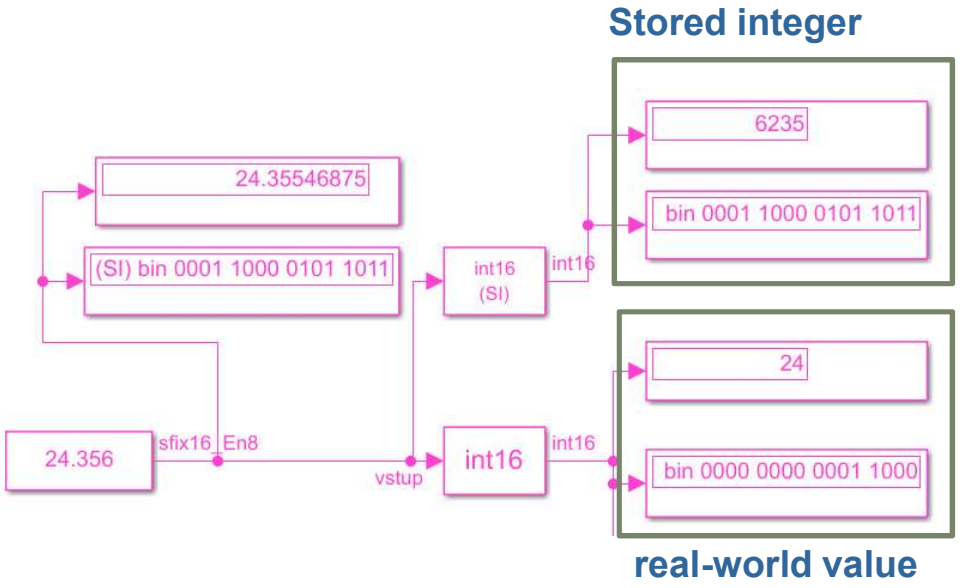


Simulink a datové typy

Převod real-world value vs stored integer

Real-world value konverze konvertuje reálnou hodnotu čísla, stored integer pouze reinterpretuje data





Simulink a datové typy

Převod real-world value vs stored integer

Real-world value konverze konvertuje reálnou hodnotu čísla, stored integer pouze reinterpretuje data

real-world value

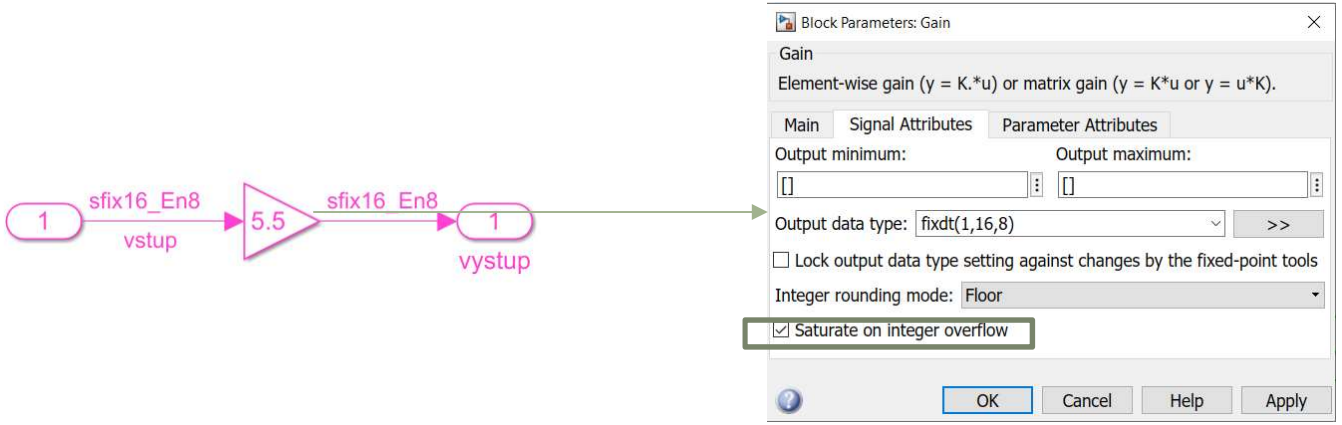
```
/* Outport: '<Root>/vystup' incorporates:
 *  DataTypeConversion: '<S1>/Data Type Conversion'
 *  Inport: '<Root>/Input'
 */
Nasobeni_Y.vystup = (int16_T)((int16_T)((Nasobeni_U.vstup < 0 ? 255 : 0) +
  Nasobeni_U.vstup) >> 8);
```

Stored integer

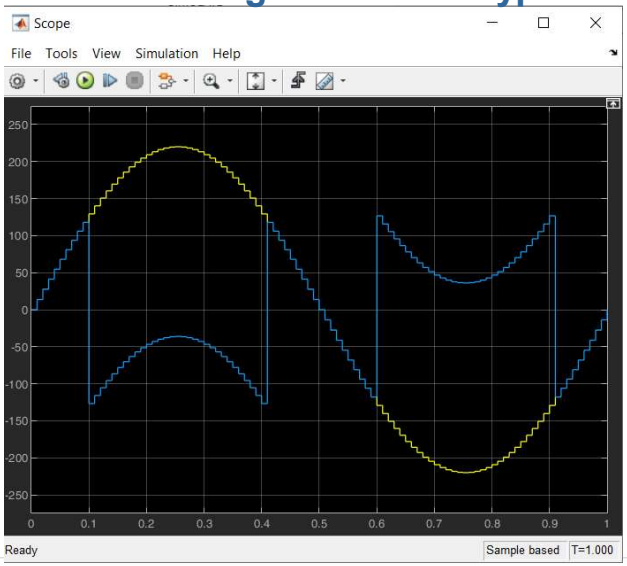
```
/* Outport: '<Root>/vystup' incorporates:
 *  Inport: '<Root>/Input'
 */
Nasobeni_Y.vystup = Nasobeni_U.vstup;
```

Simulink a datové typy

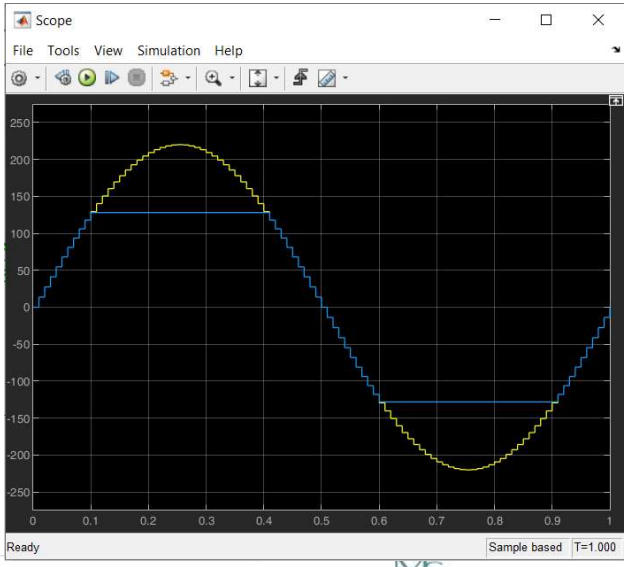
Simulink a datové typy – Saturate on integer overflow



Saturate on integer overflow vypnuté



Saturate on integer overflow zapnuté



Simulink a datové typy

Simulink a datové typy – Saturate on integer overflow a vygenerovaný kód

Saturate on integer overflow vypnuté

```
void Nasobeni_step(void)
{
  /* Outport: '<Root>/vystup' incorporates:
   *  Gain: '<S1>/Gain'
   *  Inport: '<Root>/In1'
   */
  Nasobeni_Y.vystup = (int16_T)((11 * Nasobeni_U.vstup) >> 1);
}
```

Saturate on integer overflow zapnuté

```
/* Model step function */
void Nasobeni_step(void)
{
  int32_T tmp;

  /* Gain: '<S1>/Gain' incorporates:
   *  Inport: '<Root>/In1'
   */
  tmp = (11 * Nasobeni_U.vstup) >> 1;
  if (tmp > 32767) {
    tmp = 32767;
  } else {
    if (tmp < -32768) {
      tmp = -32768;
    }
  }
```

Tvorba vlastních uživatelských funkcí v Simulinku

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI
OP Výzkum a vývoj pro inovace

Uživatelské funkce v Simulinku

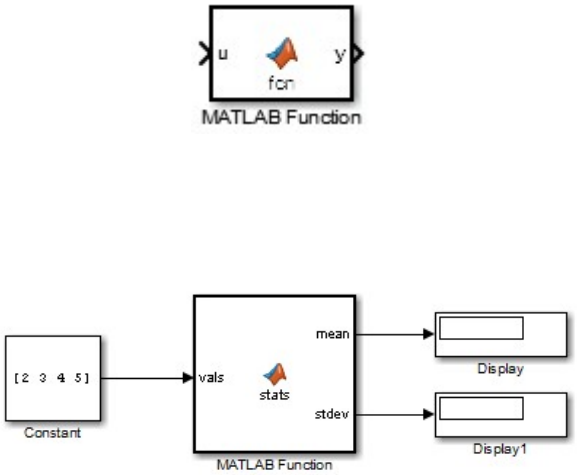
Tvorba vlastních uživatelských funkcí v Simulinku s podporou automatického generování kódu:

- **Pomocí základních Simulinkovských bloků**
- **Matlab function (pomocí jazyka Matlab)**
- **Stateflow**
- **S-function (pomocí C/C++)**

Matlab function

Matlab function (Matlabovská funkce)

- Matlab function – umožňuje implementaci Matlab kódu v Simulinku
- Umožňuje automatické generování kódu (C/C++,VHDL)

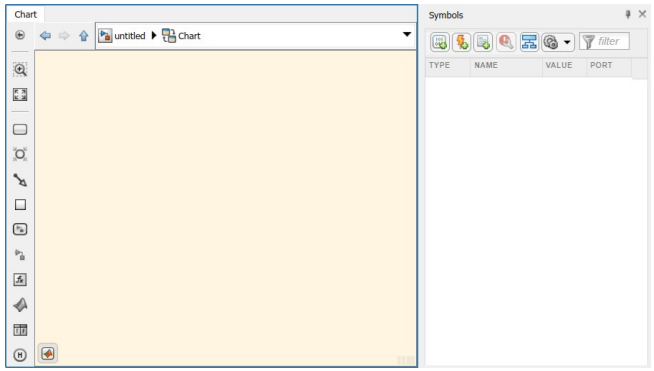



```
1   function [mean,stdev] = stats(vals)
2   % #codegen
3
4   % calculates a statistical mean and a standard
5   % deviation for the values in vals.
6
7 -  len = length(vals);
8 -  mean = avg(vals,len);
9 -  stdev = sqrt(sum(((vals-avg(vals,len)).^2))/len);
10 -  coder.extrinsic('plot');
11 -  plot(vals,'-+');
12
13  function mean = avg(array,size)
14 -  mean = sum(array)/size;
```

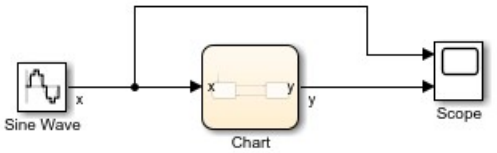
Stateflow

- Stateflow je grafické programovací prostředí pro modelování stavových automatů
- Umožňuje automatické generování kódu

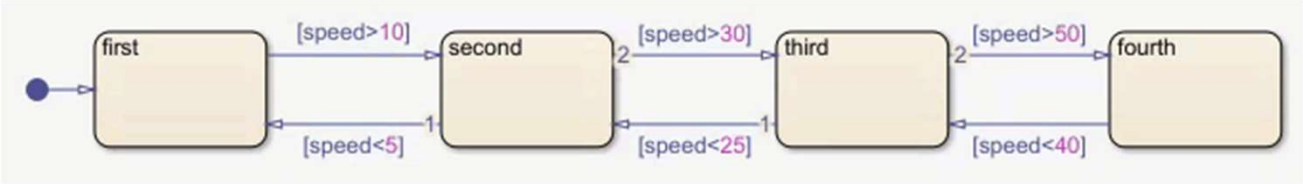
Prostředí Stateflow



Stateflow Chart v Simulinku



Stavový automat ve Stateflow



S-function (Systémová funkce)

S-function (Systémová funkce)

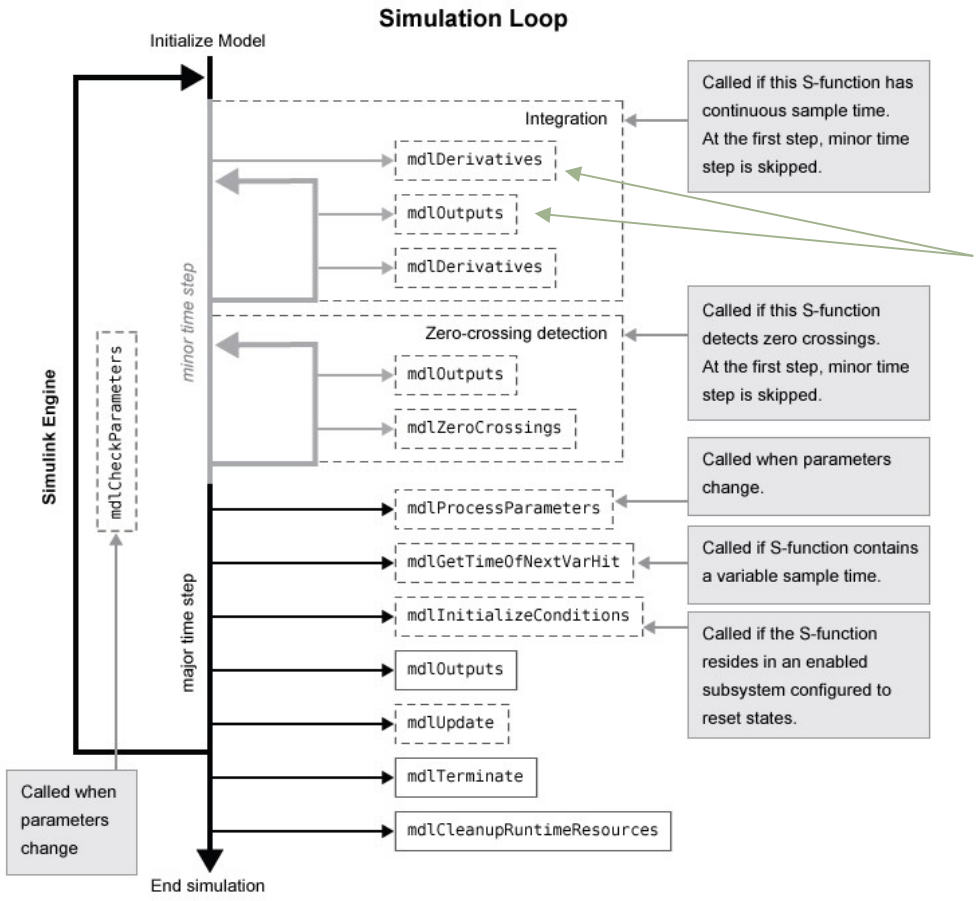
S-funkce je způsob jak vytvořit blok v Simulinku pomocí Matlabu, C, C++, nebo Fortranu

Varianty S-funkcí:

- **Level-2 Matlab S-function** (blok slouží k manuálnímu importu napsané S-funkce)
- **C MEX S-Function** (slouží k importu mex S-funkce vytvořené v Matlabu z vlastního C)
- **The S-Function Builder** (GUI pro jednoduchou tvorbu S-funkce z vlastního C)
- **The Legacy Code Tool** (tool se sadou příkazů na jednoduchou tvorbu S-fce v Matlabu)

S-funkce

Základní průběh S-funkce



Průběh výpočtu S-funkce odpovídá základnímu průběhu simulace Simulinku.

Jednotlivé funkce v jazyce C popisující danou S-funkci

Typická S-funkce má stovky řádek C kódu proto je výhodné využívat např. S-function builder

S-Function Builder

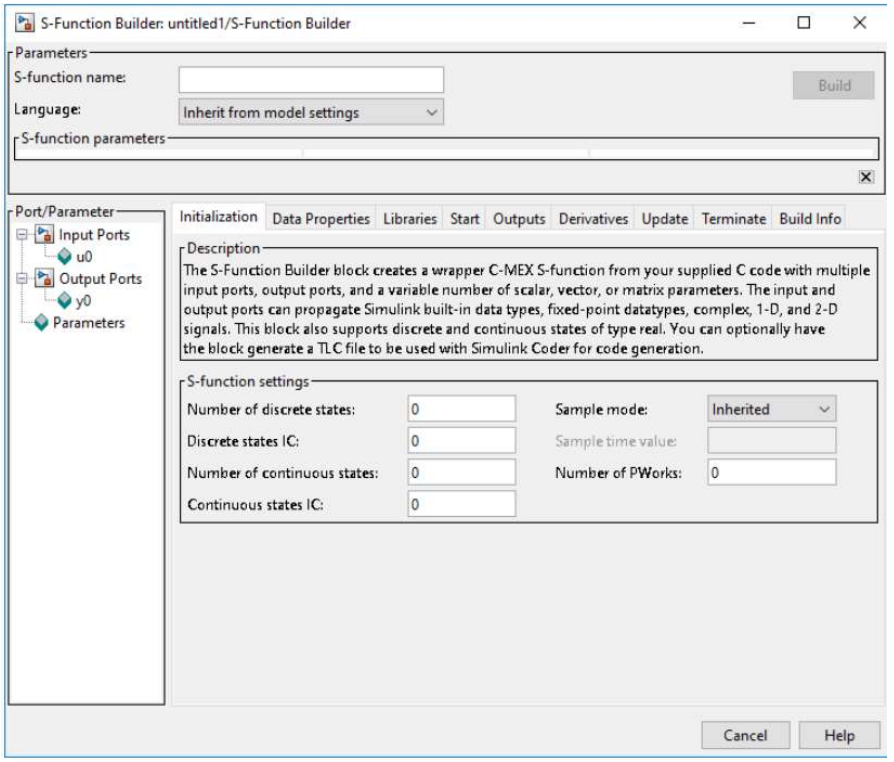
S-Function Builder

Vygenerované soubory:

- **sfun.c** (zdrojový C kód S-fce, obsahuje volání wrappovací/obalové funkce)
- **sfun_wrapper.c** (tzv. obalová funkce, obsahuje veškerý C kód napsaný v S-function builderu)
- **sfun.tlc** (slouží ke správnému vygenerování kódu S-funkce a jeho zařazení v rámci vygenerovaného kódu)

Zkomplilovaná S-funkce:

- **sfun.mexw64** (zkompilovaná a spustitelná S-funkce pomocí 64 bitového compileru pro Windows)



FAKULTA ELEKTROTECHNICKÁ ZÁPADOČESKÉ UNIVERZITY V PLZNI

RICE

Regionální inovační centrum elektrotechniky
Fakulta elektrotechnická
Západočeská univerzita v Plzni

Příští přednáška

Návrh software a automatické generování kódů v Matlab Simulink Část 2.

Jakub Talla

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI
OP Výzkum a vývoj pro inovace

FAKULTA
ELEKTROTECHNICKÁ
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

RICE

Regionální inovační centrum elektrotechniky
Fakulta elektrotechnická
Západočeská univerzita v Plzni

Děkuji za pozornost!

Adresa:
Univerzitní 26
306 14 Plzeň
Česká republika

Tel: +420 377 634 443
Fax: +420 377 634 402

Email: talic@kev.zcu.cz

www.rice.zcu.cz

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj pro inovace