Gyors prototípus fejlesztés Mitmót platform és Matlab-Simulink segítségével

TDK dolgozat

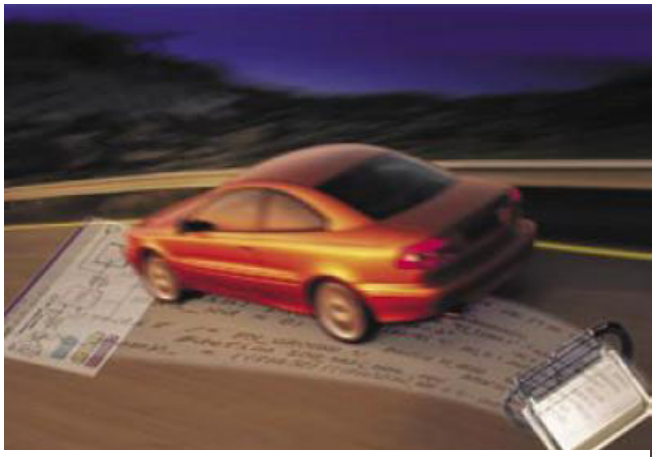
Huszár Viktor
IV. éves villamosmérnök hallgató

Konzulens: Márkus János

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék
Budapest, 2005

Tartalomjegyzék

1. Előszó



[dSPACE, 2005]

„A felfüggesztések aktivizálódnak, a fékek okosodnak – az elektronika számos területen növeli a teljesítményt, ez alól a járművek sem kivételek” – írja egy automatikus kódgenerálást végző szoftvercsomag dokumentációjának első sora [dSPACE, 2005].

Fejlesztéseink, technikánk napjainkra soha nem látott mértékű összetettséget ért el. Óriási kihívást jelent megfelelni a növekvő komplexitásnak, csökkenteni a fejlesztési időt, de nem engedni semmit a megszokott minőségből.

Nem elég, ha tökéletes munkát végzünk, amennyiben adott idő alatt mások kétszerannyi, hasonló bonyolultságú munkát képesek elvégezni. Nem elég, ha minimális idő alatt tudjuk elvégezni a ránk bízott feladatot, ha ugyanakkor a konkurencia sokkal hatékonyabban megoldja a problémát. Nem elég, ha a leggyorsabb és legtökéletesebb munkát végezzük, ha mások adott idő és adott minőség mellett fele áron elő tudnak állni a termékükkel.

Az automatikus kódgenerálás már jó néhány éve a fejlesztések számos területén igyekszik megfelelni e kívánalmaknak. Automatikus kódgenerálást végző eljárások és szoftverek százai lelhetők fel az élet legváltozatosabb területein.

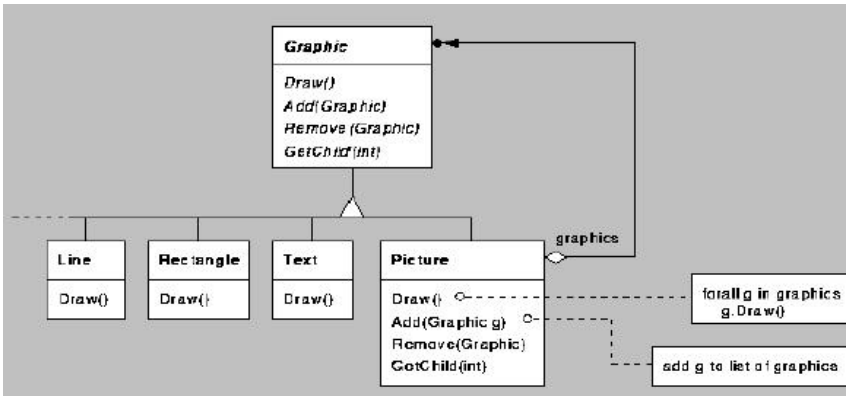
1.1. Az „okos gépelés”

A legtöbb programozó már rutinszerűen használja az utóbbi években szinte minden szoftverfejlesztő-csomag által támogatott „okos gépelés” legkülönfélébb technikáit. Nem kell mást tenni, csak megnyomni egy gombot, és az adott objektumhoz létrejön egy, az objektum összes ismert tulajdonságát tartalmazó forrásfájl. Elég egy kattintás egy nem létező, de éppen hivatkozott függvényre, és generálódik hozzá egy megfelelő keret. Elég egy gombnyomás és

egy teljes lista ugrik elő, hogy a néhány addig begépelt karakterhez megfelelő folytatást biztosítson.

1.2. Tervezői Sémák

Magasabb szintű kódgenerálást biztosít az úgynevezett Tervezői Sémák (Design Patterns) használata. Az objektum-orientált nyelvek, áttekinthetőségi és újrahasználhatósági szempontokat figyelembe véve, a terv egészére nézve összefüggő, ám igen sok különböző forrásból álló kódot követelnek. Amennyiben a programozó tisztában van azzal, hogy a terv „fa-struktúrája" hogyan fog kinézni, alkalmas program felhasználásával csak meg kell „rajzolnia" a tervet a szükséges objektumokkal. Alkalmas kódgenerálás után, a programozó tetszése szerint alakíthatja a létrejött vázban az egyes objektumok, tagfüggvények belső leírását, szerkezetét.



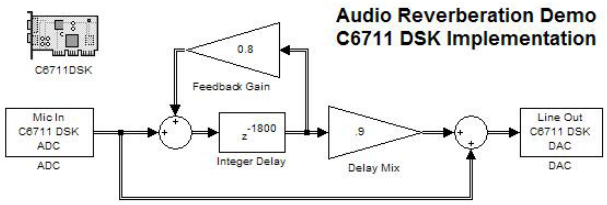
1.1. ábra Objektum-orientált program blokkdiagramja

1.3. A „grafikus programozás"

[dSPACE, 2005]

Egyre több olyan eszközünk van, amelyek vezérlőegységét nem kifejezetten az adott hardverhez fejlesztették ki. A mikrovezérlők ill. jelfeldolgozó-processzorok az élet számos területén igen jelentős teret foglalnak el. Elég csak körbetekinteni otthonunkban, láthatjuk, hogy a mosogatógép és a mosógép saját magának szabályozza a megfelelő vízhőmérsékletet, miközben figyelembe

veszi a tartály telítettségét, a ruhamennyiséget. A legfejlettebb autókban közel 200 egymástól független mikrovezérlős rendszer több ezer különböző mérési és vezérlési feladatot láthat el. Ilyen mértékű, és mennyiségű program megvalósítása szinte reménytelen feladatnak tűnik, figyelembe véve, hogy azokat későbbi fejlesztésekben is fel kell használni, ezért mindenki számára áttekinthetőnek, értelmezhetőnek kell lennie. Emberi mivoltunkból következik, hogy sokkal egyszerűbben megértünk és felidézünk olyan dolgokat, amelyeket nem csupán szöveg, de képek, magyarázó ábrák segítségével próbálunk értelmezni. Miért ne lehetne a nagy komplexitású rendszereink bonyolult programjainak nagy részét is „ábrába" kódolni?



1.2. ábra Simulink modell visszhang generálásához [TI C6711, 2005]

A „grafikus programozás" egy evolúciós lépés a programozás terén. A fejlesztési, pályázati, céges tervek vezetőinek meg kell fontolnia a kérdést, hogy érdemes-e minden betűnyi kódot emberekkel gépeltetni. A grafikus programozás lehetővé teszi a programozónak, hogy alkalmas programok, blokkcsomagok, terv- ill. hardver-specifikus blokkok segítségével, mintegy „megrajzolja" a leendő kód azon részeit, amelyek a rendelkezésre álló blokkok segítségével egyszerűen, a leendő kód biztonságára és hatékonyságára nézve minden követelménynek eleget téve megvalósíthatók.

Fontos kérdés ugyanakkor egy elkészítendő projekt terveinek tesztelése. A tesztelés sok olyan hibára deríthet fényt, amelyek ha csak a kész hardver legyártása után jelentkeznének, óriási pénzeket ölelne fel a projekt átvizsgálása, módosítása.

A terv, a megrajzolást követően, a készenléti állapot különböző szintjein tesztelhető.

A megrajzolt terv már mindjárt a megrajzolást követően tesztelhető egy megfelelő program segítségével (ún. futtatható specifikáció). A grafikus programozást támogató tervező eszközök általában képesek az általuk készített tervek ún. „modell-a-hurokban" (model-in-the-loop) tesztelésére is. Ebben az esetben a kész tervet ültetjük be egy megfelelően előkészített stimuláló környezetbe (általában szintén rajzolt blokk-diagramm), és kimenetként funkcionáló blokkok segítségével értelmezzük a működését.

Amennyiben a blokk-diagramm a kívánalmaknak megfelelően működik, legeneráltathatjuk a fejlesztőkörnyezetünkkel a forráskódot, ami már futtatható a céleszközön. Mielőtt letöltenénk a tervet az eszközre még lehetőségünk van az ún. „szoftver-a-hurokban"

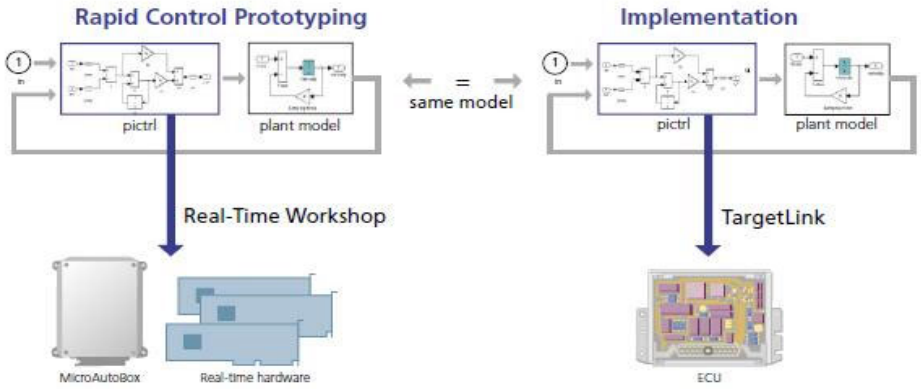
(software-in-the-loop) tesztelésre, ahol a fentebb említett tesztelő környezetbe nem a modellt, hanem a kész kódot ültetjük.

Ha megfelelőnek találtuk a kódunk viselkedését, lehetőségünk nyílik tesztelni a célhardver működését is a beletöltött kóddal, ez az ún. „hardver-a-hurokban" (hardware-in-the-loop) tesztelés, amihez szintén használható a megfelelően előkészített és eddigiekben is felhasznált stimuláló környezet. [dSPACE, 2005]

1.4. A megvalósítás

Tekintettel a fentebb említett témák méretére és komplexitására, e dolgozat csak a grafikus programozás automatikus kódgenerálással kapcsolatos vonatkozásaival foglalkozik, és nem tér ki a tesztelési eljárások kifejlesztésére, de nyitva hagyja a kérdést későbbi fejlesztések céljából.

A dolgozat fő célkitűzése egy mikrokontrolleres alkalmazás „grafikus programozásának" kialakítása. A választott platform egy 8-bites mikrokontroller és a köré épülő perifériák által megvalósított beágyazott rendszer (Mitmót, lásd 3. fejezet), a modellező nyelv a MathWorks által kifejlesztett Simulink programcsomag (lásd 2. fejezet), a kódgenerálást pedig a Rel-Time Workshop végzi (lásd 2. fejezet). A megvalósítás részletei a 4. fejezetben találhatók. Az 5. fejezet foglalja össze a dolgozatban elért eredményeket, valamint tárgyalja a továbbfejlesztés lehetőségeit.



1.3. ábra Kódgenerálás RTW ill. TargetLink segítségével [dSPACE, 2005]

Automatikus kódgenerálás vonatkozásában az autóiparban különösen elterjedt a dSPACE által fejlesztett TargetLink nevű program, amely szintén a Simulink-re épül. Tekintettel arra, hogy ez a termék és dokumentációi egyetemi szinten nem hozzáférhetők, továbbá a MathWorks által biztosított eszközök is alkalmasak gyors prototípus-fejlesztésre, a MathWorks termékeit használtuk a feladat megoldására.

A dolgozat kialakítása közben igyekeztem a felmerülő angol kifejezések magyar megfelelőit használni. Ahol nem állt rendelkezésemre elfogadott magyarítás, ott saját kifejezésekkel éltem, de az első előfordulásukkor feltüntettem az angol kifejezést. Előfordulnak olyan szerkezetek is, ahol inkább maradtam az angol kifejezésnél (pl. system target file)

2. A Matlab programcsalád

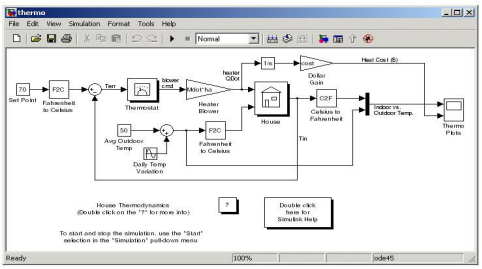
2.1. A Matlab

A MathWorks cég Matlab® nevű programja [MathWorks] gyakorlatilag egy óriási programcsalád, amelyben a mérnöki és tudományos számítások jelentős részéhez található valamilyen szintű támogatás. Felhasználható a szabályozástechnika szabályozóinak tervezéséhez, de segítséget nyújthat például pénzügyi modellezéshez, ezek számításainak elvégzéséhez és kiértékeléséhez. Napjainkra olyan mértékű fejlettséget ért el a Matlab, hogy a hozzá kapható felhasználói csomagokat meg sem kísérlem bemutatni, lehetőségeinek azon szűk részére szorítkozom, amelyeket a dolgozatom fejlesztése során felhasználtam.

2.2. A Simulink

A Simulink® egy szintén a MathWorks által fejlesztett programcsomag, amely a Matlab-ra épül, nélküle nem futtatható. A Simulink nagyon jól használható dinamikus rendszerek modellezésre, szimulálására és analizálására. Támogatja a lineáris és nemlineáris rendszerek tervezését, valamint a folytonos, diszkrét és kevert idejű modellezést. [Simulink, 2005]

A modellezéshez a Simulink egy grafikus felhasználói környezetet biztosít, ahol, akár egy asztallapon az építőkockákkal magunk állíthatjuk össze a modellünket a fellelhető blokkok segítségével. Nem kell mást tennünk, csak lepakolni a megfelelő építőkockákat és „megrajzolni” a közöttük lévő összeköttetéseket. A program biztosít egy szinte minden részletre kiterjedő blokk-könyvtár rendszert is. A könyvtárakban fellelhetők a legkülönfélébb források, nyelők, lineáris és nemlineáris komponensek és kapcsolók. Mindazonáltal a felhasználó lehetőséget kap saját blokkcsomagjának fejlesztésére és használatára is.

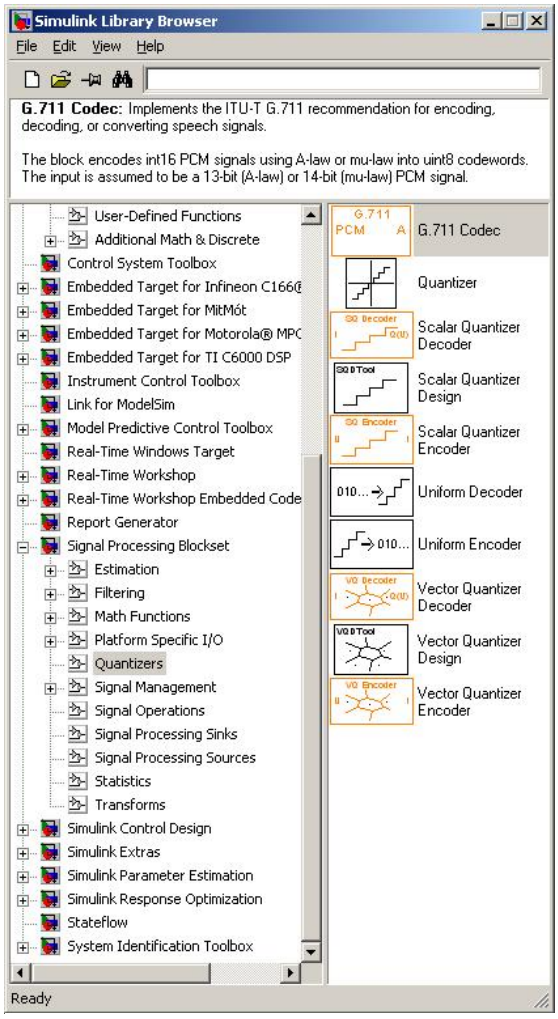


2.1. ábra Hierarchikus Simulink modell

[Simulink, 2005]

A Simulinkben készített modellek hierarchikusak, így a felhasználó megközelítheti a probléma megvalósítását mind felülről-lefelé, mind alulról-felfelé. A tervet bárki megtekintheti és értelmezheti egy felső szintjén, de probléma nélkül bele lehet tekinteni bármelyik nem elemi blokk belsejébe és megfigyelni annak működését. A modellben megfigyelhető, hogy az egyes szintjei hogyan épülnek egymásra, és adott szinten milyen kapcsolatok vannak kialakítva a blokkok között. (lásd M.1. melléklet)

A modell kialakítása után a fejlesztő lehetőséget kap a modell tesztelésére. Megfelelő ki- és bemeneti blokkokat elhelyezve a modellben, elég egy gombnyomás és a Simulink kiértékeli a modell kimeneteinek értékeit, az adott gerjesztések mellett. A tesztelés végezhető Matlab környezetből, parancssorból vezérelve, vagy Simulink alól menük segítségével. Grafikus mérő-blokkok használatával (pl. oszcilloszkóp, multiméter) az eredmény azonnal megjeleníthető. A szimulációs eredmények exportálhatók a Matlab munkaterületre, ahol kiértékelhetők, feldolgozhatók és megjeleníthetők.



2.2. ábra A Simulink könyvtár

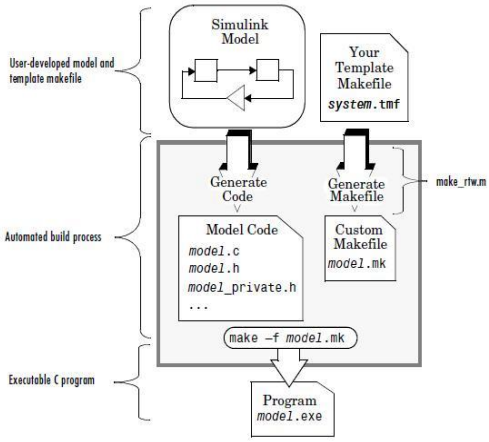
A tervező természetesen kialakíthatja a saját blokkcsomagját is. Egy-egy adott blokk megvalósítására különböző megoldások lehetségesek. Amennyiben egy Matlab-fájlt szeretnénk Simulink blokként működtetni, lehetőségünk van azt egy ún. M-MEX fájlba fordítani. Megfelelő blokk kiválasztásával és a mex fájlal való kapcsolat kialakításával, a blokk a kívánalmaknak megfelelően, Simulinken belül fogja végrehajtani a Matlab szubrutint. Hasonló eljárással felhasználható akár C vagy C++ nyelven írt program is (C-MEX fájl készíthető belőlük).

A saját Simulink blokkok kialakítására mégis a leginkább testhezálló megoldás az ún. S-függvények kialakítása. Az S-függvények létrehozása gyakorlatilag egy M-, vagy C program írásának felel meg, ahol a felhasználó speciálisan Simulinkhez kialakított függvények hívásával megadja a készítendő blokk ki-, bemenetei és belső állapotai közötti kapcsolatot.

2.3. A Real-Time Workshop

A Real-Time Workshop® további lehetőségekkel bővíti a Matlab és Simulink programok felhasználhatóságát. Simulink blokk-diagrammokból a Real-Time Workshop (RTW) segítségével alkalmazásokat készíthetünk prototípus fejlesztéshez, teszteléshez, valós-idejű rendszerek telepítéséhez a legkülönfélébb platformokra. A felhasználónak lehetősége van forráskódot generálni, amely illeszkedik a fordítóhoz, ki- és bemeneti egységekhez, memória modulokhoz, különféle kommunikációkhoz. [RTW, 2005]

A Real-Time Workshop az automatikus programkészítő mechanizmusa segítségével képes futtatható állományokat készíteni valós-idejű alkalmazásokhoz, a legkülönfélébb támogatott platformra. Az 2.3. ábra szemlélteti a programkészítés menetét, ahol a középső vastag doboz munkáját végzi el a Real-Time Workshop.



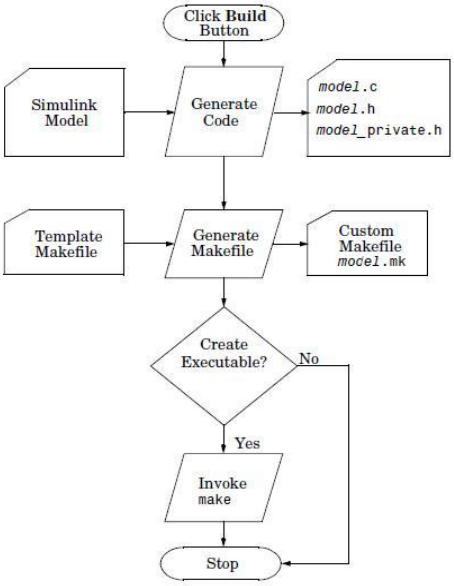
2.3. ábra Az RTW programkészítő mechanizmusa [RTW, 2005]

A Real-Time Workshop programkészítő mechanizmusa négy fő lépésből áll.

A programkészítés első lépése a Simulink modell analizálása. A Simulink modellek egy *model*.mdl kiterjesztésű szöveges fájlban találhatók, ezek azonban igen sok információt tartalmaznak a modellel kapcsolatban, amelyek nem szükségesek a készítendő program működése szempontjából (pl. blokkok helyzete a modell térben, programkészítési opciók stb.). Az analizálást követően a Real-Time Workshop létrehoz egy *model*.rtw nevű közvetítő fájlt, amely az ilyen információkat nem, de a kódgeneráláshoz szükséges összes információt tartalmazza. Ez a fájl általában jóval többet mond el a modellről, mint a modell fájl, hiszen ide már importálódtak az egyes blokkok olyan információi is, amelyek az adott modellel kapcsolatosak (pl. szimulációs és blokk paraméterek, mintavételi idők, blokkok kiértékelési sorrendje, stb.). A fájl gyakorlatilag egy óriási méretű objektumot foglal magába, ez, a `CompiledModel` nevű objektum tartalmazza a modellünk minden lényeges információját, mezőit, blokkjait és paramétereit.

A második lépés a kódgenerálás. A kódot a Real-Time Workshop egy beépített rendszere, a Target Language Compiler készíti. A Target Language Compiler a *model*.rtw-ben található

közvetítő modell leírást alakítja át céleszköz-specifikus kóddá. A Target Language Compiler áttanulmányozása és magas fokú felhasználása volt jelen dolgozat fő célkitűzése.



2.4. ábra A programkészítés lépései [RTW, 2005]

A programkészítés harmadik lépése a testreszabott készítőfájl (makefile) létrehozása. Amennyiben a felhasználónak valamelyik készen kapható céleszköz-specifikus TLC programot használja, akkor elegendő, ha a generált kód-fájlokat egy szerkesztett séma-készítőfájl (system template makefile – `target.tmf`) segítségével fűzi össze megfelelő projekté és fordítja azt futtatható programmá. A séma készítőfájl segítségével megadható a megfelelő fordító, beállíthatók a megfelelő fordítási opciók. A futás eredménye egy készítőfájl, ami gyakorlatilag azonos a sémával, csak a sémában szereplő változók cserélődnek le valós értékekre, elérési utakra, fájlokra.

A Real-Time Workshop programkészítő mechanizmusának negyedik és egyben utolsó állomása opcionális, amennyiben a felhasználó csak forráskódot szeretne, akkor ez nem fut le. Ebben a fázisban készül el a futtatható állomány, amely akár (ha ez része a készítőfájlnak) automatikusan le is töltődhet a céleszközre.

2.4. A Real-Time Workshop Embedded Coder

A Real-Time Workshop kiegészíthető az ún. Real-Time Workshop Embedded Coder nevű termékkel. Az Embedded Coder a megfelelő választás akkor, ha kritikus a célhardveren futó alkalmazás sebessége, memória igénye, ill. egyszerűsége, ilymódon a termék ideális eszköz beágyazott alkalmazások kialakításához. [RTW Embedded Coder, 2005]

Az Embedded Coder a kód létrehozásakor, hardverigény kritikus alkalmazások szempontjából különösen fontos megoldásokat tart szem előtt:

- A valós-idejű modell adat struktúrák használata optimalizálja a modell specifikus memória felhasználást.

- Az egyszerűsített függvény határfelületek lehetőséget biztosítanak rá, hogy a felhasználó probléma nélkül saját maga, kézzel alakítsa az automatikusan generált kódot.
- A statikus memóriafoglalás csökkenti az átlapolódást és támogatja a minél magasabb teljesítményt.

Az Embedded Coder lehetőséget biztosít különböző beágyazott modellek választására (pl. operációs rendszerrel vagy anélkül futó alkalmazások, egy- vagy többszálú program-futtatás, megszakítás vezértelt alkalmazások kialakítása).

2.5. A Target Language Compiler

A Target Language Compiler (TLC) a Real-Time Workshop szerves része. A Target Language Compiler segítségével a fejlesztő teljes mértékben céleszköz-specifikus kódot tud kialakítani. [TLC, 2005]

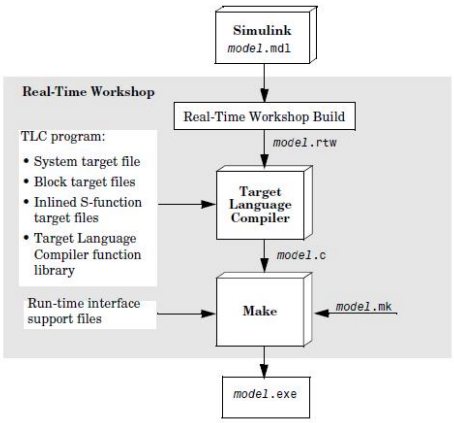
A TLC egy modell fordításkor előre meghatározott szöveges fájlcsomagot (tlc fájlok) használ fel a kód létrehozásához. Ezt a fájlcsomagot nevezzük TLC programnak.

A Target Language Compiler gyakorlatilag egy sor-interpreter nyelv, mely folyamatosan végigszalad a kijelölt tlc fájlokon, felhasználva a deklarált függvényeket, könyvtárakat, a RTW hatékonyságnövelő mechanizmusait és ezek segítségével alakítja ki a megfelelő kódot.

Az ábrán megfigyelhetjük, hogy a TLC hogyan kapcsolódik a Real-Time Workshop programkészítő mechanizmusához.

A Target Language Compiler feladata, hogy a készített közvetítő fájlból (*model*.rtw) kinyerve minden szükséges információt, létrehozza a kódot.

A TLC ún. célrendszer fájlokat (system target file) használ a készítendő kód egészére vonatkozó, hardver specifikus kód kialakításához, és célblokk fájlokat (block target files) a blokk-specifikus kód kialakításához.



2.5. ábra A TLC működése [TLC, 2005]

A Target Language Compiler nyelvezete, tekintve, hogy *kódot generáló kódot* kell leírni, némileg különbözik például egy C nyelvtől. Minden, a TLC által kiértékelendő sor `%` jellel kell kezdődjön. Amennyiben egy sorban (például `%if` után) több kiértékelendő utasítás is áll, elég az első elé tenni a % jelet. Komment sor `%%` jelekkel adható meg.

Olyan sorokat ahol nem szerepel % jel, a TLC automatikusan továbbít a beállított kimeneti egység felé (tipikusan a készítendő forrásfájl).

A kódgeneráláshoz szükséges TLC változók (például egy kapcsoló kimenetét jelölő változó) nem kötelezően értékeket, hanem értékeket hordozó változóneveket tartalmaznak. Amennyiben a kód kialakításához van szükségünk egy TLC változóra (például a kapcsoló kimeneti változójának a megjelenítéséhez a kódban), akkor a `%<TLC_változó_neve>` hivatkozást kell használnunk.

Mivel a TLC nyelvet direkt a *`modell`*`.rtw` fájl adatainak kinyerésére hozták létre, ezért az ismert nyelvi eszközökön kívül (`%if, %for` stb.) nagy hangsúlyt kap a %with és a %roll formulák. Egy %with blokkon belül a `CompiledModel` objektumhoz vagy annak egy részéhez tudunk úgy hozzáférni, hogy nem kell minden alkalommal végigírni a hivatkozát (pl. `CompiledModel.Name`), helyette elég csak a hivatkozni kívánt változó (pl. `Name`).

A %roll segítségével a modell.rtw-ben előforduló olyan objektumokon tudunk végigmenni, amelyek egy blokkon belül valamiben nagyon hasonló tulajdonságokkal rendelkeznek (pl. egy több kimenetű és több bemenetű blokk kimeneteihez és bemeneteihez való hozzáférés).

A könnyebb megértés kedvéért álljon itt egy kiragadott tlc forráskód-részlet, ahol bemutatok néhány nyelvi elemet és egyszerű függvényt. A kód a későbbiekben részletezett block target fájlokhoz hasonló. A segítségével a hozzá tartozó blokk bementére érkező jel függvényében beállítjuk a blokk kimenetének értékét.

```
%% először kialakítunk két változót, amelyek segítségével
%% hozzáférhetünk  a ki- és bemeneti értékeket
%% tartalmazó tlc struktúrákhoz

%assign u0 = LibBlockInputSignal(0, "", "", 0)
%assign y0 = LibBlockOutputSignal(0, "", "", 0)
```

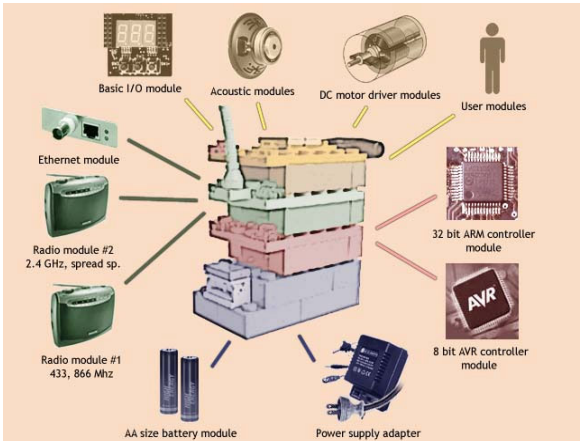
```
  %% I. A tlc kód:

    if (%<u0> >= 1)
       %<y0> = 1;
    else
       %<y0> = 0;


%% II. Amennyiben létezik
%% egy pl. valt nevű tlc
%% változó, akkor
%% a tlc kód lehet ilyen is:
   %if (valt >= 1)
       %<y0> = 1;
   %else
       %<y0> = 0;
   %end if
```

```
// I. És az általa generált
// C forráskód
if (input >= 1)
     output = 1;
else
     output = 0;



// II. És ekkor az általa
// generált C forráskód

// VAGY EZ:
   output = 1;

// VAGY EZ:
    output = 0;
```

A ténylegesen megvalósított, futtatható kódot generáló tlc fájlok a mellékletek között (lásd M.6) hozzáférhetők.

Az előbbiekben csak egy kis ízelítőt adtam a TLC nyelvi lehetőségeiből, azonban ez megfelelő információt biztosít a következőkben részletezett Mitmót TLC keretrendszer és Mitmót blokkcsomag megértéséhez.

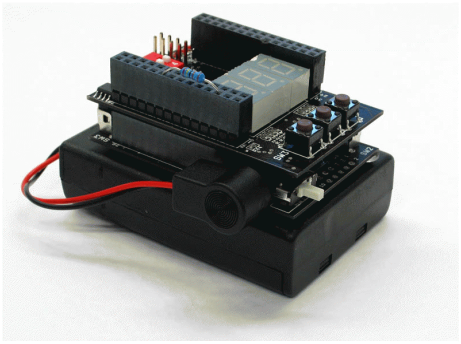
3. A Mitmót



3.1. ábra A Mitmót rendszer [Mitmót]

A Beágyazott Információs Rendszerek szakirányt elkezdő hallgatók már évek óta küzdöttek egy nagy problémával, nevezetesen, hogy elő-tapasztalat nélkül, egy félév alatt kellett elsajátítaniuk a NYÁK tervezés, a hardvertervezés és a szoftvertervezés mikrokontrolleres vonatkozásait. A Mitmót platform tervezői e problémára keresték a megoldást.

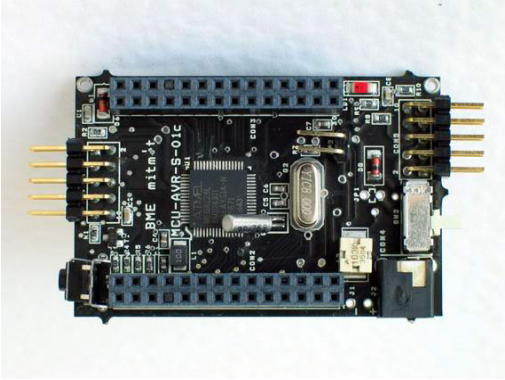
A Mitmót egy mikrokontrolleres „beágyazott rendszer”, mely egyszerű vezérlési feladatok megvalósítására éppúgy alkalmas, mint nagyobb bonyolultságú mérési eljárások kivitelezésére. A platform létrejöttével a szakirányt elkezdő hallgatók testközelből ismerkedhetnek meg egy megvalósított rendszer programozásával, használatával, életszerű problémákat oldhatnak meg vele, esetleg később (pl. önálló laboratórium keretében) maguk is hozzájárulhatnak a platform fejlesztéséhez.



3.2. ábra A Mitmót platform [Mitmót vezérlő, 2005]

Az általunk kézhez kapott alapmodul tartalmaz egy vezérlőkártyát és egy egyszerű ki- és bemeneti egységeket hordozó perifériakártyát.

3.1. A vezérlőkártya



3.3. ábra A Mitmót vezérlőkártyája [Mitmót vezérlő, 2005]

A vezérlőkártya gyakorlatilag egy szabványosított interfészt biztosít egy adott mikrokontroller és a hozzákapcsolt perifériák között, biztosítja a vezérlőegység táp- és órajel ellátását, továbbá interfészt szolgáltat az ISP és JTAG programozáshoz, hibakereséshez.

Az általam használt kártyára ültetett vezérlőegység egy 8 bites Atmel ATmega128L mikrokontroller. Az egység RISC felépítésű, rendelkezik128 kbyte program flash-, 4 kbyte adat SRAM és 4 kbyte EEPROM memóriával, JTAG támogatással, napjainkban használt legfontosabb mikrokontrolleres kommunikációk támogatásával (UART, SPI, I^2C), 2 darab 8-bites és 2 darab 16-bites időzítő/számláló egységgel, egy 10-bites szukcesszív-approximációs A/D mintavevő tartóval, valamint támogat hatféle kisfogyasztású üzemmódot. A kártyán található továbbá egy 8MHz-es kvarckristály, amely stabil órajelet biztosít a mikrokontroller számára. [Mitmót vezérlő, 2005]

A kártya tápellátása biztosítható egy, a processzorpanelen lévő tápfeszültség csatlakozóról, vagy a panel aljára szerelt akkumulátor-tartóba helyezett elemekről.

A kártya programozása történhet az ISP és a JTAG csatlakozókon keresztül. Az ISP csatlakozó az AVR processzorok programozóinál használt szabványos 10-pólusú csatlakozó, ezen keresztül történhet az In-System programozás (természetesen erre a JTAG is képes, megfelelő külső hardver támogatással). A JTAG csatlakozó segítségével lehet a processzort In-Circuit debuggolni, programozni, ill. a processzor és környezete Boundary scan alapú tesztelésére is alkalmas.

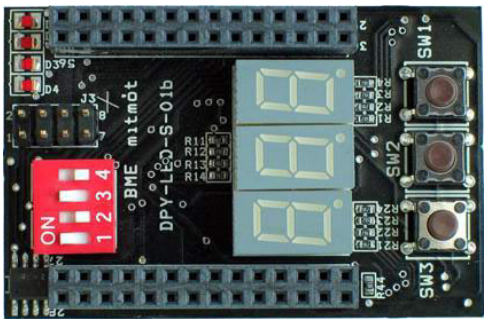
A kártyán található egy RESET nyomógomb, amely a buszra is ki van vezetve, ezzel biztosítható a teljes rendszert érintő RESET jel.

A Mitmót egyes egységei, kártyái közötti kapcsolatot egy egységes busz teremti meg. A buszon keresztül elérhető a mikrokontroller lábainak nagy része. A buszra ki van vezetve a rendszer digitális földje, a digitális stabil 3,3V-os tápfeszültség, a reset jel. Hozzáférhetők a mikrokontroller által biztosított ki- és bemeneti lábak (GPIO0-GPIO15), a támogatott kommunikációk megfelelő jelei (UART → Tx, Rx; I^2C → SDA, SCL; SPI → SSEL, SCK,

MOSI, MISO), az időzítőhöz kapcsolódó külső trigger jelek (eltárolható az időzítő aktuális értéke), a mikrokontroller külső megszakításkérő lábai. Kivezetésre került továbbá az egyik 16-bites időzítő számláló pulzus szélesség modulátorának (PWM) 2 kimenete (OC1A, OC1B) és a 10-bites AD átalakító néhány csatornája (ADC0-3)

A hozzáférhető mikrokontroller jelek és azok buszkiosztása megtalálható az M.2 mellékletben.

3.2. A perifériakártya



3.4. ábra A Mitmót perifériakártyája [Mitmót periféria, 2005]

Az alapmodul által magában foglalt perifériakártya segítségével a kezdő „beágyazott rendszer” fejlesztő lehetőséget kap rá, hogy egyszerű, de látványos példákon keresztül szerezzen tapasztalatokat és csiszolja tudását a mikrokontrollerek programozása terén. [Mitmót periféria, 2005]

A kártya tartalmaz négy darab két-állapotú kapcsolót, négy darab kisfogyasztású felületszerelt LED izzót, három darab 7+1 szegmenses kijelzőt, szintén három darab nem pergés-mentesített nyomógombot, és egy felületszerelt LM75 típusú hőmérő IC-t.

A alap-perifériák (gombok, kapcsolók, LED-ek) segítségével a felhasználó tapasztalatokat szerezhet a legegyszerűbb ki- és bemeneti egységek vezérlése terén.

A kapcsolók a Mitmót busz GPIO8-GPIO11 jelein keresztül csatlakoznak a mikrokontroller PC0-PC3 ki/bemeneti lábaihoz. A kapcsolók OFF állásban logikai alacsony, míg ON állásban logikai magas szintet kényszerítenek a portra.

A LED-ek szintén egyszerű kimeneti egységként kezelhetők, a Mitmót busz GPIO4-GPIO7 jelein keresztül, közvetlenül a mikrokontroller PA4-PA7 ki/bemeneti jeleivel vezérelhetők.

A nyomógombok, a LED-ekhez és a kapcsolókhoz hasonlóan, könnyen használhatók bemeneti jelként kezelve (a Mitmót busz GPIO1-GPIO3 jeleivel, a kontroller PA1-PA3 lábain keresztül).

A 7-szegmenses kijelzők már egy magasabb szintű kommunikáció felhasználásába adnak betekintést a felhasználó számára, ugyanis a kijelzők nem mind külön-külön ki/bemeneti jelekre vannak kapcsolva, hanem három darab 8-bites sorba kötött shift-regiszteren keresztül

vezérelhetők, amelyek az SPI buszra csatlakoznak. Ezzel a kommunikációval a 7-szegmenses kijelzők vezérlése 2 kimeneti jelre korlátozódik (SPI_MOSI, SPI_SCK), ahol a felhasználónak sorban egymásután kell a megfelelő vezérlő adatot kiadni, a kijelzők működtetéséhez.

A hőmérő IC beüzemelése segítséget nyújt a mikrokontroller programozónak, hogy betekintést nyerjen az I^2C protokoll használatába, amely napjaink egyik legelterjedtebb beágyazott rendszeres kommunikációja. A hőmérő két kommunikációs cikluson keresztül tudja a teljes általa mért adatot közölni a mikrokontrollerrel, első körben megkapjuk a hőmérséklet egész részét ($-55^oC \rightarrow +125^oC$), a második ciklusban bedig a törtrészét ($0,5^oC$–os felbontásban).

3.3. Az RS232-es szabvány → USART kommunikáció

Nem kimondottan a Mitmót kártyáihoz kapcsolódik a szabvány, de a TDK témáját közelebbről érinti ez a kommunikáció.

Programfejlesztés közben szükség van a programok tesztelésére, futtatására, hibák megtalálására és kijavítására. A hibakeresés legelterjedtebb módja, hogy üzeneteket generáltatunk a programmal a futtatáskor és ezek segítségével megállapítjuk, hogy megfelelően működik-e a program.

Mikrokontrollerek esetében az üzenetek kiíratása nem is tűnik olyan könnyű feladatnak, hiszen az esetek többségében nincsen megfelelő kimeneti egység csatlakoztatva a vezérlőhöz. Erre a problémára kínál kézzelfogható megoldást az USART kommunikáció, amely (illeszkedve az RS232 szabványhoz) képes a fejlesztő számítógépére (vagy más standard kimeneti perifériára) üzeneteket küldeni.

A jelenlegi platformon nincs külön erre a célra fejlesztett csatlakozó. Szerencsére a Mitmót buszon hozzáférhetőek a kommunikációhoz szükséges mikrokontroller jelek. Megfelelő szintillesztés és csatlakozók elkészítése után a felhasználónak lehetősége nyílik egy terminál program segítségével üzenetek fogadására, és vezérlő parancsok továbbítására a Mitmót felé.

3.4. A Mitmót programozása szükséges programok

A Mitmót mikrokontrollere programozható assembly és C nyelven, megfelelő szoftver támogatással.

Az Atmel cég minden felhasználó számára biztosít egy ingyenes, könnyen hozzáférhető AVR Studio nevű assembly-fordítót. A program továbbá segítséget nyújt JTAG interfészen keresztüli programozásra és hibakeresésre is. [Atmel]

Az AVR Studio által lefordított gépi kód egy .hex file-ba kerül, amely ISP programozó és megfelelő szoftver segítségével (pl. PonyProg 2000) felhasználható, amennyiben a felhasználónak nem áll rendelkezésére JTAG modul. [PonyProg]

Az ATmega128L mikrovezérlő programozható C nyelven is. Ehhez szükséges egy megfelelő C fordító. A WinAVR nevű fordító csomag [WinAVR] megfelelő megoldást kínál a legtöbb fejlesztési feladathoz, óriási eszköztárával, de ezt kiegészíthetjük még számos fellelhető csomaggal, amelyekkel igen szerteágazó feladatok is megoldhatók. Hozzáférhető továbbá nagy számú ingyenes fejlesztőkörnyezet a WinAVR-hez, ezek segítségével az operációs rendszerünknél megszokott keretek között írhatjuk a programjainkat, és egy gombnyomással fordíthatjuk azokat (pl. KamAVR). [KamAVR]

Ha nem használunk JTAG interfészt, akkor a hibakeresést más módon kell megoldanunk. Erre megfelelő megoldás például a fentebb említett USART kommunikáción keresztüli üzenetek generálása, melyhez szükségünk van egy terminál programra, a szerző ehhez a Bray++ által fejlesztett Terminal v1.9b-t használta. [Bray++]

4. A Mitmót TLC keretrendszer és blokkcsomag

A TDK dolgozatom fő célkitűzése egy Mitmót specifikus Simulink blokkcsomag kialakítása volt. A kialakított blokkok lehetővé teszik Simulink blokk-diagrammok létrehozását és azokból forráskód generálását, amely a modellnek megfelelő alkalmazást képes futtatni a Mitmóton.

A fejezet célja, hogy bemutassa a Mitmót TLC keretrendszerének és blokkcsomagjának kialakítását. A fejezet első részében részletezem egy TLC program kialakításának szükségességét és nehézségeit. A második rész tárgyalja a Mitmót TLC keretrendszer megvalósításának részleteit. A fejezet harmadik része a blokkcsomag kialakításával foglalkozik. A negyedik részben pedig a blokkcsomag felhasználhatóságára mutatok néhány példát.

A Texas C6711 DSP

Mielőtt hozzáfogtam volna egy saját blokkcsomag létrehozásához szükség volt némi tapasztalatra az automatikus kódgenerálás terén. Tapasztalatszerzéshez egy, a tanszéken fellelhető, a Texas Instruments cég által készített jelfeldolgozó processzorpanelt hívtunk segítségül (TI C6711 DSP), melyhez készítettek Simulink alatt használható blokkcsomagot. Mivel e kutatások nem kapcsolódnak szorosan a téma kialakításához, ezért a velük foglalkozó dokumentáció az M.4 mellékletben található.

4.1. A TLC program

Amint azt a Target Language Compiler-el foglalkozó fejezetben említettem, egy új, célhardver-specifikus blokkcsomag kialakításához szükség van system target és block target fájlokra.

Amennyiben a készen kapható system target fájlcsomagok közül valamelyik illeszkedik az adott hardverhez, akkor elegendő a blokk-specifikus fájlok kialakítása. Ellenkező esetben egy jóval nagyobb problémával áll szemben a fejlesztő, amelynek megoldásához közel sem elegendő csupán áttanulmányozni a fellehető Matlab, Simulink és TLC dokumentációkat.

A system target fájlok olyan magasfokú integráltságban állnak egymással (nem csak a saját fejlesztésűekkel, hanem a fellelhető TLC könyvtárakkal és TLC programokkal is), hogy ezek

értelmezése és felgöngyölítése megfelelő dokumentációk és tudományos értekezések híján óriási feladat.

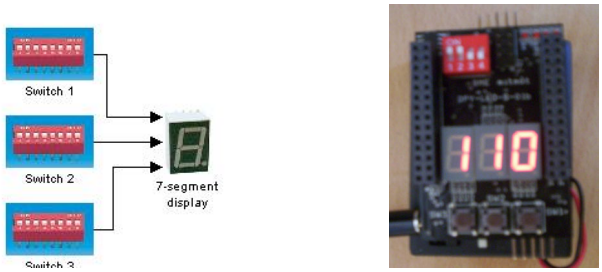
A Target Language Compile felhasználói dokumentációjának [TLC, 2005] segítségével a fejlesztő betekintést nyer a TLC programok kialakításába, a TLC nyelv szintaktikájába, felhasználási lehetőségeibe. Sajnos azonban az ott közölt adatok (terjedelmes mennyiségük ellenére is) csupán alapot adhatnak egy blokkcsomag fejlesztéséhez (amennyiben nincs szükség system target fájlok kialakítására), de (a fejlesztőeszközök jelenlegi fejlettségi fokán) a további munka merőben intuitív jellegű, elengedhetetlen a megfelelő elmélyülés és felkészültség.

Az alábbiakban részletezett TLC program elkészítésekor törekedtem egy áttekinthető, hierarchikus rendszer kialakítására. A könnyebb érthetőség kedvért fastruktúra-ábrán próbálom majd szemléltetni a fájlok kapcsolatait és azok fontosabb függvényeit (lásd M.5), a mellékletben (lásd M.6 és M.7) pedig fellelhető az összes kialakításra került tlc fájl.

4.2. A Mitmót TLC keretrendszer

A feladat megvalósításának jelenlegi fokán a blokkcsomag felhasználója lehetőséget kap rá, hogy mindenféle programozási ismeretek nélkül, egy Matlab/Simulink és egy AVR C fordító segítségével, a számára megfelelő (készen kapható Simulink blokkokat sem nélkülöző) modellt hozzon létre, amelyet kódgenerálás után lefordítva a Mitmót megfelelően működtethető.

A kialakított system és block target fájlok a felhasználó számára láthatatlanok, azok működéséről nem kell semmit sem tudnia a blokkcsomag helyes használatához. Elegendő az adott modellt összeállítani, kialakítani a felhasználó számára megfelelő kapcsolatokat, és a hardver a kívánalmaknak megfelelően fog viselkedni (lásd 4.1. ábra).



4.1. ábra Kapcsolókból és kijelzőkből kialakított modell.
A kijelzők rendre az adott kapcsolók állapotát jelzik

A system target fájlok hivatottak kialakítani a céleszköz-specifikus kódgenerálás átfogó struktúráját. A TLC program a system target fájloktól tudja, hogy hol kell kezdeni a fordítást, hol vannak megszakítások, honnan szereztetők be az egyes blokkokhoz szükséges blokk-specifikus fájlok stb.

Megfelelő hierarchia kialakításával átláthatóvá tehető a fejlesztett TLC program. A MathWorks is egy minden részletre kiterjedő hierarchiát próbált kialakítani a TLC könyvtáraknál (itt találhatók kész függvények, amelyek a fordítás különböző fázisaiban felhasználhatók), de dokumentáció híján ennek felgöngyölítése igen bonyolult feladat.

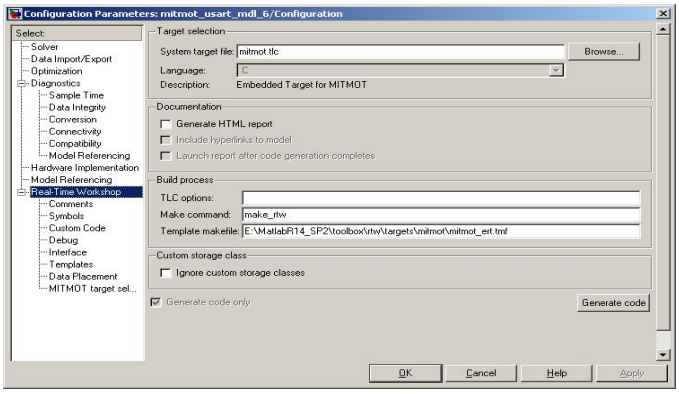
A továbbiak megértéséhez fontos tudnunk, hogy a tlc fájlokban szereplő `include` kulcsszót nem egészen úgy kell elképzelni, mint például egy C programban. Egy C programhoz hozzákapcsolt külső fájl általában vagy további csatolásokat végez, definíciókat állapít meg, vagy függvényeket deklarál.

Egy tlc fájl csatolások és függvények kialakításán kívül, külön futtatható részleteket is tartalmazhat. Ezek a részletek az adott tlc fájl csatolásakor lefutnak, mintha egy függvényhívást végeztünk volna a tlc fájl csatolásával.

4.2.1. A mitmot.tlc

Amennyiben a fejlesztő elhelyez egy számára alkalmas könyvtárban egy *target*`.tlc` fájlt (a dőlt betűs szavak helyére a céleszköz ill. esetenként a modell neve helyettesítendő), és azt a Matlab számára elérhetővé teszi, akkor már meg is tette az első lépést egy saját hardver-specifikus automatikus kódgeneráló fejlesztése felé. Ettől természetesen még semmi nem fog történni, még nagyon sok a tenni- és megfontolni való.

A mitmot.tlc nem csupán a kódgenerálás első lépése, a Simulink egy *target*`.tlc` fájl segítségével tudja megjeleníteni a céleszköz Konfigurációs Paraméter (Configuration Parameter) opcióinak beállítására szolgáló ablakot (lásd 4.2. ábra).



4.2. ábra A Mitmót konfigurációs beállításai

A system target fájl elején beállítandó egy `TargetName` nevű globális változó, ezek után a fordító tudni fogja, hogy mi a célhardver neve. A fordítás folyamán a hardvernév többször felhasználásra kerül (pl. könyvtár-hierarchia kialakítása, forrásfájlok létrehozása).

Mivel egy mikrokontrolleres alkalmazásról van szó, szükséges, hogy a lehető legáttekinthetőbb és legkisebb memória igényű programot tudjuk készíteni. Ezen a kívánalmaknak a teljesítéséhez leginkább egy, a Real-Time Workshop Embedded Coder–el készített kód felelne meg. Közölni kell a fordítóval, hogy „Embedded-C" stílusú forrást generáljon, ez a `CodeFormat` nevű globális változóval tehető meg.

Egy `ERTCustomFileTemplate` nevű változó segítségével megadhatjuk, hogy az Embedded Real Time Target kódmodell szerint fordítandó Simulink modellünkhöz milyen séma szerint generálódjon a forráskód. A séma ebben az esetben egy tlc file, a `mitmot_ectemplate.tlc` (lsd. később)

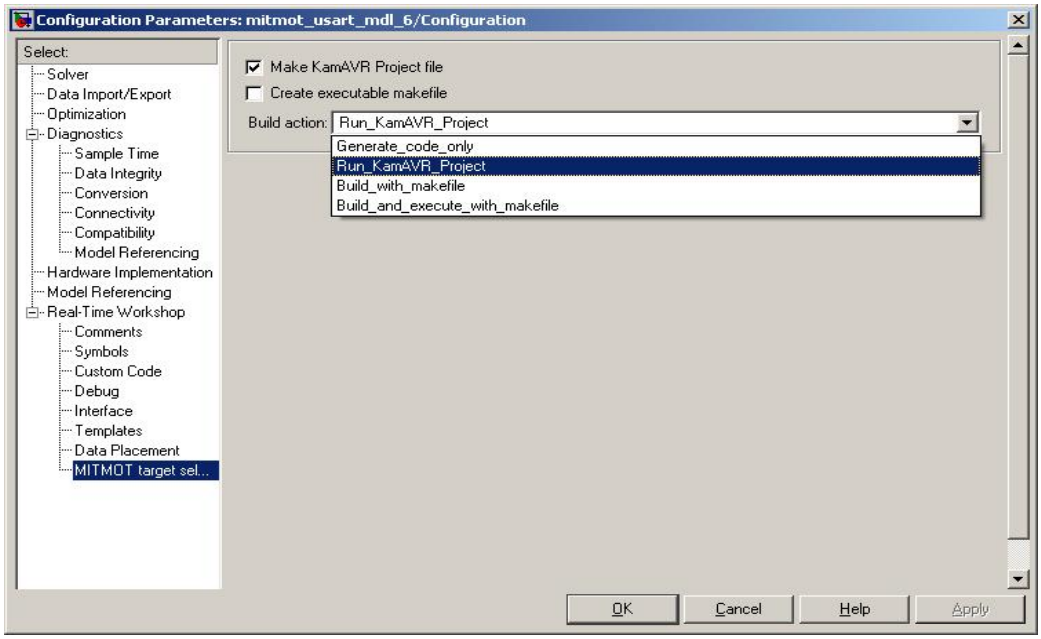
A `mitmot.tlc`-ben elhelyezett `RTW_OPTIONS` blokk gondoskodik a Konfigurációs ablak céleszköz-specifikus részeiről.

Az `RTW_OPTIONS` blokk `rtwgensettings` nevű objektumának változóinak beállításával lehetőségünk nyílik a kódgenerálás és a TLC programunk néhány saját opciójának beállítására.

A kódgenerálás folyamán a Target Language Compiler-nek tudnia kell, hogy az eredeti tlc kódmodellek közül (`grt – generic real time target`, vagy `ert – embedded real time target`) melyiket óhajtjuk használni. A `DerivedFrom` változóval közölhetjük a fordítóval, hogy amennyiben adott függvény nem szerepel a saját tlc fájlaink között, akkor melyik kódmodell függvényét hívja meg.

A céleszköz kiválasztásakor előfordulhat, hogy az adott eszköz korrekt beállításához szükségesek különböző elő-, ill.. utóbeállítások, szükség lehet például néhány olyan opció letiltására, amelyek eredetileg is szerepelnek a Konfigurációs ablak beállításai között, ezeket alkalmasan megírt ún. `callback` függvényekkel tehetjük meg. A TLC pogramunkhoz rendelt callback függvények megadását, az `rtwgensettings` objektum `Activate`-, `Select`-, ill. `UnselectCallback` változóinak beállításával tehetjük meg. Céleszköznek Mitmótot kiválasztva például letiltódik a Csak kódot generál (Generate Code Only) opció, mivel a Mitmót esetében a felhasználó maga választhatja ki, hogy mi történjen a kódgenerálást követően.

Az `rtwoptions` objektum változói beállíthatók egy alkalmasan elkészített Matlab fájl segítségével (`mitmot_getRtwOptions.m`, lsd. később), így egyéni beállításokat adhatunk hozzá az általános Konfigurációs Paraméter ablakhoz. A készített Matlab fájl közli a Simulink-kel a céleszközhöz tartozó paraméter ablakok neveit és a hozzájuk csatolt beállítási lehetőségeket. A Mitmót TLC program jelenleg egy paraméter ablakkal rendelkezik (Mitmot target selection), ahol beállítható, hogy a kódgenerálást követően a felhasználó kíván-e projekt ill. készítőfájlt (makefile) létrehozni (lásd 4.3. ábra).



4.3. ábra A Mitmót target selection panel

A `mitmot.tlc`-hez további tcl-fájlok is hozzá vannak rendelve, amelyek biztosítják a kódgenerálás menetét. A részletezett fájlok közül azok, amelyek nem `mitmot`-al kezdődnek az említett könyvtárban lelhetők fel.

A Target Language Compiler által biztosított függvények és eljárások nagy része fellelhető a `matlabroot/rtw/c/tlc` könyvtárban. Sajnos, megfelelő dokumentáció híján ezek a függvények csak hosszadalmas tanulmányozás után deríthetők fel, előfordulhat, hogy adott problémára a fejlesztő nem is találja meg a már létező megoldást, csupán a dokumentációhiány és a létező függvények óriási mennyisége miatt.

A `mitmot.tlc`-hez hozzárendelt fájlok közül az első a `utillib.tlc`, amely gyakorlatilag egy általánosan felhasználható TLC függvény-könyvtárnak tekinthető. Ebben a könyvtárban található meg a `LibIsContinuous()` függvény, amely ellenőrzi, hogy a blokkok vagy beállítások bármelyike folytonos idejű modellre utal-e, ugyanis folytonos modellből nem generálható beágyazott kód.

A következő csatolt fájl a `codegenentry.tlc`, gyakorlatilag egy gyűjtő fájlnak tekinthető, amelyik csatolja a három, következőkben részletezendő fájlt a fordításhoz. Az első a `genmap.tlc`, amely biztosítja a megfeleltetést a Simulink-ben fellelhető blokkok és a blokkhoz tatozó block target fájl között. A `commonsetup.tlc` meghív számos függvényt, amelyek a fordításhoz elengedhetetlen globális változókat állítják be, továbbá készít a `CompiledModel` objektumhoz (lásd 2.3. fejezet) néhány további szükséges mezőt. Az utolsó - a `codegenentry.tlc` által meghívott - fájl a `commonentry.tlc`, ezzel kezdetét veszi az eddigi beállításoknak megfelelő kódgenerálás.

Amennyiben a fejlesztő elhelyez a `target.tlc`-ben egy `target_genfiles.tlc` nevű csatolást (`mitmot_genfiles.tlc`, lásd később), akkor a kódgenerálást követően lehetőségünk van további munkálatokat végeznünk (pl. projekt fájl létrehozása, kereszt-fordító program indítása)

A mitmot_getRtwOptions.m

Ennek a Matlab fájlnak a segítségével alakíthatjuk ki a céleszközünkhöz tartozó panel kinézetét. Elkészítése nem különösebben bonyolult, egyszerűen fel kell sorolni a leendő al-objektumok nevét, típusát, hivatkozásait (nyomógomb, lista, stb.). Minden egyes al-objektumhoz rendelhetők ún. callback függvények, amelyekkel például letilthatjuk valamelyik opciónkat, amennyiben az nem illeszkedik egy másik kiválasztott opció értelmezéséhez.

4.2.2. A kódgenerálásért felelős system target fájlok

E fejezet eddigi részeiben megismerkedhettünk egy általános TLC program magjával, annak kialakításával és a hozzá tartozó Paraméter Konfigurációs panel létrehozásával.

A továbbiakban áttekintjük a TLC program kódgenerálással foglalkozó részét, megismerkedhetünk a generált fájlokkal, struktúrájukkal, felépítésükkel.

Az egyértelműség kedvéért minden modellnév függő elnevezést dőlt betűvel írtam.

A kódgenerálás folyamán a mindenképpen kialakításra kerülő fájlok a következők:

- *modell*_main.c : Az alkalmazás belépési pontját (main() függvény) tartalmazó fájl.
- *modell*.c : A modellünknek megfelelően kialakított C kód, az egyes blokkok által végzett műveletek forrásai találhatók itt.
- *modell*_data.c : A modellünkben szereplő konstansok, a blokkok modellen belül beállított paramétereinek elérési pontja.
- *modell*.h : Általános fejlécfájl, amely tartalmazza a modellünkhöz szükséges adat struktúrákat, külső függvényeket, fejlécfájl csatolásokat.
- *modell*_private.h : A saját blokkjainkhoz szükséges definíciók és külső függvények találhatók ebben a fejlécfájlban.
- *modell*_types.h : A modellünk által igényelt új típusok deklarálása itt történik.
- rtwtypes.h : Az adott céleszközhöz beállított típusdefiníciókat itt közli a TLC a majdani fordítóval (olyan típusok, amelyek ugyan nem minden fordítóban megtalálható típusok, de azokra egyszerűen visszavezethetők és egyértelműek)

A készített alkalmazás tartalmazhat további fájlokat, a fejlesztő igényei szerint. A felsorolt fájlok közül a *modell*_main.c és a *modell*_private.h gyakorlatilag teljes egészében a fejlesztő kialakítása szerint jön létre, míg a többi a Target Language Compiler által létrehozott adat- és kódstruktúrák eredményeképpen generálódik.

A mellékletben (lásd M.8 és M.9) mindegyik fentebb részletezett fájlra található automatikusan generált példa forráskód.

A mitmot_ectemplate.tlc

A fájl csak formális szerepet tölt be a kódgenerálási folyamatban. Mivel egy Embedded Real-Time Target esetén beállítandó az ERTCustomFileTemplate, ezért készült ez a fájl is. Jelen állapotában a fájl csak meghív egy SLibCreateMITMOTERTMain(fname) nevű tlc függvényt, ami elindítja a készítendő alkalmazás main() függvényét tartalmazó fájl generálását. A meghívott függvény a mitmot_ertmain.tlc fájlban található. Az átadott paraméter a készítendő fájl kiterjesztés nélküli neve (*modell*_main).

A mitmot_ertmain.tlc

Ebben a fájlban található a generálandó *modell*_main.c fájl elkészítéséhez tartozó kód nagy része.

A fájl akkor fut le, amikor az előző fájl (mitmot_ectemplate.tlc) meghívja az SLibCreateMITMOTERTMain(fname) nevű függvényt.

Ez a függvény közli a fordítóval, hogy az fname változó által képviselt fájlt adja hozzá a generálandó C fájlok listjához (SLibAddModelFile(…) fgv.), majd kialakítja a készítendő fájl megjegyzésekből álló fejlécét (SLibSetModelFileAttribute(…"Banner"…)) és hozzáadja a szükséges fejlécfájl csatolásokat (SLibSetModelFileAttribute(…"Includes"…)).

A kódgenerálásnak ezen a pontján még csak a *modell*_main.c fájlunkhoz adhatjuk hozzá a szükséges csatolásokat, a *modell*_private.h kialakítására majd csak a block target fájlok futása alatt van lehetőségünk.

Az SLibCreateMITMOTERTMain (…) ezután befejezi a működését és a Target Language Compiler főkönyvtárában található ert.tlc veszi át a vezérlést. Az ert.tlc a fordítandó alkalmazásnak beállításainak megfelelő függvényt hívja meg a továbblépéshez.

Amennyiben tisztában vagyunk vele, hogy melyik függvény kerül majd meghívásra, akkor akár felül is definiálhatjuk azt, ezzel kényszerítve a Target Language Compiler-t, hogy a saját függvényünket futtassa le. Mivel a Mitmóthoz tartozó blokkcsomag és TLC program jelenleg csak az operációs rendszer nélküli, egyszálas alkalmazások fordítását támogatja, ezért elegendő volt a FcnSingleRateWithoutOS(cFile) és a FcnSimpleNonOSMain() nevű, ert.tlc-ben is szereplő függvényeknek a felüldefiniálása.

A FcnSingleRateWithoutOS(cFile) segítségével a *modell*_main.c további részeit alakíthatjuk ki. Lehetőségünk van statikus ill. globális változók kialakítására, amennyiben a saját kódjaink igénylik ezeket (bár ezek nem igazán szerencsés megoldások).

Ahhoz, hogy az alkalmazás egy „lépésében" (a főciklus egyszeri lefutásakor) lezajló folyamatokat befolyásolni tudjuk, ki kell alakítani egy függvényt, ami összefogja a teendőket. Az rt_OneStep() nevű függvény szintén a FcnSingleRateWithoutOS(cFile)-en belül kerül kialakításra. A függvény szerkezete messze túllép jelenlegi felhasználhatóságán (lásd 4.4. ábra).

Az `rt_OneStep()` minden lefutásakor meghívja a *`modell`*`_output()` és *`modell`*`_update()` automatikusan generált (tehát a *`modell`*`.c` fájlban található) függvényeket, ezzel biztosítva, hogy minden a modellben szereplő blokk be- és kimenetei, valamint belső állapotai megfelelő értékeket vegyenek fel.

A `FcnSimpleNonOSMain()` függvényt a `FcnSingleRateWithoutOS(cFile)` hívja meg, ezzel megkezdve a `main()` függvény kialakítását. A `main()` gyakorlatilag három fontos dolgot tesz: elvégzi az inicializálásokat (*`modell`*`_initialize()`), az alkalmazás főciklusának minden lefutásakor meghívja az `rt_OneStep()`-t, és amennyiben hiba lép fel, befejezi az alkalmazást (`modell_terminate()`).

```
43 void rt_OneStep(void)
44 {
45   /* Disable interrupts here */
46
47   /* Save FPU context here (if necessary) */
48   /* Re-enable timer or interrupt here */
49   /* Set model inputs here */
50
51   mitmot_usart_mdl_6_output();
52   mitmot_usart_mdl_6_update();
53
54   /* Get model outputs here */
55
56   /* Disable interrupts here */
57   /* Restore FPU context here (if necessary) */
58   /* Enable interrupts here */
59 }
60
61 int main(void)
62 {
63
64     /*Initialize the target*/
65   mitmot_usart_mdl_6_initialize(1);
66
67
68   puts("**starting the model**\r\n");
69   while (rtmGetErrorStatus(mitmot_usart_mdl_6_M) == NULL) {
70     rt_OneStep();
71   }
72   puts("**stopping the model**\r\n");
73
74   /* Disable rt_OneStep() here */
75
76   /* Terminate model */
77   mitmot_usart_mdl_6_terminate();
78   return 0;
79 }
```

4.4. ábra A *`modell`*`_main.c` fájl

A `main()` által meghívott inicializáló és lezáró függvények (az `update` és `output` függvényekhez hasonlóan) automatikusan generálódnak, tehát a `modell.c` fájlban találhatóak. Az ilyen módon automatikusan generált fájloknak jellemzője, hogy míg a kialakításáért a Target Language Compiler felel, addig a benne szereplő tartalom nagyban függ a modellben szereplő blokkoktól, és így a hozzájuk tartozó block target fájloktól.

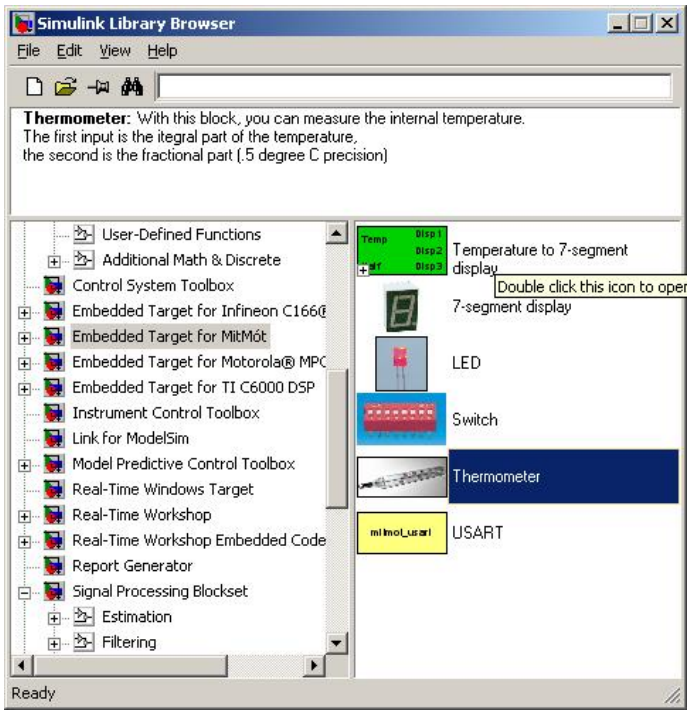
```
25 /* Model step function */
26 void pelda1_step(void)
27 {
28
29   /* S-Function Block: <Root>/Switch 1 (mitmot_switch) */
30   pelda1_B.Switch1 = getSwitch(pelda1_P.Switch1_P1);
31
32   /* S-Function Block: <Root>/Switch 2 (mitmot_switch) */
33   pelda1_B.Switch2 = getSwitch(pelda1_P.Switch2_P1);
34
35   /* S-Function Block: <Root>/Switch 3 (mitmot_switch) */
36   pelda1_B.Switch3 = getSwitch(pelda1_P.Switch3_P1);
37
38   /* S-Function "mitmot_7segment_wrapper" Block: <Root>/7-segment display */
39   dyp3(int2SegNumber(pelda1_B.Switch1),int2SegNumber(pelda1_B.Switch2),int2SegNu
40
41   /* (no update code required) */
42 }
43
44 /* Model initialize function */
45 void pelda1_initialize(boolean_T firstTime)
46 {
47
48   if (firstTime) {
49     /* registration code */
50
51     /* initialize error status */
52     rtmSetErrorStatus(pelda1_M, (const char_T *)0);
53
54     /* block I/O */
55     (void)memset(((void *) &pelda1_B), 0, sizeof(BlockIO_pelda1));
56
57     SPI_MasterInit();
58
59     Dyp_PortInit();
60
61     /* S-Function Block: <Root>/7-segment display */
62   }
63 }
```

4.5. ábra A *modell.c* fájl tartalma (részlet) (*modell*_step() és *modell*_initialize() függvények)

4.3. A Mitmót blokkcsomag

Miután áttekintettük, hogy a generálandó fájlok általános kialakításáért, kinézetétért mely tlc fájlok és mi módon felelősek, most nézzük meg, miképpen alakul ki a kód, amit valójában „megrajzoltunk”.

Amikor a modellt „papírra vetettük”, gyakorlatilag kiválogattunk a nekünk tetsző blokkokat és kialakítottuk közöttük a megfelelő kapcsolatokat. A háttérben, mint tudjuk, a blokkok mögött a block target fájlok állnak, ezek felelősek a megfelelő kódsorok elhelyezéséért a generált forráskódokban. Azonban amíg például egy `main()` függvény kialakítása egy teljes függvény megadásával járt, annak különböző szintjeivel és hivatkozásaival, addig egy blokkhoz esetleg elég egy kódsort generálni, hiszen csak a ki- és/vagy bemeneteinek aktuális állapotaira és azok közötti kapcsolatokra vagyunk kíváncsiak. Ezek persze szintén megadhatók függvény formájában külön fájlban elhelyezve, a könnyebb áttekinthetőség kedvéért.



4.6. ábra A Mitmót blokkcsomag

4.3.1. Az S-függvények

Ahhoz, hogy a fejlesztő saját blokkokkal rendelkezzen, voltaképpen két kódra van szüksége. Az egyik az, amelyik a blokk Simulink-en belüli viselkedését felügyeli (valamilyen MEX fájl), a másik pedig a block target fájl, ami a Target Language Compiler viselkedését írja le, ha találkozik az adott blokkal.

A Simulinken belül a megfelelő függvénnyel megoldhatók a tesztelés különféle lehetőségei, de még a TLC futása közben is használható, például a blokk ki-, ill. bemeneteinek kiértékelésére.

Mivel a tesztelés jelen dolgozatnak nem témája, ezért szükség volt egy olyan megoldásra, amivel könnyen előállíthatók a kellőszámú ki-, bemenettel és paraméterekkel rendelkező, a blokkok mögött elhelyezkedő S-függvények.

Az S-függvény készítő (User-Defined Functions/S-Function Builder) egy grafikus kezelőfelülettel ellátott Simulink program, ahol a felhasználó megadhatja, hogy a kialakítandó blokk milyen tulajdoságokkal rendelkezzen. A futtatás után a program generál egy *block*.c fájlt (ez egy S-függvény), amit aztán le is fordít Matlab alatt így kapunk egy C-MEX S-függvényt (Windows-os környezetben egy *blokk*.dll fájl).

A blokkunk grafikus kialakítása az S-függvény nevű blokkal (User-Defined Functions/S-Function) történik, ahol meg kell adni a blokkhoz használt S-függvényt (jelen esetben az előbb generált *block*.dll). A kapott blokknak kialakíthatjuk a kinézetét (maszkját), így kapcsolhatunk hozzá bármilyen grafikus kinézetet. Rendelhetünk a maszkhoz különböző paramétereket (pl. hogy a céleszköz melyik kapcsolójáról van szó), amennyiben ezt megtettük az S-függvényünk készítésekor is. Az így átadott paraméterek neveinek nem kell egyezniük, csak arra kell ügyelni, hogy a megfelelő számú és megfelelő típusú paramétert jutassuk el az S-függvényünknek.

4.3.2. A block target fájlok

Amennyiben kialakítottuk a blokkunk Simulinken belüli kinézetét és viselkedését, „már csak” annyi a teendőnk, hogy megadjuk, miként viselkedjen a blokk a kódgeneráláskor. Ez a block target fájlokkal tehető meg.

Egy S-függvényt tartalmazó blokkhoz tartozó block target fájl leírja, hogy a Target Language Compiler az adott kódrészleteket a futtatandó kód melyik részébe helyezze. Megkülönböztetünk tíz fajta megoldást, amelyek mindegyikét egy, a block target fájlban elhelyezett és kialakított függvény írja le (nem kötelező mindet felhasználni) [TLC, 2005]:

Az első két függvény használható a blokkhoz tartozó előmunkálatok és tlc-inicializálások elvégzéséhez:

- BlockTypeSetup: Az adott blokktípus első megjelenésekor fut le (blokktípus függő tlc-inicializálást tesz lehetővé).
- BlockInstanceSetup: Az adott blokk minden egyes megjelenéséhez biztosíthatunk tlc-inicializálást.

A következő függvények mindegyike futtatható kódot eredményez, amelyek a Real-Time Workshop a kód különböző részein helyez el a függvényeknek megfelelően:

- Enable: Amennyiben elhelyezünk a modellünkben egy, az adott blokkal egy hierarchiai szinten lévő Engedélyező (Enable) blokkot, akkor az alkalmazás futtatásakor az Enable függvény által megadott kód fog lefutni.
- Disable: Az Enable függvénnyel analóg működésű, csak ez a modellszint tiltásakor futtatja le a megfelelő kódot.
- Start: Olyan kód generálható itt, amelyik egyetlen-egyszer fut le a program teljes ciklusa alatt (pl. inicializáló függvények)
- Outputs: Itt helyezhető el a generálandó kódnak azon része, ami a blokk mindenkori kimeneteinek írását/beállítását és/vagy bemeneteinek olvasását végzi, a futtatandó program főciklusának minden egyes periódusában.
- Update: Ezzel a függvénnyel alakíthatók ki azok a kódrészletek, amelyek a belső állapotváltozók frissítéséért felelősek.
- Derivates: Itt számíttathatók a blokk esetleges folytonos idejű részletei.
- Terminate: A modell leállásakor meghívásra kerülő függvények alakíthatók ki ezzel a tlc-függvénnyel.

A következőkben rendre bemutatom a kialakított blokkokat és a mögöttük lévő kód megvalósítási formáit.

A blokkok kialakításánál azt az elvet követtem, miszerint az adott blokkhoz tartozó block target file a lehető legkevesebb függvényhívást tartalmazza az áttekinthetőség kedvéért. A fájl által meghívott függvények megvalósítása pedig egy másik, csatolt tlc-fájban történik (például `mitmot_usart.tlc` és `mitmot_usart_init.tlc`). Az összes kialakításra került block target fájl megtalálható a mellékletben (lásd M.7).

A Mitmóttal foglalkozó fejezetben tárgyalt blokkokat a következő megvalósítások követték:

- USART blokk: megvalósítja az RS232 kommunikációt, a kódba elhelyez néhány üzenetet, további üzenetek könnyen beszúrhatók és azonnal használhatók
- LED, Switch, 7-segment display, Thermometer: A tárgyaltaknak megfelelően működő blokkok.
- SPI kommunikáció: jelenleg külön blokk nincs hozzá kialakítva, bármely a perifériakártyán lévő blokk használatakor inicializálódik, és három nullát küld, a kijelző inicializálásához.

- Temperature to 7-segment display: csupán Simulink blokkokból kialakított modul, amelynek segítségével a hőmérőről érkező számokat a megjelenítéshez megfelelő formátumúra alakíthatjuk.

4.3.3. Az USART blokk

A blokk jelenlegi formájában nem generál felhasználó által változtatható kódot. A blokk által készített kód állandó, de nagyon jól használható, hiszen általa bármilyen üzenet közvetítésére képesek vagyunk a kapcsolatban lévő számítógép felé. Amennyiben a felhasználó az elkészült forrásfájlokban szeretne kézzel írt kódrészleteket elhelyezni, szüksége lehet hibakeresésre. Ennek a blokknak a használatával nincs szükség megírni az üzenetközvetítő függvényeket, azok azonnal használhatóak.

A blokkhoz tartozó block target file (`mitmot_usart.tlc`), az előbbieknek megfelelően csak az alkalmazás indulásakor lefutó kódot generál, tehát csak a `Start()` függvény került kialakításra. A függvény semmi mást nem tesz, mint lefuttat egy inicializáló-függvényt (ez még szintén TLC függvény) és generál egy üzenet kiíratást a készítendő kódhoz.

Az inicializáló-függvény egy másik tlc fájlban érhető el (`mitmot_usart_init.tlc`). A függvény első teendője, hogy létrehoz és beállít egy globális tlc változót, amellyel védelmezi magát az estlege újra futtatástól. Ezt követően a függvény előállítja a megfelelő USART specifikus definíciókat (pl. `UART_BAUD_RATE`) a *`modell`*`_private.h` fejlécfájlba, az `SLibCacheDefine()` függvény segítségével.

A következő lépésben bejegyzi az általa készített függvényeket, a külső függvényeket tartalmazó fejlécfájlba (szintén a *`modell`*`_private.h`), a `LibCacheFunctionPrototype()` függvény hívásával.

Ezek után elkészíti a csatolandó külső forrásfájlt (`mitmot_usart.c`), amit be is jegyez a külső fájlok listájába (`LibAddToModelSources()`).

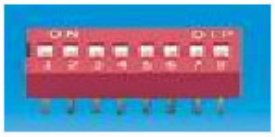
Utolsó lépésként az inicializáló-függvény beírja a *`modell`*`.c`-ben található `model_initialize()` függvénybe a készített inicializáló-függvény hívást (`USART_Init()`).

4.3.4. A perifériakártya blokkjai

A következő négy tárgyalandó blokk (Switch, LED, 7-segment display, Thermometer) mindegyike a perifériakártyán helyezkedik el. Amennyiben a perifériakártya aktívvá válik, a 7-szegmenses kijelzők különböző jeleket kapnak, ez esetleg zavaró lehet. Ezért az alkalmazásunk elején érdemes beállítani az SPI kommunikációt és küldeni 3 darab nullát a kijelzőnek (mivel shift regisztereken keresztül kapják a jeleiket), ezzel törölve a kijelzőkön megjelenítendő jeleket. E blokkok mindegyike elvégezheti ezt az inicializálást – amennyiben az még nem futott le - a saját inicializálásukkal együtt.

A blokkok block target fájljainak kialakítása nagyon hasonlít az előbbiekben részletezett USART blokkéra (block target fájl + külön inicializáló fájl), ezért csak az adott blokkok specifikus részeit fejtem ki bővebben.

A Mitmót Switch blokk



A Mitmót perifériakártyán négy darab kétállapotú kapcsoló található, amelyek egyszerű bemenetként kezelve logikai „1”-et vagy „0”-t továbbíthatnak a vezérlő egység felé. Tehát a kapcsolókkal nincs más teendő (a megfelelő inicializálás után) mint a főciklus minden lefutásakor mintavételezni az értékét és azt eltárolni egy alkalmas változóban.

A blokk egy maszkal rendelkezik, így a felhasználó információkat képes továbbítani a TLC felé. Az egyetlen továbbítható paraméter, hogy a négy lehetséges kapcsoló közül melyikről van szó. Ezzel a megoldással egy fajta blokk használható fel mind a négy kapcsolóhoz, különböző paraméter beállítások mellett.

Az előbbieknek kritériumoknak megfelelő block target fájl (`mitmot_switch.tlc`) `Start()` függvényében nem is található más, mint egy megfelelő inicializáló-függvény hívás (`%<DypInit>`). A `DypInit()` tlc függvény a perifériakártya tlc fájljában (`mitmot_dyp.tlc`) található, ezt a függvényt hívja meg a következő három blokk is az inicializáláshoz, a saját `Start()` függvényében, ezért a fájl részletezését csak itt ejtem meg.

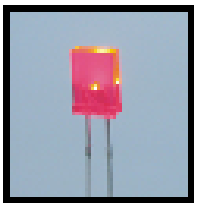
A `mitmot_dyp.tlc` kialakítása nagyon hasonló a `mitmot_usart_init.tlc` fájléhoz. Itt is megtalálható egy globális változó (`MITMOT_DYP_INIT`), ami itt még fontosabb, hiszen ez a fájl véletlen sorrendben a modellben szereplő több blokkból is meghívódhat. Ezek után megtalálhatók a külső függvények, a perifériakártyához tartozó forrásfájl kialakítása (`mitmot_dyp.c`), végül a `model1.c` fájl inicializáló függvényében a

megfelelő függvényhívás kialakítása. A fájl befejező részében azonban található még egy tlc függvényhívás (%<SPIInit()>), itt kell ugyanis meghívni a már említett SPI inicializáló függvényeket.

Az `SPIInit()` tlc függvény a `mitmot_spi.tlc` fájlban található. A fájl kialakítása teljesen analóg az előbbi két inicializáló-függvényt tartalmazó fájléhoz (`mitmot_usart_init.tlc, mitmot_dyp.tlc`).

Az iniciallizálást követően a `mitmot_switch.tlc` utolsó feladata, hogy az `Output()` függvény segítségével létrehozza a kapcsolók futtatható forráskódját. A tlc fájlban látható kód gyakorlatilag a blokk kimenetét állítja be a kapcsoló állapotának megfelelően. A kialakított kód így lényegében nem tesz mást, mint a paraméternek megfelelően az adott kapcsolóhoz tartozó változóba beírja a kapcsoló értékét.

A Mitmót LED blokk



A LED egy kimeneti eszköz, amennyiben a mikrokontroller LED-hez tartozó kimeneti lábán logikai „1”-es értéket jelez, akkor a LED világít.

A blokk, az összes többihez hasonlóan, szintén maszkolt, a beállított paraméterrel jelezhető, hogy melyik LED-ről van szó.

A hozzá tartozó block target fájlnak, az előbbiekben részletezett inicializálást követően, nincs más dolga, mint beolvassa a blokk bementi lábának az értékét. A forráskód teendője így mindössze annyi, hogy (a paraméternek megfelelően) beállítsa a mikrokontrollernek a LED-hez tartozó kimeneti lábát

A Mitmót 7-segment display blokk

A kijelzőkhöz tartozó blokk megvalósításánál figyelembe kellett venni, hogy a 3 kijelző sorba van kötve. A lehetőségek mérlegelése után úgy döntöttem, nem paraméter fogja befolyásolni, hogy melyik kijelzőnek vezérlem a viselkedését, hanem a kialakított blokknak három bemeneti lába van, amelyek rendre megfeleltethetők a három kijelzőnek.

A hozzá tartozó tlc fájl (`mitmot_7segment.tlc`) a már megszokott módon először elvégzi a szükséges inicializálásokat, ezt követően pedig elkészíti a hozzátartozó függvényhívásnak megfelelő kódot.

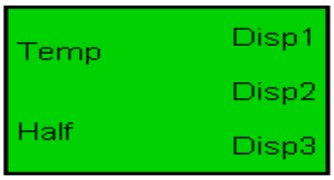
A kijelzőket leginkább számok megjelenítésére használhatjuk, valamint tartalmaznak nyolcadik szegmensként tizedespontot is. Ahhoz, hogy a tizedes pont is kihasználható legyen, de a blokk felhasználása ne legyen különösebben bonyolult, a blokk által generált kód bemeneti értékként 0 és 31 közötti egész számot fogad el (a modellben ilyen érték lehet a blokk bemenetén). Nulla és tizenöt közötti értékek esetén a nekik megfelelő hexadecimális karakter jelenik meg a kijelzőn. Amennyiben az adott számnál 16-al nagyobb bemeneti értéket küldünk a blokk adott bemenetére, a szám mellett a tizedespont is megjelenik. A tizedespont felhasználhatóságát biztosító Simulink modellrészletnek így elég tartalmaznia egy kétbemenetű összeadót és egy konstans blokkot.

A Mitmót Thermometer blokk

A hőmérőhöz tartozó blokk egy két-kimenetű egység, ahol az egyik kimenet szolgáltatja a hőmérséklet egészrészét, míg a másik a törtrészét. Mivel az IC 0,5°C-os pontossággal rendelkezik, ezért a blokk törtrész kimenetén elég két értéket közölni (van vagy nincs fél-fok).

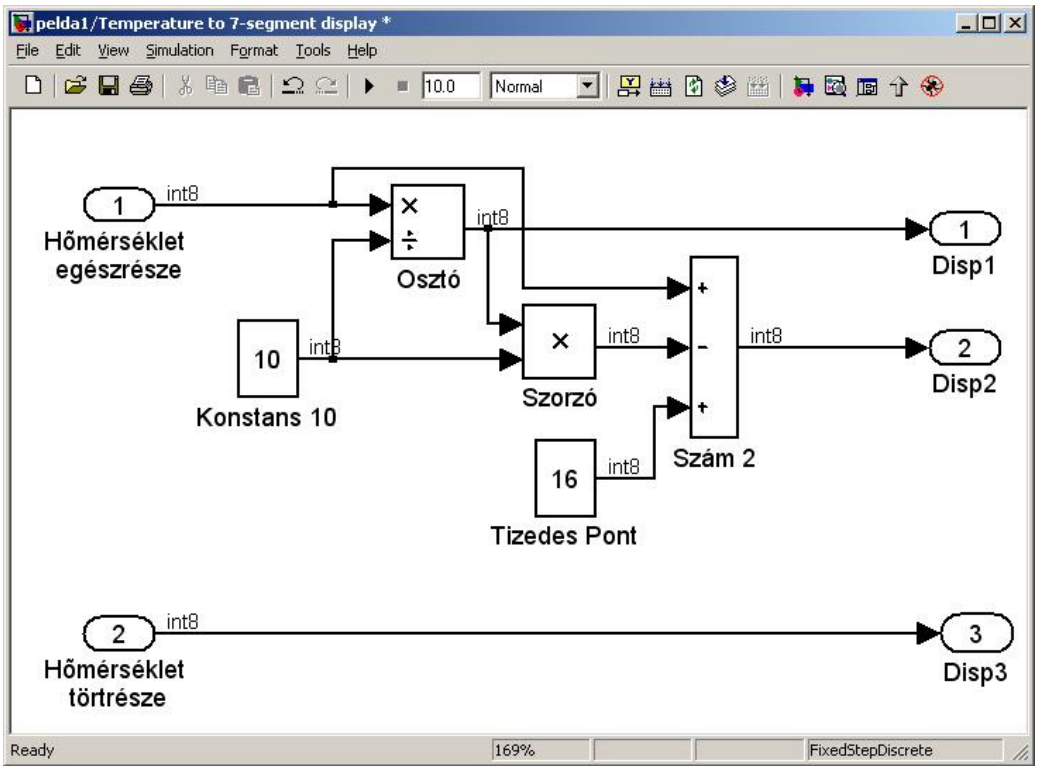
A blokk tlc fájljainak (`mitmot_thermo.tlc`) a kialakítása hasonló az előbbiekben tárgyaltakéhoz, csak itt az adat beolvasási feladatokat az I^2C kommunikáció látja el. Az I^2C kommunikációt tartalmazó forrásfájl generálásáért a `mitmot_thermo_init.tlc` felelős, itt vannak ugyanis kialakítva a kommunikációt megvalósító függvények.

4.3.5. A hőmérséklet kijelzése



A hőmérséklet az esetek többségében szeretnénk kijelezni. A hőmérő blokk kimenetei és a 7-szegmeneses kijelző blokk bemenetei azonban nem kompatibilisek egymással. Tekintettel arra, hogy a kialakított TLC program képes támogatni a Simulink által biztosított blokkok nagy részét, ezért minden probléma nélkül kialakítható egy ún. alrendszer (subsystem), amely végrehajtja a konverziót, továbbá blokkba tehető és maszkolható (lásd 4.7. ábra).

A kialakított alrendszer ebben az esetben egy két-bemenetű és három-kimenetű blokkal elfedett modell. Az alrendszer képes az egyik bemenetén beérkező 0 és 99 közötti egész számot szétbontani karaktereire és azokat továbbítani két kimenetre (a második karakter után tizedespontot tesz). A harmadik kimeneten, a második bemenet függvényében 5 vagy 0 jelenik meg.

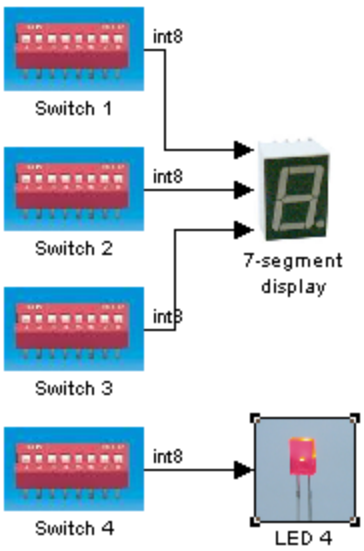


4.7. ábra A hőmérőről érkező számok konvertálása megjeleníthető értékekre

4.4. Példák a blokkcsomag használatára

A Mitmót blokkcsomaggal ugyanúgy készíthetünk egészen egyszerű reprezentációs példákat, mint bonyolult számításokat is igénylő megvalósításokat. A teendő minden esetben csak annyi, hogy a blokkok segítségével megrajzoljuk a modellünket, majd kódgenerálást és fordítást követően letöltjük a Mitmótba.

4.4.1. Első példa



4.8. ábra Modell az első példához

A példa (`pelda1.mdl`) egy egészen egyszerű modell kialakítását szemlélteti.

Az első három kapcsoló kimenetei vezérlik a 7-szegmenses kijelzők bemeneteit. Tekintettel arra, hogy a kapcsolók két állapotúak, és így kimenetükön csak „0” vagy „1” jelenhet meg, ezért a három kijelzőn is ezen számok valamelyike fog megjelenni.

A negyedik kapcsoló vezérli a negyedik LED-et: a LED világítani fog, ha a kapcsoló felkapcsolt állapotban van.

A korábban leírtaknak megfelelően (4.2.2 fejezet) több forrásfájl kerül kialakításra. A blokkok által generált kód a `pelda1.c` fájlban lelhető fel, az inicializáló függvényhívások a `pelda1_initialize()`, míg a mindig lefutó kódok a `pelda1_step()` függvényben találhatóak.

A `main()` és az rt_OneStep() függvény a `pelda1_main.c` fájlban található.

A `pelda1_data.c` fájl a felhasználásra kerülő adatokat tartalmazza.

A modellhez szükséges inicializáló függvényeket a `mitmot_dyp.c` és a `mitmot_spi.c` fájlok tartalmazzák.

A modellből készített kód forrásfájljai megtalálhatóak a M.8 mellékletben. A mitmot kezdetű forrásfájlokat ott nem közöltem, hiszen azok megegyeznek az őket generáló tlc fájlban szereplő kóddal. A könnyebb megérthetőség kedvéért ábrákon keresztül nézzük meg a generált kód néhány sorát.

Az első ábrán (4.9. ábra) láthatjuk, hogy egy kapcsoló beolvasásához nincs más teendőnk, mint meghívni egy alkalmasan kialakított függvényt (`getSwitch()`), aminek a visszatérési értéke függ a kapcsoló aktuális állapotától

```
/* S-Function Block: <Root>/Switch 1 (mitmot_switch) */
pelda1_B.Switch1 = getSwitch(pelda1_P.Switch1_P1);
```

4.9. ábra Egy kapcsoló értékének beolvasása

A következő ábra (4.10. ábra) szemlélteti a kijelzők és a LED értékeinek frissítését. Itt elég meghívni a megfelelő függvényeket (`dyp3()` és `led()`), amelyek beállítják a kimeneti perifériákat.

```
/* S-Function Block: <Root>/LED 4 (mitmot_led) */
led(pelda1_P.LED4_P1,pelda1_B.Switch4);

/* S-Function Block: <Root>/7-segment display */
dyp3(int2SegNumber(pelda1_B.Switch1),int2SegNumber(pelda1_B.Switch2)
```

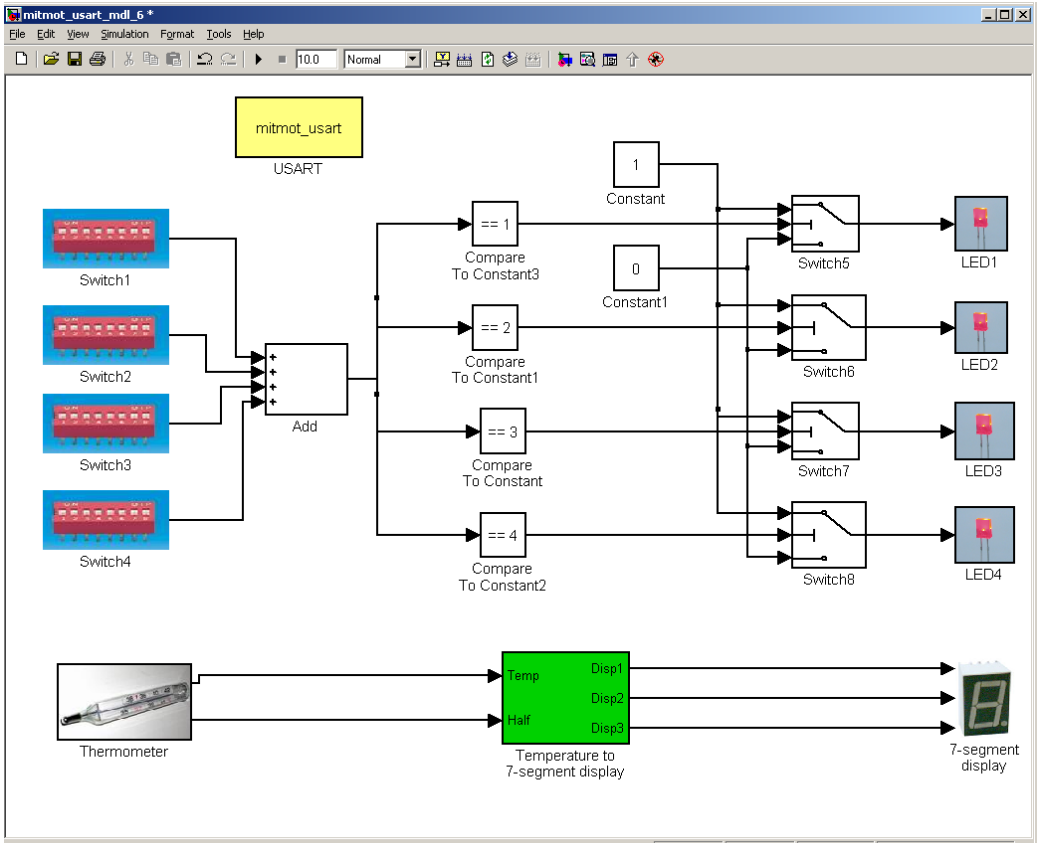
4.10. ábra A kimeneti perifériák bemeneti értékeinek beállítása

A példa utolsó ábráján megfigyelhetjük, hogy a `main()` függvény a neki megfelelő három feladatot végzi el (inicializálás, főciklus futtatása, program befejezés).

```
48
49 int main(void)
50 {
51
52   pelda1_initialize(1);
53
54   puts("**starting the model**\r\n");
55   while (rtmGetErrorStatus(pelda1_M) == NULL) {
56     rt_OneStep();
57   }
58   puts("**stopping the model**\r\n");
59
60   /* Terminate model */
61   pelda1_terminate();
62   return 0;
63 }
64
```

4.11. ábra A main() függévny

4.4.2. Második példa



4.12. ábra Komplex modell a második példához

A második példához választott modell több feladatot lát el egyszerre. Feladatának egyik része, hogy a mért hőmérsékletet megjeleníti a 7-szegmenses kijelzőkön. (4.12. ábra alsó része). A modell ezen kívül egy összetettebb feladatot is ellát: Az alkalmazás futásakor az ON állapotban lévő kapcsolók számának megfelelő sorszámú LED fog világítani.

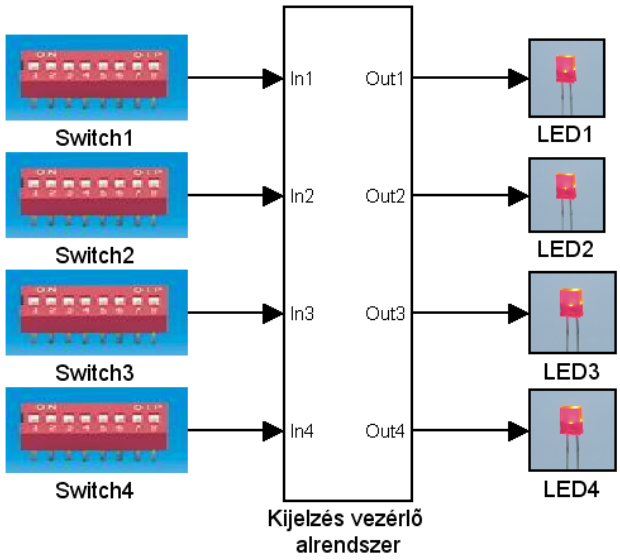
A modell második feladatának elvégzéséhez össze kell adni a kapcsolók értékeit, és kideríteni, hogy azok közül hány darab világít. A kiértékeléshez össze kell hasonlítani az összeadás eredményét a lehetséges 4 értékkel. A megfelelő összehasonlító egy kapcsolót vezérel, amelyik „1” értéket ad az adott LED lábára, míg a többi LED „0”-t kap (így nem kell foglalkozni a csupa OFF állapottal sem külön).

A modellben továbbá megtalálható az USART blokk is, így üzenetek közvetítésére is van lehetőség az alkalmazás futtatása közben.

Első pillantásra feltűnik a modellel kapcsolatban, hogy a kapcsolók és LED-ek között elhelyezkedő modellrészlet meglehetősen komplex. Ha lenne még néhány hasonló komplexitású részlet, akkor a modell teljesen áttekinthetetlen lenne. Mint tudjuk, egy

Simulink modellen belül kialakítható a nekünk tetsző hierarchia: a bonyolult részek elfedésére ki kell alakítani a megfelelő alrendszereket és a felhasználó számára érthetővé válik, hogy a modell melyik részén pontosan mi is történik (lásd 4.13. ábra).

A példához tartozó forráskódok közül csak a `pelda2.c` fájlt csatoltam a mellékletekhez (lásd M.9), hiszen csak ez különbözik lényegesen az első példa forrásaitól.



4.13. ábra A második példa modelljének részlete, ahol a bonyolult megvalósításokat elfedtük egy alrendszerrel

5. Összefoglalás, kitekintés

A dolgozat fő célkitűzése egy Simulink blokkcsomag kialakítása volt, amely képes a megvalósított modellnek megfelelő forráskódot generálni a Mitmót platformhoz.

Az első fejezet megismertette az olvasót az automatikus kódgenerálás lehetőségeivel, továbbá tárgyalta a probléma szükségességét egy olyan világban, ahol nem lehetünk meg olyan eszközök nélkül, amelyek elfedik a szemünk elől a végletekig komplex megvalósításokat és lehetőséget biztosítanak a gyors, precíz, olcsó és mégis minőségi munkavégzésre.

A második fejezetben az olvasó megismerkedhetett a Matlab programcsaláddal, betekintést nyert olyan eszközök használatába, amelyekkel a felkészült fejlesztő automatikus kódgenerálásra képes keretrendszert hozhat létre.

A harmadik fejezetben láthattuk, hogy a Mitmót rendszer miként épül fel. Megismerhettük a platform fő tulajdonságait, szerkezeti egységeit továbbá egységeinek kapcsolatát egymással, a felhasználóval és a külvilággal.

A negyedik fejezet tárgyalta a feladat megoldásának lépéseit, bemutatta a kialakított Mitmót TLC keretrendszert és a Mitmót blokkcsomagot. A továbbiakban szeretném összefoglalni e fejezet főbb motívumait és felvillantani a továbbfejlesztés számos lehetőségét.

Ahhoz, hogy a Mitmóthoz megfelelő kódot tudjunk generálni, elengedhetetlen volt egy Mitmót specifikus TLC program kialakítása. A TLC program tartalmazza a system és block target fájlokat, amelyek elengedhetetlenek a blokkcsomag helyes működéséhez.

A Real-Time Workshop elkészít egy, a modellünknek megfelelő közvetítő fájlt (`modell.rtw`). A közvetítő fájl létrejötte után a system target fájlok veszik át a vezérlést.

A TLC programhoz tartozó system target fájlok (`mitmot.tlc`, `mitmot_ertmain.tlc`, stb.) alakítják ki a Simulink *Konfigurációs Paraméterek* ablakában található *Mitmót panel* kinézetét, továbbá ugyanezek a fájlok felelősek a kódgenerálás helyes lefutásáért, a generált kód kinézetéért és a megfelelő blokkokhoz tartozó tlc fájlok helyes eléréséért és értelmezéséért.

A kódgenerálás folyamán készített forrásfájlok között található olyan, amelyiket teljes egészében céleszköz-specifikus TLC program alakít ki (pl. `modell_main.c`), a kötelezően generált fájlok többségének kialakítása azonban a Target Language Compiler feladata (pl. `modell.c`), ezek tartalmának befolyásolására csak részleges lehetőségeink vannak. A forrásfájlok között persze akadhatnak olyanok is (pl. `mitmot_spi.c`), amelyek kialakítása nem kötelező, létrehozásuknak elsősorban praktikussági okai vannak (pl. hierarchia, átláthatóság).

A system target fájlok csak a céleszköz-specifikus kód átfogó strukturáját alakítják ki, az adott blokkokhoz tartozó blokk-specifikus kódgenerálásért a block target fájlok felelősek. Minden kialakított blokk mögött egy block target fájl áll, ezáltal befolyásolva a Target Language Compiler működését.

A Mitmót blokkcsomagban eddig megvalósításra került blokkok segítségével hozzáférhetünk a Mitmót perifériakártyáján elhelyezett ki- és bemeneti eszközökhöz, valamint felhasználhatjuk a Mitmót által támogatott USART kommunikációt a kényelmes hibakereséshez.

A Mitmót blokkcsomag felhasználója által kialakított modellből készülő kód azonnal futtatható, könnyen áttekinthető és ennek megfelelően későbbi felhasználása, szerkesztése könnyen és egyszerűen elvégezhető.

A Mitmót TLC keretrendszer és Mitmót blokkcsomag kialakítása korántsem lezárt feladat. A fejlesztés folyamatosan zajlik, rengeteg a megvalósítandó lehetőség.

A későbbi megvalósításokban például a felhasználónak lehetősége lesz választani a keretrendszer által felkínált mikrokontrolleres kódmodellek között. Futtatható lesz a kód megszakításos programvezérléssel, egy- vagy többszálú programfuttatással, vagy akár operációs rendszer bevonásával is.

A blokkcsomag fejlesztéséhez szintén számos ötlet áll rendelkezésre. A napokban kaptam kézhez a Mitmót fejlesztőitől az újonnan kialakított rádiós kártyát, amelynek bevonásával rádiós kommunikáció is a blokkcsomag felhasználójának rendelkezésére fog állni.

A tervek között szerepel továbbá a jelenlegi kommunikációk felhasználhatóságának szélesítése, az adott kommunikáció által biztosított minden funkció állítható lesz a blokkhoz tartozó paraméterekkel.

Fontos megemlíteni, hogy a Mitmót mikrokontrollerének belső perifériái között szerepel egy 10-bites analóg-digitális átalakító is, annak minden lehetőségével együtt. Egy

megfelelően kialakított blokkal a felhasználó képes lesz a Mitmót buszon fogadott analóg jelet digitalizálni, eltárolni majd azzal műveleteket végezni.

A nagyobb tároló kapacitás érdekében a jövőben kialakításra fog kerülni két blokk, amelyekkel hozzáférhetővé válnak a Mitmót megfelelő memóriái (EEPROM, SRAM). Az alkalmazás által felhasználásra nem került Flash memória szintén kihasználható lesz megfelelően kialakított blokk segítségével.

A felhasználási formák tehát végtelenek. Az automatikus kódgenerálás egy roppant hatékony eszköz az ember kezében, amellyel gyorsan, könnyen készíthető igényes kód, pl. gyors prototípus készítéséhez vagy új ötletek kipróbálásához anélkül, hogy a fejlesztőnek a bitszintű részletekkel kellene vesződnie napokat vagy heteket.

Csak idő kérdése, hogy hamarosan mindannyian „papírlapon rajzoljuk” programjainkat.

Irodalomjegyzék

Könyvek

Simulink - The MathWorks, *Simulink – User's Guide*, 2005

RTW - The MathWorks, *Real-Time Workshop – User's Guide*, 2005

RTW Embedded Coder - The MathWorks, *Real-Time Workshop Embedded Coder - User's Guide*, 2005

TLC - The MathWorks, *Target Language Compiler – Reference Guide*, 2005

TI C6711 - The MathWorks, *Embedded Target for TI C6000 DSP – Reference Guide*, 2005

Felhasználói kiadványok

dSPACE - dSPACE – *TargetLink 2.0*, 2004

Mitmót vezérlő - BME-MIT, *The modular mitmót system, Az AVR mikrovezérlős kártya*, Budapest 2005

Mitmót perifériakártya - BME-MIT, *The modular mitmót system, A DPY-LED perifériakártya*, Budapest 2005

Internet

MathWorks - A MathWorks cég honlapja; http://www.mathworks.com/...

Mitmót - A Mitmót rendszer honlapja; http://bri.mit.bme.hu/...

Atmel - A Mitmót mikrokontrollerét gyártó cég honlapja; http://www.atmel.com/...

WinAVR - C-fordító a mitmóthoz; http://winavr.sourceforge.net/...

KamAVR - Integrált fejlesztőkörnyezet a WinAVR-hez; http://www.avrfreaks.com/...

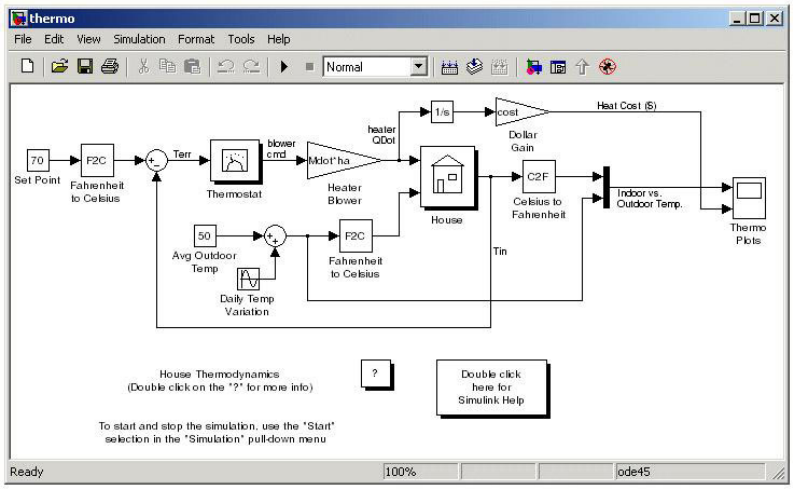
PonyProg - ISP programozó a Mitmóthoz; http://www.LancOS.com/...

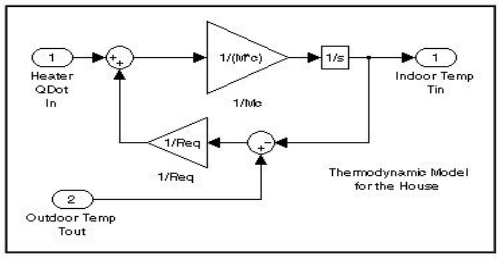
Bray++ - Terminálprogram soros port vizsgálatához; http://bray.velenje.cx/avr/terminal/...

Melléklet

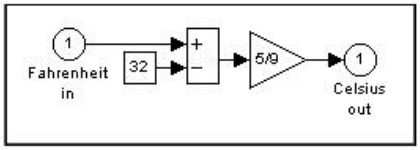
M.1. A Simulink modell hierarchikus felépítése

A Simulink használati utasításban fellelhető példa a hierarchikus blokk-diagrammokra. [Simulink, 2005]. A kialkított ház fűtési-rendszerének minden szintje jól áttekinthető, egyszerű kialaítású modell.

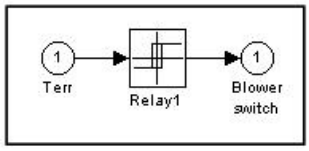




House subsystem



Fahrenheit to Celsius conversion (F2C)



Thermostat subsystem

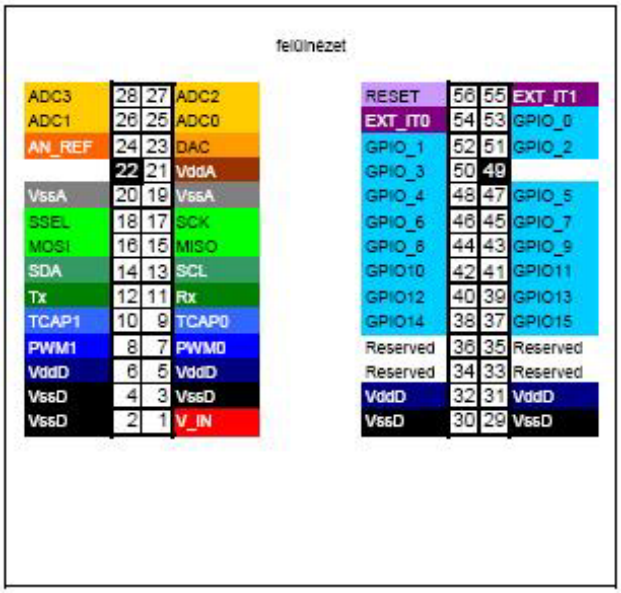
M.2. A Mitmót jelei és a busz jelkiosztása

A Mitmót buszjelei

Pozíció	Leírás	Blokknév	Jelnév	Megjegyzés
1	Stabilizálatlan tápfesz. bemenet	V_input	V_IN	Kötelező
2	Digitális föld	VssDigital	VssD	Kötelező
3	Digitális föld	VssDigital	VssD	Kötelező
4	Digitális föld	VssDigital	VssD	Kötelező
5	Stabilizált digitális tápfeszültség	VddDigital	VddD	Kötelező
6	Stabilizált digitális tápfeszültség	VddDigital	VddD	Kötelező
7	Pulzusszélesség modulált kimenet	PWM	PWM0	Kötelező
8	Pulzusszélesség modulált kimenet	PWM	PWM1	Opcionális
9	Időzítő kimenet/számláló bemenet	Timer-Capture	TCAP0	Kötelező
10	Időzítő kimenet/számláló bemenet	Timer-Capture	TCAP1	Opcionális
11	Soros vétel	USART	Rx	Kötelező
12	Soros adás	USART	Tx	Kötelező
13	Inter IC (I2C) interfész	I2C	SCL	Kötelező
14	Inter IC (I2C) interfész	I2C	SDA	Kötelező
15	SPI interfész	SPI	MISO	Kötelező
16	SPI interfész	SPI	MOSI	Kötelező
17	SPI interfész	SPI	SCK	Kötelező
18	SPI interfész	SPI	SSEL	Kötelező
19	Analóg föld	VssAnalog	VssA	Opcionális
20	Analóg föld	VssAnalog	VssA	Opcionális
21	Analóg tápfeszültség	VddAnalog	VddA	Opcionális
22	Pozicionáló vakdugó*			Kötelező
23	Digitál/analóg átalakító (kimenet)	DAC	DAC	Opcionális
24	Analóg referenciafeszültség	An_Ref	AN_REF	Opcionális
25	Analóg/digitál bemenet	ADC	ADC0	Opcionális
26	Analóg/digitál bemenet	ADC	ADC1	Opcionális
27	Analóg/digitál bemenet	ADC	ADC2	Opcionális
28	Analóg/digitál bemenet	ADC	ADC3	Opcionális

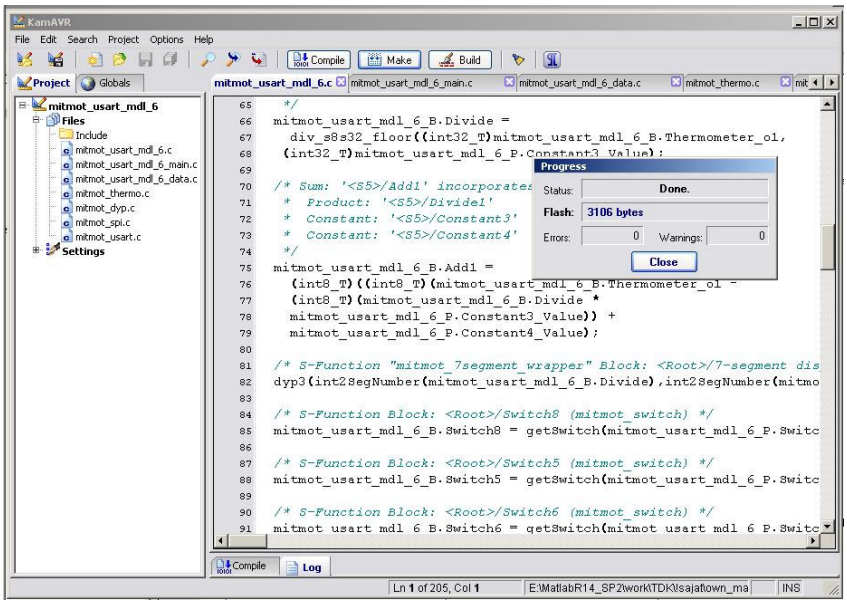
Pozíció	Leírás	Blokknév	Jelnév	Megjegyzés
29	Digitális föld	VssDigital	VssD	Kötelező
30	Digitális föld	VssDigital	VssD	Kötelező
31	Stabilizált digitális tápfeszültség	VddDigital	VddD	Kötelező
32	Stabilizált digitális tápfeszültség	VddDigital	VddD	Kötelező
33	Későbbi bővítésre foglalt	Reserved		Opcionális
34	Későbbi bővítésre foglalt	Reserved		Opcionális
35	Későbbi bővítésre foglalt	Reserved		Opcionális
36	Későbbi bővítésre foglalt	Reserved		Opcionális
37	Általános célú I/O	GPIO	GPIO15	Opcionális
38	Általános célú I/O	GPIO	GPIO14	Opcionális
39	Általános célú I/O	GPIO	GPIO13	Opcionális
40	Általános célú I/O	GPIO	GPIO12	Opcionális
41	Általános célú I/O	GPIO	GPIO11	Opcionális
42	Általános célú I/O	GPIO	GPIO10	Opcionális
43	Általános célú I/O	GPIO	GPIO_9	Opcionális
44	Általános célú I/O	GPIO	GPIO_8	Opcionális
45	Általános célú I/O	GPIO	GPIO_7	Kötelező
46	Általános célú I/O	GPIO	GPIO_6	Kötelező
47	Általános célú I/O	GPIO	GPIO_5	Kötelező
48	Általános célú I/O	GPIO	GPIO_4	Kötelező
49	Pozicionáló vakdugó*			Kötelező
50	Általános célú I/O	GPIO	GPIO_3	Kötelező
51	Általános célú I/O	GPIO	GPIO_2	Kötelező
52	Általános célú I/O	GPIO	GPIO_1	Kötelező
53	Általános célú I/O	GPIO	GPIO_0	Kötelező
54	Külső megszakítás	EXT_IT	EXT_IT0	Kötelező
55	Külső megszakítás	EXT_IT	EXT_IT1	Kötelező
56	Reset	RESET	RESET	Kötelező

A Mitmót-busz jelkiosztása



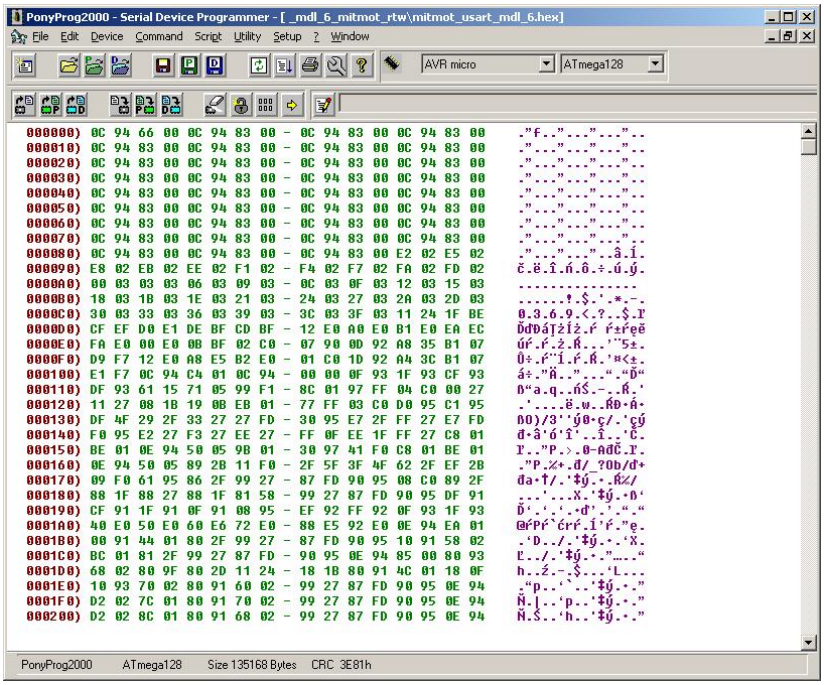
M.3. A fejlesztéshez felhasznált egyéb programok

A KamAVR



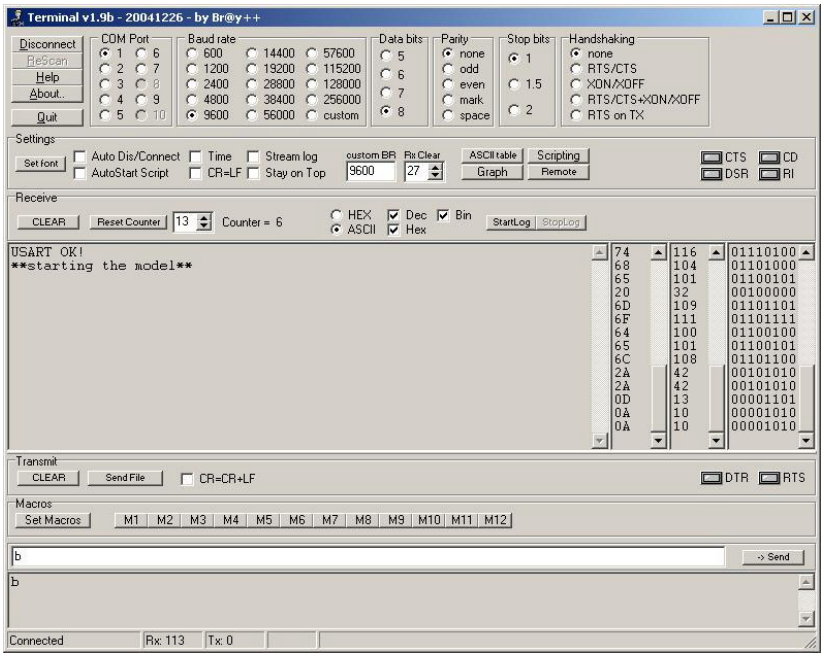
A KamAVR egy ingyenes szoftverfejlesztő környezet, AVR mikrokontrollerek C nyelvű programozásának megkönnyítésére. [KamAVR]

A PonyProg2000



A PonyProg2000 számos mikokontroller soros programozását támogató ingyenes szoftver. [PonyProg 2000]

A Terminal v1.9b



A Terminal v1.9b egy ingyenesen letölthető terminál program, amelynek segítségével könnyen kommunikálhatunk az RS232 porton a mikrokontrollerünkkel.

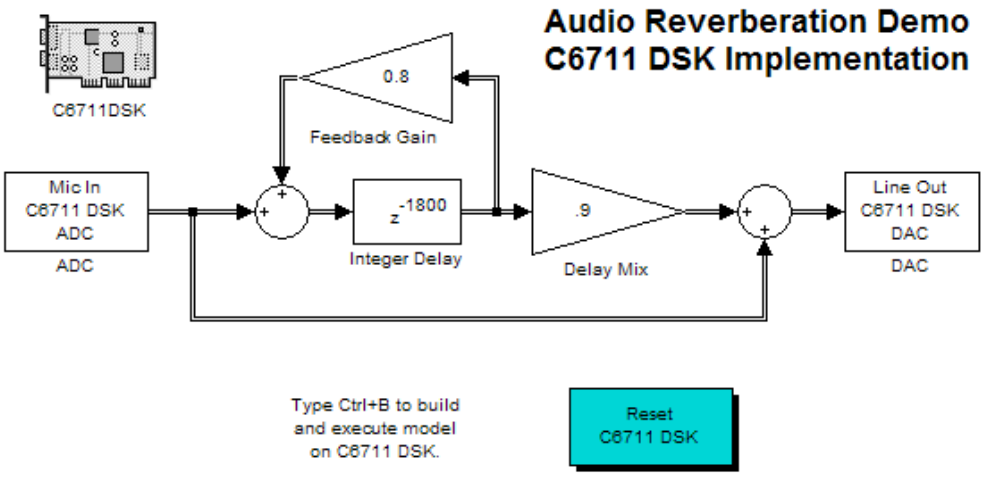
M.4. A TI DSP projekt leírása

Mielőtt hozzáfogtam volna a Mitmót TLC keretrendszer és a Mitmót blokkcsomag kialakításához tapasztalatszerzésre volt szükségem automatikus kódgenerálás terén. A tanszéken fellelhető volt egy Texas DSP (C6711 DSP), melyhez készítettek Simulink alatt használható blokkcsomagot.

A DSP-t egy ún. Starter Kit-ben forgalmazták. A főmodulja egy, a DSP köré kialakított platform, mellyel hangfeldolgozási feladatokat lehet elvégezni.

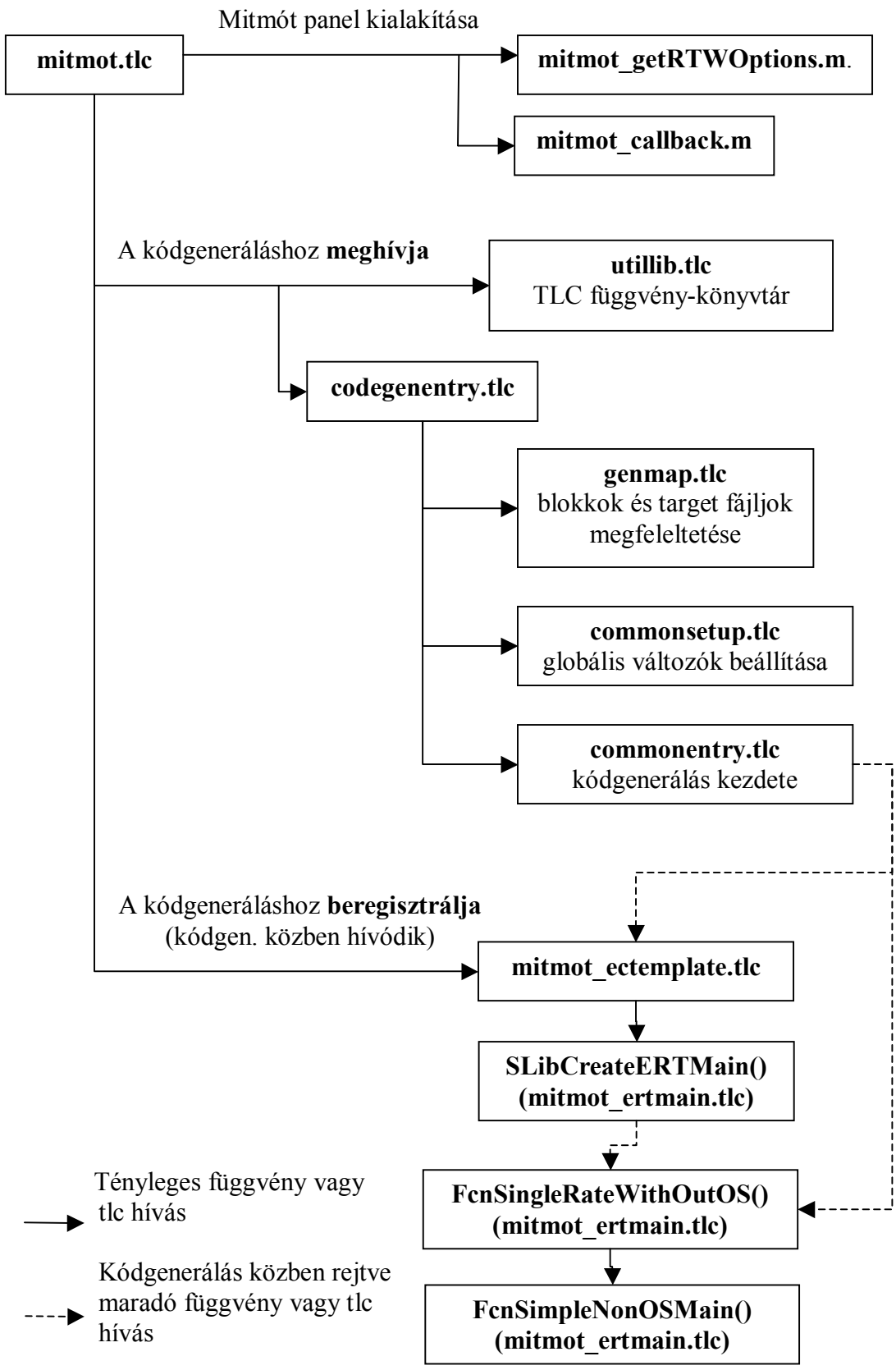
A téma szempontjából fontosnak tartottam, hogy egy viszonylag komplexebb modellt kialakítsak, majd pedig a modellhez kódot generálva vizsgáljam a DSP működését a generált kód függvényében.

A választott modellben (lásd M.4.1. ábra) található egy alkalmasan kialakított visszacsatolt hurok, melynek a segítségével visszhang kelthető (a kártya kimenetén a bemeneti jel és a hozzáadott visszhang jelenik meg).



M.4.1. ábra Visszhangkeltő modell a C6711 Texas DSP-re

M.5. A Mitmót keretrendszer működése



M.6. A kialakításra került system target fájlok

A mitmot.tlc

```
%% SYSTLC: Embedded Target for MITMOT TMF: E:\MatlabR14_SP2\toolbox\rtw\targets\mitmot\mitmot_ert.tmf
MAKE: make_rtw \
%% EXTMODE: no_ext_comm
%%
%% $Revision: 1.9 $ $Date: 2005/10/16 $
%%
%% Abstract:  System target file for Embedded Target for MITMOT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%selectfile NULL_FILE

%assign TargetName  = "MITMOT"
%assign TargetType  = "RT"
%assign Language    = "C"
%assign CodeFormat  = "Embedded-C"
%assign ERTCustomFileTemplate = "mitmot_ectemplate.tlc"

%include "utillib.tlc"
%if LibIsContinuous(0)
   %assign errTxt = "\n\nThe Embedded-C code format does not support " ...
     "continuous sample time blocks.  "
   %<LibReportError(errTxt)>
%endif

%include "codegenentry.tlc"

%%Generate project file
%include "mitmot_genfiles.tlc"

%% The contents between 'BEGIN_RTW_OPTIONS' and 'END_RTW_OPTIONS' are strictly
%% written by the standard format. We need to use this cell structure to construct
%% RTW target structure and UI.
%%
/%
  BEGIN_RTW_OPTIONS

  rtwoptions = mitmot_getRtwOptions(rtwoptions);

  %------------------------------------------%
  % Configure RTW code generation settings %
  %------------------------------------------%

  rtwgensettings.DerivedFrom        = 'ert.tlc';
  rtwgensettings.BuildDirSuffix     = '_mitmot_rtw';
  rtwgensettings.Version            = '1'; % Specify callbacks' compliance with DAStudio dialog
  rtwgensettings.StructByteAlignment = '8';

  rtwgensettings.SelectCallback     = 'mitmot_callback("SelectCallback",hSrc,hDlg);';
  rtwgensettings.UnselectCallback   = 'mitmot_callback("UnselectCallback",hSrc,hDlg);';
  rtwgensettings.ActivateCallback   = 'mitmot_callback("ActivateCallback",hSrc,hDlg);';

  END_RTW_OPTIONS
%/
%% [EOF] mitmot.tlc
```

A mitmot_getRtwOptions.m

```
function rtwoptions = mitmot_getRtwOptions(rtwoptions)

% Add MITMOT-specific RTW Options.
% This is called from the RTW_OPTIONS section of the  system target file.
%
% $Revision: 1.2 $ $Date: 2005/10/01 $

rtwoptions(end+1).prompt      = 'MITMOT target selection';
rtwoptions(end).type          = 'Category';
rtwoptions(end).enable        = 'on';
rtwoptions(end).default       = 3; % Number of items in category (max of 7)

rtwoptions(end+1).prompt      = 'Make KamAVR Project file';
rtwoptions(end).type          = 'Checkbox';
rtwoptions(end).default       = 'on';
rtwoptions(end).tlcvariable   = 'makeKAMfile';
rtwoptions(end).makevariable  = 'MAKE_KAM_FILE';
rtwoptions(end).tooltip       = ...
   ['If checked, TLC will make a KamAVR project file'];

rtwoptions(end+1).prompt      = 'Create executable makefile';
rtwoptions(end).type          = 'Checkbox';
rtwoptions(end).default       = 'off';
rtwoptions(end).tlcvariable   = 'createMakeFile';
rtwoptions(end).makevariable  = 'CRERATE_MAKE_FILE';
rtwoptions(end).tooltip       = ...
   ['If checked, TLC will make an executable make file'];

rtwoptions(end+1).prompt     = 'Build action:';
rtwoptions(end).type         = 'Popup';
rtwoptions(end).default      = 'Run_KamAVR_Project';
rtwoptions(end).tlcvariable  = 'mitmotBuildAction';
rtwoptions(end).makevariable = 'BUILD_ACTION';
rtwoptions(end).popupstrings = ['Generate_code_only|',...
   'Run_KamAVR_Project|',...
   'Build_with_makefile|',...
   'Build_and_execute_with_makefile'];
rtwoptions(end).tooltip      = ...
   ['Select action to be performed after code generation'];

% EOF mitmot_getRtwOptions.m
```

A mitmot_ectemplate.tlc

```
%% $RCSfile: mitmot_ectemplate.tlc,v $
%% $Revision: 1.1 $ $Date: 2005/10/01 $
%%
%% Abstract:
%%    Embedded Coder file processing template
%%    For Embedded Target for MITMOT.
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%selectfile NULL_FILE
%include "mitmot_ertmain.tlc"
%assign fname = "%<CompiledModel.Name>" + "_main"
%<SLibCreateMITMOTERTMain(fname)>
%%[EOF] mitmot_ectemplate.tlc
```

A mitmot_ertmain.tlc

```
%% mimot_ertmain.tlc
%% $Revision: 1.4 $
%% $Date: 2005/10/10 $
%%
%% Abstract:
%%   Generation of ERT-based main.c file for
%%   Embedded Target for MITMOT.
%%   External mode, mat-file logging, and continuous states
%%   are not supported by this target.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%selectfile NULL_FILE
%include "mitmot_main_func.tlc"

%% These are inherited from ertmainlib.tlc.

%% ============================================================
%% Function: FcnSimpleNonOSMain
%% Abstract:  Render main() function.
%%          This tlc function overrides the one in ertmainlib.tlc
%%          in order to add MITMOT-specific operations and to remove
%%          unwanted material such as external mode.
%%
%function FcnSimpleNonOSMain() Output
   int main(void)
   {
      %assign rootSystem = System[NumSystems-1]
      %assign reqInsts = LibGetSystemField(rootSystem, "ReqRootPrmHdrDataInsts")

      %<FcnMdlName()>_initialize(%<SLibModelFcnArgs("Initialize",TLC_TRUE,"")>);

      %if !reqInsts.SimStructInst && !EmptyRealTimeObject
         %assign simstructArg = tSimStruct
      %else
         %assign simstructArg = ""
      %endif
      puts("**starting the model**\r\n");
      while (%<RTMGetErrStat()> == NULL) {
         rt_OneStep(%<simstructArg>);
      }
         puts("**stopping the model**\r\n");
         %<FcnGenerateModelTerminate()>\
      return 0;
   }
%endfunction  %% FcnSimpleNonOSMain()

%% ============================================================
%function FcnSingleRateWithoutOS(cFile) void
 %openfile tmpFcnBuf
 %<FcnRTOneStepDescription()>\
 %assign rootSystem = System[NumSystems-1]
 %assign reqInsts = LibGetSystemField(rootSystem,"ReqRootPrmHdrDataInsts")
 %if !reqInsts.SimStructInst && !EmptyRealTimeObject
  %assign simstructArg = "%<tSimStructType> *%<tSimStruct>"
 %else
  %assign simstructArg = "void"
 %endif
 void rt_OneStep(%<simstructArg>)
 {
  /* Disable interrupts here */

  /* Save FPU context here (if necessary) */
  /* Re-enable timer or interrupt here */
```

```
    /* Set model inputs here */

    %<FcnCallMdlStep("")>\

    /* Get model outputs here */
    /* Disable interrupts here */
    /* Restore FPU context here (if necessary) */
    /* Enable interrupts here */
    %if ExtMode == 1
      rtExtModeCheckEndTrigger();
    %endif
  }
  %<FcnSimpleNonOSMain()>\
  %closefile tmpFcnBuf
  %return tmpFcnBuf
%endfunction %% SingleRateWithoutOS

%%  ********************************************************************
%%  This is the entry point for generating main.c for ERT-based target
%%  ********************************************************************

%function SLibCreateMITMOTERTMain(fName) void

    %assign cFile = SLibAddModelFile("SystemBody","Simulink",fName)

    %%openfile tmpFcnBuf
    %<SLibDeclareModelFcnArgs(TLC_TRUE)>\

    %%<SLibSetModelFileAttribute(cFile, "Definitions", tmpFcnBuf)>

    %openfile tmpFcnBuf
      /*
       * Real-Time Workshop code generation for Simulink model "%<FcnMdlName()>"
       *
       * Real-Time Workshop file version      : %<Version>
       * Real-Time Workshop file generated on : %<GeneratedOn>
       * C source code generated on           : %<TLC_TIME>
       *
       *
       * Compiler specified defines:
       *   RT
       *   MODEL          = %<CompiledModel.Name>
       *   NUMST          = %<CompiledModel.NumSampleTimes> (Number of sample times)
       *   NCSTATES       = %<CompiledModel.NumContStates> (Number of continuous states)
       *   TID01EQ        = %<CompiledModel.FixedStepOpts.TID01EQ>
       *                  (Set to 1 if sample time task id's 0 and 1 have equal rates)
       *
       * For more information:
       *   o Real-Time Workshop User's Guide
       *   o Real-Time Workshop Embedded Coder User's Guide
       *   o Embedded Target for MITMOT User's Guide
       */
    %closefile tmpFcnBuf
    %<SLibSetModelFileAttribute(cFile, "Banner", tmpFcnBuf)>

    %<SLibSetModelFileAttribute(cFile, "Functions", ...
      FcnGenerateMainFunctions(0,cFile))>

    %openfile tmpFcnBuf
    #include <avr/io.h>
    #include <stdio.h>
    #include "rtwtypes.h"
    #include "%<FcnGetPublicModelHeaderFile()>"
    %closefile tmpFcnBuf
    %<SLibSetModelFileAttribute(cFile, "Includes", tmpFcnBuf)>
%endfunction %% [EOF]  mitmot_ertmain.tlc
```

A mitmot_main_func.tlc

```
%% mitmot_main_func.tlc
%% $Revision: 1.2 $
%% $Date: 2005/10/10 $
%%
%% Abstract:
%%   TLC functions that are used in the generation
%%   of main.c for both ERT and GRT-based MITMOT
%%   targets.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%selectfile NULL_FILE

%% ============================================================================
%% Function: render_defines
%% Abstract:
%%
%function render_defines() void
%openfile buff

   #ifndef cbi
      #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
   #endif
   #ifndef sbi
      #define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
   #endif

%closefile buff
%return buff
%endfunction %% render_defines

%% ============================================================================
%% Function: render_target_setup
%% Abstract:
%%
%function render_target_setup() void
%openfile buff
   //target setup by vikk
%closefile buff
%return buff
%endfunction %% render_target_setup


%% ============================================================================
%% Function: debugMsgPrint
%% Abstract:  Pring a message to the appropriate debugging output
%%            medium.
%%
%function debugMsgPrint(msg) void
%openfile buff

   puts("%<msg>\n");

%closefile buff
%return buff
%endfunction %% debugMsgPrint

%% EOF  mitmot_main_func.tlc
```

A mitmot_genfiles.tlc

```
%% File: mitmot_genfiles.tlc
%%
%% Abstract:
%%   This tlc file will generate a project file needed to build a
%%   mitmot project in KamAVR.
%%
%% $Revision: 1.2 $
%% $Date: 2005/10/10 $

%selectfile NULL_FILE

%selectfile STDOUT
### Generating KamAVR project file : %<CompiledModel.Name>.kam
%selectfile NULL_FILE

%openfile project_file = "%<CompiledModel.Name>.kam"
<?xml version="1.0"?>
<Project>%<CompiledModel.Name>
   <Files>
      <Folder>Include
      </Folder>
            <File path="%<CompiledModel.Name>.c" open="1"/>
            <File path="%<CompiledModel.Name>_main.c" open="1"/>
            <File path="%<CompiledModel.Name>_data.c" open="1"/>
 %assign sources = ""
 %foreach idx = ::CompiledModel.NumSources
  %if (::CompiledModel.Sources[idx] != "%<CompiledModel.Name>")
     %assign sources = "" + ::CompiledModel.Sources[idx] + "." + ::LangFileExt + ""
            <File path="%<sources>" open="1"/>
  %endif
 %endforeach
   </Files>
   <Settings>
      <CPU>
         <type>atmega128</type>
         <clock>8000000</clock>
      </CPU>
      <Flags>
         <optim>s</optim>
         <unsign_char>1</unsign_char>
         <unsign_bit>1</unsign_bit>
         <printf>Standard</printf>
         <scanf>Standard</scanf>
      </Flags>
      <gen_files>
         <OutFormat>ihex</OutFormat>
         <DbgFormat>ELF/DWARF-2</DbgFormat>
         <list>1</list>
         <map>1</map>
         <makefile>1</makefile>
      </gen_files>
   </Settings>
</Project>
%closefile project_file

%with CompiledModel
%assign current_path = RTWGenSettings.RelativeBuildDir + "\\" + Name + ".kam"
%endwith
%%<FEVAL("OpenMitmotProject", current_path)>
%<FEVAL("dos","C:\\WinAVR\\KAM\\KamAVR.exe
E:\\MatlabR14_SP2\\work\\tdk\\!sajat\\own_matlab7\\code_gen_harmadik\\dolgozat\\" + current_path + " &")>
```

M.7. A Mitmót blokkcsomag block target fájljai

A mitmot_switch.tlc

```
%% File : mitmot_switch.tlc
%% Created: Sun Sep 25 16:01:01 2005
%%
%implements  mitmot_switch "C"
%include "mitmot_dyp.tlc"
%include "mitmot_spi.tlc"

%% Function: Start ===========================================================
%%
%function Start(block, system) Output
   %<DypInit()>
%endfunction %%Start


%% Function: Outputs =========================================================
%%
%% Purpose:
%%      Code generation rules for mdlOutputs function.
%%
%function Outputs(block, system) Output
  %assign y = LibBlockOutputSignal(0, "", "", 0)
  %assign param = LibBlockParameter(P1,"","",0)

    /* %<Type> Block: %<Name> (%<ParamSettings.FunctionName>) */
    %<y> = getSwitch(%<param>);
  %%
%endfunction %%Outputs
%% [EOF] mitmot_switch.tlc
```

A mitmot_led.tlc

```
%% File : mitmot_led.tlc
%% Created: Sun Sep 25 16:01:29 2005
%%
%implements  mitmot_led "C"
%include "mitmot_dyp.tlc"
%include "mitmot_spi.tlc"
%% Function: Start ===========================================================
%%
%function Start(block, system) Output
   %<DypInit()>
%endfunction %%BlockTypeSetup
%% Function: Outputs =========================================================
%%
%% Purpose:
%%      Code generation rules for mdlOutputs function.
%%
%function Outputs(block, system) Output
  %assign u = LibBlockInputSignal(0, "", "", 0)
  %assign param = LibBlockParameter(P1,"","",0)
    /* %<Type> Block: %<Name> (%<ParamSettings.FunctionName>) */
    led(%<param>,%<u>);
  %%
%endfunction
%% [EOF] mitmot_led.tlc
```

A mitmot_7segment.tlc

```
%% File : mitmot_7segment.tlc
%% Created: Mon Oct 10 00:49:18 2005
%%
%% Function: Start ============================================================
%%
%function Start(block, system) Output
  /* %<Type> Block: %<Name> */
  %<DypInit()>
%endfunction

%% Function: Outputs ==========================================================
%%
%function Outputs(block, system) Output
  /* S-Function "mitmot_7segment_wrapper" Block: %<Name> */
 %assign u0 = LibBlockInputSignal(0, "", "", 0)
 %assign u1 = LibBlockInputSignal(1, "", "", 0)
 %assign u2 = LibBlockInputSignal(2, "", "", 0)
 dyp3(int2SegNumber(%<u0>),int2SegNumber(%<u1>),int2SegNumber(%<u2>));

 %%
%endfunction
%% [EOF] mitmot_7segment.tlc
```

A mitmot_usart.tlc

```
%% File : mitmot_usart.tlc
%% Created: Sun Oct  9 19:14:07 2005
%%
%include "mitmot_usart_init.tlc"

%% Function: Start ============================================================
%%
%function Start(block, system) Output
  /* %<Type> Block: %<Name> */
   %<UsartInit()>
   puts("USART OK!\r\n");
%endfunction

%% [EOF] mitmot_usart.tlc
```

A mitmot_dyp.tlc

```
%% File : mitmot_dyp.tlc
%% Created: Sun Sep 25 16:01:01 2005
%%
%include "mitmot_spi.tlc"

%if !EXISTS("MITMOT_DYP_TLC")
%assign ::MITMOT_DYP_TLC = 1

%% ============================================================================
%% Function: DypInit
%% Abstract:
%%

%function DypInit() void

%if !EXISTS("MITMOT_DYP_INIT")
  %assign ::MITMOT_DYP_INIT = 1

%%openfile dyph = "mitmot_dyp.h"
%openfile dyph

   extern void Dyp_PortInit(void);
   extern void led(unsigned short index, unsigned short on);
   extern unsigned short getSwitch(unsigned short index);
   extern void dyp3(unsigned short d2,unsigned short d1,unsigned short d0);
   extern unsigned short int2SegNumber(unsigned short i);


%closefile dyph
%<LibCacheFunctionPrototype(dyph)>

%<LibAddToCommonIncludes("<avr/delay.h>")>

%openfile dypc = "mitmot_dyp.c"
   #include "%<CompiledModel.Name>.h"
   #include "%<CompiledModel.Name>_private.h"

   //Inicializálja a portokat, hogy müxön a kártya
   void Dyp_PortInit(void) {
      DDRC = 0;
      DDRA = ~((1<<1) | (1<<2) | (1<<3));
      PORTA = 0;

      puts("DYP PORTS OK!\r\n");
      dyp3(0,0,0);
   }

   //Ezzel fog müxeni a led
   void led(unsigned short index, unsigned short on) {
      if (on)
         sbi(PORTA, index+3);
      else
         cbi(PORTA, index+3);
   }

   //Ezzel fog müxeni a switch
   unsigned short getSwitch(unsigned short index) {
      unsigned short temp = 0xff;

      cbi(temp, index-1);
      if ((PINC | temp) == 0xff)
         return 1;
      else
```

```
        return 0;
    }

    //Ezzel lehet datát a 7 szegmenses kijelzőre
    void dyp3(unsigned short d2,unsigned short d1,unsigned short d0) {
      SPI_MasterTransmit(~d2);
      SPI_MasterTransmit(~d1);
      SPI_MasterTransmit(~d0);

      _delay_ms(32);
    }

    //A beadott egy karakteres számot "7segmenses számmá" lakítja
    //ha 0x10-el nagyobb számot küldünk, akkor a tizedes pont is megjelenik
    unsigned short int2SegNumber(unsigned short i) {
        switch (i) {
            //hexa számok 0x00->0x0F
            case 0 : return ~0x28;
            case 1 : return ~0xf9;
            case 2 : return ~0x1c;
            case 3 : return ~0x58;
            case 4 : return ~0xc9;
            case 5 : return ~0x4a;
            case 6 : return ~0x0a;
            case 7 : return ~0xf8;
            case 8 : return ~0x08;
            case 9 : return ~0x48;
            case 10 : return ~0x88;
            case 11 : return ~0x0b;
            case 12 : return ~0x2e;
            case 13 : return ~0x19;
            case 14 : return ~0x0e;
            case 15 : return ~0x8e;

            //hexa számok 0x00->0x0F + tizedespont
            case 16: return ~(0x28) | 0x08;
            case 17: return ~(0xf9) | 0x08;
            case 18: return ~(0x1c) | 0x08;
            case 19: return ~(0x58) | 0x08;
            case 20: return ~(0xc9) | 0x08;
            case 21: return ~(0x4a) | 0x08;
            case 22: return ~(0x0a) | 0x08;
            case 23: return ~(0xf8) | 0x08;
            case 24: return ~(0x08) | 0x08;
            case 25: return ~(0x48) | 0x08;
            case 26 : return ~(0x88) | 0x08;
            case 27 : return ~(0x0b) | 0x08;
            case 28 : return ~(0x2e) | 0x08;
            case 29 : return ~(0x19) | 0x08;
            case 30 : return ~(0x0e) | 0x08;
            case 31 : return ~(0x8e) | 0x08;
            default : return i;
        }
        return 0;
    }
%closefile dypc
%<LibAddToModelSources("mitmot_dyp")>
%openfile buff
        %<SPIInit()>
        Dyp_PortInit();
%closefile buff
%return buff
%endif %%volt-e már ez
%endfunction %% DypInit
%endif  %%MITMOT_DYP_TLC
%% [EOF] mitmot_dyp.tlc
```

A mitmot_spi.tlc

```
%% File : mitmot_spi.tlc
%% Created: Sun Sep 25 16:01:01 2005
%%
%if !EXISTS("MITMOT_SPI_TLC")
%assign ::MITMOT_SPI_TLC = 1

%% ============================
%% Function: DypInit
%% Abstract:
%%
%function SPIInit() void

%if !EXISTS("MITMOT_SPI_INIT")
   %assign ::MITMOT_SPI_INIT = 1

%%openfile spih = "mitmot_spi.h"
%openfile spih
   #define SPI_MISO                    3
   #define SPI_MOSI                    2
   #define SPI_SCK                     1
   #define SPI_SS                     0
%closefile spih
%<LibCacheDefine(spih)>

%openfile tmpFcnBuf
   #ifndef cbi
      #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
   #endif
   #ifndef sbi
      #define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
   #endif
%closefile tmpFcnBuf
%<LibCacheDefine(tmpFcnBuf)>
%<LibAddToCommonIncludes("<avr/io.h>")>

%openfile spih
   extern void SPI_MasterInit(void);
   extern unsigned short SPI_MasterTransmit(unsigned short d);
%closefile spih
%<LibCacheFunctionPrototype(spih)>

%openfile spic = "mitmot_spi.c"
   #include "%<CompiledModel.Name>.h"
   #include "%<CompiledModel.Name>_private.h"

   //Inicializálja a portokat, hogy müxön a kártya
   void SPI_MasterInit(void) {
     DDRB = (1<<SPI_MOSI) | (1<<SPI_SCK) | (1<<SPI_SS);
     SPCR = (1<<SPE) | (1<<MSTR) | (1<<SPR0);

     puts("SPI OK!\r\n");
   }

   //Ezzel lehet datát kivinni az SPI-re
   unsigned short SPI_MasterTransmit(unsigned short d) { // send character over SPI
     unsigned short received = 0;
     SPDR = d;
     while(!(SPSR & (1<<SPIF)));
     received = SPDR;
     return (received);
   }
%closefile spic
%<LibAddToModelSources("mitmot_spi")>
```

```
%openfile buff
   SPI_MasterInit();
%closefile buff
%return buff

%endif %% ti. ez lefutott-e már
%endfunction %% SPIInit
%endif  %%MITMOT_SPI_TLC
%% [EOF] mitmot_spi.tlc
```

A mitmot_usart_init.tlc

```
%% File : mitmot_usart_init.tlc
%% Created: Sun Sep 25 16:01:01 2005
%%
%include "mitmot_spi.tlc"

%if !EXISTS("MITMOT_USART_INIT_TLC")
%assign ::MITMOT_USART_INIT_TLC = 1

%function UsartInit() void

%if !EXISTS("MITMOT_USART_INIT")
  %assign ::MITMOT_USART_INIT = 1

%<LibAddToCommonIncludes("<stdio.h>")>

%openfile usarth
   #define F_OSC 8000000          /* oscillator-frequency in Hz */
   #define UART_BAUD_RATE 9600
   #define baud 51
   #define UART_BAUD_CALC(UART_BAUD_RATE,F_OSC) ((F_OSC)/((UART_BAUD_RATE)*16)-1)
%closefile usarth
%<LibCacheDefine(usarth)>

%openfile usarth
   extern void USART_Init(void);
   extern int USART_Putc(unsigned char c);
   extern int USART_Getc(void);
   extern void USART_Puts (char *s);
   extern void SerialTerminate(void);
%closefile usarth
%<LibCacheFunctionPrototype(usarth)>

%openfile usartc = "mitmot_usart.c"
   #include "%<CompiledModel.Name>.h"
   #include "%<CompiledModel.Name>_private.h"

   void USART_Init(void) {
      // set baud rate
      UBRR1H = (unsigned char)(baud>>8);
      UBRR1L = (unsigned char)baud;

      // Enable receiver and transmitter; enable RX interrupt
      UCSR1B = (1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1);
      //asynchronous 8N1
      UCSR1C = (3<<UCSZ10) /* |(1<<UMSEL1) */;

      stdout = fdevopen(USART_Putc, USART_Getc,0);
   }

   int USART_Putc(unsigned char c) {
     // wait until UDR1 ready
```

```
    while(!(UCSR1A & (1 << UDRE1)));

    UDR1 = c;    // send character
    return 0;
  }

  int USART_Getc(void) {
    // wait until UDR1 ready
    uint16_t c;
    c = UDR1;
    return (unsigned char)c;
  }

  void USART_Puts (char *s) {
    //  loop until *s != NULL
    while (*s) {
      USART_Putc(*s);
      s++;
    }
  }

  void SerialTerminate(void) { // terminate sent string!!!
    while(!(UCSR1A & (1 << UDRE1)));
    UDR1 = 0x0d;
    while(!(UCSR1A & (1 << UDRE1)));
    UDR1 = 0x0a;
  }

    // INTERRUPT can be interrupted
    // SIGNAL can't be interrupted
  SIGNAL (SIG_UART1_RECV) { // USART RX interrupt
    uint16_t c;
    c = UDR1;
    //ew(0x0400, c);
  }

%closefile usartc
%<LibAddToModelSources("mitmot_usart")>

%openfile buff
      USART_Init();
%closefile buff
%return buff

%endif %%volt-e már ez
%endfunction %% UsartInit
%endif  %%MITMOT_DYP_TLC
%% [EOF] mitmot_dyp.tlc
```

M.8. A pelda1.mdl modellhez generált forrásfájlok

A pelda1.c

```
#include "pelda1.h"
#include "pelda1_private.h"

/* Block signals (auto storage) */
BlockIO_pelda1 pelda1_B;

/* Real-time model */
RT_MODEL_pelda1 pelda1_M_;
RT_MODEL_pelda1 *pelda1_M = &pelda1_M_;

/* Model step function */
void pelda1_step(void)
{

  /* S-Function Block: <Root>/Switch 1 (mitmot_switch) */
  pelda1_B.Switch1 = getSwitch(pelda1_P.Switch1_P1);

  /* S-Function Block: <Root>/Switch 2 (mitmot_switch) */
  pelda1_B.Switch2 = getSwitch(pelda1_P.Switch2_P1);

  /* S-Function Block: <Root>/Switch 3 (mitmot_switch) */
  pelda1_B.Switch3 = getSwitch(pelda1_P.Switch3_P1);

  /* S-Function "mitmot_7segment_wrapper" Block: <Root>/7-segment display */

dyp3(int2SegNumber(pelda1_B.Switch1),int2SegNumber(pelda1_B.Switch2),int2SegNumber(
pelda1_B.Switch3));

  /* S-Function Block: <Root>/Switch 4 (mitmot_switch) */
  pelda1_B.Switch4 = getSwitch(pelda1_P.Switch4_P1);

  /* S-Function Block: <Root>/LED 4 (mitmot_led) */
  led(pelda1_P.LED4_P1,pelda1_B.Switch4);

  /* (no update code required) */
}

/* Model initialize function */
void pelda1_initialize(boolean_T firstTime)
{

  if (firstTime) {
    /* registration code */

    /* initialize error status */
    rtmSetErrorStatus(pelda1_M, (const char_T *)0);

    /* block I/O */
    (void)memset(((void *) &pelda1_B), 0, sizeof(BlockIO_pelda1));

    SPI_MasterInit();

    Dyp_PortInit();

    /* S-Function Block: <Root>/7-segment display */
  }
}
```

A pelda1_main.c

```
#include <avr/io.h>
#include <stdio.h>
#include "rtwtypes.h"
#include "pelda1.h"

void rt_OneStep(void)
{
  /* Disable interrupts here */

  /* Save FPU context here (if necessary) */
  /* Re-enable timer or interrupt here */
  /* Set model inputs here */

  pelda1_step();

  /* Get model outputs here */

  /* Disable interrupts here */
  /* Restore FPU context here (if necessary) */
  /* Enable interrupts here */
}

int main(void)
{

  pelda1_initialize(1);

  puts("**starting the model**\r\n");
  while (rtmGetErrorStatus(pelda1_M) == NULL) {
    rt_OneStep();
  }
  puts("**stopping the model**\r\n");

  /* Disable rt_OneStep() here */

  /* Terminate model */
  pelda1_terminate();
  return 0;
}
```

A pelda1_data.c

```
#include "pelda1.h"
#include "pelda1_private.h"

/* Block parameters (auto storage) */
Parameters_pelda1 pelda1_P = {
  1 ,                                  /* Switch1_P1 : '<Root>/Switch 1' */
  2 ,                                  /* Switch2_P1 : '<Root>/Switch 2' */
  3 ,                                  /* Switch3_P1 : '<Root>/Switch 3' */
  4 ,                                  /* Switch4_P1 : '<Root>/Switch 4' */
  4 ,                                  /* LED4_P1 : '<Root>/LED 4' */
};
```

A pelda1.h

```
#ifndef _RTW_HEADER_pelda1_h_
#define _RTW_HEADER_pelda1_h_

#include <math.h>
#include <float.h>
#include <string.h>
#include <avr/delay.h>
#include <avr/io.h>
#include "rtwtypes.h"
#include "pelda1_types.h"

/* Macros for accessing real-time model data structure  */

#ifndef rtmGetErrorStatus
# define rtmGetErrorStatus(rtm) ((rtm)->errorStatus)
#endif

#ifndef rtmSetErrorStatus
# define rtmSetErrorStatus(rtm, val) ((rtm)->errorStatus = (val))
#endif

/* Block signals (auto storage) */
typedef struct _BlockIO_pelda1 {
  int8_T Switch1;                       /* '<Root>/Switch 1' */
  int8_T Switch2;                       /* '<Root>/Switch 2' */
  int8_T Switch3;                       /* '<Root>/Switch 3' */
  int8_T Switch4;                       /* '<Root>/Switch 4' */
} BlockIO_pelda1;

/* Parameters (auto storage) */
struct _Parameters_pelda1 {
  int8_T Switch1_P1;                    /* Expression: int8(sw_mask)
                                         * '<Root>/Switch 1'
                                         */
  int8_T Switch2_P1;                    /* Expression: int8(sw_mask)
                                         * '<Root>/Switch 2'
                                         */
  int8_T Switch3_P1;                    /* Expression: int8(sw_mask)
                                         * '<Root>/Switch 3'
                                         */
  int8_T Switch4_P1;                    /* Expression: int8(sw_mask)
                                         * '<Root>/Switch 4'
                                         */
  int8_T LED4_P1;                       /* Expression: int8(led_mask)
                                         * '<Root>/LED 4'
                                         */
};
/* Real-time Model Data Structure */
struct _RT_MODEL_pelda1_Tag {
  const char *errorStatus;
};
/* Block parameters (auto storage) */
extern Parameters_pelda1 pelda1_P;
/* Block signals (auto storage) */
extern BlockIO_pelda1 pelda1_B;
/* Model entry point functions */
extern void pelda1_initialize(boolean_T firstTime);
extern void pelda1_step(void);
extern void pelda1_terminate(void);
/* Real-time Model object */
extern RT_MODEL_pelda1 *pelda1_M;
```

A pelda1_private.h

```
#ifndef _RTW_HEADER_pelda1_private_h_
#define _RTW_HEADER_pelda1_private_h_

#include "rtwtypes.h"

/* Private macros used by the generated code to access rtModel */

#define SPI_MISO                       3
#define SPI_MOSI                       2
#define SPI_SCK                        1
#define SPI_SS                         0

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

#ifndef __RTWTYPES_H__
#error This file requires rtwtypes.h to be included
#else
#ifdef TMWTYPES_PREVIOUSLY_INCLUDED
#error This file requires rtwtypes.h to be included before tmwtypes.h
#else
/* Check for inclusion of an incorrect version of rtwtypes.h */
#ifndef RTWTYPES_ID_C08S16I32L32N32F1
#error This code was generated with a different "rtwtypes.h" than the file included
#endif                                 /* RTWTYPES_ID_C08S16I32L32N32F1 */
#endif                                 /* TMWTYPES_PREVIOUSLY_INCLUDED */
#endif                                 /* __RTWTYPES_H__ */

extern void Dyp_PortInit(void);
extern void led(unsigned short index, unsigned short on);
extern unsigned short getSwitch(unsigned short index);
extern void dyp3(unsigned short d2,unsigned short d1,unsigned short d0);
extern unsigned short int2SegNumber(unsigned short i);

extern void SPI_MasterInit(void);
extern unsigned short SPI_MasterTransmit(unsigned short d);
```

A pelda1_types.h

```
#ifndef _RTW_HEADER_pelda1_types_h_
#define _RTW_HEADER_pelda1_types_h_

/* Parameters (auto storage) */
typedef struct _Parameters_pelda1 Parameters_pelda1;

/* Forward declaration for rtModel */
typedef struct _RT_MODEL_pelda1_Tag RT_MODEL_pelda1;

#endif /* _RTW_HEADER_pelda1_types_h_ */
```

Az rtwtypes.h (részletek)

```
#ifndef __RTWTYPES_H__
#define __RTWTYPES_H__
#ifndef __TMWTYPES__
#define __TMWTYPES__

#include <limits.h>
/*=======================================================*
 * Fixed width word size data types:                     *
 *   int8_T, int16_T, int32_T     - signed 8, 16, or 32 bit integers     *
 *   uint8_T, uint16_T, uint32_T  - unsigned 8, 16, or 32 bit integers   *
 *   real32_T, real64_T           - 32 and 64 bit floating point numbers *
 *=======================================================*/

typedef signed char int8_T;
typedef unsigned char uint8_T;
typedef short int16_T;
typedef unsigned short uint16_T;
typedef int int32_T;
typedef unsigned int uint32_T;
typedef float real32_T;
typedef double real64_T;

/*=====================================================*
 * Generic type definitions: real_T, time_T, boolean_T, char_T, int_T,       *
 *                           uint_T and byte_T.                              *
 *=====================================================*/

typedef double real_T;
typedef double time_T;
typedef unsigned char boolean_T;
typedef int int_T;
typedef unsigned int uint_T;
typedef char char_T;
typedef char_T byte_T;


/*=====================================================*
 * Min and Max:                                        *
 *   int8_T, int16_T, int32_T     - signed 8, 16, or 32 bit integers     *
 *   uint8_T, uint16_T, uint32_T  - unsigned 8, 16, or 32 bit integers   *
 *=====================================================*/

#define MAX_int8_T                     ((int8_T)(127))
#define MIN_int8_T                     ((int8_T)(-128))
#define MAX_uint8_T                    ((uint8_T)(255))
#define MIN_uint8_T                    ((uint8_T)(0))
#define MAX_int16_T                    ((int16_T)(32767))
#define MIN_int16_T                    ((int16_T)(-32768))
#define MAX_uint16_T                   ((uint16_T)(65535))
#define MIN_uint16_T                   ((uint16_T)(0))
#define MAX_int32_T                    ((int32_T)(2147483647))
#define MIN_int32_T                    ((int32_T)(-2147483647-1))
#define MAX_uint32_T                   ((uint32_T)(0xFFFFFFFFU))
#define MIN_uint32_T                   ((uint32_T)(0))

/* Logical type definitions */
#if !defined(__cplusplus) && !defined(__true_false_are_keywords)
# ifndef false
# define false (0)
# endif
# ifndef true
# define true (1)
# endif
#endif
```

```
#ifndef TRUE
# define TRUE (1)
#endif
#ifndef FALSE
# define FALSE (0)
#endif

/* States of an enabled subsystem */
typedef enum {
  SUBSYS_DISABLED = 0,
  SUBSYS_ENABLED = 2,
  SUBSYS_BECOMING_DISABLED = 4,
  SUBSYS_BECOMING_ENABLED = 8,
  SUBSYS_TRIGGERED = 16
  } CondStates;

/* Enumeration of built-in data types */
typedef enum {
  SS_DOUBLE = 0,                        /* real_T    */
  SS_SINGLE = 1,                        /* real32_T  */
  SS_INT8 = 2,                          /* int8_T    */
  SS_UINT8 = 3,                         /* uint8_T   */
  SS_INT16 = 4,                         /* int16_T   */
  SS_UINT16 = 5,                        /* uint16_T  */
  SS_INT32 = 6,                         /* int32_T   */
  SS_UINT32 = 7,                        /* uint32_T  */
  SS_BOOLEAN = 8                        /* boolean_T */
  } BuiltInDTypeId;

#define SS_NUM_BUILT_IN_DTYPE           ((int)SS_BOOLEAN+1)
```

M.9. A pelda2.mdl modellhez generált forrásfájlok

A pelda2.c

```
#include "pelda2.h"
#include "pelda2_private.h"

/* Block signals (auto storage) */
BlockIO_pelda2 pelda2_B;

/* Real-time model */
RT_MODEL_pelda2 pelda2_M_;
RT_MODEL_pelda2 *pelda2_M = &pelda2_M_;

int8_T div_s8s32_floor(int32_T numerator, int32_T denominator)
{
  int8_T quotient;
  uint32_T absNumerator;
  uint32_T absDenominator;
  uint32_T tempAbsQuotient;
  uint32_T quotientNeedsNegation;
  if(denominator == 0) {
    quotient = numerator >= 0 ? MAX_int8_T : MIN_int8_T;
    /* divide by zero handler */
  } else {
    absNumerator = (uint32_T)(numerator >= 0 ? numerator : -numerator);
    absDenominator = (uint32_T)(denominator >= 0 ? denominator : -denominator);
    quotientNeedsNegation = (numerator < 0 != denominator < 0);
    tempAbsQuotient = (uint32_T)(absNumerator / absDenominator);
    if(quotientNeedsNegation) {
      absNumerator %= absDenominator;
      if(absNumerator > 0) {
        tempAbsQuotient++;
      }
    }
   return quotientNeedsNegation ? (int8_T)-(int32_T)tempAbsQuotient :
    (int8_T)tempAbsQuotient;
  }
  return quotient;
}

/* Model step function */
void pelda2_step(void)
{

  /* local block i/o variables */

  int8_T rtb_Add;

  /* S-Function "mitmot_thermo_wrapper" Block: <Root>/Thermometer */
  getTemperature(&pelda2_B.Thermometer_o1,&pelda2_B.Thermometer_o2,0);

  /* Product: '<S2>/Divide' incorporates:
   *  Constant: '<S2>/Constant3'
   */
  pelda2_B.Divide = div_s8s32_floor((int32_T)pelda2_B.Thermometer_o1,
   (int32_T)pelda2_P.Constant3_Value);

  /* Sum: '<S2>/Add1' incorporates:
   *  Product: '<S2>/Divide1'
   *  Constant: '<S2>/Constant3'
   *  Constant: '<S2>/Constant4'
```

```
   */
  pelda2_B.Add1 = (int8_T)((int8_T)(pelda2_B.Thermometer_o1 -
    (int8_T)(pelda2_B.Divide * pelda2_P.Constant3_Value)) +
    pelda2_P.Constant4_Value);

  /* S-Function "mitmot_7segment_wrapper" Block: <Root>/7-segment display */

dyp3(int2SegNumber(pelda2_B.Divide),int2SegNumber(pelda2_B.Add1),int2SegNumber(peld
a2_B.Thermometer_o2));

  /* S-Function Block: <Root>/Switch1 (mitmot_switch) */
  pelda2_B.Switch1 = getSwitch(pelda2_P.Switch1_P1);

  /* S-Function Block: <Root>/Switch2 (mitmot_switch) */
  pelda2_B.Switch2 = getSwitch(pelda2_P.Switch2_P1);

  /* S-Function Block: <Root>/Switch3 (mitmot_switch) */
  pelda2_B.Switch3 = getSwitch(pelda2_P.Switch3_P1);

  /* S-Function Block: <Root>/Switch4 (mitmot_switch) */
  pelda2_B.Switch4 = getSwitch(pelda2_P.Switch4_P1);

  /* Sum: '<S1>/Add' */
  rtb_Add = (int8_T)((int8_T)((int8_T)(pelda2_B.Switch1 + pelda2_B.Switch2) +
    pelda2_B.Switch3) + pelda2_B.Switch4);

  /* Switch: '<S1>/Switch5' incorporates:
   */
  if((int16_T)rtb_Add == (int16_T)pelda2_P.Constant_Value_o) {
    pelda2_B.Switch5 = pelda2_P.Constant_Value;
  } else {
    pelda2_B.Switch5 = pelda2_P.Constant1_Value;
  }

  /* S-Function Block: <Root>/LED1 (mitmot_led) */
  led(pelda2_P.LED1_P1,pelda2_B.Switch5);

  /* Switch: '<S1>/Switch6' incorporates:
   */
  if((int16_T)rtb_Add == (int16_T)pelda2_P.Constant_Value_c) {
    pelda2_B.Switch6 = pelda2_P.Constant_Value;
  } else {
    pelda2_B.Switch6 = pelda2_P.Constant1_Value;
  }

  /* S-Function Block: <Root>/LED2 (mitmot_led) */
  led(pelda2_P.LED2_P1,pelda2_B.Switch6);

  /* Switch: '<S1>/Switch7' incorporates:
   */
  if((int16_T)rtb_Add == (int16_T)pelda2_P.Constant_Value_n) {
    pelda2_B.Switch7 = pelda2_P.Constant_Value;
  } else {
    pelda2_B.Switch7 = pelda2_P.Constant1_Value;
  }

  /* S-Function Block: <Root>/LED3 (mitmot_led) */
  led(pelda2_P.LED3_P1,pelda2_B.Switch7);

  /* Switch: '<S1>/Switch8' incorporates:
   */
  if((int16_T)rtb_Add == (int16_T)pelda2_P.Constant_Value_h) {
    pelda2_B.Switch8 = pelda2_P.Constant_Value;
  } else {
    pelda2_B.Switch8 = pelda2_P.Constant1_Value;
```

```
  }

  /* S-Function Block: <Root>/LED4 (mitmot_led) */
  led(pelda2_P.LED4_P1,pelda2_B.Switch8);

  /* (no update code required) */
}

/* Model initialize function */
void pelda2_initialize(boolean_T firstTime)
{

  if (firstTime) {
    /* registration code */

    /* initialize error status */
    rtmSetErrorStatus(pelda2_M, (const char_T *)0);

    /* block I/O */
    (void)memset(((void *) &pelda2_B), 0, sizeof(BlockIO_pelda2));

    /* S-Function Block: <Root>/Thermometer */
    I2C_PortInit();

    /* S-Function Block: <Root>/7-segment display */
    SPI_MasterInit();

    Dyp_PortInit();

    /* S-Function Block: <Root>/USART */
    USART_Init();

    puts("USART OK!\r\n");
  }
}
```