Információfeldolgozás laboratórium
Magasszintű kódgenerálás mérés

VIMIM322
BME MIT: MSc beágyazott információs rendszerek szakirány

Mérési útmutató
2010

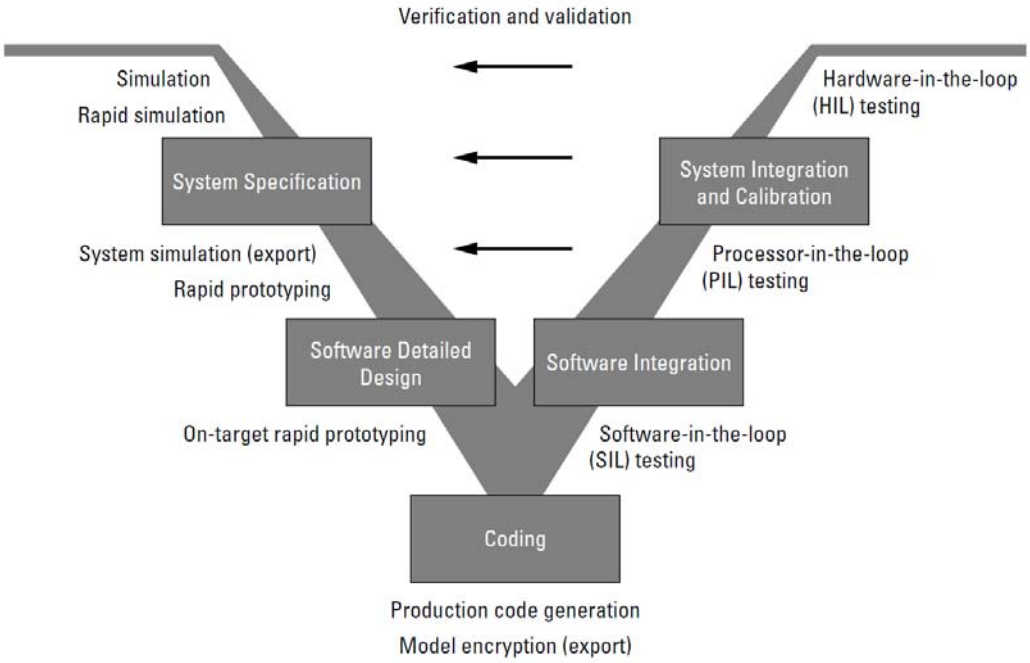
Csak belső használatra

Scherer Balázs

1. Bevezetés: modellalapú fejlesztés és kódgenerálás alkalmazása a rendszertervezésben

Az MSc képzés Rendszertervezés tárgya (VIMIM238) röviden bemutatta a beágyazott rendszerek tervezésére használt általános lépéseket. Ugyanakkor a tárgyban viszonylag kevés szó esett arról, hogy a fejlesztés során hogyan alkalmazzák a modernek számító és egyre jobban elterjedő modellalapú technológiákat és a kódgenerálást. Ezek a technológiák annyira elterjedtek már az iparban, hogy például az autóipari elektronikai egységek vezérléseihez, szabályozásaihoz kötődő programok akár 30-40%-át is így állítják elő, és ez a tendencia várhatóan folytatódni fog a jövőben. A mérés célja, hogy ezeknek a technológiáknak a használatáról adjon egy áttekintést a Matlab Simulink Real-Time workshop alkalmazásával.

Mielőtt elkezdenénk az kódgenerálás mikéntjével foglalkozni, tekintsük át, hogy ez a modellalapú fejlesztési módszer hogyan épül be a korábbi tanulmányok során már ismertetett V-modellbe (1.1 ábra).



***1.1 ábra**. A modellalapú fejlesztés és kódgenerálás illeszkedése a rendszertervezés folyamatába* [1]

Nézzük meg az 1.1-es ábrán szereplő kulcsmozzanatok szerepét:

Szimuláció

A szimuláció egy PC-n végzett ellenőrzése a kívánt vezérlési, szabályozási funkciónak. A szimulációhoz rendelkezésre álló szoftverkörnyezetek, mint például a Simulink, jelentősen lerövidítik és olcsóbbá teszik az algoritmusok kipróbálását és a vezérlési paraméterek behangolását. Természetesen ehhez az szükséges, hogy a szimulációnál a külső környezetről rendelkezésre álljanak a megfelelő modellek. Ezeket a külső környezetet szimuláló modelleket az egyes alkalmazásokhoz – mint repülőgépipar, autóipar – külön ezzel foglalkozó cégek állítják elő (ilyen például a CarSim http://www.carsim.com/), de vannak például magában a Simulink-ben is környezet specifikus toolbox-ok, modellek is.

Rapid prototyping, gyors prototípuskészítés

A gyors prototípuskészítés célja a szimuláció során feltárt kritikusabb funkcióknak a valóságot megközelítő szituációban való kipróbálása. Ebben az esetben a vezérlési modell már nem a PC-n fut, de még nem is a tényleges hardveren, hanem valamilyen rapid prototípus eszközön. A Simulink autóipari alkalmazásainál például bevett ilyen prototípuskészítő platform a dSpace Autobox-a.

1.2 ábra. *A dSpace MicroAutoBox hardvere* [2]

Ezek a speciális gyors prototípusfejlesztő eszközök általában valamilyen ipari PC platformra épülnek és a számítási kapacitásuk sokszorosa a végtermékben szereplő beágyazott hardvernek (természetesen az áruk is csillagászati). Ugyanakkor többnyire real-time viselkedésre képesek és rendelkeznek azokkal a hardver interfészekkel és software driverekkel, amelyeket a tényleges platformnak is nyújtania kell (a MicroAutoBox esetében ilyen perifériák a CAN, LIN és különböző analóg/digitális I/O-k). Ezeket a gyors prototípus eszközöket már a tényleges környezetben használva a szoftver az algoritmusok és azok paramétereinek további gyors finomítása, és esetleg új lehetőségek kipróbálása is lehetővé válik.

On-target Rapid prototyping, gyors prototípuskészítés a célhardveren

A célhardveren történő prototípus feladata, hogy a vezérlési koncepciót még tovább lehessen finomítani a tényleges target-hez fűződő időzítési viszonyokkal és I/O késleltetésekkel. A célhardver ebben az esetben még az esetek többségében nem a végleges vezérlőt jelenti, hanem annak egy azonos hardware platformmal bíró korábbi változatát, vagy a processzorhoz tartozó gyári fejlesztőkészletet.

SIL Software In the Loop testing

Ebben a fázisban a szoftver egyes moduljai már készen állnak és jóval komplexebb funkciókkal rendelkeznek, mint a gyors prototípus fejlesztés során használt vezérlés/funkció töredékek. Ugyanakkor még nem teszteltek, és ez a tesztelés először a gyors kipróbálást lehetővé tevő PC-s környezetben történik meg. Amennyiben a szimulációban már az összes modul megfelelően működik, akkor lépnek tovább. Természetesen ebben a szimulációban még nem lehet a real-time viselkedést ellenőrizni és a targetfüggő számábrázolást sem nagyon tudjuk vizsgálni (fixpontos aritmetika stb).

PIL Processor In the Loop testing

Ez a lépés történhet szimulátorban vagy egy célprocesszort tartalmazó fejlesztőkártyán. Célja hogy a processzorarchitektúra-függő számábrázolási problémákat felfedje.

HIL Hardware In the Loop testing

Ez a lépés már a tényleges hardveren ellenőrzi a szoftver működését. Ellenőrizhetőek a pontos időzítési viszonyok és a real-time viselkedés is.

A mérés célja – azon túl, hogy betekintést adjon arról, hogy a Matlab Simulink Real-Time workshop hogyan generál kódot a Simulink blokkokból –, hogy megismertesse a hallgatókat

azzal, hogy a valóságban hogyan zajlik egy ilyen fejlesztési folyamat. Ezért a tényleges fejlesztési sorból a szimuláció és az on-target rapid prototyping lépést fogjuk végrehajtani a labor során.

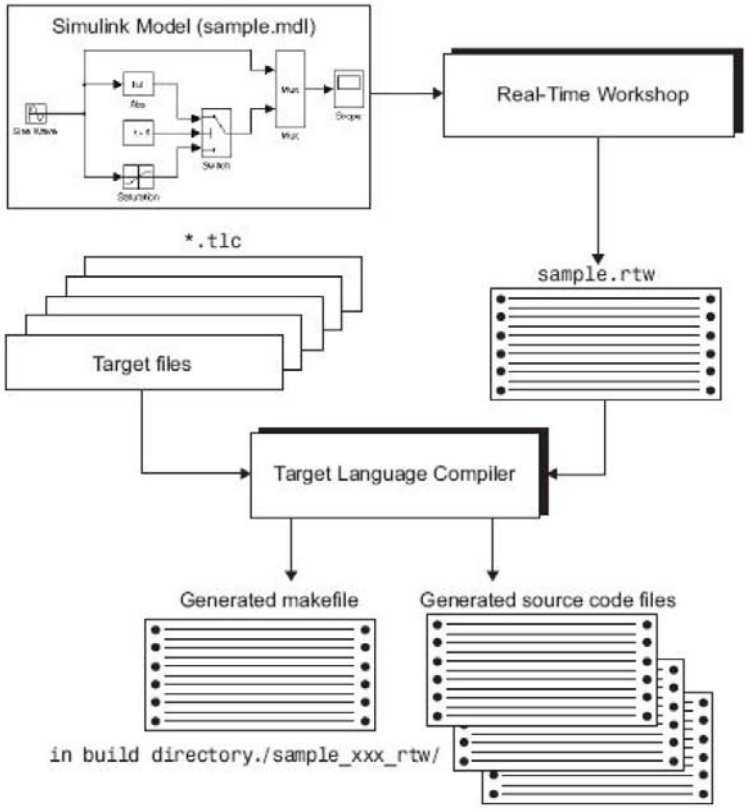
2. Kódgenerálás a Matlab Simulink Real-Time workshop Embedded coder segítségével

Mielőtt hozzákezdenénk egy rendszer fejlesztéséhez, először nézzük meg, hogyan működik a Simulink Real-Time workshop Embedded coder kódgenerálási folyamata, hiszen ezt fogjuk a későbbiekben felhasználni.

2.1. A kódgenerálás folyamata

A részletekre való kitérés nélkül a kódgenerálás folyamata a következő ***(1.3 ábra)***:

1. A Simulink szimulációban futtatható *sample.mdl* modell fájlból a Real-Time workshop előállítja a *sample.rtw* fájlt. Ez a sample.rtw fájl a Simulink modell fájl köztes fordítása, ami leírja a modellben szereplő blokkokat, azok ki- és bemeneteit, paramétereit, állapotait és egyéb, a modellhez kapcsolódó jellegzetességeket.
2. A *sample.rtw* fájlt a Target Language Compiler célhardver specifikus kóddá fordítja. Ehhez a fordításhoz viszont fel kell használnia az egyes modulokhoz tartozó **.tlc* fájlokat, amik egy script nyelvű leírásként tartalmazzák a célhardvertől függő kód generálásához szükséges információkat.



***2.1 ábra**. A Real-Time workshop Embedded coder kódgenerálási folyamata* [2]

3. A generált fájlokat a felhasználó lefordítja és letölti a target-re. Ezt a folyamatot megkönnyítendő a Target Language Compiler képes *make* fájlok előállítására is, amennyiben erre fel van készítve (nálunk ez nem valósul majd meg, mi külső *make* file-t használunk).

TLC Target Language Compiler fájlok

A TLC fájlok egyszerű ASCII script fájlok. Gyakorlatilag ezek a fájlok mondják meg a Target Language Compiler-nek, hogy milyen funkciókra milyen kódot illesszen a célhardvertől függő forrásba.

A TLC fájloknak két csoportja van: a System Target File (STF), ez az adott célhardverhez tartozó fő scrip file és egyben a Target Language Compiler belépési pontja is. A System Target File leginkább a C program main() függvényéhez hasonlítható, bár felépítése és jellege egészen más, csak a funkciója hasonló. Gyakorlatilag ez felügyeli az egész kódgenerálási folyamatot, és az esetek többségében ez felelős a főprogram célhardvertől függő inicializációs részének specifikálásáért is. System Target File környezetenként egy létezik, nálunk ez a file a stm3210c.tlc lesz (ez a file *c:\Program Files\MATLAB\R2007a\ toolbox\rtw\targets\stm3210c \stm3210c* könyvtárban található, de ez csak egy „main" konfiguráló funkciót tölt be, a tényleges inicializációs részeket a stm3210c_main.tlc file-ban találjuk).

A második számosabb csoportot az egyes modulokhoz tartozó TLC fájlok alkotják. Gyakorlatilag minden célhardver függő Simulink blokkhoz létre kell hoznunk egy-egy ilyen script fájlt, ami megadja, hogy milyen kódrészletnek kell lefutnia inicializációnál és milyen kódrészletnek kell akkor lefutnia, amikor a bemenetből előállítjuk a kimenetet. Természetesen a TLC fájlok ezeknél a korlátozott funkcióknál jóval többre is képesek, de ehhez a több száz oldalas dokumentáció mélyére kellene ásni [3], [4]. Az általunk használt blokkokhoz tartozó TLC file-ok a *c:\ProgramFiles\MATLAB\R2007a\toolbox\rtw\targets\stm3210c\blocks* könyvtárban találhatóak meg.

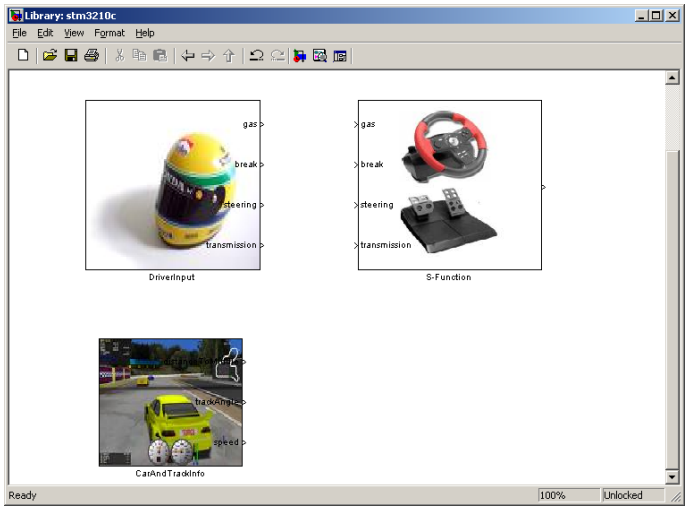
2.2. Példa saját modul készítésére

A mérés egyik célja, hogy megismerkedjünk a kódgenerálás folyamatával. Ezt úgy tudjuk a legegyszerűbben megtenni, ha egy mintapéldán keresztül bemutatjuk, hogy hogyan lehet egy a célhardware-hez kötődő kódot előállító Simulink blokkot létrehozni. *(Ez a példa egy már meglévő blokk elkészítésének lépéseit mutatja be. Nem az a feladat, hogy ezt a blokkot még egyszer elkészítsük.)*

A célhardware jelen esetben az STM3210C kártyán található 4 darab LED lesz. Ezek kezelését kell ellátnunk Simulink blokkok segítségével.

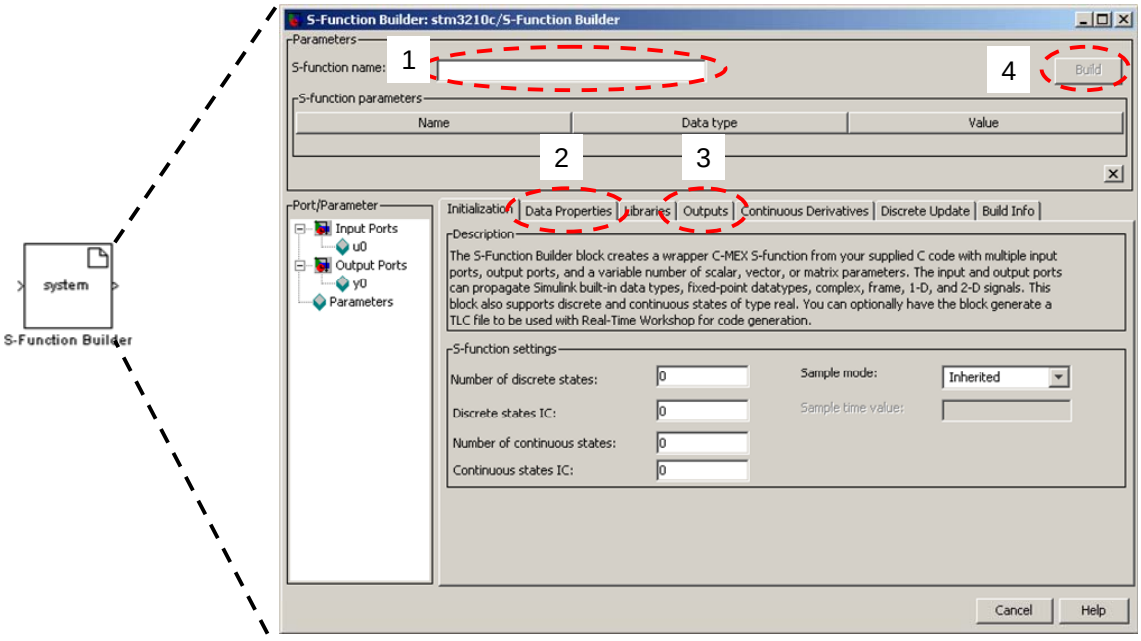
Első lépésként a Simulink-ot kell a Matlab alól elindítani a *simulink* paranccsal. Következő lépésként workspace-t kell a megfelelő könyvtárra állítani, ebben az esetben ez a *c:\Program Files\MATLAB\R2007a\toolbox\rtw\targets\stm3210c\blocks*, mert itt tároljuk a fejlesztőkártyához tartozó Simulink blokkokat.

A *blocks* könyvtárban találjuk az *stm3210c.mdl* Simulink könyvtárat, amihez a példában hozzá fogjuk adni a LED vezérlést ellátó modult.



***2.2 ábra**. Az stm3210c.mdl modell blokk könyvtár*

A LED vezérlést úgynevezett S-Function segítségével fogjuk megoldani. Az S-Function egy Simulink blokk körítést hoz létre a C nyelvű utasításainkhoz, lehetőséget adva arra, hogy magához a LED vezérléshez egy „nativ" már meglévő C nyelvű kódot használjunk, amit a fordítási folyamat során majd a Simulink beépít a generált kódba. Egy ilyen S függvényt úgy tudunk legegyszerűbben elkészíteni, ha a Simulink beépített *S-Function Builder* blokkját használjuk. Ezt a blokkot a Simulink Library Browser-ben a *Simulink/User-Defined Functions* könyvtár alatt találjuk.

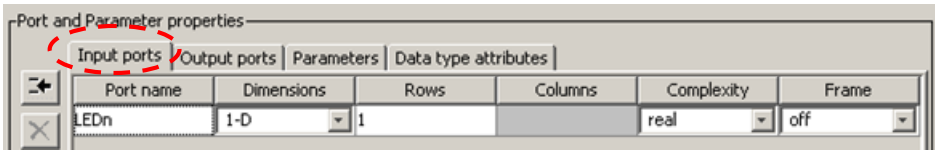


***2.3 ábra**. Az S-Function Builder*

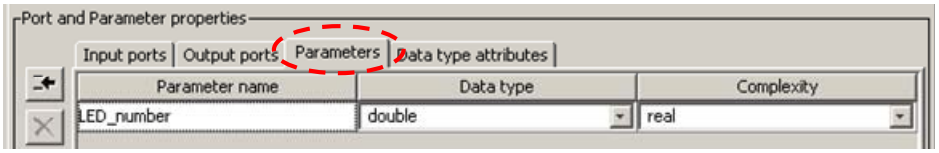
Ezt a modult megnyitva lehetőségünk van a nevét megváltoztatni és a különböző bemeneteit és kimeneteit meghatározni. Az *„1" S-Function name* ablakban tudjuk a blokk nevét megadni, de ami ennél fontosabb, az a *„2" Data Properties* és *„3" Outputs* fül, amivel lehetőségünk lesz a modul bemeneteit, kimeneteit és viselkedését megadni, valamint a *„4" Build* fül, amivel a tulajdonságok megadása után el tudjuk készíteni a blokkhoz tartozó alap file-okat (később lesz róla szó, hogy ezek milyen file-ok) ***2.3 ábra***.

A LED blokkunk működéséhez a *„2” Data Properties* ablakban meg tudjuk adni a modul Simulinkban használt bemeneteit, kimeneteit és paramétereit. A példában átnevezzük a bemenetet LEDn-re, *„1.”* a kimenetet változatlanul hagyjuk (azért nem töröljük, mert a Simulink szereti, ha a blokkoknak van kimenete). Megadunk egy paramétert a *LED_number*-t *„2.”*, amit arra fogunk használni, hogy a blokk egy legördülő menüjében kiválasszuk, hogy melyik LED-et akarjuk vezérelni (a paraméter típusának meghagytuk a *double*-t, mert ez nem fog jelen esetben a kódba legenerálódni, csak a kód előállításánál használjuk fel). Beállítjuk a bemenet és a kimenet adat típusát *„3.”*. Majd végül meghatározzuk a blokk viselkedését a szimulációban, ami jelen esetben a bemenet kirakása a kimenetre *„4.”* ***2.4 ábra***.

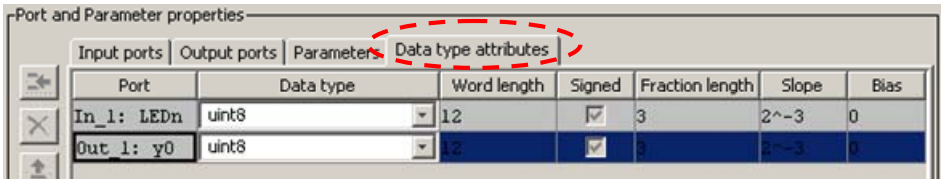
1.



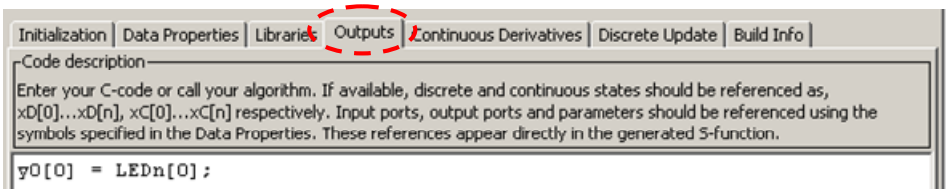
2.



3.



4.



***2.4 ábra**. Az S-Function Builder konfigurálása*

Ezekkel a lépésekkel elkészítettük a S függvényünket, amihez a jobb felső sarokban található *„Build”* gomb segítségével automatikusan le tudjuk generálni a blokk viselkedését leíró file-okat.

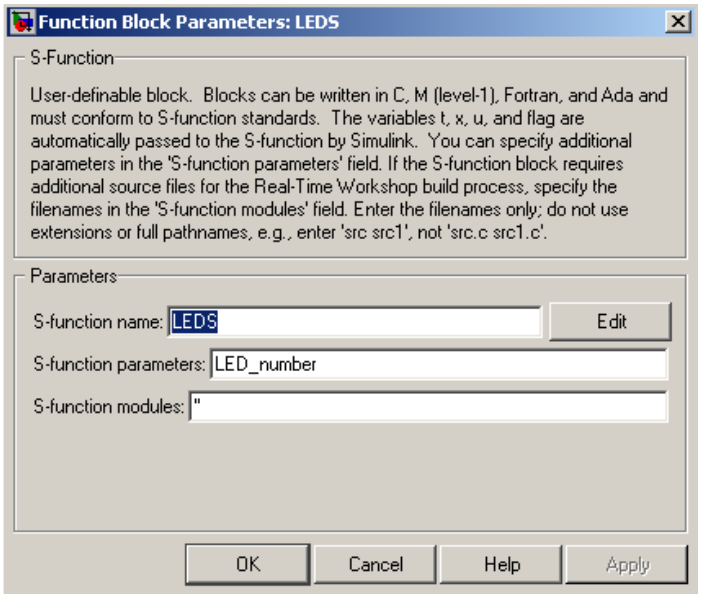
A *„Build”* eredményeként a következő file-ok állnak elő.

- **LEDS.c**: a LEDS blokk szimulációban mutatott viselkedését leíró forrásfile.
- **LEDS_wrapper.c**: a generált target-en futtatható kódhoz tartozó vezérlőfüggvény (jelen pillanatban ugyanazt a funkciót látná el, mint a szimulációban, tehát a bemenetet átmásolná a kimenetre)
- **LEDS.tlc:** az a Target Language Compiler file, ami levezényelné a blokkhoz tartozó kód generálását.
- **LEDS.mexw32:** a LEDS blokk szimulációban mutatott viselkedését leíró Simulink által végrehajtható lefordított file.

A file-ok közül a **LEDS.c** -vel és a **LEDS.mexw32**-vel a továbbiakban nincs semmi dolgunk, ezek a Simulink szimulációkban a megadott funkciót fogják elvégezni. A kódgeneráláshoz

tartozó **LEDS_wrapper.c**, **LEDS.tlc** file-okkal pedig a későbbiekben fogunk foglalkozni, de először fejezzük be a Simulink blokk elkészítését.

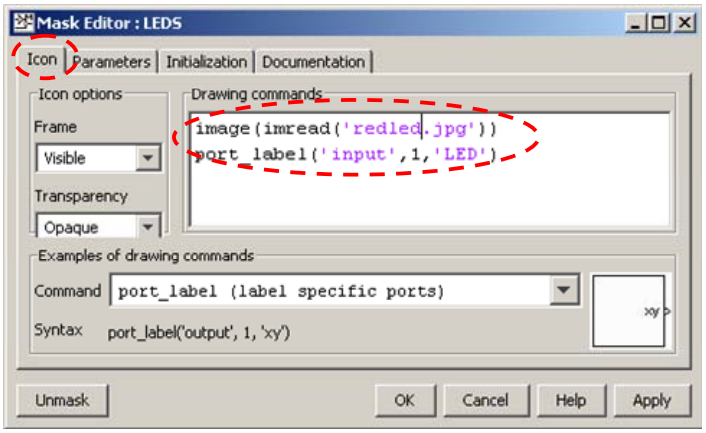
A Simulink blokk elkészítéséhez, véglegesítéséhez le szokták cserélni az *S-Function Builder* blokkot egy másik, az alkalmazásokban praktikusabb „sima" *S-Function* blokkra. Ennek az a magyarázata, hogy az *S-Function Builder* blokk nagy segítségét nyújtott abban, hogy létrehozzuk a szimulációhoz és kódgeneráláshoz tartozó alap file-okat, de miután ezeket egyszer létrehoztuk nem célszerű engedni, hogy más esetleg véletlenül felülírja őket a *„Build"* gomb lenyomásával. A lecserélés folyamata igen egyszerű: letöröljük az *S-Function Builder* blokkot (adott esetben akár egy másik saját könyvtárba át is másolhatjuk, hogy ne vesszen el), és lerakunk a helyére egy „sima" *S-Function* blokkot (ugyanúgy a *Simulink/User-Defined Functions* könyvtár alatt találjuk). Majd megadjuk ennek a blokknak a legenerált S függvény nevét és paramétereit (Az *„Edit"* ikonnal ellenőrizhetjük is, hogy megtalálja-e a rendszer az előbb generált file-okat) **2.5 ábra**.



***2.5 ábra**. Az S-Function Builder blokk lecserélése „sima" S-Function blokkra*

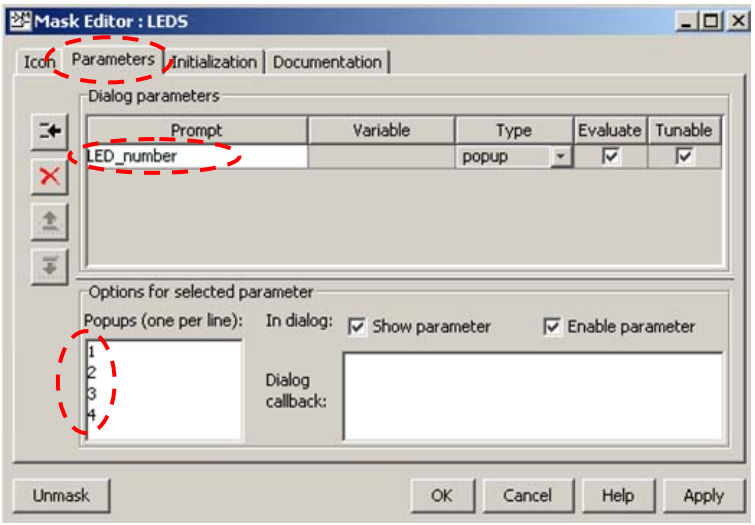
Fontos tanács, hogy azokban az esetekben, amikor új file-okat adunk a Matlab toolbox-aihoz, mint most ennél a LED blokknál tettük, célszerű lefuttatni a Matlab-ban a „rehash TOOLBOX" parancsot, hogy frissítse a path-hoz tartozó file nyilvántartását.

Az S-Function védetté tétele után a következő lépés a *LED_number* paraméter megadási módjának specifikálása és a modul külsejének megváltoztatása. A Simulink környezetben ez nevezik *„Mask"*-olásnak. A LEDS blokk maszkjának elkészítése elég egyszerű folyamat, és a LEDS blokk-hoz tartozó jobb egérgomb menüben látható *Mask S-Function* parancs segítségével tehető meg.



***2.6 ábra**. Az S-Function maskolása: Icon testreszabás*

A *„Mask"*-olás *„Icon"* menüjében a blokk ikonját tudjuk kialakítani. Itt a *Drawing command* résznél gyakorlatilag le tudjuk programozni a kinézetet **2.6 ábra** (jelen esetben elneveztük a bemeneti portot LED-nek és az ikon képének a *redled.jpg*-t választottuk ki). A *„Parameters"* fülben pedig az ikonra klikkelve előugró konfigurációs menüt tudjuk előállítani.



***2.7 ábra**. Az S-Function maskolása: Paraméter beállítás*

Ebben az esetben a *LED_number* paramétert úgy specifikáltuk, hogy a felhasználó egy legördülő menüből ki tudja választani a fejlesztő kártyán található 1-4 LED bármelyikét.

Ezekkel a lépésekkel előállítottuk a LED Simulink blokkjának ikonját és konfigurálhatóvá tettük azt. A következő feladat, hogy a tényleges LED villogtató funkcionalitást hozzáadjuk a blokkhoz.

Ehhez egy picit bele kell néznünk a **LEDS.tlc** és a **LEDS_wrapper.c** file-okba.

A LEDS.tlc számunkra érdekes részlete a kimenet előállítását leíró *„Outputs"* függvény:

```
%%Function: Outputs ============================================
%%
%% Purpose:
%%      Code generation rules for mdlOutputs function.
%%
```

```
%function Outputs(block, system) Output
  /* S-Function "LEDS_wrapper" Block: %<Name> */

  %assign pu0 = LibBlockInputSignalAddr(0, "", "", 0)
  %assign py0 = LibBlockOutputSignalAddr(0, "", "", 0)
  %assign nelements1 = LibBlockParameterSize(P1)
  %assign param_width1 = nelements1[0] * nelements1[1]
  %if (param_width1) > 1
    %assign pp1 = LibBlockMatrixParameterBaseAddr(P1)
  %else
    %assign pp1 = LibBlockParameterAddr(P1, "", "", 0)
  %endif
  %assign py_width = LibBlockOutputSignalWidth(0)
  %assign pu_width = LibBlockInputSignalWidth(0)
  LEDS_Outputs_wrapper(%<pu0>, %<py0>, %<pp1>, %<param_width1>);

  %%
%endfunction
```

Ebben a TLC script nyelv különösebb ismerete nélkül is jól követhető a folyamat, amiben a **pu0, py0, pp1** változókban eltárolja a program a Simulink blokk bemeneteit, majd ezeket átadja a LEDS_Outputs_wrapper függvénynek (azok a sorok, amiknél nincs % karakter és valamilyen TLC parancs azok direktben a generált C kódban meg fognak jelenni, így a LEDS_Outputs_wrapper függvényhívás is.).

A **LEDS_wrapper.c** file LEDS_Outputs_wrapper függvénye pedig egyszerűen az *S-Function Builder-ben* megadott funkcionalitást hajtja végre. Ebbe a függvénybe tudnánk beírni a tényleges műveleteket is.:

```
void LEDS_Outputs_wrapper(const uint8_T *LEDn,
                          uint8_T *y0,
                           const real_T  *LED_number, const int_T p_width0)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
          y0[0] = LEDn[0];

/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}
```

Ez így teljesen működőképes is, de ami egy picit bosszantó, hogy sok esetben nem akarunk külön függvényhívást végrehajtani minden LED villogtatásért, hanem sokkal jobb lenne, ha a generált kód ún. ***inline***-lenne, tehát egy az egyben beépülne a modellhez létrejövő fő C file-ba. Természetesen erre is lehetőségünk van, mert a **LEDS.tlc** file-t módosíthatjuk ennek megfelelően is. Ehhez a módosításhoz azt kell tudnunk, hogy a fejlesztőkártya LED-jei a következő módon kezelhetőek:

- **STM_EVAL_LEDInit()**: LED-ek inicializálása, lehetséges paraméter LED1, LED2, LED3, LED4.
- **STM_EVAL_LEDOn()**: LED-ek bekapcsolása, lehetséges paraméter LED1, LED2, LED3, LED4.
- **STM_EVAL_LEDOff()**: LED-ek kikapcsolása, lehetséges paraméter LED1, LED2, LED3, LED4.

Ezek alapján a következőképpen alakulna az ***inline*** funkcionalitást megvalósító **LEDS.tlc**-file (Ez található a mérésen használt verzióban is):

```
%% Function: Start ============================================================
%%
%% Purpose:
%%      LED initialization code.
%%
%function Start(block, system) Output
          %<LibAddToCommonIncludes("stm32_eval.h")>
       %assign p0 = CAST("Number",LibBlockParameterValue (P1, 0))
       STM_EVAL_LEDInit(LED%<p0>);

%endfunction


%%Function: Outputs ==========================================================
%%
%% Purpose:
%%      Code generation rules for mdlOutputs function.
%%
%function Outputs(block, system) Output

  %assign u0 = LibBlockInputSignal (0, "", "", 0)
  %assign p0 = CAST("Number",LibBlockParameterValue (P1, 0))

 if(%<u0> == 1)
      STM_EVAL_LEDOn(LED%<p0>);
     else
      STM_EVAL_LEDOff(LED%<p0>);

%endfunction
```

Ezzel a módosítással gyakorlatilag végére értünk a saját Simulink blokk elkészítésének. A következő rész a blokkok használatát, tehát egy modell elkészítését és az abból való kódgenerálást tárgyalja.

2.3. Simulink Modell készítése és az abból való kódgenerálás

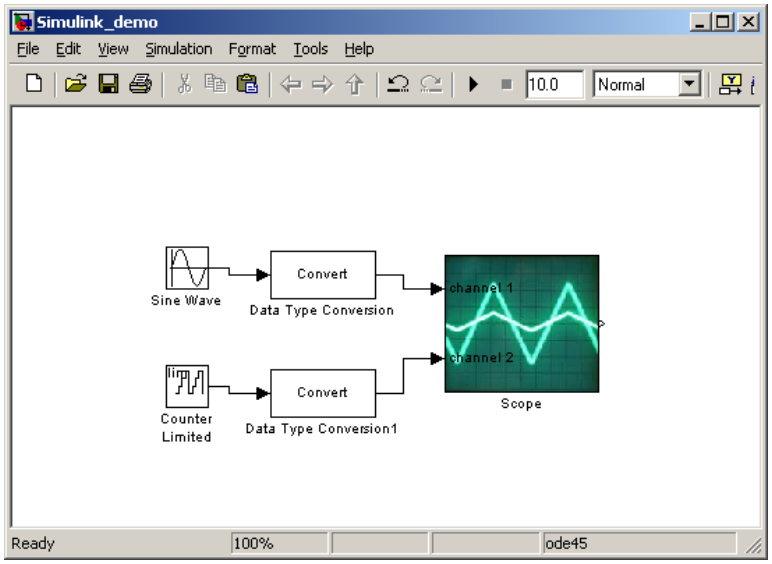
Ebben a fejezetben egy példán keresztül bemutatjuk, hogy hogyan kell az előző fejezetben létrehozott blokkokból egy szimulációs modellt készíteni, majd abból kódot generálni, és letölteni az így generált kódot az STM3210C fejlesztőkártyára.

Első lépésként át kell állítani a Matlab workspace-ét egy nem a Matlab termékcsalád alá tartozó könyvtárhoz, és itt létre kell hozni egy új Simulink modell-t (*Simulink/File/New-> Model*). *(A modellt majd célszerű átnevezni Simulink_demo névre, ezzel biztosítva, hogy a fordításnál ne kelljen a make file tartalmát megváltoztatni)*.

A modellbe gyakorlatilag tetszés szerint pakolhatunk Simulink könyvtári blokkokat. Fontos, hogy a Simulink Library browser-ben megtalálható az az Stm3210c könyvtár, amit az előző fejezetben kiegészítettünk egy LED vezérlő blokkal.

A mintaalkalmazást ennek a könyvtárnak a segítségével készítettük el. Ez a mintaalkalmazás egy *Scope kijelző* használatával az STM3210C demókártyán megjelenít egy szinuszjelet és egy számláló értéket. A Modellben használt *Scope* blokk elkészítése a mérés egyik feladata,

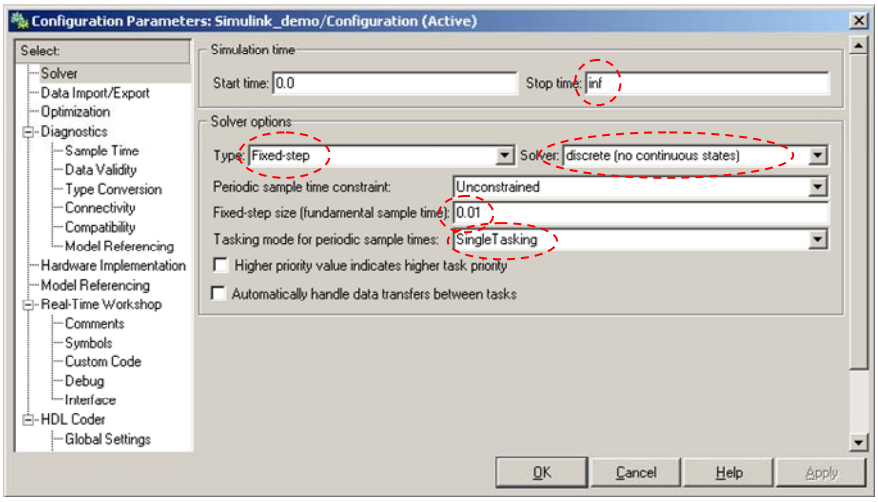
tehát alapértelmezésként nincs benne a könyvtárban. Fontos megjegyezni, hogy a beágyazott kódgenerációra készülő modelleknél nagyon fontos a *Convert* típuskonverziós blokkok alkalmazása ahol ez szükséges, mert ezek használata nélkül biztos, hogy hibajelzést fogunk kapni akkor, ha nem azonos típusú bemenetet és kimenetet kötünk össze. Ez blokk a *Simulink / Commonly Used Blocks* könyvtárában található.



***2.8 ábra**. Minta szimulációs modell*

Az ilyen úton elkészülő modell szimulálható a PC-s környezetben (célszerű mindig ezt megtenni), de elsősorban a cél a kódgenerálás ebből a modellből. A kódgenerálás első lépéseként állítsuk be a szimuláció paramétereit a *Simulation / Configuration Parameters* segítségével.

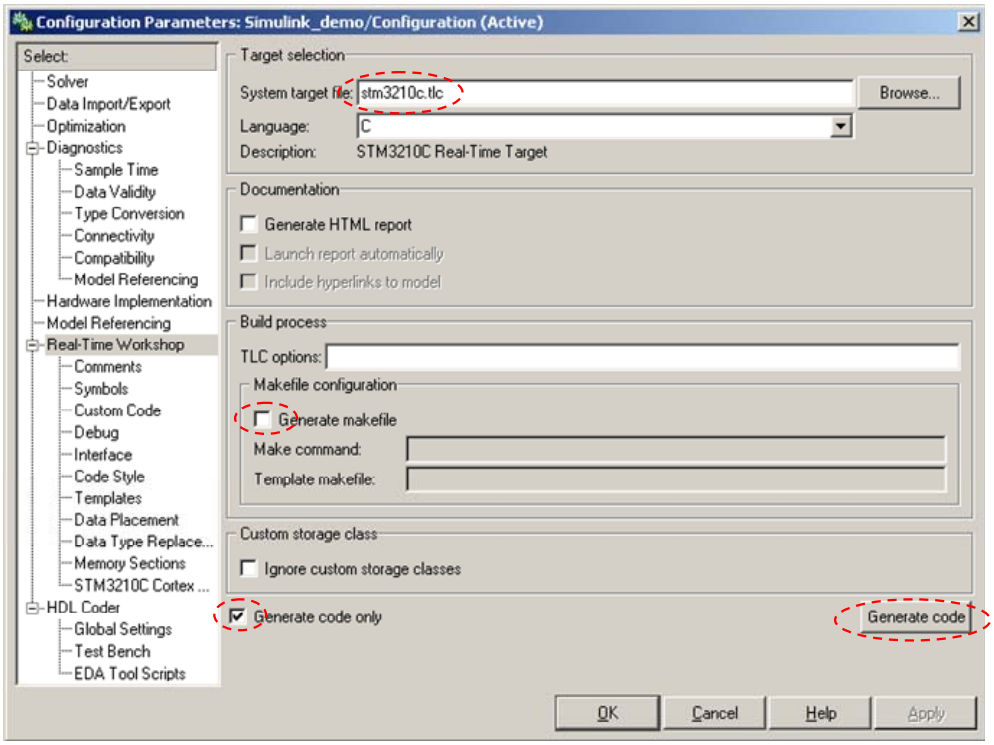
Az első lépés az alap szimulációs tulajdonságok beállítása. Természetesen egy kódgeneráláshoz használt szimuláció mindig csak *Fixed-step* és *discrete* lehet. További lépésként állítsuk még be a szimulációs lépésközt 10 ms-ra (a jelenlegi generált kód csak ezzel működőképes) valamint a szimulációs idő végét állítsuk végtelenre: *inf*-re, hiszen a letöltött programot nem szeretnénk 10 másodperc után felfüggeszteni, ami az alapbeállításban meg van adva. A Tasking mode-ot válasszuk *Single Tasking*-nak



***2.9 ábra**. Minta szimulációs modell Solver beállításai*

A következő lépésként a baloldalt található menüből válasszuk ki a *Hardware Implementation* fület, és állítsuk be a hardware típusát ARM7/8/9-re, hogy a számábrázolási mód megfelelő legyen a demókártyánk számára.

A következő konfigurációs lehetőség már közvetlenül a kódgeneráláshoz tartozik. Válasszuk ki a baloldali menüből a Real-Time Workshop fület, és a System target file-nak adjuk meg az *stm3210c.tlc*-t, ami a demókártyánk alap hardware leíró file-ja. Kapcsoljuk ki továbbá a *make file* generálási opciót, mert azt jelenleg nem támogatjuk a demókártyánkhoz, és kapcsoljuk be a *Generate code only* opciót, ami után már akár a *Generate code* gombbal le is generálhatjuk a modellhez tartozó C file-okat.



***2.10 ábra**. Minta szimulációs kódgeneráláshoz tartozó beállításai*

A generált file-ok a workspace alatti *Simulink_demo_stm3210c_rtw* könyvtárba jönnek létre. Ezek a file-ok jelen esetben:

- **ert_main.c:** Minta main file, nekünk nincs rá szükségünk, törölhető.
- **Simulink_demo_main.c:** A demó kártya szoftver környezetéhez tartozó main file. A programunk belépési pontja. Szükségünk van rá.
- **Simulink_demo_data.c:** A modellhez és annak blokkjaihoz tartozó különféle adatok. Szükségünk van rá.
- **Simulink_demo.c:** A tényleges működését megvalósító file.

Valamint az ezekhez tartozó fejléc file-ok. A generált header és sorce file-okat másoljuk át a *c:\user\ws_devel\Simulink_demo* könyvtárba, majd indítsuk el az *eclipse* fejlesztőkörnyezetet, amelynek a segítségével le lehet fordítani és tölteni a generált kódot (Ezt már más laborok alkalmával hasonló környezeten kipróbáltuk, itt is a már ismert lépéseket kell megtenni: *make all,* majd *program*).

A demó kártyán a *reset* megnyomása után el fog indulni az automatikusan generált kód.

3. Mérési feladatok

A laborban az előzőekben bemutatott ismereteket felhasználva a következő mérési feladatokat kell elvégezni:

3.1. A kódgenerálás gyakorlása

1. Feladat: A 2. fejezetben bemutatott lépéseket felhasználva készítse el a 2.8-as ábrán látható Scope Simulink blokkot!

A blokk a *c:\user\devenv\Libraries\STM3210C_EVAL\stm3210c_eval_lcd.c*-ben található *LCD_ScopeInit* és *LCD_ScopeDraw* függvényeket használhatja fel (a függvények működésének leírása megtalálható a forrásfile kommentjeiben).

2. Feladat: Készítsen egy Simulink modellt a Scope blokk kipróbálására. Generáljon a modellből kódot és töltse le a demókártyára!

A jegyzőkönyvben ismertesse a generált kód szerkezetét és az ezzel kapcsolatos észrevételeit.

3.2. A modellalapú rendszerfejlesztés gyakorlása

Itt a feladat az IE225-ös Bosch Beágyazott Rendszerek laborban található szimulált autóhoz egy tempomat és egy automatikus kormányzási vezérlés elkészítése. Ezekből a tempomat feladatnak mindenképpen bele kell férnie az időbe.

1. Feladat: A szimulátor gázpedál – sebesség összefüggéseinek identifikálása.

Vezesse az összeállított Simulátor programot (lehetőleg válasszon egy nem lejtős pályát és ne nagyon rángassa a kormányt, valamint ne fékezzen, mert csak a gázpedál – sebesség kapcsolatra vagyunk kíváncsiak).

A vezetés közben a mérőtárs készítsen a CANalayzer programban egy *log*-ot a vezetésről. Az elkészített *log*-ból exportálja ki a gázpedál és sebesség adatokat Matlab változókként.

Mivel a kiexportált adatok nem egyenletesen mintavételezettek interpolálja azokat egy egyenletes 10 ms-os mintavételezésre (használja például az *interp1* Matlab függvényt).

Az egyenletesen mintavételezett adatokat importálja be az *System Identification* toolbox-ba és készítse el ennek segítségével a gázpedál – sebesség összefüggés modelljét.

Ezzel a lépéssel megteremtjük magunknak a környezetet leíró blokkokat, amelyeket a vezérlés szimulációjánál fel tudunk használni.

2. Feladat: Készítsen Simulinkban egy 50 km/h-s tempomat vezérlést!

Ez a feladat a modellalapú rendszertervezés első, szimulációs lépését mutatja be. Az identifikált környezethez itt készítjük el és próbáljuk ki a később élesben a hardware-en is alkalmazott algoritmusokat. A szimulációnál ügyeljen arra, hogy a gázpedálvezérlés

paramétere csak 0-255-ig terjedhet (precízebbek csinálhatnak egy külön identifikációt is, amibe a fék hatását is elemzik és felhasználhatják ezt is a szimulációban).

Azoknak, akik egyből szeretnének szimuláció nélkül a vezérlésükből kódot generálni és letölteni az a kártyára, majd élesben kipróbálni annak a viselkedését, ajánlom, hogy gondolják végig azt a szituációt, hogy egy valós környezetben ezt egy tesztvezetőnek kellene kipróbálnia egy tesztpályán. Ha ez sem bír elég meggyőző erővel, akkor gondolják át azt, hogy nekik kellene a mérőtársuk által összerakott vezérlést tesztpilótaként kipróbálni egy valós autóval. Vállalnák ezt anélkül, hogy a szimulátorban ellenőriznék a vezérlés működését?

3. Feladat: A szimulációban kipróbált, működő algoritmusból generáljon kódot és töltse le a mérőkártyára!

Ehhez használja fel az stm3210c könyvtárban található blokkokat, amelyek fogadják a vezetőtől jövő kormány és gázpedál parancsokat (*DriverInput)*, rendelkezésünkre bocsájtják a gépkocsi állapotát: pálya közepétől való távolság, a következő kanyar görbülete, és a gépkocsi sebessége (*CarAndTrackInfo),* valamint lehetővé teszik számunkra, hogy a szimulált autót vezessük (*DrivingOutputToCar) (**2.2 ábra**)*.

Ez a lépés felel meg a modellalapú rendszertervezés *On-target Rapid prototyping* fázisának.

3.3. Bonyolultabb vezérlési feladat

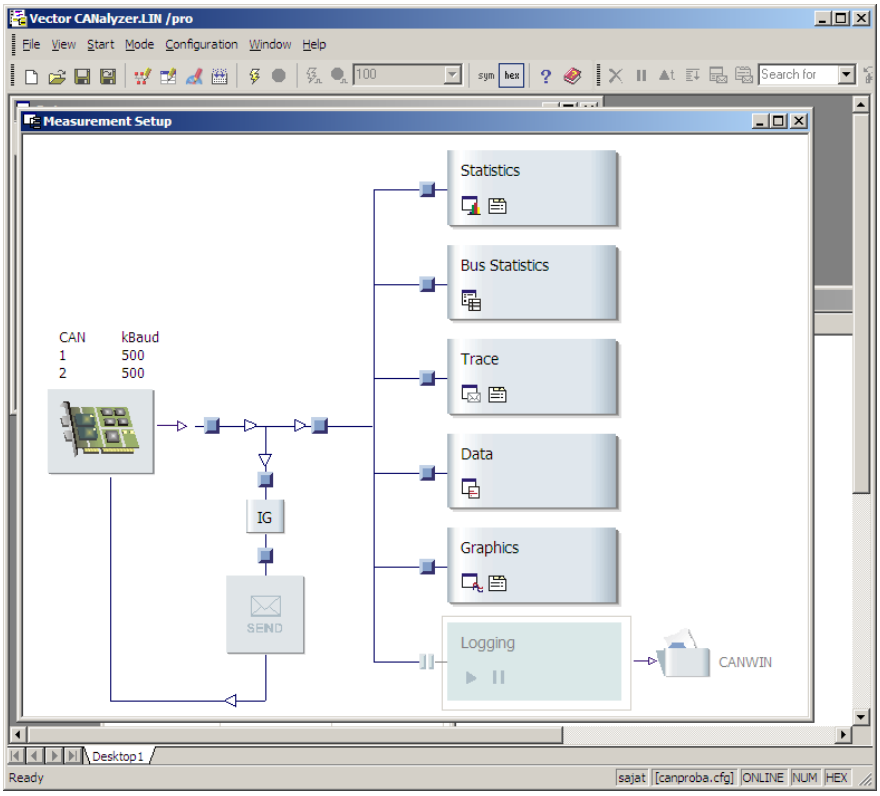
A 3.2 feladat tanulságait felhasználva készítsen egy a tempomat bekapcsolt állapota alatt működő automatikus kormányzást úgy, hogy az automatikus kormányzás igyekezzen középen tartani a gépkocsit.

4. Segítség a mérési feladatokhoz

4.1. Adatgyűjtés CANalayzerrel

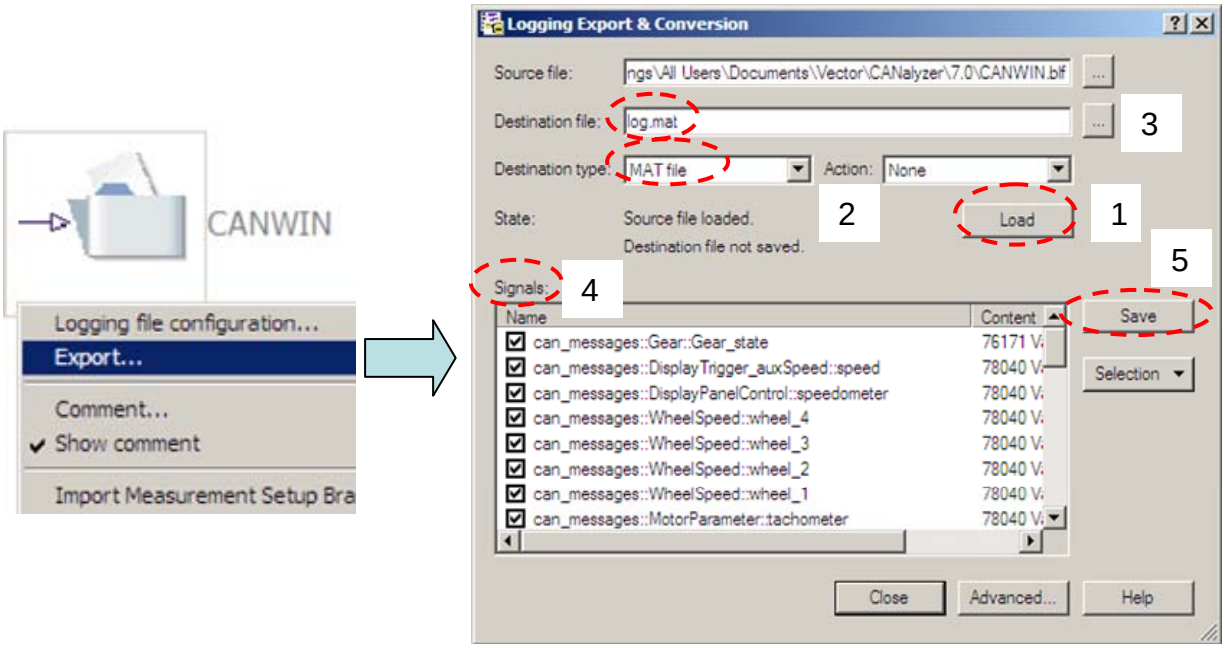
Ahhoz, hogy a Simulink-es szimulációkat el tudjuk készíteni, szükségünk van arra, hogy az autónk viselkedését identifikáljuk. Ennek pedig első lépése az adatgyűjtés, aminek a legkézenfekvőbb módszere a CANalayzer segítségével logokat készíteni az autónk CAN adatforgalmáról. Ennek a folyamatnak a leírása következik most.

A CANalayzer elindítása után a *Measurement Setup* ablakában tudjuk a *Logging* blokkot csatlakoztatni a méréshez (általában alapesetben csatlakoztatva van). A loggolás ezek után automatikausan elkezdődik, amikor elindítjuk a mérést a *Start*  gombbal.



***4.1ábra**. A CANalyzer Measurement Setup ablaka csatlakoztatott Logging blokkal*

A mérés leállítása után a *Logging* blokk mellett található *CANWIN* ikonra jobb egérgombbal kattintva lehetőségünk adódik a loggolt adatokat kiexportálni a CANalyzerből.



***4.2ábra**. A loggolt adatok kiexportálása mat file-ként*

Az előugró ablakban először is a „1” *Load* gombbal be kell töltenünk a loggolt adatokat, majd meg kell határoznunk az exportálás formátumát a „2” *Destination type* menüben, valamint

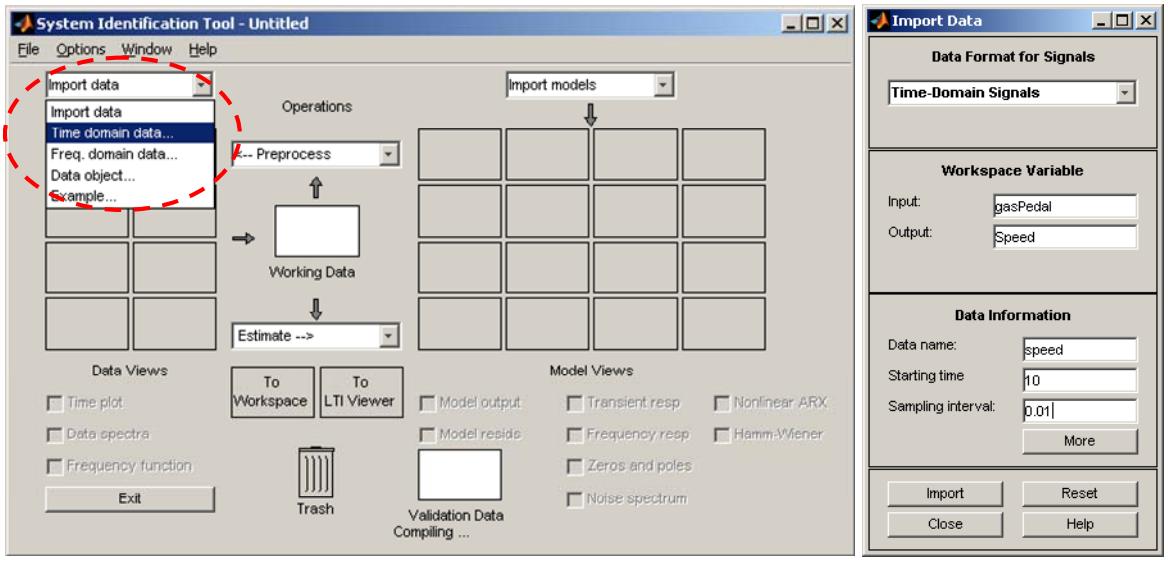
meg kell adnunk a cél file nevét: „3” *Destination file*. Ezek után a „4” *Signal* ablakban ki tudjuk választani, hogy mely paramétereket szeretnénk kiexportálni (célszerű csak azokat, amelyekre valóban szükség lesz), majd végül a „5” *Save* gombbal végre tudjuk hajtani az exportálást. Az így létrejött *mat* file-t egyszerűen be tudjuk tölteni a Matlab workspace-ébe.

4.2. A Matlab rendszeridentifikációs toolbox-ának használata

Ahhoz, hogy a 3.2 fejezet feladatait végre tudjuk hajtani, szükségünk lesz egy szimulációs modellre, amely leírja az autó reakcióját a gázpedál lenyomására. Ezt a legegyszerűbben a Matlab Rendszeridentifikációs toolbox-ával tudjuk elkészíteni, felhasználva a CANalyzerből gyűjtött adatokat.

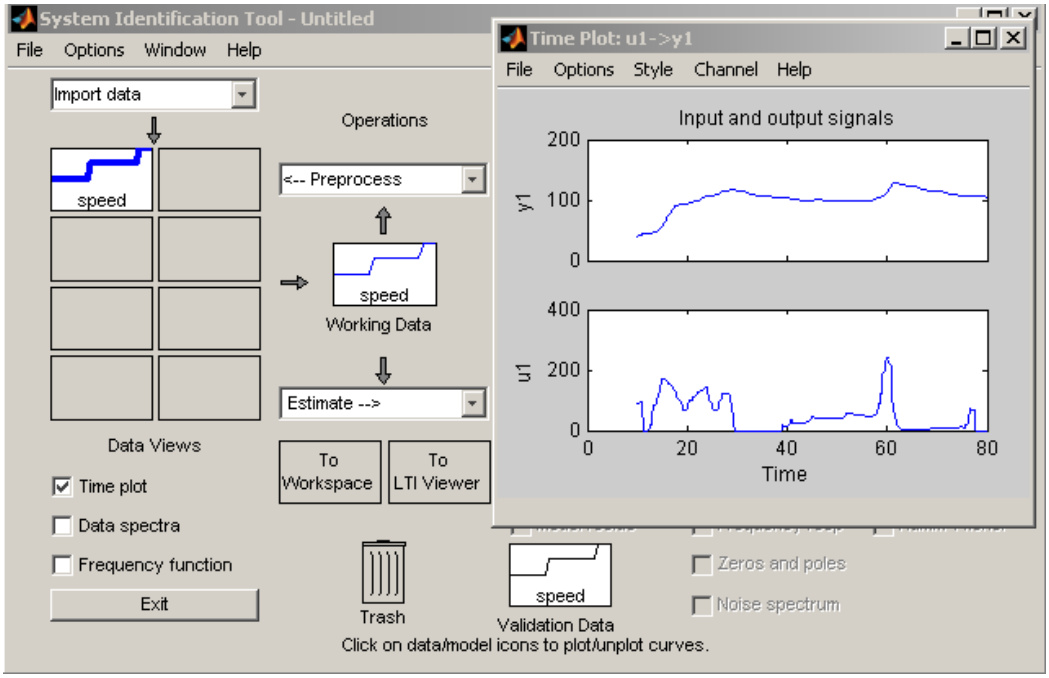
Jelen példánkban feltételezzük, hogy rendelkezésünkre áll a CANalyzerből gyűjtött gázpedál és sebesség paraméter azonos és egységes 10 ms-os mintavételezéssel, tehát már átalakítottuk őket az *interp1* függvény segítségével. Ez a két változó a *gasPedal* és *Speed*.

A System Identification Toolbox-ot a Matlab parancssorából az ***ident*** kulcsszóval tudjuk elindítani. Az első feladatunk, hogy a workspace-ben előállított változókat beimportáljuk a rendszeridentifikációs toolbox-ba. Erre a baloldalt található Import data opciót tudjuk használni. Figyeljünk a mintavételezési gyakoriság, és – amennyiben szükséges – a kezdési időpont helyes megadására.



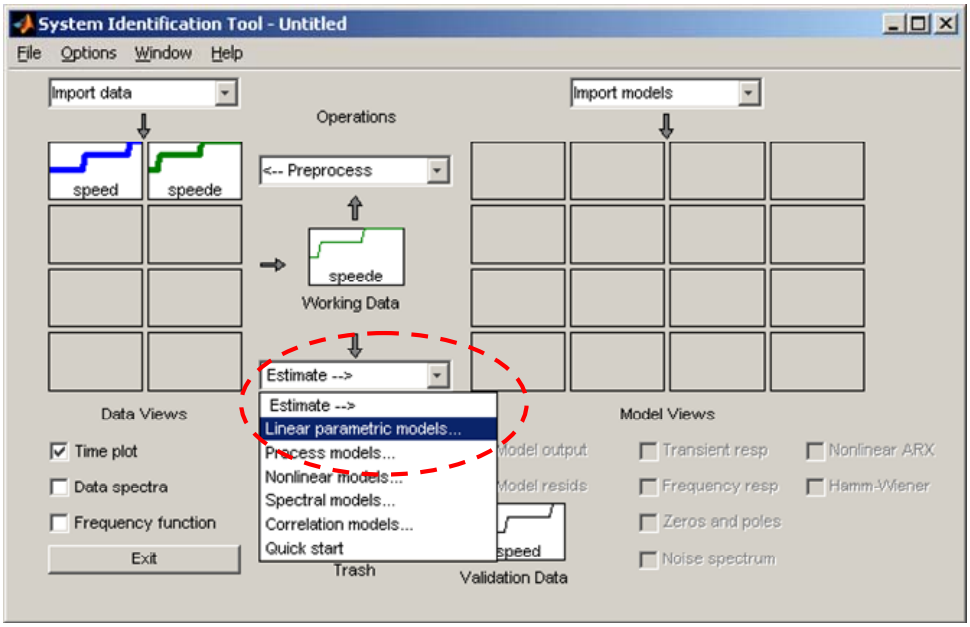
***4.3 ábra**. Adatok importálása a System Identification tool számára*

A beintegrált adat rögtön automatikusan bemásolódik a *Working Data és Vaildation data* blokkba, valamint a bal oldali checkbox-ok segítségével lehetőségünk van ábrázoltatni is az importált értékeket.



***4.4 ábra**. Az importált adatok megjelenítése*

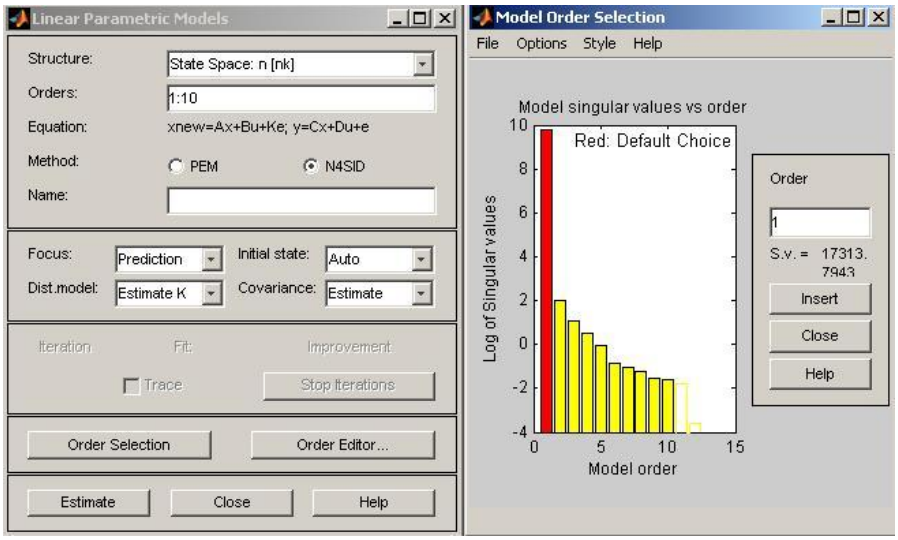
Fontos megjegyezni, hogy a műveletek mindig a *Working Data* részben látható adatokon hajtódnak végre. Következő lépésként ajánlott lehet egy tartományt kiválasztani az importált adatokból, hogy ne ugyanazt az adathalmazt használjuk fel az identifikációhoz és a validációhoz is, tehát a validációnál legyenek független eredményeink is. Ez egyszerűen megtehető a *Working Data* felett lévő *Preprocess/ Select Range* opció segítségével. Az identifikációs adatok kiválasztása után elvégezhetjük a rendszer identifikációját az *Estimate / Linear parametric modells…* paranccsal ***4.5 ábra***.



***4.5 ábra**. Identifikáció*

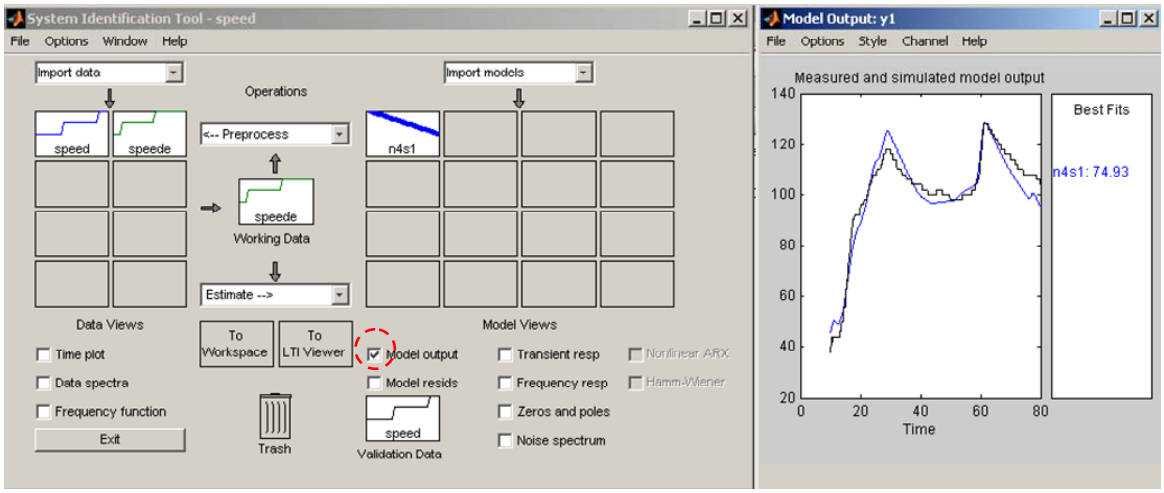
Legegyszerűbb talán, ha egy *State Space* struktúrájú identifikációt próbálunk meg létrehozni. Az identifikálandó rendszer *„order”*-jének meghatározásához kérjük a toolbox segítségét. Ezt

úgy tudjuk megtenni, hogy rálépünk az *Order Selection,* gombra, majd ezek után lenyomjuk az *Estimate* gombot, amely hatására megjelenik a javasolt *order,* ami jelen esetben 1 ***(4.6 ábra)***.



4.6 ábra. *Az identifikáció fel paraméterezése*

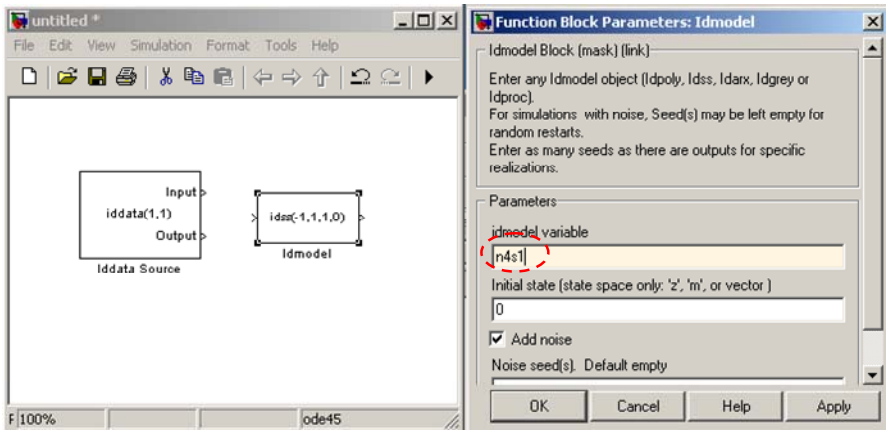
A javasolt *order-t* beírva a megfelelő helyre (*Orders*) és újra lenyomva az *Estimate* gombot a toolbox elkészíti a rendszer identifikációját. Az identifikált modellt lehetőségünk van ellenőrizni, például felrajzoltathatjuk az eredeti és az identifikált rendszer válaszát (sebesség) a bemenő gerjesztésre (gázpedál állása) a *Model Views* rész alatt található *Model Output* checkbox-al ***4.7 ábra*** (látható, hogy egész jól sikerült az identifikáció).



4.7 ábra. *Az identifikáció ellenőrzése*

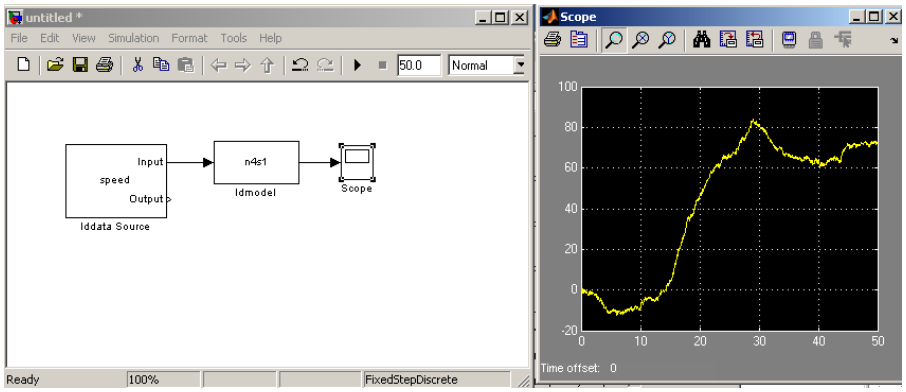
A rendszeridentifikációs toolbox eredményeit nagyon egyszerűen fel tudjuk használni a Simulink szimulációkban. Ennek a felhasználásnak a legegyszerűbb módja, hogy az egér segítségével mind a gerjesztő adatot, mind az identifikált modellt ráhúzzuk a 

ikonra, ezzel hozzáférhetővé téve ezeket a paramétereket a Matlab bármely programja számára. A Simulink-ban pedig úgy tudjuk felhasználni ezeket a kimentett ún. *cell structure* –okat, hogy a Library browserben a *System Identification Toolbox* könyvtárból kiválasztjuk az *Iddata Source* és *Idmodel* blokkokat és azokban megadjuk a *workspace* be kimentett változók neveit.



***4.8 ábra**. Az identifikáció eredményének felhasználása Simulinkban*

Ezek után kedvünkre használhatjuk az identifikáció eredményeit.



***4.9 ábra**. Szimuláció a rendszeridentifikációs tool-ból kiexportált adatokkal és modellel*

5. Irodalomjegyzék

[1] The MathWorks, Inc.: *„MATLAB™ Simulink® Real-Time Workshop® 7 Getting Started Guide”* 2002–2009 The MathWorks, Inc.
[2] dSpace MicroAutoBox sorozat: http://www.dspaceinc.com
[3] The MathWorks, Inc: *„Real-Time Workshop 6 Target Language Compiler (rtw_tlc.pdf), version 6.6”* 2007 The MathWorks, Inc.
[4] Fábián Laura: *„Automatikus kódgenerálás MITMÓT ARM platformra Matlab-Simulink környezet segítségével”* DIPLOMATERV FELADAT BME MIT 2007