BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MÉRÉSTECHNIKA ÉS INFORMÁCIÓS RENDSZEREK TANSZÉK

DIPLOMATERV FELADAT

Fábián Laura

szigorló műszaki informatikus hallgató részére
(nappali tagozat műszaki informatikai szak)

Automatikus kódgenerálás MITMÓT ARM platformra Matlab-Simulink környezet segítségével
(a feladat szövege a mellékletben)

A tervfeladatot összeállította és a tervfeladat tanszéki konzulense:

dr. Márkus János
egy. adjunktus

A záróvizsga tárgyai:

Beágyazott információs rendszerek
Távközlő hálózatok
Peer-to-peer hálózatok

A tervfeladat kiadásának napja:
A tervfeladat beadásának határideje:

dr. Görgényi András
adjunktus, diplomaterv felelős

dr. Péceli Gábor
egyetemi tanár, tanszékvezető

A tervet bevette:
A terv beadásának dátuma:
A terv bírálója:

Melléklet

Automatikus kódgenerálás MITMÓT ARM platformra Matlab-Simulink környezet segítségével

dr. Márkus János
egy. adjunktus

Nyilatkozat

Alulírott *Fábián Laura*, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

_____

Fábián Laura
hallgató

Tartalomjegyzék

Kivonat **VI**

Abstract **VII**

Előszó **1**

1. Rendszerfejlesztés az iparban **3**
1.1. Az ipari fejlesztési folyamat . . . 3
1.2. A V-modell . . . 4
1.3. Automatikus kódgenerálás . . . 6
1.4. Modell-alapú tervezés . . . 8

2. Automatikus kódgenerálás beágyazott környezetben **11**
2.1. A beágyazott rendszer . . . 11
2.2. MBD a szabályozástechnika és jelfeldolgozás területén . . . 12
2.3. Modellezési stílus irányelvek . . . 14
2.3.1. MISRA-C . . . 14
2.3.2. MAAB . . . 16
2.3.3. DO-178B . . . 19
2.4. Néhány létező megvalósítás bemutatása . . . 22
2.4.1. SCADE . . . 22
2.4.2. LabVIEW . . . 24
2.4.3. TargetLink . . . 26
2.4.4. ASCET . . . 27

3. A használt fejlesztői környezet bemutatása **29**
3.1. A MATLAB-Simulink programcsalád . . . 29
3.1.1. Simulink . . . 31
3.1.2. Real-Time Workshop . . . 32
3.1.3. Real-Time Workshop Embedded Coder . . . 33
3.1.4. Kódgenerálás Simulink modellhez . . . 34

3.2. A MITMÓT . . . 35
3.3. eCOS . . . 37

4. Automatikus kódgenerálás MATLAB-Simulink eszközökkel 38
4.1. S-függvények . . . 38
4.1.1. A Simulink szimuláció menete . . . 39
4.1.2. C-MEX S-függvény Callback eljárások . . . 40
4.2. A TLC nyelv . . . 44
4.2.1. A modell.rtw fájl . . . 47
4.2.2. A TLC és az S-függvények összekapcsolása . . . 47
4.2.3. A TLC nyelv szintaktikája . . . 49
4.2.4. Blokk target TLC függvények . . . 51
4.2.5. System target fájlok . . . 53

5. A MITMÓT ARM target 59
5.1. A MITMÓT ARM system target fájl . . . 59
5.2. Segédfüggvények a kódgeneráláshoz . . . 61
5.2.1. mm_arm_template.tlc . . . 61
5.2.2. mm_arm_main_common.tlc . . . 62
5.2.3. mm_arm_ertmain.tlc . . . 63
5.3. A MITMÓT blokk-könyvtár . . . 65
5.3.1. A MITMÓT blokk-könyvtár S-függvényei . . . 65
5.3.2. A MITMÓT blokk-könyvtár TLC kódjai . . . 74
5.4. Egy kódgenerálási példa bemutatása . . . 76

6. Összefoglalás 82
6.1. Eredmények . . . 82
6.2. Továbbfejlesztési lehetőségek . . . 83

Irodalomjegyzék 84

Függelék 89
F.1. A CD-melléklet tartalma . . . 90
F.1.1. MITMÓT ARM target könyvtár . . . 90
F.1.2. MITMÓT ARM szoftverkörnyezet . . . 90

Ábrák jegyzéke 91

Rövidítések 93

Kivonat

Ahogy a valósidejű rendszerek egyre bonyolultabbakká válnak, úgy csökken az idő, ami megtervezésükhöz és fejlesztésükhöz rendelkezésre áll, hiszen a piacra kerülési idő csökkentése egyre fontosabb szempont. Emiatt már régóta az érdeklődés középpontjában állnak azok a módszerek és megoldások, amelyekkel a fejlesztési folyamat ideje hatékonyan lerövidíthető.

A fejlesztési idő csökkentését az emberi beavatkozást igénylő részek redukálásával próbálják elérni, azáltal, hogy a rendszerfejlesztési folyamat lépéseinek egyre bővülő halmaza válik automatizálttá. Ezen automatizált folyamatok körében az iparban viszonylag új szereplő az automatikus kódgenerálás, noha múltja már pár évtizedre is visszanyúlik.

Az automatikus kódgenerálás többek között a Modell-alapú tervezési séma (Model-Based Design) térhódításával magyarázható. Modell-alapú tervezésnél már a tervezési folyamat legelején rendelkezésre áll a modell grafikus formában, amely grafikus modell egyben a rendszerspecifikáció is. Így már a fejlesztési folyamat korai szakaszától kezdve lehetőség van a rendszer tesztelésére – ami jelentősen felgyorsítja magát a fejlesztési folyamatot, hiszen minél később derül ki egy hiba , annál időigényesebb és költségesebb a javítása. A grafikus modellből – átugorva a kézi kódolás hosszas és bosszantó hibákat bevezető fázisát – automatikus kódgenerálással a rendszer kódja azonnal megkapható, így a modell bármilyen változtatása rögtön követhető annak kódjában is.

A diplomatervben egy Modell-alapú fejlesztési környezet tervezési folyamata kerül bemutatásra, amely fejlesztési környezet célja, hogy a BME MIT tanszéken fejlesztett MIT-MÓT szenzorkártya perifériáinak működését MATLAB-Simulink környezetben szimulálni lehessen, valamint az ott elkészült alkalmazást automatikus kódgenerálással azonnal át lehessen ültetni a gyakorlatba.

Abstract

As systems become more and more complex and development time is getting shorter due to the pressure of time-to-market, engineers have to search for efficient development techniques, in which errors and design flaws are catched early, reusability of modules is easy and the time of moving from prototypes to final hardware is short. Model-Based Design (MBD) is a technique developed by The Mathworks, Inc., which lets engineers use state-of-the-art software development methods for rapid prototyping.

This master thesis discusses the design procedure of a model-based development environment for simulation and automatic code generation using the Mathworks products Simulink and Real-Time Workshop Embedded Coder. The main goal of the work was to create an environment for seamless integration of simulation and real-time operation. The simulation target is the MITMOT, a modular platform consisting of a 32-bit ARM-based microcontroller and some custom I/O peripherial developed at the Dept. of Measurement and Information Systems of the Budapest University of Technology and Economics. The development consisted two main parts: the first was to create models of the different I/O units in the Simulink environment using S-functions, while the second was to write the TLC (target language compiler) files which are responsible for the C-code generation.

Előszó

A napjainkban gyártott mikrochipek 98 százaléka beágyazott rendszerekbe kerül. A hardver-ipar egyre általánosabb elemek tömeggyártására állt át, ami miatt a különböző alkalmazásokhoz kapcsolódó speciális feladatok megvalósítása mindinkább szoftveres úton történik.

Emellett, a beágyazott rendszerek fejlesztése komoly kihívássá vált számos alkalmazás esetén, egyrészt a megoldandó feladat bonyolultsága, másrészt a megoldáshoz rendelkezésre álló idő rövidsége miatt. A hagyományos kézi kódfejlesztés, implementálás és verifikáció mind az idő, mind a komplexitás kapcsán elégtelennek bizonyult. Ebből kifolyólag erőteljes elmozdulás következett be a magasabbszintű Modell-alapú tervezési eszközök és módszerek (Model-Based Design, MBD), valamint a hozzájuk kapcsolódó automatikus kódgenerálás irányába.

A Modell-alapú tervezési módszer lényege, hogy egy magas szintű, általában grafikus leíró nyelvet használnak a rendszertervezéshez. A tervezés folyamán a rendszer grafikus modellje készül el, amely egyben a specifikációt is magában foglalja. A mai MBD eszközök már lehetővé teszik egyetlen referencia terv használatát a tervezés minden fázisában.

A Modell-alapú tervezési módszer fontos része az automatikus kódgenerálás, hiszen segítségével a megtervezett rendszer grafikus modelljéből egy egérkattintással megkaphatjuk a rendszert megvalósító kódot. Ennek az eljárásnak több előnye is van: egyrészt nem kell a kézi kódolással időt vesztegetni, másrészt a kézi kódolás által bevezetett hibák automatikus kódgenerálás esetén nem fordulhatnak elő. Ha még azt is nézzük, hogy egyes kódgenerátor programok biztonságkritikus kódra vonatkozó szabványoknak is eleget tesznek, a használatukból adódó előny nyilvánvaló.

Az automatikus kódgenerálási technikák terén bekövetkezett hatalmas fejlődés és a hardver-platform gyártók által a felhasználók rendelkezésére bocsátott testreszabható blokkok egy integrált és optimalizált megoldást biztosítanak a tervvalidációtól kezdve a szimuláción és implemetáción át a verifikációig, mindezt egy egységes Modell-alapú tervezési környezetben [1, 2].

A diplomában egy MATLAB-Simulink szimulációs és automatikus kódgenerálási környezet fejlesztése kerül bemutatásra, MITMÓT ARM kártyára.

Az első fejezetben rövid áttekintést kapunk az ipar által követett fejlesztési sémáról, a V-modellről és a Modell-alapú tervezési technikáról.

A második fejezetben ezután áttérünk a beágyazott rendszerek által megkövetelt speciális feltételekre, a biztonságkritikus beágyazott rendszerek szoftveréhez kapcsolódó szabványok ismertetésére, majd a piacon jelenleg kapható automatikus kódgenerátorok bemutatására.

A harmadik fejezetben a MATLAB-Simulink környezet automatikus kódgeneráláshoz kapcsolódó részeinek ismertetése következik, az S-függvények és a TLC nyelv áttekintésén keresztül.

A negyedik fejezet a MITMÓT ARM kártyára fejlesztett S-függvények és TLC kódok részletes elemzéséből áll, majd működésük egy konkrét példán keresztül is bemutatásra kerül.

Az utolsó fejezet a diplomamunka összefoglalása, valamint kitekintés a lehetséges továbbfejlesztési lehetőségek irányába.

1. fejezet

Rendszerfejlesztés az iparban

1.1. Az ipari fejlesztési folyamat

Egy ipari fejlesztésnél a két fő szempont a fejlesztési idő rövidsége és a költséghatékonyság. Az algoritmustervezés és -megvalósítás a mai komplex alkalmazások esetén jóval több tesztelési lépésen kell átmenjen, mint akár 20-30 éve, hiszen a biztonságkritikusság az egyik fő szempont.

Az ipari rendszer alatt most nem csak szoftvert értünk, hanem a hozzá tartozó hardver alapot is, hiszen egy célspecifikus alkalmazás esetén gyakran nem csak a szoftvert fejlesztik, hanem a hardvert és a szoftvert együtt.

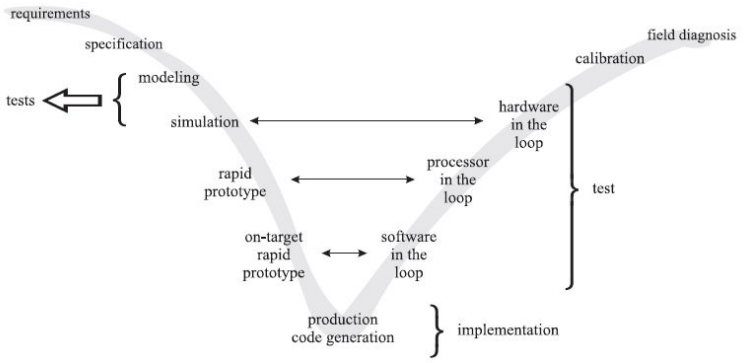
Egy ipari fejlesztési folyamat látható az 1.1 ábrán.

Legelőször a modell elméleti terve készül el, a megvalósítandó funkciók, a célkörnyezet és a követelmények meghatározásával. Mivel az elméleti tervezés témája a dolgozat keretein túlmutat, ezen fázis részletes elemzését nem tárgyaljuk.

Az elméleti terv első próbája általában egy megfelelő szoftveres környezetben végzett számítógépes szimuláció, majd aztán a szimulációs eredmények alapján a modellen további finomítások és változtatások következnek.

Amikor a számítógépes modell már a felállított követelményeknek megfelelően működik, egy előzetes megvalósíthatóság-ellenőrzés következik, melynek terméke a *rapid prototype*. Ez a „gyors prototípus” már tartalmazza a nem funkcionális, így a nem is feltétlenül modellezett karakterisztikák egy részét. Ilyen karakterisztikák például a válaszidő, jitter, késleltetések – amelyek miatt akár az eredeti modell változtatására is szükség lehet.

A *rapid prototype* tesztelése után a következő lépés a célhardveren történő *rapid prototyping*, azaz, amikor a prototípust a kitűzötteknek megfelelő hardverkörnyezetben tesztelik. Ez a lépés olyan karakterisztikákat validál, mint a használt fordítóprogram, a fixpontos számábrázolás hatása és együttműködés más rendszerekkel [3].



1.1. ábra. Egy ipari fejlesztés állomásai [3]

Végül, mikor mindezen prototípus tesztelése azt mutatja, hogy sikerült eljutni egy megvalósítható és működő megoldáshoz, következik a végső, az alkalmazás tényleges kódjának gyártása. Az itt készülő kódnak megfelelő mértékben optimalizáltnak kell lennie – alaposan ellenőrzött, tesztelt és validált kell legyen.

Ahogy az eddig leírtakból is látszik, a termékfejlesztési folyamat alatt a kód számos ellenőrzési-tesztelési-validációs állomáson megy keresztül, melyek során az alap kódot fokozatosan finomítják (*fine tuning*).

Egy-egy finomítási szakasz több konfiguráción is végrehajtásra kerül, hiszen az alapos helyzetfelméréshez software-in-the-loop, processor-in-the-loop, és hardware-in-the-loop konfigurációt is ajánlott tesztelni.

A fejlesztési folyamat ezen séma alapján való haladása azonban magával vonja sok különböző kód használatát. Rapid prototyping, on-target rapid prototyping, processor-in-the-loop szimuláció: mindegyik esetben más-más kódra lehet szükség. Ilyen sokféle kód megírása kézzel rengeteg időbe telne, amire egy mai fejlesztési projektnél már nincs lehetőség. Mi jelenthet mégis megoldást?

Sok, ám eltérő kód egy modellből való generálásához hatásos segítség az automatikus kódgenerálás, melynek jelentősége az évek során folyamatosan nő, mára már szinte nélkülözhetetlen minden nagyobb léptékű ipari fejlesztésnél. Automatikus kódgenerálással megkaphatjuk a tervezett rendszer modelljéből a kitűzött működést megvalósító kódot mindenféle kézi kódolási lépés nélkül [3, 4].

1.2. A V-modell

Kezdetben egy szoftverfejlesztés főleg kis, kompakt programokra irányult, amelyekhez akár egy ember is elég volt. Mostanra nagyobb léptékű, összetettebb programok kellenek, melyeket már nem lát át egy ember, így a fejlesztői munka csoportokra bontva történik, az egyes fejlesztői csoportok külön dolgoznak a saját részükön. Mindehhez az új fejlesz-

tési sémához a szoftverfejlesztés folyamatát is adaptálni kellett, ekkor született elsőként a Vízesés-, majd később a Spirál- és a V- szoftverfejlesztési modell. Minden modell sajátsága, hogy a fejlesztési folyamatot élesen elhatárolható lépésekre bontja.

Részletesen a V-modell kerül itt bemutatásra, hiszen az iparban jelenleg ez a leginkább elterjedt.

A V-modell előnyei:

- Jól elhatárolt lépésekre bontja a fejlesztési folyamatot és egy logikai sorrendet javasol ezen lépések végrehajtására, az egyes lépések közti logikai kapcsolatot is pontosan meghatározza,
- A tesztelési dokumentáció megírása a lehető legkorábbi szakaszban történik, azaz amint kész egy komplett részfeladat, azonnal elkészül a hozzá kapcsolódó tesztelési leírás is,
- Azonos súllyal kezeli a fejlesztési és tesztelési fázist,
- Egyszerű és könnyen átlátható képet ad a szoftverfejlesztés folyamatáról.

A V-modell jellegzetes alakja azt jelképezi, hogy az egyes fejlesztési lépésekhez mindig tartozik egy tesztelési rész is, ezek a V két ágán az egymással egy szinten levő elemek. A V-modell erőssége és egyben újítása is ez a felépítés, hiszen így minden tesztelési szakasz közvetlenül a hozzá kapcsolódó fejlesztésből származik, így lehetőség van az egyes fejlesztési szakaszok külön-külön való ellenőrzésére [5].

A V-modell egy tipikus szakaszfelbontása és az egyes szakaszok jelentése (1.2. ábra):

Követelmény-analízis A fejlesztendő rendszerhez kapcsolódó követelmények összegyűjtése, a megvalósítandó funkciók és esetleges korlátozások felmérése. A fázis célja a *Követelmény specifikációs dokumentáció* összeállítása, amely a következő fázis kiindulási alapja.

Rendszertervezés Ebben a szakaszban készül el a rendszer felépítésének pontos terve, azaz, hogy milyen blokkokból és komponensekből fog állni, az egyes részek között milyen interfészek és interakciók lesznek megvalósítva. Meghatározásra kerül a szükséges hardver, továbbá a szoftver dekompozíciója is megtörténik. A szakasz végére elkészül a *Rendszerfelépítés dokumentáció*.

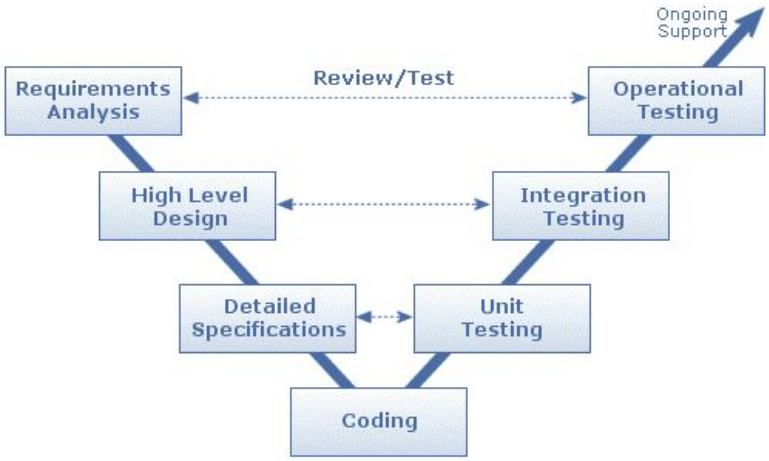
Szoftvertervezés Itt az előző szakaszban meghatározott rendszerarchitektúra alapján kijelölt szoftver-blokkok finomítása történik, a kijelölt modulokat kisebb, immár kódolási egységekre bontják fel. Meghatározásra kerülnek az egyes modulok interfészei és interakciói, a funkcionalitásukkal együtt. Definiálják a szoftver viselkedését az összes lényeges rendszerállapotban, mint például indulás, leállítás, hibaesetek, diagnosztika. A végtermék a *Szoftvertervezési dokumentáció*.

Kódolás A *Szoftvertervezési dokumentáció* alapján ezen részben történik a kijelölt modulok átültetése kóddá. A rendszer először kis részenként (unitonként), kerül kódolásra. Egy unit egy különálló funkciót valósít meg, ezen unitok egyesítésével fog a teljes rendszer kialakulni.

Szoftververifikáció Itt az elkészült unitok tesztelése történik, funkciók és interakciók terén, majd végül a unitokat egyesítve összeáll a teljes rendszer.

Rendszerverifikáció Ezen fázisban már a kijelölt célkörnyezetben történik a teljes rendszer tesztelése, a követelmények teljesítésének ellenőrzése.

Rendszer jóváhagyás A termék a megrendelőhöz kerül kipróbálásra, kiderül, hogy a megvalósítás mennyiben egyezik a vásárló által támasztott igényekkel, felmerülhet igény a változtatásra is.



1.2. ábra. A V-modell [7]

Ugyanakkor a V-modellel kapcsolatban egyre több kétség merül fel [7, 6]:

- Túl egyszerű ahhoz, hogy pontosan leírja a szoftverfejlesztési folyamatot,
- Nem rugalmas, a változásokhoz nem tud alkalmazkodni,
- Nem megfelelő tesztelési metodológia.

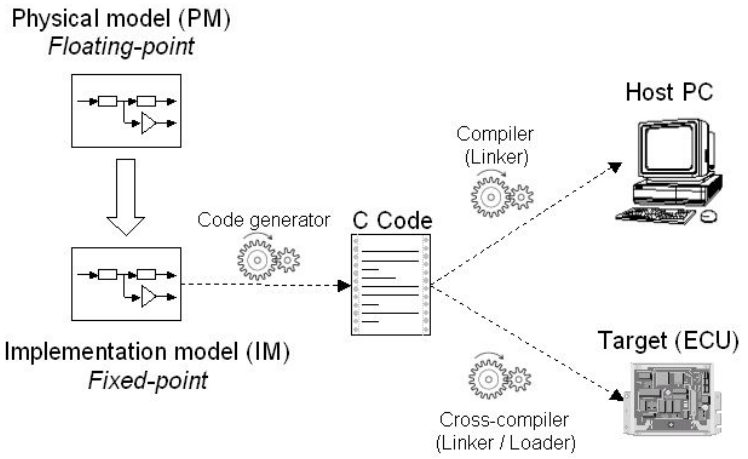
1.3. Automatikus kódgenerálás

A fejlesztő-tervező mérnöki munka során egyre fontosabb szerepet kapnak a rapid prototyping, processor-in-the-loop és hardware-in-the-loop szimulációk. Az ipar folyamatosan

és gyorsan változó igényei megkövetelik a fejlesztési és tesztelési idők csökkenését, ami természetesen nem mehet a minőség rovására. Ehhez azonban vagy rengeteg új emberre, vagy a fejlesztési folyamat minél jelentősebb automatizálására van szükség.

A kódolás napjainkra az ún. magasabb szintű nyelvek irányába tolódott át, az assembly kódolás helyét lassanként teljesen átveszik a C, a Java és egyéb hasonló nyelvek. Mára ez az eltolódás a grafikus leíró nyelvek irányába mutat (UML, Petri hálók, Stateflow, Simulink). Természetesen erőforrás-kritikus rendszerekben továbbra is a hatékony C- és assembly nyelven írt programokra van szükség. Erre jelent megoldást az automatikus kódgenerálás, ahol egy magasabb rendű nyelven leírt modellt a kódgenerátor a kívánt nyelvre lefordít [4] (1.3. ábra).

Ez a módszer egyrészt költséghatékony, másrészt versenyképes, hiszen egy tervezési feladathoz elég egyetlen magas szintű modellt megalkotni, és ezen modell alapján automatikus kódgenerálás segítségével különböző platformokon alkalmazható megoldásokat kaphatunk.



1.3. ábra. Az automatikus kódgenerálás alapelve (autocode tool-chain) [4]

Az automatikus kódgenerálással kapcsolatban azonban felmerülnek új problémák: például, mi biztosítja, hogy a kódgenerálási folyamat nem vezet be hibákat? Ez a dilemma ismerős lehet, hiszen a magas szintű nyelvek elterjedésekor komoly aggodalmat okozott a fordító programok megjelenése: a C kódból kapott assembly program működése vajon a tervezett lesz? Ez az aggodalom az évek során megalapozatlannak bizonyult, így napjainkban egyre kevesebb kézzel írtott assembly kód születik.

De az automatikus kódgenerátorok esetén a várakozás nem elég, formális bizonyító eljárásokra van szükség, amelyek segítségével bizton lehet állítani, hogy a modell és a belőle generált kód működése megegyezik. Ez különösen akkor létfontosságú, ha a termék tesztelése a modellezési fázisban történik. A témakörrel a 2. fejezet foglalkozik részletesen.

A másik fő probléma automatikusan generált kódok esetén az optimalitás kérdése. A kézzel írott kódhoz képest egy modellből generált kód jóval lassabb lehet, memóriahasználata is kevésbé optimális. Azonban a hibamentes kód és a töredékére csökkent imple-

mentációs idő miatt az automatikus kódgenerálás mégis egyre szélesebb körben használt megoldássá válik [8].

1.4. Modell-alapú tervezés

Egy hagyományos fejlesztési projekt három teljesen különálló részre volt bontható: a követelmények meghatározása, prototípus létrehozása, és végül a prototípus tesztelése. Modell-alapú tervezésnél (Model-based Design, MBD) szintén megvan ez a három fázis, de a köztük levő határ elmosódik. Ahogy a tervezési feladatok egyre nagyobb léptékűvé és bonyolultabbakká válnak, szükségessé válik először a tervezendő rendszer magas absztrakciós szinten való leírása.

„A számítógép-alapú modellezésnél a fejlesztési folyamat középpontjában a tervezendő rendszer modellje áll, kezdve a rendszerkövetelményektől végig a tervezésen, megvalósításon és tesztelésen át. A modell egy futtatható specifikáció, amit a tervezés során folyamatosan finomítanak, a modell működése minden fázisban szimulációval ellenőrizhető" [9].

A számítógépes modellezés napjainkra a fejlesztési folyamatban nélkülözhetetlenné vált, hiszen a tervezett rendszerek egyre összetettebbek: már nem elég a kész rendszeren történő próbálgatáson alapuló hibakeresés, helyette előnyösebb a fejlesztés korai fázisában szűrni a hibákat, amely feladatra a rendszer számítógépes modellje kitűnően alkalmas.

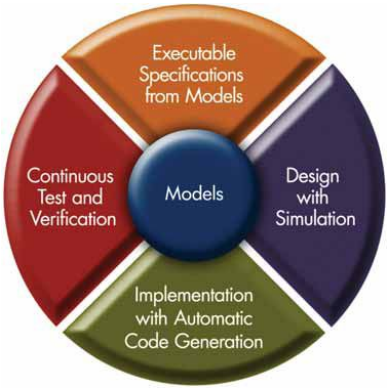
A Modell-alapú tervezéssel lehetővé válik a termék prototípusának már korai fázistól kezdődő tesztelése és verifikációja szoftveres szimulációval. Az MBD módszer használatával szükségtelenné válik a termékfejlesztési folyamat egyes lépései során különböző fizikai prototípusok létrehozására, hiszen az egész fejlesztés végigkövethető egyetlen modellen, így pénz és fejlesztési idő takarítható meg, és a piacra kerülésig tartó idő is jelentősen lecsökken [9]. Az MBD séma támogatja a korai és folyamatos tesztelést és verifikációt, ami igen előnyös tulajdonság, hiszen minél koraibb fejlesztési fázisban derül ki egy hiba, annál költséghatékonyabb a javítása.

A fejlesztés kezdetektől végig egységes kezelése régebben kivitelezhetetlen volt – a szoftveres rendszerfejlesztés és szimulációs fázis után az implementációs fázisban a mérnökök kezdhették az egész munkát elölről, mert a tervező eszközök és a valós implementációs platform között semmiféle kapcsolat nem volt. Az utóbbi években ezen tervezési szakadék áthidalására jelentős erőfeszítés történt, így valósulhatott meg mára az MBD elterjedése [1].

A Modell-alapú tervezéssel lehetővé válik:

- Egységes fejlesztői környezet használata projekt-teamek között,
- A fejlesztés összekapcsolása a követelményekkel,
- A fejlesztés és tesztelés összekapcsolása, folyamatos hibakeresés és javítás,

- Algoritmus finomítás többszintű szimulációval,
- Automatikus kódgenerálás,
- Tesztsorozat fejlesztése és újrafelhasználása,
- Automatikus dokumentáció generálás, valamint
- Elemek újra felhasználása a rendszer más platformon való alkalmazása esetén.



1.4. ábra. A Modell-alapú tervezés elemei [10]

MBD megközelítést alkalmazva a fejlesztés során először *futtatható specifikáció* rendszermodellek létrehozása történik, hozzájuk kapcsolódva *teszt bemenet – várt kimenet* párok (1.4. ábra). Az egyes scenáriók szimulációs futtatása lehetőséget ad a speciális követelmények, fejlesztések és tesztek validálására. Ezen teszteket futtatva ráadásul nem csak a hibákat lehet felfedezni még korai szakaszban, de egyben a későbbi fizikai rendszer teszteléséhez is hasznos információk gyűlnek össze.

Jon Friedmann, a MathWorks autóipari marketing menedzsere, így foglalta össze a MBD teszteléshez kapcsolódó előnyeit: „Mára olyan új eszközök állnak rendelkezésre, amelyek segítségével jelentősen lecsökkenthető a rendszer teljes verifikációjához szükséges tesztek száma, a redundás tesztek szűrésével" [9].

Az MBD tervezési fázisában bevezetett új analizációs technikák a hagyományos szimulációs eljárásokat messze felülmúlják a rendszerkövetelmények és megvalósítás közti verifikáció terén. A különböző elemzési módszerek, a fejlesztési környezetbe ágyazva, automatikus debug és modell-ellenőrzési funkciókat biztosítanak, melyek fényt deríthetnek alapvető tervezési hibákra.

Az MBD modell fő elemei [10]:

- Végrehajtható specifikációk: azonnali visszajelzés az adott specifikáció viselkedéséről (ellentétben az írott specifikációval, amely később félreértésekre adhat okot),
- Tervezés szimulációval: a tervezési tér szűkítésével gyorsabbá teszi a fejlesztési folyamatot,

- Implementáció kódgenerálással: kiiktatja a hosszadalmas és új hibákat bevezető kódolási fázist,
- Folyamatos tesztelés és verifikáció: már a tervezés korai szakaszától kezdve lehet tesztelni, olyan belső változók megfigyelése is lehetséges, amelyek tesztje a hardver prototípusnál már nehezen kivitelezhető lenne.

2. fejezet

Automatikus kódgenerálás beágyazott környezetben

2.1. A beágyazott rendszer

A beágyazott rendszer egy olyan speciális célú rendszer, ahol a rendszert vezérlő számítógép be van építve abba az eszközbe/rendszerbe vagy dedikált ahhoz az eszközhöz/rendszerhez, amit vezérel. Az általános célú számítógépekkel ellentétben, mint például egy PC, a beágyazott rendszer csak néhány, speciális követelményű, előre definiált taszkot hajt végre. Mivel a rendszer speciális feladatokat hajt végre, a tervezésnél erre optimalizálva a termékméret és -költség jelentősen csökkenthető. A beágyazott rendszereket gyakran tömeggyártással állítják elő, így a költséghatékonyság kiemelkedő.

A rendszertervezés az a folyamat, amely során a követelményekből egy modellt származtatnak, mely modellből aztán a rendszer többé-kevésbé automatizált módon generálható. Például a szoftvertervezés során egy fordítható program a cél, hardvertervezésnél egy olyan hardverleírás, amelyből a kívánt áramkör szintetizálható [11, 12].

Beágyazott rendszer esetén három fontos tényező van: hardver, szoftver, és a környezet. Ebben nagyjából az összes számítógépes rendszerhez hasonlít. Azonban van egy fontos különbség beágyazott és általános számítógépes rendszerek között: a beágyazott rendszer fizikai kényszerekhez kapcsolódó számításokat végez, így a számítástechnika fejlődése során kialakult általános felbontás, azaz a számítást végző (szoftver) és fizikai (hardver és környezet) jellemzők határozott különválasztása nem működik beágyazott rendszerek esetén; helyette a beágyazott rendszer tervezése olyan átfogó szemléletet követel, ami következetesen integrálja az alapvető hardver és szoftver tervezési paradigmákat és a szabályozástechnika elméletét [11].

2.2. MBD a szabályozástechnika és jelfeldolgozás területén

Szakértők szerint a jó szoftver tervezéséhez mindössze négy alaptételt kell betartani: törekedni kell az egyértelműségre, moduláris alapon kell gondolkodni, automatizálni kell, ahol csak lehet, és tesztelni kell már korán és amilyen gyakran csak lehet. Ezek az alapelvek már 30 éve aktuálisak, azonban napjainkig csak és kizárólag a szoftverfejlesztéshez kapcsolták őket. Ezt a korlátot döntötte le a Modell-alapú tervezés megjelenése.

A modellalkotás és szimuláció már régóta kulcseleme az autó- és repülőgép-iparnak. A légirendszerek komplexitásának növekedtével egyre inkább felmerült az igény olyan megoldások iránt, amelyek segítségével megvalósítható lenne a célhardver és a vezérlési algoritmusok minél valósághűbb szimulációja, minél korábbi szakasztól kezdve és a fejlesztés különböző fázisaiban [13].

A grafikus modellezési technikák és az MBD alkalmazásával a tervezőmérnökök számára lehetővé vált olyan dinamikus rendszermodellek készítése, amelyek alkalmasak a jármű és a vezérlő algoritmusok együttes szimulációjára, és amely modellek a fejlesztési folyamat követelményfelmérési szakaszától kezdve az implementáción és a tesztelésen át végig használhatóak.

Az 1960-as és '70-es években az abszolút programozási standard az assembly volt. Az évek múltával, ahogy a beágyazott processzorok egyre fejlettebbekké váltak, egyre bonyolultabb és összetettebb funkcionalitást megvalósító szoftverek futtatására lettek alkalmasak. A bonyolultság növekedtével a szoftverfejlesztés elmozdult a magasabb absztrakciós szintű nyelvek irányába, a gépi kód felől a hangsúly átkerült a C és FORTRAN típusú nyelvekre. Ezen magasszintű nyelveket elsajátítva a fejlesztők már csak a magasabb absztrakciós szinten dolgoztak, az alacsonyabb szintre való fordításra fordító programok használata lett a megoldás. Ma újabb elmozdulásnak lehetünk tanúi, az absztrakciós szint (rendszer-szintű tervezés) és automatizálás (kódgenerálás) új állomásához érkeztünk, melyek segítségével a megvalósítható rendszerek bonyolultsága tovább növekedhet [14].

Ahogy növekszik az igény a nagyintegráltságú, biztonságkritikus szoftverekre, a szoftvertervezési módszertanok és eszközök is folyamatosan fejlődnek hatékonyság és minőség terén, amivel lehetővé válik a fejlesztők számára a folyamatosan változó követelményekhez való gyorsabb igazodás. A legújabb módszertanok, az *Extrém programozás* (Extreme Programming) és a *Gyors fejlesztés* (Agile Development), a szervezeten belüli egységek valamint a beszállítók és a megrendelők közti kommunikáció fejlesztésére koncentrálnak: dokumentációhalmok helyett futtatható szoftveren keresztüli kommunikáció, valamint a korai fázisban kezdődő tesztírás, amely akár még a kódírási fázis kezdetét is megelőzi. További jellegzetesség, hogy projektkezdéskor még csak egy viszonylag egyszerű terv áll készen, amelyből később bontakozhat ki az összes benne foglalt lehetőség [13, 15].

A Modell-alapú tervezés nem egy önmagában megálló eljárás, adaptálni lehet mindenféle tervezési folyamhoz, a V-modelltől kezdve a Vízesés és Spirál modelleken át. A mo-

dell alkalmazható, amint a projektkövetelmények már körvonalazódtak. A követelmények blokk-diagramm formában kellenek, mely diagram tartalmazza a rendszer dinamikus karakterisztikáját, beleértve az olyan környezeti építőelemeket, mint beavatkozók, érzékelők, mechanikai eszközök, elektronika és egyéb fizikai elemek.

A követelmények meghatározása alatt a modell a rendszer idealizált képe, és ehhez az idealizált sémához adódnak hozzá a fejlesztési folyamat során további részletek, melyekkel a modell immáron *végrehajtható specifikáció* lesz. Ez a végrehajtható specifikáció már tartalmazza az összes olyan információt, ami a szoftver vagy hardver implementáció elkészítéséhez szükséges, mint például fix-pontos és időzítési viselkedés. Mivel a modell egyben a dokumentáció is, a végső tervből egy az egyben készülhetnek a valós kísérleti szimulációk, mindenféle hibabekerülési esély nélkül és minimális erőráfordítással. Továbbá amiatt, hogy a rendszer végrehajtható specifikáció, a modellből lehet automatikusan kódot generálni a valós-idejű rapid prototyping-hoz a célplatformra. Mint ahogy a modell, a kód is bármely ponton tesztelhető és jóváhagyható. Az esetleges hibák javításához elég a modellt változtatni, és majd a már átalakított modellhez újra kódot generálni, hogy megmaradjon az egységesség a kód és a modell között.

Az újgenerációs grafikus fejlesztőeszközök – mint például a Simulink termékcsalád a The MathWorks-től – már biztosítják az egységes fejlesztőeszközök használatát az interaktív grafikus szerkesztőn át a grafikus debugger, modell-analizátor és diagnosztikai funkciókon keresztül. A testreszabható blokk-könyvtárakkal pontosan lehet tervezni, szimulálni, implementálni és tesztelni többek között a szabályozástechnikai, jelfeldolgozási és kommunikációs rendszereket. A bonyolult tervek menedzselése is egyszerűbb, hiszen lehetőség van a modell hierarchikus felbontására. Az egyes hierarchiaszintek a terv főbb összetevői, és még ezek is lebonthatók egészen a funkcionális egységekig, tetszés szerinti méretre és struktúrára. Így elérhető a többszintű modellhűség, amelyben egy modellelem bármikor helyettesíthető egy másikkal.

Ez a megközelítés korszerűsíti a nagy rendszerek tervezését, mert egymással párhuzamosan tudnak az egyes csoportok dolgozni a rendszer finomításán és optimizálásán. Azok a vállalatok, akik bevezették a Modell-alapú tervezés használatát, látványos eredményekről számolnak be: több ezernyi hibátlan működésű automatikusan generált légi-irányításban használt kódsor, 500 százalék termelékenységnövekedés, valamint az átlagos fejlesztési idő felére csökkenése. Ezek és az ehhez hasonló eredmények nem csak egyszerűen abból adódnak, hogy az MBD fejlesztési módszer felgyorsítja a fejlesztést, egyszerűsíti a kommunikációt és növeli a hatékonyságot, hanem mert elősegíti a jó tervezési gyakorlatokat a követelmény-felméréstől a tervezésen át az implementációig és a tesztelésig [13].

2.3. Modellezési stílus irányelvek

Beágyazott repülőgép- és autóvezérlő rendszerek egyre gyakrabban készülnek Modell-alapú tervezés és automatikus kódgenerálás útján. Ebben a megközelítésben a modellezés és a szimuláció blokk-diagrammok, állapotgépek és adatszótárak használatával történik. A modellekből automatikus kódgenerálással készült kód már egyből a beágyazott mikroprocesszoros környezetbe kerül [15].

A modellfelépítés, blokkválasztás, adattípusok kijelölése és a kódgenerálás beállítása mind-mind jelentékenyen befolyásolja az automatikusan generált kódot. Még a szoftverfejlesztési folyamat megkezdése előtt ezért hasznos, ha készül egy összegzés a követendő modellezési stílusról. Sok minőségbiztosítási szabvány tekinti alapfeltételnek tervezési és kódolási szabványok használatát.

A modellezési irányelvek meghatározása különösen fontos nagy és összetett projektek esetén. A szimulációs idő és a modellfrissítés percek helyett órákba telhet, ha a modell architektúra, ami a komponensek felosztását írja le, nem pontosan meghatározott. Nagy projekteknél az is fontos lehet, hogy számításba vegyék a több felhasználó és földrajzilag több helyen történő fejlesztésből adódó nehézségeket.

Az ipari tapasztalatok alapján egy projekt nagynak számít, ha a modell egy ember számára már túl nagy ahhoz, hogy teljes részletességében megismerje, több, mint 10000 blokkból áll, vagy több, mint 10 saját könyvtárat használ [15]. A modellezési stílus alapelvek bevezetése főleg ilyen nagy léptékű feladatok esetén hasznos.

Végül, modellezési stílus alapelvek használata elősegíti a minőségbiztosítást, mert pontosan meghatároz egy sémát, amit be kell tartani a szoftverfejlesztésnél, a séma betartásának ellenőrzése pedig már egyértelmű feladat.

A repülőgép- és autóiparban a nagyobb gyártók általában saját maguk által kialakított szabályrendszert használnak, a továbbiakban pár ilyen konkrét modellezési stílus irányelvgyűjtemény kerül bemutatásra [16].

2.3.1. MISRA-C

A MISRA (Motor and Industry Software Reliability Association) egy, az autógyártó és autóipari beszállító cégek által alapított egyesület. Célja, hogy támogassa az autóiparon belül a biztonságos és megbízható szoftverfejlesztést és annak üzembe helyezését autókban.

Ilyen típusú támogatás azért vált szükségessé, mert az iparban széleskörűen elterjedt a C nyelv, a specifikációjában előforduló hiányosságokkal együtt. Hiába tesz eleget minden vonatkozásban az alkalmazott C kód a szabványnak, a nyelv specifikációs hiányosságai nagy valószínűséggel kiváltó okai lesznek a teljes rendszer hibás működésének [17].

Néhány alapvető ok, amik a C programozás hibáit eredményezik:

- A nyelv értelmezése fordítótól függően eltérő lehet,

- Platformfüggőség,
- A nyelv többértelműsége,
- Végtelen, bonyolult szerkezetek,
- Gépelési hibák.

A kódból a fent említett hibaforrásokat a lehető legnagyobb mértékben kiiktatandó, a MISRA Konzorcium 1998-ban kiadott egy ajánlást 127 szabállyal, ez volt az eredeti MISRA-C. Az évek során az eredeti szabvány számos gyengeségére derült fény, ez vezetett a MISRA-C:2004 (MISRA-C 2.0) szabvány létrejöttéhez [18].

Az új szabvány 141 szabályból áll, ezek MISRA irányelvek néven is ismertek. A szabályok huszonegy tárgykörre és két kategóriára vannak felosztva. Témakör például a *Környezet* és *Futás-idejű hibák*. A két kategória pedig:

- Szükséges szabályok: teljesítésük kötelező,
- Ajánlott szabályok: betartásuk általános esetben ajánlott.

A MISRA-C:2004 121 szükséges és 20 ajánlott szabályból áll.

A Modell-alapú szoftvertervezés elterjedése a jövőben tovább fog nőni. Míg a MISRA szabályok könnyen áthághatóak nem megfelelő szoftvermodellek és komponensek használatával, addig automatikusan generált kód esetén a kézi kódolásból adódó szabálysértéseket teljes mértékben ki lehet küszöbölni.

Az implementációból adódó szabályszegéseket azonban gyakran nem lehet kivédeni, ha az implementáció célplatform-függő. A MISRA Konzorciumnak ezért külön bizottságai vannak az automatikus kódgenerálással kapcsolatban felmerülő kérdések részletekbe menő vizsgálatára.

A MISRA-C szabvány egy 100 oldal körüli, a MISRA Konzorciumtól megvásárolható dokumentum. Ebben szerepelnek a MISRA szabályok sorszámozva. Bemutatásra kerül példákon keresztül az is, hogy az egyes szabályokat hogyan kell értelmezni, és az egyes szabályok bevezetésének indoklása is szerepel a leírásban.

A MISRA szabványnak való megfelelőség bizonyításához a kódnak a szükséges szabályok mindegyikét teljesítenie kell, a javasolt szabályok használata azonban már egyéni döntéstől függ, azaz, elég csak a feladat szempontjából praktikusak alkalmazása. A minősítés számos szoftvertermékkel elvégezhető, ugyanakkor a MISRA által specifikált minősítési eljárás nincs kidolgozva [17].

A MISRA-C kötelezően alkalmazandó szabályai olyanok, mint például:

- A kódnak meg kell felelnie az ISO 9899 C szabványban leírtaknak,
- Minden objektumot és függvényt deklarálni kell az első használat előtt,

- Nem konstans függvény-mutatók használata tilos,
- A belső és külső azonosítók egyaránt maximum 31 karakterrel meghatározhatóak legyenek.

A javasolt szabályra példa:

- A nulla ellenőrzés legyen explicit, kivéve, ha az operandus logikai típusú.

A MISRA szabályok esetén egyértelműen látszik a törekvés, hogy minél inkább statikusan ellenőrizhetőek legyenek. A statikus analízis nagy előnye, hogy automatikusan végezhető, azaz elég hozzá egy ellenőrző programot megírni. A maradék, automatikusan nem ellenőrizhető szabályokat csak a kód átnézésével lehet vizsgálni. Automatikus kódellenőrzéshez több szoftver is van kereskedelmi forgalomban, sőt, a PC-Lint ingyenes MISRA-megfelelőségi ellenőrzést nyújt.

Összefoglalva, a MISRA-C nem hivatott a megvalósított algoritmus semmiféle vizsgálatára, nem erőltet semmilyen konkrét stílushasználatot, és a szabályokat betartva is lehet bármilyen, a valóságtól teljesen elrugaszkodott kódot írni. A MISRA-C célja, hogy a C nyelvet próbálja minél inkább leszűkíteni úgy, hogy a C nyelven írt kód egyértelmű legyen [17].

2.3.2. MAAB

1998-ban alakult meg a MathWorks Automotive Advisory Board (MAAB). Célja, hogy összehangolja az autóipari felhasználóik (mint például a Toyota, Ford, DaimlerChrysler) MathWorks-terméktulajdonságokra vonatkozó kéréseit. Mára a szervezet ülésein szinte az összes nagy alkatrészgyártó és -ellátó képviselteti magát.

2001-re a MAAB kidolgozta a saját modellezési irányelveit, melyek bárki számára szabadon elérhetőek a MathWorks honlapjáról [19, 20]. A MAAB irányelveket a MathWorks a saját termékeihez dolgozta ki, azaz a Simulinkhez és a Stateflowhoz [16].

Az irányelvek kidolgozásának fő motivációja az volt, hogy megalapozza a sikeres projektet és támogassa a csapatmunkát. Alkalmazása megkönnyíti a csapatmunkát nem csak cégen belüli, hanem külső partnerekkel és alvállalkozókkal való együttműködés esetén is [19].

A MAAB stílus irányelveket alkalmazva megvalósítható [Erkkinen04]:

- A problémamentes rendszerintegrálás,
- Egyértelmű interfészek,
- Modellek, kód és dokumentáció egységes megjelenése,
- Modell újrafelhasználhatósága,

- Könnyen olvasható modell,
- Modellek problémamentes cseréjének lehetősége,
- Letisztult folyamatok fejlesztése,
- Professzionális dokumentáció,
- Gyors szoftverváltoztatás,
- Egyszerű projekt átadás.

A MAAB irányelveket figyelembe vevő vállalatoknak természetesen lehetnek olyan saját belső szabályaik, melyek kivételt, bővítést vagy szigorítást jelentenek az eredeti MAAB ajánláshoz képest.

Sajnálatos módon az irányelveket már jó néhány éve nem frissítették, így a jelenlegi termékadottságokat nem tükrözi, de az aktulizáció már előkészítés alatt áll. És habár az autóipar által fejlesztett szabványról van szó, légiipari vállalatok is használják, sőt, egyes repülőgépgyártók még a szabálygyűjtemény kidolgozásában is részt vesznek.

Szabály ID	db_0042: Portok a Simulink modellekben
Prioritás	erősen ajánlott
Hatáskör	MAAB
Automatizálás	lehetséges
Előfeltétel	-
Leírás	A Simulink modellben minden port esetén teljesülnie kell, hogy • a bemenő portok bal oldalon legyenek, de a jelkereszteződést elkerülendő beljebb mozgathatók • a kimeneti portok helye jobb oldalon legyen, de a jelkereszteződést elkerülendő beljebb mozgathatók Az összes ki- és bemenő port neve látható kell legyen.
Előny	A szabály betartása biztosítja, hogy átlátható interfészek, könnyen értelmezhető és egységes megjelenésű modellek készüljenek, továbbá segíthet a jelkereszteződések megszüntetésében
Hátrány	A szabály áthágása átláthatatlan modellhez vezethet

2.1. táblázat. Egy MAAB irányelv

Egy MAAB szabály (2.1. táblázat) az alábbi felépítésű [15, 20]:

Szabály ID Két kisbetűből és négy számból áll, alulvonással elválasztva. Számozása minden szerző esetén 0-tól kezdődik és nem lehet szünet az ID sorszámok között. Amint

egy új szabály megkapja az ID-ját, onnantól fixen a sajátja, ez a szabályhivatkozásokhoz és fájlnév-ellenőrzéshez kell. A szabály ID két betűjének fel kell osztania a lehetséges ID-k terét – általában a szerzők monogramja szokott lenni, de bármilyen betűpár megfelel.

Szabály cím A cím rövid, de egyedi leírása a szabály alkalmazási területéről (például *azonosítók hossza*), amit az *Előkövetelmények* mező és egyedi ellenőrző eszközök használnak. Egy linknek is kell tartoznia a címhez, aminek segítségével egyszerűen elérhető az adott irányelv. A cím nem lehet a szabály kissé bővített leírása, mert az irányelvek tartalma megváltozhat, de a címnek állandónak kell lennie.

Prioritás Minden irányelvnek kötelező prioritást adni, a prioritás lehet *kötelező*, *erősen ajánlott* és *javasolt*. A prioritás nem egyszerűen az irányelv fontosságát jelöli, hanem az áthágásából származó következményeket is mutatja.

Kötelező Ezek azok az irányelvek, melyekben minden vállalat egyetértett, hogy nélkülözhetetlen, és hozzá 100 százalékban alkalmazkodik. A kötelező szabályok be nem tartásának következménye, hogy alapvető dolgok hiányozhatnak, és esetleg a modell működése sem lesz megfelelő. Kötelező irányelv be nem tartásának indokait dokumentálni kell.

Erősen ajánlott Azon irányelvek, amik betartása kifejezetten előnyös gyakorlat, de a már használatban levő modellek miatt eleve ki van zárva, hogy a vállalat 100 százalékban be tudja tartani. A be nem tartásából származó következmények, hogy a minőség és a külső megjelenés romlik, és megemelkedik a kockázat a karbantarthatóság, hordozhatóság és újrafelhasználhatóság terén.

Javasolt Az irányelv betartása javasolt a külső megjelenés javítása érdekében, de nem alapvető a modell működéséhez. Ezek az irányelvek jellemzően projektspecifikusak. Ha egy javasolt szabály nem teljesül, az általában azt jelenti, hogy a végeredmény nem alkalmazkodik a többi projekthez.

Hatáskör Az 1.0-ás verzióban az alábbiak egyike lehet: "MAAB", "DC", "DC-POWERTRAIN", "DC-CHASSIS", "FORD", "FORD-POWERTRAIN", "GM", "TOYOTA".

- MAAB: a The MathWorks és a hozzá csatlakozó autógyártók csoportja
- DC: a DaimlerChrysler Vállalat
- DC-POWERTRAIN: erőátvitel-specifikus DaimlerChrysler szabályok
- DC-CHASSIS: alváz-specifikus DaimlerChrysler szabályok
- FORD: Ford Vállalat
- FORD-POWERTRAIN: erőátvitel-specifikus Ford szabályok
- GM: General Motors Vállalat

- TOYOTA: Toyota Vállalat

Egyéb hatáskörök hozzáadása, mint például a projekt előtagja a projekthez kapcsolódó irányelveknél szintén lehetséges.

Automatizálás Lehet: *javítás*, *ellenőrzés*, *lehetőséges*, *semmilyen*.

Javítás Az automatikus javítás (részlegesen) és ellenőrzés lehetséges és már rendelkezésre áll

Ellenőrzés Automatikus ellenőrzés lehetséges, és van is az adott irányelvhez ellenőrző eszköz. Az ellenőrző neve a szabály ID-je.

Lehetséges Automatikus ellenőrzés megvalósítható, de még nem implementált.

Semmilyen Az irányelv teljesülését nem lehet automatizáltan ellenőrizni.

Előkövetelmény Ezzel a mezővel az irányelv az előkövetelményeit adó többi irányelvhez kapcsolható (logikai kapcsolat). Az előkövetelményként való megadásánál szerepelnie kell az irányelv azonosítójának (a konzisztencia miatt) és a címének (az értelmezhetőség miatt) – ezeken kívül más szöveget nem tartalmazhat a mező.

Leírás A leírás tartalmazza az irányelv részletes kifejtését, ha szükséges, képek és táblázatok is szerepelhetnek. Amennyiben formális leírás is rendelkezésre áll (mint például matematikai képlet, reguláris kifejezés, szintakszis diagram, pontos szám vagy határérték), azt meg kell adni a többértelműség elkerülése érdekében, továbbá ez a későbbiekben az automatizált ellenőrzést is segítheti. Egy mindenki számára érthető informális leírás hozzáadása mindenképpen szükséges, a napi használathoz.

Előny Itt van megfogalmazva az irányelv betartásának összes előnye, ahol, amennyiben szükséges, képek és táblázatok is helyet kaphatnak. A szabály betartására vonatkozó döntésben ez a mező kiemelkedően fontos, ezért tartalma kellően részletes kell legyen, azaz nem elég például annyi, hogy „javítja a minőséget”.

Hátrány A mezőben szerepelnek azok a hátrányos következmények, amit a szabály be nem tartása von magával. Képek és táblázatok itt is szerepelhetnek. A szabály alkalmazásának eldöntésekor a mező tartalma kiemelkedően fontos, ezért tartalmának kellően részletesnek kell lennie, azaz nem elég például annyi, hogy „csökken a minőség”.

2.3.3. DO-178B

Hivatalosan: *DO-178B Szoftver Megfontolások Légi Rendszerek és Felszerelések Hitelesítéséhez*, az amerikai RTCA (Radio Technical Commission for Aeronautics, Repülési Rádiótechnikai Bizottság) és az európai EUROCAE (European Organization for Civil Aviation

Equipment, Polgári Repülési Berendezésekkel Foglalkozó Európai Szervezet) közösen kidolgozott szabványa. Az EUROCEA ED-12B néven iktatta a szabványt, így Európában ezen a néven is ismert (2.1 ábra).

Az RTCA Szövetségi Tanácsadó Bizottság, nemzetközi tagságának több jelentős légi vállalat és kormányzati szabályozó intézmény is tagja, mint például az FAA (Federal Aviation Administration, Szövetségi Légügyi Hivatal) és a JAA (Joint Aviation Authorities, Társult Légügyi Hatóságok) [21].

A szabvány múltja egészen 1981-ig nyúlik vissza, akkor jött ki az első verzió, a DO-178. Ezt követte 1985-ben a DO-178A, majd végül 1992-ben a DO-178B. Mára a repülőgépipar *de facto* szabványa lett, számos nagy RTOS (Real-Time Operating System, valósidejű operációs rendszer) gyártó (Aonix, Enea, Lynuxworks, Green Hills Software, Wind River, BAE Systems) jelentette be termékének olyan biztonságkritikus verzióját, amely kifejezetten a DO-178B szabvány tanúsítási vizsgálatát veszi figyelembe.

DO-178B / ED-12B	
Software Considerations in Airborne Systems and Equipment Certification	
Latest Revision	December 1, 1992
Prepared by	RTCA SC-167 EUROCAE WG-12

2.1. ábra. DO-178B logó [22]

A szabvány kizárólag a civil szféra légiipari szoftvertermékeire vonatkozik, a katonai alkalmazásokhoz más, általában országonként eltérő szabványok vannak. Annak ellenére, hogy a DO-178B kifejezetten a repülőgép-ipar számára készült, számos más ipari szektorban is figyelmet keltett.

Az a szoftver, amely eleget tesz a DO-178B irányelveknek, teljességgel kompatibilis más iparágak biztonsági előírásaival is, mint például az orvosi eszközökre vonatkozó FDA Class III [510(k)] „élet-biztonsági", vagy az ipari IEC 61508 általános biztonsági szabvánnyal.

Az immár nemzetközileg elterjedt DO-178B a repülőgépekben alkalmazott biztonságkritikus rendszerek szoftverfejlesztéséhez, kódolásához ad útmutatást, és a megfelelőség bizonyításának követelményeiről is leírást ad. A szabvány biztosítani igyekszik, hogy a repülőgép szoftverébe a fejlesztés során ne kerülhessen be olyan kódrész, mely a működésben bármilyen hibát okozhat vagy elősegítheti hibás állapot kialakulását [23].

A DO-178B A-szintű biztonságkritikai irányelv megköveteli az aprólékosan megtervezett és dokumentált szoftverfejlesztést. Olyan kódkövetési folyamatot kell megvalósítani, mely tisztázni képes a kód minden egyes sorának funkcióját, és amely szigorúan befolyásolni tudja a szoftver működését.

A minősítési eljárást a Tanúsítvány Kapcsolat (Certification Liaison) és a Szoftver Minőség Biztosítás (Software Quality Assurance, SQA) független nemzetközi auditorai felügyelik. A két fél a teljes szoftverfejlesztési folyamatot vizsgálja a minősítés megállapításához. A kész kódot már azelőtt is jóvá kell hagyniuk, mielőtt az utolsó, on-board teszt következik.

A minősítési eljárás során kiadott tanúsítvány ugyanis nem csak magát a szoftvert minősíti, mert az RTCA irányelvek alapján csak teljes repülőgép vagy repülőgép-motor lehet minősített. Azaz a minősített termék tulajdonképpen egy teljes repülőgép, melynek minden összetevője minősíthető.

A repülőgép minősítését elősegítendő, a szoftverhez *tanúsítvány ereklye* (certification artifacts) nevű dokumentáció készül. Ez a dokumentáció azt hivatott bizonyítani, hogy a szoftver tényleg a DO-178B szabványnak megfelelően készült. A biztonsági előírásoknak megfelelő szoftver és hozzá a tanúsítvány ereklyék jelentik a minősíthetőséget.

A tanúsítvány ereklyék első részét a tervezési dokumentumok adják. Ezeket még a kód első sorának megírása előtt el kell készíteni és jóvá kell hagyatni. A legfelső szintű tervezési dokumentum a Szoftveres Szempontok Terve Tanusításhoz (Plan for Software Aspects for Certification, PSAC)

Ha a PSAC-ot jóváhagyják, a DO-178B szoftvert fejlesztő csoportnak el kell készítenie egy sor további, még részletesebb tervet, amelyek majd a kulcsfolyamatokat irányítják.

Ezek a részletes tervek a:

1. Szoftverfejlesztési Terv (Software Development Plan, SDP),
2. Szoftververifikációs Terv (Software Verification Plan, SVP),
3. Szoftverkonfiguráció Menedzsment Terv (Software Configuration Management Plan, SCMP),
4. Szoftverminőség Biztosítási Terv (Software Quality Assurance Plan, SQAP).

Az SDP-hez több részletes szabvány is kapcsolódik, amelyek megszabják, hogy hogyan kell szoftverkódot fejleszteni a DO-178B szabványnak megfelelően. Az említett szabványok:

1. Szoftvertervezési Szabvány (Software Design Standards, SDS),
2. Szoftverkövetelmény Szabvány (Software Requirements Standards, SRS),
3. Szoftverkódolási Szabvány (Software Coding Standard, SCS).

A nyomonkövethetőség a DO-178B szabvány kulcseleme, ami egyben a fő különbség a biztonságkritikus és az általános szoftverfejlesztés között. A szabványt használó fejlesztések során a szoftverfejlesztéssel párhuzamosan rengeteg, a követhetőséget biztosító dokumentáció készül. Ezek a dokumentumok bizonyítják, hogy:

- A szoftver tökéletesen eleget tesz minden rendszerkövetelménynek,
- Minden egyes kódutasítás szükséges és a kitűzött feladatát ellátja,
- Semmilyen előre meg nem fontolt kódsor nincs a szoftverben, és bármilyen, a hordozhatóság, ellenállóság vagy egyéb okból létező, nem nélkülözhetetlen kód nem befolyásolja a szoftvert biztonsági szempontból.

A követhetőségi és követelmény dokumentációk magukban foglalnak második szintű tanúsítvány ereklyéket, amelyeket ismételten jóvá kell hagyatni a minősítő bizottsággal. A nyomonkövethetőség bizonyítása kifejezetten erre a célra fejlesztett szoftvereszközök használatával történik.

A bizonyítási eljárás során az auditorok először a kód legalsó szintjén vizsgálják a követelmények teljesülését, majd folytatják tovább egyre feljebb, egészen a rendszer szintjéig.

Minden szoftverfejlesztés és tanúsítvány ereklye készülését független belső SQA felügyeli, gondoskodva arról, hogy a tervekhez a fejlesztés folyamán végig ragaszkodjanak. A tanúsító hatóság képviselője, vagy a megbízott auditor rendszeresen felülvizsgálja a fejlesztési folyamatot és a szolgáltatott bizonyítékokat, hogy valóban megfelelnek-e a DO-178B szabvány előírásainak.

A sok tervezés, fejlesztés, tesztelés és felülvizsgálat eredményeként kiemelkedő minőségű termék készül, mely eleget tesz a legmagasabb szintű nemzetközi szoftver-megbízhatósági követelményeknek [21, 23].

2.4. Néhány létező megvalósítás bemutatása

2.4.1. SCADE

A SCADE (Safety Critical Application Development Environment) gyártója a francia Esterel Technologies, a SCADE pedig az általuk „Szabvány a repülőgépipari beágyazott rendszerek fejlesztéséhez” névvel ellátott termékcsalád.

Az állítás valóban igaz, hiszen a SCADE széria tervezésénél a biztonságkritikus rendszerek igényeit vették figyelembe. SCADE-t használnak többek között a repülőgép-, vasút-, védelmi- és autóiparágakban is. A szoftvercsomag olyan szolgáltatásokat foglal magában, amelyek támogatják a tanúsítási folyamatot – ez kifejezetten a D0-178 szabványra érvényes. Azaz a SCADE képes olyan kód generálására, amelyen nyomon követhető, hogy a D0-178 szabványnak megfelel.

A SCADE által kínált lehetőségek tökéletesen megfelelnek a repülőgép- és autóipari kívánalmaknak. A repülőgépiparban a vezérlő rendszerek folyamatosan különböző értékeket ellenőriznek, mint például az aktuális magasság, a szél sebessége vagy az irány. A SCADE

azon tulajdonsága, hogy képes reaktív rendszer modellezésére, épp ezért kiemelkedően fontos. Az első nagy és sikeres repülőgépipari SCADE-projekt az AIRBUS A380 fedélzeti szoftvere volt [24] (2.2 ábra).

2.2. ábra. Az A380 első repülése (2005. április 27.) [25]

Egy másik problémakör, ahol a SCADE kiválóan alkalmazható lehet, az áramkörtervezés és verifikáció. Az áramköröknél szintén a folytonos vezérlési séma a jellemző, azaz bizonyos jelek vissza vannak csatolva a rendszerben. A tervezési verifikációt használva olyan biztonsági jellemzők, mint például deadlock kialakulásának lehetősége, gyorsan és kiválóan szűrhetők.

A SCADE képes Ada, SPARK Ada, Qualifiable C (v3.1 vagy v4.2) és normál C nyelvű kód generálására. A rendszerimplementációt egy automatikus kódgenerátor szintetizálja, amely képes a SCADE modellt kóddá leképzeni – az ehhez szükséges felhasználói beavatkozás mindössze a célnyelv és a gyökérpont kiválasztása. A gyökérpont az a blokk-diagram, mely a modell legfelsőbb szintű leírása, a kód ehhez és a hozzá kapcsolódó alkomponensekhez generálódik (2.3 ábra).

Ahhoz, hogy biztonságos kódot kapjunk, a generátor bizonyos szabályokat kell kövessen.

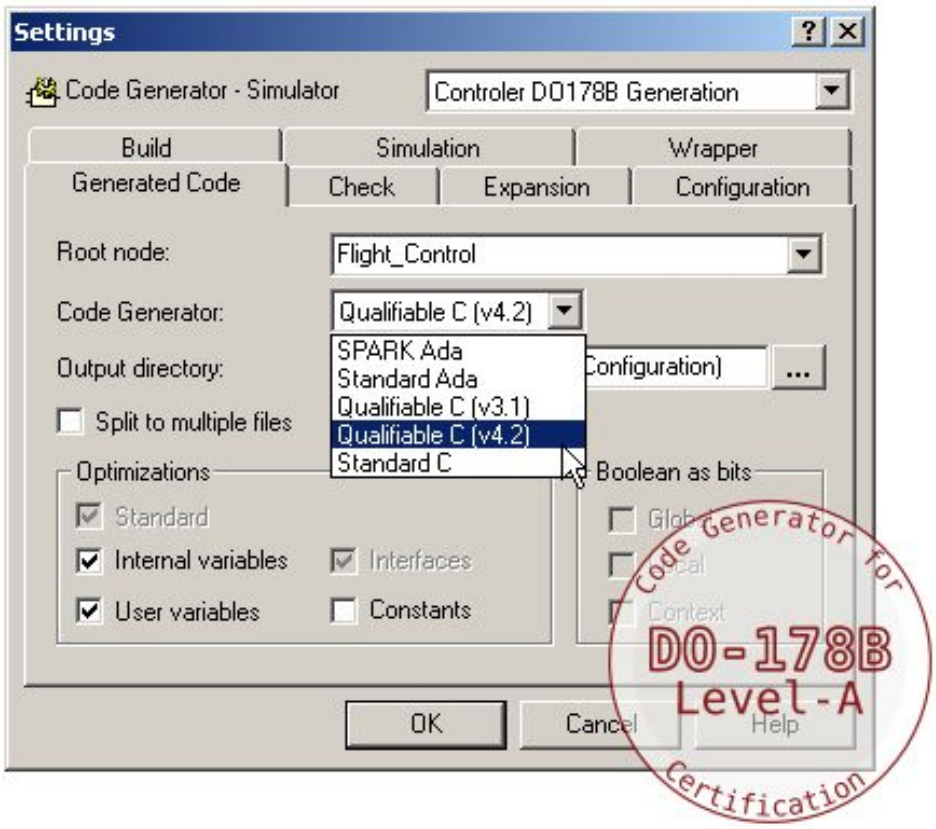
Először is, a generátor nem készít felesleges kódot. Ez azt jelenti, hogy csak és kizárólag a rendelkezésre álló modellnek közvetlenül megfelelő kódot generál. Vagyis ha a modellen változtatás történik és újrafordítják, a régi kód bármely része megváltozhat.

Másodszor, a generálás során soha nem kerül be halott kód, azaz olyan kód, ami soha nem hajtódna végre. Mindkét szabály elősegíti, hogy az alapkód kisebb, és a DO-178B level A követelményeknek megfelelő lehessen.

Harmadszor, a kódgenerátor nem készít olyan kódot, ami mutató aritmetikát használ. Ezzel lecsökken a mutató index túllépéséből adódó memória-felülírás esélye.

És végül, a generált kód statikus memória allokációt használ, azaz minden memóriafoglalás fordítási időben történik. Így lehetőség van statikus analízisre, hogy kiderüljön, hogy az architektúrán rendelkezésre áll-e elég memória az alkalmazás futtatásához.

Az elkészült kódban semmiféle kézi módosítás nem történhet annak tudomásulvétele nélkül, hogy azt a modell már nem fogja tükrözni és hatása a modellel kapcsolatos állításokban sem fog szerepelni. Annak ellenére, hogy nem módosítható, a kód így is jól



2.3. ábra. Kódgenerálás SCADE módra [25]

olvasható, sok kommenttel és az eredeti modellt jól tükröző függvénynevek használatával [25].

A kódbiztonság növelésének másik módja egy pontosan definiált nyelv használata, mint például a SPARK Ada. A SPARK Ada nyelv leszűkíti az Ada nyelvet pontos definíciókat használva oly módon, hogy minden SPARK Ada kód kizárólag egyféleképpen értelmezhető. Ez azt jelenti, hogy olyan alkalmazásokat lehet készíteni, melyek ugyanazt a funkciót valósítják meg, eltérő fordítások esetén is. Továbbá, a nyelv pontosan definiált limitált részhalmazának használata lehetővé teszi a szigorú szemantika ellenőrzést [24].

2.4.2. LabVIEW

A LabVIEW (Laboratory Virtual Instrumentation Engineering Workbench) a National Instruments grafikus fejlesztőkörnyezete. Jelenleg a 8.20-as verzió az aktuális, ahol a 20 utalás a termék huszadik évfordulójára, hiszen az első LabVIEW 1986-ban jelent meg, amelyet még Apple Macintosh-ra írtak [26].

A LabView programozási nyelve, a "G" egy úgynevezett adatfolyam leíró nyelv, azaz a programvégrehajtás a grafikus diagram – irányított gráf – alapján történik.

A beágyazott rendszerek összetettségét figyelembe véve, azaz hogy szükség van elektronika-, szoftver- és mechanika-tervezésre is, a LabVIEW integrált megoldást nyújt. Ez az összetett megoldás azt jelenti, hogy a termék a rendszeridentifikációtól kezdve a szimuláción, modellfejlesztésen, kódgeneráláson át a teljes termékfejlesztési szakaszban használható [27].

LabVIEW Technology	Compilation Method	Target Operating Systems	Common Hardware Targets
Development System	Native LabVIEW compiler[1]	Windows, Linux, Mac	Desktop, laptop, CompactPCI/PXI, industrial PCs, single-board computers
LabVIEW Real-Time Module	Native LabVIEW compiler[1]	Phar Lap ETS, RTX	All of the above plus NI Compact FieldPoint, CompactRIO, Compact Vision System
LabVIEW PDA Module	Cross-compiler, third-party tools[1]	Palm, Windows Mobile for Pocket PC, Windows CE[4]	Handhelds, cell phones, touch panel displays
LabVIEW FPGA Module	VHDL code generation, integrated Xilinx ISE tools[1]	N/A (result is hardware)	CompactRIO chassis, PCI/PXI R Series intelligent DAQ devices
NI LabVIEW Embedded Module for ADI Blackfin Processors[3]	C code generation, integrated ADI VisualDSP++ tools[1]	VisualDSP++ Kernel (VDK)	600 MHz ADI Blackfin DSP/MPU hybrid
LabVIEW Embedded Development Module	C code generation, third-party compiler[2]	eCos, VxWorks, UNIX, Windows (any OS)	Any 32-bit microprocessor or DSP

[1]OEM-ready -- compilation is fully automated and no knowledge of the third-party toolchain is required.

[2]Requires knowledge of a third-party toolchain.

[3]NI announced the first public beta version in August 2005.

[4]Windows CE support for select devices.

2.4. ábra. LabVIEW 8 targetek [27]

A LabVIEW alkalmazás virtuális műszerekből épül fel (VI – Virtual Instrument), egy virtuális műszer tulajdonképpen három részből áll: egy blokkdiagramból, egy hozzá szorosan kapcsolódó vezérlőpanelből és egy előlapból. Az egyes VI-k tovább finomíthatók subVI-kre, azaz a program támogatja a moduláris felépítést.

Fix-pontos, egy mintavételi frekvenciájú modellek esetén a LabVIEW SignalExpress a grafikus leírásból képes automatikusan C, Integer LabVIEW és LabVIEW FPGA kódot generálni, illetve egy LabVIEW alkalmazás tartalmazhat C, VHDL, MathScript kódot is.

A fejlesztőkörnyezethez több könyvtár tartozik, amelyek függvényeket biztosítanak adatgyűjtéshez, jelgeneráláshoz, matematikai és statisztikai műveletek elvégzéséhez, jelkondícionáláshoz és analízishez is. Emellett számos grafikus felületelem is rendelkezésre áll a különböző LabVIEW csomagokban.

A LabVIEW Professional Development System segítségével stand-alone futtatható alkalmazások készíthetők, amelyeket aztán korlátlan mennyiségben lehet megosztani.

A LabVIEW környezet nagy előnye, hogy a G nyelvű kód platformfüggetlen (néhány platform-specifikus függvénytől eltekintve), így a kód az egyes LabVIEW rendszerek között szabadon hordozható, operációs rendszertől függetlenül (2.4 ábra).

A kész G kódból automatikusan ANSI C kódot lehet generálni a programmal. Az új LabVIEW Embedded Development Module a natív kódgenerálás és fordítás felé nyit szabad utat, így tapasztalt fejlesztők bármilyen mikroprocesszorhoz és operációs rendszerhez készíthetnek speciálisan a kártyát támogató csomagot [26, 27].

2.4.3. TargetLink

A TargetLink a német dSPACE vállalat automatikus termékkód generátora. A dSPACE termékeket főképp elektronikus vezérlő eszközök és mechatronikus vezérlők szoftverének fejlesztéséhez használják. A termékcsalád fő profilja az autóipar, de a repülőgépiparban és hajtások tervezéséhez is alkalmazzák.

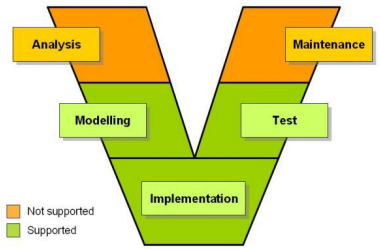
A dSPACE volt az alapja a gyors prototípus és a hardver-in-the-loop szimulációs technológiák kialakulásának, dSPACE fejlesztés volt az első termékkód generátor a MATLAB/Simulink környezethez is – ez az utóbbi ma a TargetLink. A termékcsalád része továbbá egy kifinomult és egyszerűen használható eszköz elektronikus vezérlők kalibrálásához és mérési adatok összegyűjtéshez.

A termék karrierje a kezdetektől meredeken ívelt felfelé. A DaimlerChrysler már 1999-ben TargetLink által generált kódot használt ECU fejlesztésnél. 2000-ben a Nissan és a MAN is bejelentette, hogy TargetLinket használnak új fejlesztéseikhez, és 2001-re a TargetLink kódokat a kézi kódolással megegyező hatékonyságúnak tartották, ekkora már a Siemens és a Honda cég is használja.

2002-ben a TargetLink D0-178B A-szintű besorolás alá került, azaz immár alkalmas repülőgépipari fejlesztésekhez is. 2004-től kód lefedési analízist is biztosít, és az OSEK (Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen, Nyílt Rendszerek és Interfészeik Gépjárművek Elektronikájához, autóipari szoftverfejlesztési szabvány) implementációkat is támogatja. Az Audi és a BMW is TargetLinket használ [28].

2007-re a dSPACE automatikus kódgenerátora többek között alkalmas MATLAB/Simulink/Stateflow modellből kódot generálni, szimuláción alapuló direkt tesztelésre és támogatja az autógyártók, beszállítók és eszközfejlesztők által közösen fejlesztett, nyílt és szabványos autóipari szoftverarchitektúrát, az AUTOSAR-t (AUTomotive Open System ARchitecture) is [30].

Az aktuális, 2.2-es verzió az első, ami széleskörűen támogatja a modellezést, szimulációt és kódgenerálást az AUTOSAR szoftverkomponensekhez. Ezzel lehetővé vált Simulink/TargetLink modellek direkt leképezése az AUTOSAR architektúrának megfelelő kóddá.



2.5. ábra. A V-modell ASCET-SD által támogatott fázisai [32]

A TargetLink 2.2 új blokkjaival még a modellezés szintjén AUTOSAR építőelemeket lehet meghatározni, mint például portok. A termék nemcsak az AUTOSAR-nak megfelelő kód generálására képes, hanem alkalmas a szoftverrel kapcsolatos szimulációs és tesztelési feladatok elvégzésére is. Az AUTOSAR szoftver architektúrába való beillesztéshez a TargetLink ún. szoftverkomponens-leíró fájlokat generál [29].

A TargetLink fő erőssége hagyományosan a széleskörű processzor támogatás, valamint a célkörnyezet alapján kiegészítő optimalizálás [dSPACE].

2006 április 2-án a Barracuda nevű, pilóta nélküli repülőgép első repülése igazi TargetLink sikertörténet, mivel az UAV (Unmanned Aerial Vechile) kódjának 45 százalékát TargetLink-kel generálták, Simulink és Stateflow modellekből. Többek között a TargetLink generálta a kódot a repülőgép vezérlőrendszeréhez, az autopilótához, a repülés menedzsmenthez, a repülési adatok számítását végző logikához és a navigációs rendszerhez is [31].

2.4.4. ASCET

Az ETAS Csoport (Entwicklungs-, Test- und Applikationssysteme, Fejlesztői-, Tesztelői és Felhasználói Rendszerek) 2003-ban jött létre az ETAS, Vetronix és LiveDevices cégek egyesülésével. Portfóliója szabványos fejlesztő- és diagnosztikai eszközökből áll, melyek lefedik az autóipari elektonikus vezérlőegységek teljes fejlesztési és használati életciklusát.

Az ASCET V5.2 egy univerzális és rugalmas termékcsalád, beágyazott autóipari rendszerek funkció- és szoftverfejlesztéséhez. Az autógyártók körében az ASCET eszközök széles körben elterjedtek elektonikus vezérlő alkalmazások modell-alapú fejlesztéséhez. A minőség növelése valamint a fejlesztési projektek kockázatának és költségének csökkentése érdekében, az ASCET eszközök támogatják a modellezést, szimulációt, rapid prototypingot és az automatikus kódgenerálást beágyazott autóipari alkalmazásokhoz. A 2.5-ös ábra az ASCET termékek és a V-modell kapcsolatát mutatja.

Az ASCET V5.2 egy moduláris termék, amely az ASCET-MD (Modeling and Design), ASCET-RP (Rapid Prototyping), és az ASCET-SE (Software Engineering) egységekből áll. Ezen modulok önmagukban, illetve tetszőleges kombinációkban is működőképesek. Létezik

még egy ASCET-MIP (MATLAB Integration Package) kiegészítő termék az ASCET-MD és ASCET-RP csomagokhoz, amely a MATLAB/Simulink programcsalád felé nyújt interfészt.

Az ASCET termékcsalád mindhárom tagja lehetővé teszi a létező szoftverelemek újrafelhasználását. Az ASCET-MD egy grafikus fejlesztői környezet, amellyel hardverfüggetlen funkcionális specifikáció készíthető. Az ASCET-SD segítségével blokk-diagrammokból és állapotgépekből automatikusan kódot lehet generálni. Az ASCET-RP elősegíti a célkörnyezet-specifikus rapid prototypingot, és lehetővé teszi a tesztelést a fejlesztési folyamat minden szakaszán. A végeredmény egy újrahasznosítható specifikáció, amelyből biztonságos végrehajtható termékkód készíthető egyetlen egérkattintással [33].

Az ASCET kódgenerátor MISRA-szabvány megfelelőségi vizsgálata

Az autóipari cégeknek komoly garanciára van szükségük az automatikusan generált kódok minőségét illetően. Az erősen biztonságkritikus alkalmazások kódjának automatikus generálása komoly veszélyforrást jelenthet, ha a tervezett funkciók esetleg nem az elképzeltnek megfelelően valósulnak meg. Többek között emiatt az idei (2007-es) év során a ASCET V5.2 automatikus kódgenerátor MISRA szabvány megfelelőségi vizsgálat alá került.

A szabványnak való megfelelőség elért foka mátrix formában dokumentált, amiben mind az egyes MISRA szabályoknak való megfelelőség, mind az azoktól való igazolható eltérés szerepel.

A megfelelőségi analízis során gyártósorhoz generáltak ASCET-tel kódot, operációs rendszer nélküli illetve operációs rendszert használó környezetre is. Az ASCET minősítése ezen autokódok (automatikusan generált kódok) elemzése alapján készült el.

Azon szabályok, melyek teljesülését az ASCET az implementáció során nem befolyásolhatja – mint például amelyek kizárólag a használt fordító választástól függnek – nem érintették az ASCET autokód besorolását. Ugyanígy, az autokódhoz kézzel hozzáadott kódrészek nem képezték a vizsgálat tárgyát, hiszen ezekért az ASCET nem felelős.

Mindent összevetve, a vizsgálat eredménye az lett, hogy az összes MISRA szabályt figyelembe véve, az ASCET által generált kód a MISRA irányelvek 90 százalékos betartására képes. Ám amiatt, hogy a MISRA szabvány áthágások dokumentálva vannak, az ASCET autokód akár 100 százalékban a MISRA szabványnak megfelelő lehet [34].

3. fejezet

A használt fejlesztői környezet bemutatása

3.1. A MATLAB-Simulink programcsalád

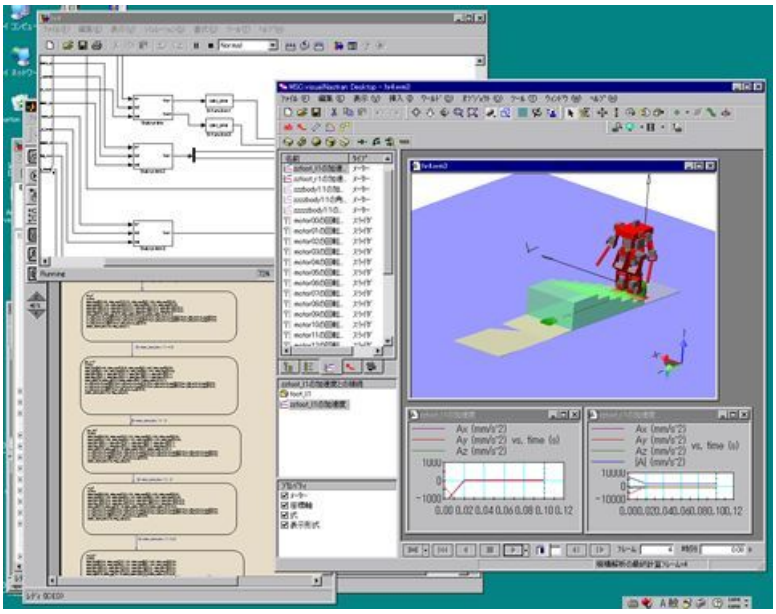
A MATLAB és a Simulink az amerikai The MathWorks vállalat termékei.

Magát a MATLAB programot az 1970-es évek végén Cleve Moler találta ki, aki akkor a Új Mexikói Egyetem számítástudományi tanszékének vezetője volt. Azért tervezte a programot, hogy hallgatói FORTRAN tudás nélkül használhassák a LINPACK és EISPACK szoftvercsomagokat (a két csomag komplex és lineáris aritmetikai kiegészítő rutinok gyűjteménye FORTAN-hoz). Az alkalmazásnak hamar elterjedt a híre az egyetemek között, főleg az alkalmazott matematikával foglalkozók körében.

Moler 1983-ban rövid látogatást tett a Stanford Egyetemen, itt ismerkedett meg a programmal a mérnök Jack Little, aki meglátta benne a hatalmas kereskedelmi lehetőséget. Steve Bangerrel hárman újraírták a MATLAB-ot C nyelven (ezek az újraírt könyvtárak JACKPAC néven váltak ismertté). Ezt követően 1984-ben megalapították a The MathWorks céget, hogy ott folytatódjon tovább a szoftver fejlesztése [35].

A MATLAB-ot elsőként a szabályozástechnika területén dolgozó mérnökök kezdték használni (amely Jack Little szakterülete), de hamar elterjedt más körökben is. Napjainkban széleskörűen használják oktatási célokra, elsősorban lineáris algebra és numerikus analízis kapcsán, de képfeldolgozással foglalkozó kutatók körében is egyre népszerűbb.

A MATLAB egy magas szintű nyelv mérnöki számításokhoz (technical computing language), valamint interaktív környezet algoritmus fejlesztéshez, adatmegjelenítéshez, adatelemzéshez és numerikus számításokhoz. MATLAB-ot használva a mérnöki számítások megoldása lényegesen gyorsabban megy, mint hagyományos nyelvek esetén, mint például C, C++ vagy Fortran [37].



3.1. ábra. MATLAB-Simulink környezet [36]

A MATLAB széleskörűen használható, beleértve olyan területeket, mint a jel- és képfeldolgozás, kommunikáció, szabályozástechnika, tesztelés és mérés, pénzügyi modellezés és analízis vagy bioinformatika. Kiegészítő eszköztárak (külön rendelkezésre álló, speciális funkciójú MATLAB függvények) még tovább bővítik a MATLAB által lefedhető problémák körét.

A MATLAB számos eszközt nyújt a dokumentálás és munkamegosztás elősegítéséhez. A MATLAB kód könnyedén integrálható más nyelven írott kóddal és alkalmazásokkal, és MATLAB-ban írt algoritmus és alkalmazás is egyszerűen megosztható [38].

Kulcs elemek:

- Magas szintű nyelv mérnöki számításokhoz,
- Fejlesztőkörnyezet kód, fájlok és adatok kezeléséhez,
- Interaktív eszközök ismétlődő felderítéshez, tervezéshez és probléma megoldáshoz,
- Matematikai függvények lineáris algebra, statisztika, Fourier analízis, szűrés, optimalizálás és numerikus integrálás elvégzéséhez,
- 2-D és 3-D grafikus funkciók adatmegjelenítéshez,
- Eszközök egyedi grafikus felhasználói interfész létrehozásához,
- Függvények MATLAB alapú algoritmusok külső alkalmazásokkal és nyelvekkel való integrációjához, mint például C, C++, Fortran, Java, COM és Microsoft Excel.

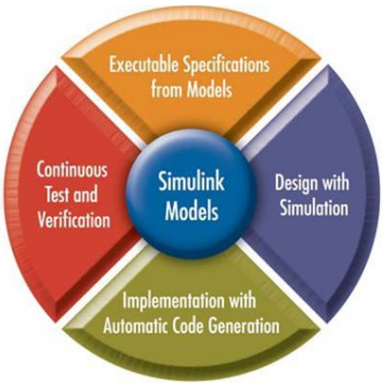
3.1.1. Simulink

A Simulink egy platform dinamikus rendszerek többrétegű szimulációjához és Modell-alapú tervezéséhez. Interaktív grafikus környezet és testreszabható blokk-könyvtárakat biztosít vezérlő, jelfeldolgozó, kommunikációs és egyéb változó idejű rendszerek pontos tervezéséhez, szimulációjához és teszteléséhez.

Kiegészítő termékek speciális modellezési és tervezési feladatokhoz, kódgeneráláshoz, algoritmus implementáláshoz, teszteléshez és verifikációhoz, tovább bővítik a Simulink környezet lehetőségeit. A hozzá tartozó alkalmazásspecifikus blokk-készletekkel, mint például a Jelfeldolgozó- (Signal Processing), Kommunikációs- (Communications) és Videó- és Képfeldolgozó (Video and Image Processing) blokk szett (Blockset), a Simulink egy kiváló grafikus fejlesztőkörnyezet beágyazott algoritmusok magas szintű leírásához.

Egy magas szintű Simulink modell számos célt szolgál:

- A Simulink modell futtatásával a szimuláció rögtön végrehajtható,
- Használható tesztelésre, verifikációs és implementációs célra - azaz a teljes fejlesztési folyamat során,
- Lehetővé válik a hibák korai észrevétele, ezzel elkerülhető a fejlesztés végén a költséges hibajavítás,
- Nincs szükség papír alapú, könnyen félreérthető specifikációra, helyette *végrehajtható specifikációt* ad,
- A fejlesztő csapat minden tagja számára könnyen érthető és használható a modell, így fókuszálni lehet a fő modell részleteinek fejlesztésére.



3.2. ábra. Simulink és MBD [3]

A Simulink a MATLAB-bal integrált, ezzel azonnali hozzáférést nyújt eszközök széles köréhez – algoritmus fejlesztéshez, adatmegjelenítéshez, adatelemzést és -elérést végző, valamint numerikus számítási eszközökhöz [39]. A 3.2-es ábra a Simulink helyét mutatja a Modell-alapú tervezésnél.

Kulcs elemek [37]:

- Előre definiált blokkok és könyvtárak kimerítő és bővíthető választéka,
- Interaktív grafikus szerkesztőfelület egyéni blokk diagramok összerakásához és kezeléséhez,
- Összetett tervek kezelése a modellek hierarchiaszintekre való felosztásával,
- Lehetőség más szimulációs programokkal való együttműködésre, és kézzel írott kód beillesztésére, beleértve MATLAB kódot is,
- Folytonos-, diszkrét- és hibrid-idejű szimulációk futtatása fix vagy változó lépésközzel,
- A modell kiértékelését segítendő függvények az interaktív input definiáláshoz és output figyeléshez,
- Grafikus debugger a szimulációs eredmény vagy a modell esetleges váratlan viselkedésének kiértékeléséhez,
- Teljeskörű hozzáférés a MATLAB adatelemzési- és megjelenítési-, GUI fejlesztési-, adat- és paraméterlétrehozási funkcióihoz,
- Modell analizációs és diagnosztikai eszközök a modell egységességének biztosításához és a modellezési hibák meghatározásához.

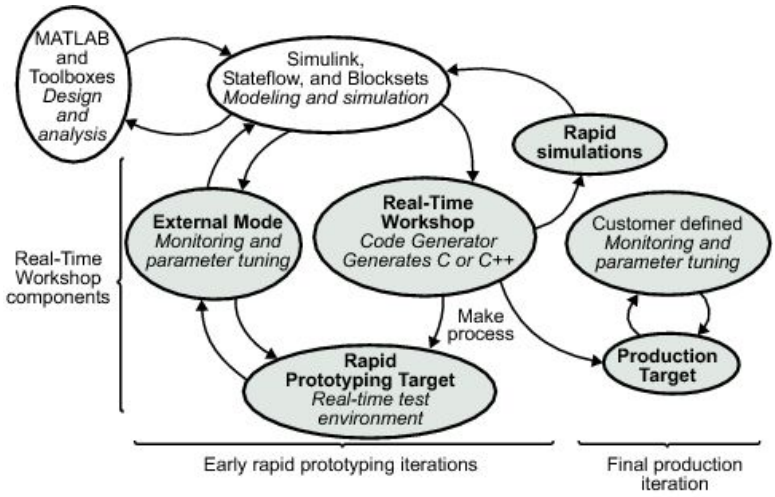
3.1.2. Real-Time Workshop

A Real-Time Workshop egy kiegészítő eszköztár a Simulinkhez. A Real-Time Workshop képes C kódot generálni illetve független C kódot végrehajtani Simulinkben írt modell teszteléséhez és fejlesztéséhez. Az eredményként kapott kód használható valós-idejű és nem valós-idejű feladatokhoz is, mint például szimuláció-gyorsítás, gyors prototípus fejlesztés vagy hardver-in-the-loop tesztelés. A kódgenerálást interaktív módon lehet hangolni és felügyelni Simulink blokkok és a beépített analizációs eszközök segítségével [37] (3.3. ábra).

A Real-Time Workshop képes ANSI/ISO-C kompatibilis kódot generálni a teljes modellhez, vagy egy különálló alrendszerhez, továbbá lehetővé teszi, hogy a kód bármilyen mikroprocesszoron és valós idejű operációs rendszeren (Real-Time Operating System, RTOS) futhasson. Külön megvásárolható kiegészítő termékek tovább bővítik a kódgenerációs lehetőségeket [40].

Kulcs elemek [37]:

- ANSI/ISO-C kód és végrehajtható fájl generálása diszkrét, folytonos és hibrid modellekhez,



3.3. ábra. A Real-Time Workshop helye a fejlesztési folyamatban [40]

- Modell blokkok használata összetett alkalmazások esetén az inkrementális kódgeneráláshoz és fordításhoz,
- A Simulink adatszótár támogatása egész, lebegőpontos és fixpontos adatok esetén,
- Kódgenerálás egy- és több mintavételi sebességű, illetve aszinkron modellekhez,
- Single-taszking és multitaszking operációs rendszer és operációs rendszer nélküli (bareboard) környezet támogatása,
- A kód végrehajtási idejének csökkentése optimalizációval,
- Lehetőség a generált kód testreszabására és már meglévő kód integrálására,
- A kód interaktívan hangolható és monitorozható Simulinken belül illetve kívülről is.

3.1.3. Real-Time Workshop Embedded Coder

A Real-Time Workshop Embedded Coder képes Simulink és Statflow modellekből a professzionális kézi kódolás egyértelműségével és hatékonyságával megegyező C kódot generálni. A generált kód rendkívül tömör és gyors – ez alapkövetelmény beágyazott rendszerek, célhardveren történő rapid prototyping, tömeggyártású mikroprocesszorok és valós idejű szimulációk esetén. Teljeskörű támogatást nyújt meglévő alkalmazások, függvények és adatok integrációjához [37].

Kulcs elemek [41]:

- ANSI/ISO-C kód generálása Simulink és Stateflow modellekből, ami memóriafelhasználás, végrehajtási idő és olvashatóság terén a kézi kóddal összemérhető,

- Támogatja az összes Simulink adatobjektumot és az adatszótár képességeket, beleértve a felhasználó által definiált tárolási osztályokat, típusokat és álneveket,
- Egyéni grafikus kezelőfelületet nyújt saját adat létrehozásához,
- Tömören felbontja a multirate (több mintavételi frekvenciájú) kódot a hatékony ütemezéshez,
- Részletes kódkommentezés,
- Automatikus dokumentáció generálás,
- A felhasználó által meghatározottak alapján egy fő programot generál, ami részletesen leírja, hogyan kell a kódot a célkörnyezetben alkalmazni.

3.1.4. Kódgenerálás Simulink modellhez

A Real-Time Workshop célkörnyezet-specifikus fájlsablont használ a Simulink modell C kódra fordításához. A célspecifikus sablonok határozzák meg azt a környezetet, ahol a generált kód futni fog. Lehetőség van saját célkonfiguráció fejlesztésére, vagy a már meglevő futáskész konfigurációk és harmadik fél Real-Time Workshop által támogatott célkonfigurációi közül választani [42]. A létező célkonfigurációk:

Generic Real-Time Target Kódot generál a modell paramétereinek interaktív hangolásához, logolja és megjeleníti a valós idejű szimulációs eredményeket, statikus memóriafoglalást használ.

Generic Real-Time Target Malloc Dinamikus memóriafoglalást használ a generált kódban, így tartalmazhatja a modell több példányát vagy több modellt egy végrehajtható fájlként.

S-Function Target A modellt leképezi Simulink S-függvény DLL-re.

Rapid Simulation Target (RSim) Gyors és rugalmas tesztplatform, batch vagy Monte Carlo szimuláció végzéséhez. Fix- és változó lépésközzel is használható.

Tornado Target Kódot generál a Wind River Systems VxWorks real-time operációs rendszerén való futtatáshoz.

Az RTW teljes keretrendszert nyújt a generált kód valós idejű végrehajtásához és a végrehajtási környezetbe helyezéséhez. Egy- vagy több mintavételi frekvenciát használó kódot generál a modellben meghatározott mintavételezési idők alapján. A kód használható valós idejű operációs rendszerrel vagy anélkül, single- és multitaszking vagy aszinkron módban is.

Single-Tasking Ebben a módban az ütemező a generált kódot egy szálként hajtja végre, az egyes mintavételi sebességek közti preempció (megszakítás) nem lehetséges.

Multitasking Itt egy determinisztikus Rate Monotonic ütemező hívja meg a kódot, engedélyezve a mintavételi frekvenciák közti váltást. Operációs rendszer nélküli környezetben a kódba beágyazott interruptokkal lehet a preemptivitást előidézni. RTOS környezet esetén van task prioritás és task preempció.

Aszinkron Az aszinkron módban a nem-periodikus vagy aszinkron mintavételi frekvenciák Simulink S-függvényekkel vannak meghatározva. A Real-Time Workshop ezeket a mintavételezési sebességeket fordítja le a végrehajtási környezetnek megfelelően célspecifikus kóddá. Modell és kód készül olyan eseményekhez, mint például hardver interrupt, és alrendszer triggerelése külön taskként. Egy aszinkron blokk könyvtár rendelkezésre áll a VxWorks valós idejű operációs rendszerhez, amely sablonként használható saját alkalmazás fejlesztésénél [40].

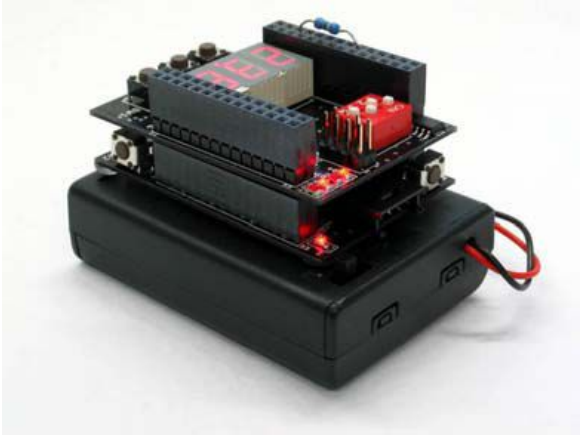
A Simulink és a Real-Time Workshop teljeskörű készletet nyújt célkörnyezet-független valósidejű alkalmazás fejlesztéséhez. Ez többek között azt jelenti, hogy [37]:

- Prioritás rendelhető a modell minden mintavételi frekvenciájához,
- Számlálók és időzítők állnak rendelkezésre az abszolút és az eltelt idő számításához,
- Van egy Rate Transition nevű blokk a mintavételi frekvenciák közti adatátviteli mechanizmus leírására, az adatintegritás, determinisztikusság és teljesítmény optimalizálásához,
- Van túlcsordulás-észlelés, hogy lehessen hibakezelő logikát rendelni minden mintavételi frekvenciához.

3.2. A MITMÓT

A MITMÓT-ot (3.4. ábra) a Budapesti Műszaki és Gazdaságtudományi Egyetemen, a Méréstechnika és Információs Rendszerek Tanszéken fejlesztették. Neve az anyatanszék rövidített nevének (mit) és a fonetikusan leírt angol mote (porszem) szónak az összetétele. A mote szó a szakirodalomban az „intelligens porszem”-et jelenti, azaz az olyan apró elektronikus készülékeket, melyek mindenhol jelenlevők, és valamiféle intelligenciával rendelkeznek.

A MITMÓT-ot elsősorban oktatási célra fejlesztették, a Méréstechnika és Információs Rendszerek Tanszék Beágyazott Rendszerek szakirányának laborgyakorlatai számára. Moduláris felépítésű, jelenleg két processzormodul, érzékelő- és beavatkozó modul valamint



3.4. ábra. A MITMÓT [43]

többféle kommunikációs modul tartozik hozzá. Ezek a modulok igény szerint variálhatók, így egy konkrét feladathoz könnyen kialakítható speciális funkcionalitású berendezés.

Processzormodulok A MITMÓT-hoz két processzormodul létezik. Az egyszerűbb feladatokhoz szánt modul alapja egy 8 bites Atmel AVR mikrovezérlő, a nagyobb bonyolultságú feladatokhoz használhatóé pedig 32 bites ARM architektúrájú Philips LPC2000 sorozatú mikrovezérlő. Mindkét modul alkalmas valós idejű, beágyazott operációs rendszer futtatására, valamint assembly és C kódú programok fejleszthetők hozzájuk. Az alkalmazásfejlesztéshez két fejlesztőrendszer nyújt segítséget, ebből az egyik otthoni használatra van.

Érzékelő/beavatkozó modulok Az alap érzékelő/beavatkozó modul az ember-gép kapcsolat biztosításához készült, kapcsolók és kijelzők találhatóak rajta. Van továbbá egy akusztikus modul is, ami rendelkezik mikrofonnal, fejhallgató kimenettel illetve analóg be- és kimenetekkel. Egy törpe villanymotorok fordulatszám-szabályozására és modellszervók vezérlésére alkalmas motorvezérlő modul is rendelkezésre áll.

Kommunikációs modulok Kétféle rádiós modul van. Az egyik a 433 MHz-es illetve a 866 MHz-es ISM sávban működik, a másik pedig 2,4 GHz-en, szórt spektrumú kommunikációval. Mindkét rádiós modul hatósugara 100 m, azaz főleg mótok közti kommunikációra alkalmas. Vezetékes alkalmazásokhoz rendelkezésre áll egy 10 Mb/s sebességű Ethernet modul, illetve egy powerline modem, amely a 230 V-os hálózaton biztosít kissebességű adatátvitelt.

Egyéb modulok A MIT tanszéken folyó önálló laboratórium, diplomatervezés és tudományos diákköri tevékenység keretében a tanszék lehetőséget biztosít hallgatói számára a feladatukhoz szükséges új modul fejlesztéséhez [43].

A MITMÓT két processzormoduljához és a kapcsolókat és kijelzőt tartalmazó érzékelő/beavatkozó modulhoz rendelkezésre áll egy C API [44, 45], ami jelentősen megkönnyíti

a kártyára való programfejlesztést, hiszen többek között megvalósítják az alapvető inicializációs, kommunikációs és érzékelés/kijelzés funkciókat.

3.3. eCOS

Az eCOS (embedded Configurable Operating System) a GNU fejlesztőeszközök családjába tartozó, jogdíjmentes, nyílt forráskódú, valós idejű operációs rendszer, beágyazott alkalmazások fejlesztéséhez.

Mivel az eCOS-t kifejezetten valósidejű alkalmazások fejlesztéséhez tervezték, olyan funkciókat biztosít, mint teljes preemptivitás, minimális megszakítás latency; minden szükséges szinkronizációs primitívvel, ütemezési policyvel és interrupt-kezeléssel együtt.

Általános beágyazott alkalmazások fejlesztéséhez az operációs rendszer eszközkezelőket biztosít, valamint támogatja a memóriamenedzsmentet, kivételkezelést, C és matematika könyvtárak használatát.

Az operációs rendszer az alkalmazás pontos igényeihez alakítható, rengeteg opcióval a lehető legjobb futásidejű teljesítmény és minimális hardver szükséglet eléréséhez.

Az eCos nyelve a C, és tartoznak hozzá POSIX és μITRON kompatibilitást biztosító rétegek és API-k is. Számos hardver-platformon futtatható, mint például ARM, Calm-RISC, FR-V, Hitachi H8, IA-32, Motorola 68000, Matsushita AM3x, MIPS, NEC V8xx, Nios II, PowerPC, SPARC és SuperH [45, 46].

A MIT tanszék ezt az operációs rendszert portolta a MITMÓT 32-bites platformra, ezt fogjuk a továbbiakban használni [45].

4. fejezet

Automatikus kódgenerálás MATLAB-Simulink eszközökkel

A MATLAB-Simulink programcsalád kódgenerálási képességeit az elmúlt években egyre több feladat kapcsán próbálták kihasználni.

Ezek közül egy kifejezetten érdekes, részletesen dokumentált példa a szlovéniai Maribor Egyetemen 2002-ben automatikus kódgeneráláshoz készített Simulink blokk-csomag. A blokk-csomagot egy saját fejlesztésű, TITMS320C32 (Texas Instruments) mikroprocesszoron és Xilinx Spartan típusú FPGA-n alapuló DSP kártyához készítették [47, 48].

A Budapesti Műszaki és Gazdaságtudományi Egyetemen 2005-ben a 8-bites MITMÓT AVR kártyához készült blokk-könyvtár, egy TDK dolgozat keretében [49].

A MATLAB-Simulink automatikus kódgenerálás megértéséhez fontos az alapoknál kezdeni: az S-függvényeknél és a TLC nyelvnél, amelyek nélkül a célplatform-specifikus kódgenerálás kivitelezhetetlen lenne. Ezek bemutatása ezen fejezet témája.

4.1. S-függvények

A Simulink könyvtárak bővítésére az S-függvények adnak lehetőséget, ahol az S-függvény egy Simulink blokk valamilyen számítógépes nyelven való leírása. Az S-függvény nevében az S a System rövidítése, így az S-függvény valójában rendszer-függvényt jelent.

Az S-függvények jelentősége abban rejlik, hogy használatukkal saját Simulink-blokkok létrehozása lehetséges, így alkalmazás-specifikus funkciók és viselkedések válnak a Simulink környezetben modellezhetővé.

S-függvényt több nyelven is lehet írni, a jelenleg rendelkezésre álló opciók: MATLAB, C, C++, Ada és Fortran. A C, C++, Ada és Fortran S-függvények MEX (Matlab EXecutable) fájlformátumra fordítódnak (ez Windows operációs rendszer esetén DLL fájlként jelenik

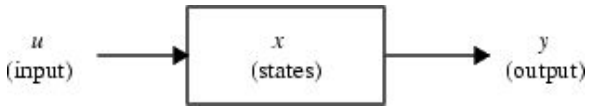
meg). Ezek a MEX fájlok lesznek az S-függvények futtatható kódjai: amikor szükség van rájuk, dinamikusan belinkelődnek a MATLAB környezetbe [50].

Az S-függvények működése nagyban hasonlít a beépített Simulink blokkok működéséhez. Ezt úgy érik el, hogy a Simulink környezettel való interakcióhoz speciális szintaxist használnak, ami a belső blokkok és a Simulink együttműködésével közel megegyező. Egy S-függvény leírása elég általános, alkalmazható folytonos, diszkrét és hibrid rendszerek esetén is.

Az S-függvények formája kötött, több, különböző kidolgozottságú template fájl is rendelkezésre áll, melyeket alapul véve elkészíthető a saját S-függvény. A megírt S-függvényt a Simulink User-Defined Functions könyvtár S-function blokk eleméhez rendelve már kész is a saját blokk. A maszkolás funkciót használva a blokk felhasználói interfészét teljes mértékben az egyedi igényekhez lehet igazítani [50].

4.1.1. A Simulink szimuláció menete

Egy Simulink blokk bemenetek, állapotok és kimenetek halmaza, ahol az egyes kimenetek a mintavételi frekvencia, a bemenetek és a blokk állapotainak függvénye (4.1. ábra).



4.1. ábra. A Simulink blokk matematikája [50]

A viszonyt a bemenetek, kimenetek és állapotok között az alábbi egyenletek határozzák meg:

Output:

$$y = f_0(t, x, u) \tag{4.1}$$

Derivate:

$$\dot{x}_c = f_d(t, x, u) \tag{4.2}$$

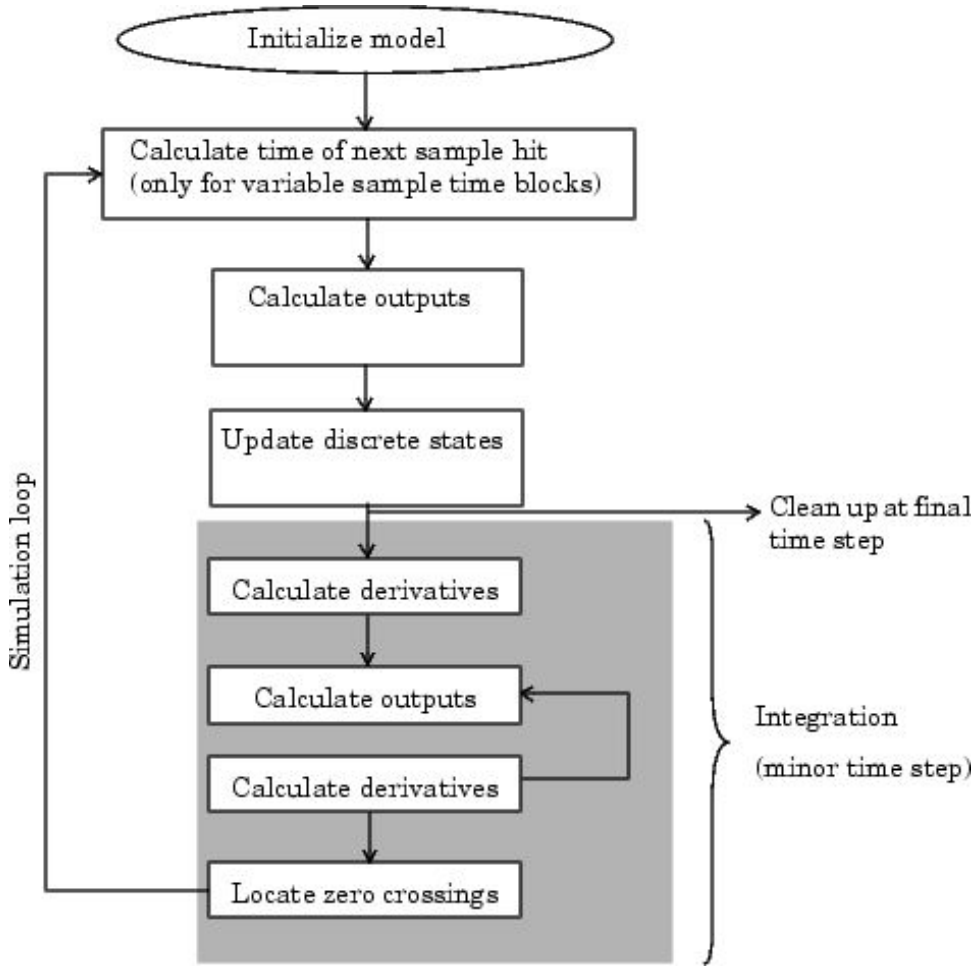
Update:

$$x_{d_{k+1}} = f_u(t, x, u) \tag{4.3}$$

ahol

$$x = x_c + x_d \tag{4.4}$$

Egy Simulink modell esetén a szimuláció végrehajtása szakaszokra bontva történik (4.2. ábra). Az első az inicializációs szakasz, ekkor a Simulink beilleszti a modellben szereplő könyvtári blokkokat, végigvezeti a modellen a jelszélességeket, adattípusokat és mintavételi időket, kiértékeli a blokkparamétereket, meghatározza a blokk végrehajtási sorrendet, és végül lefoglalja a szükséges memóriát.



4.2. ábra. A Simulink szimuláció menete [50]

Az inicializációt követően kezdődik a szimulációs szakasz, azaz a szimulációs hurok, ahol a hurkon való minden végighaladás egy szimulációs lépésnek számít. Minden egyes szimulációs lépés során a modell összes blokkja végrehajtásra kerül, az inicializáció során felállított sorrend alapján. A blokk végrehajtása azt jelenti, hogy a Simulink meghívja a blokk állapotait, deriváltjait és kimeneteit kiszámító függvényeket, az aktuális mintavételi frekvenciának megfelelően.

A szimulációs lépések végrehajtása egészen addig tart, amíg a szimuláció indításakor kijelölt időt el nem érjük, vagy a végrehajtás során valami hiba nem történik [39].

4.1.2. C-MEX S-függvény Callback eljárások

A továbbiakban kizárólag a C nyelven írt S-függvényekről lesz szó, hivatalos nevén, a fordítás utáni MEX fájlformátumra utalva, C-MEX S-függvényekről (a továbbiakban csak S-függvény).

Minden S-függvény számos Callback (meghívó) eljárást tartalmaz, amelyek az egyes szimulációs szakaszok által megkövetelt feladatokhoz kapcsolhatók. A szimuláció során a Simulink minden lépésnél meghívja az adott lépéshez tartozó eljárást a modell minden blokkjára (4.3. ábra).

A Callback eljárás tulajdonképpen egy függvényt jelent, amely a Simulink környezettel való interakciót biztosítja. A Simulink szimuláció lépéseihez hasonlóan a Callback eljárások is külön függvényekre bontva vannak megvalósítva. Egy Callback eljárás olyan általános funkcióhoz kapcsolódik, mint például:

Inicializáció Az első szimulációs lépés megkezdése előtt kerül meghívásra, inicializálja az S-függvényről lényegi információkat hordozó SimStruct struktúrát, beállítja a be- és kimenetek számát és dimenzióját, a blokk mintavételi időket és lefoglalja a szükséges memóriát.

A következő mintavételi idő kiszámítása Amennyiben változó mintavételi idejű egy blokk, ebben a szakaszban kerül kiszámításra, hogy mikor lesz a következő mintavételi ideje – azaz a következő szimulációs lépés ideje a blokkra nézve.

Kimenetek kiszámítása a fő mintavételi időben Amikor ez meghívásra kerül, minden blokk kimenete aktualizálódik az adott szimulációs lépésre nézve.

Diszkrét állapotok frissítése a fő mintavételi időben Ekkor minden diszkrét állapotot tartalmazó blokk frissíti az állapotát. Minden szimulációs lépés során egyszer végrehajtódik.

Integrálás A folytonos idejű modellek esetén, ahol az S-függvénynek vannak folytonos állapotai, úgynevezett „minor step”, azaz a fő lépésköznél kisebb időközönként frissülnek a blokk kimenetei és az állapota.

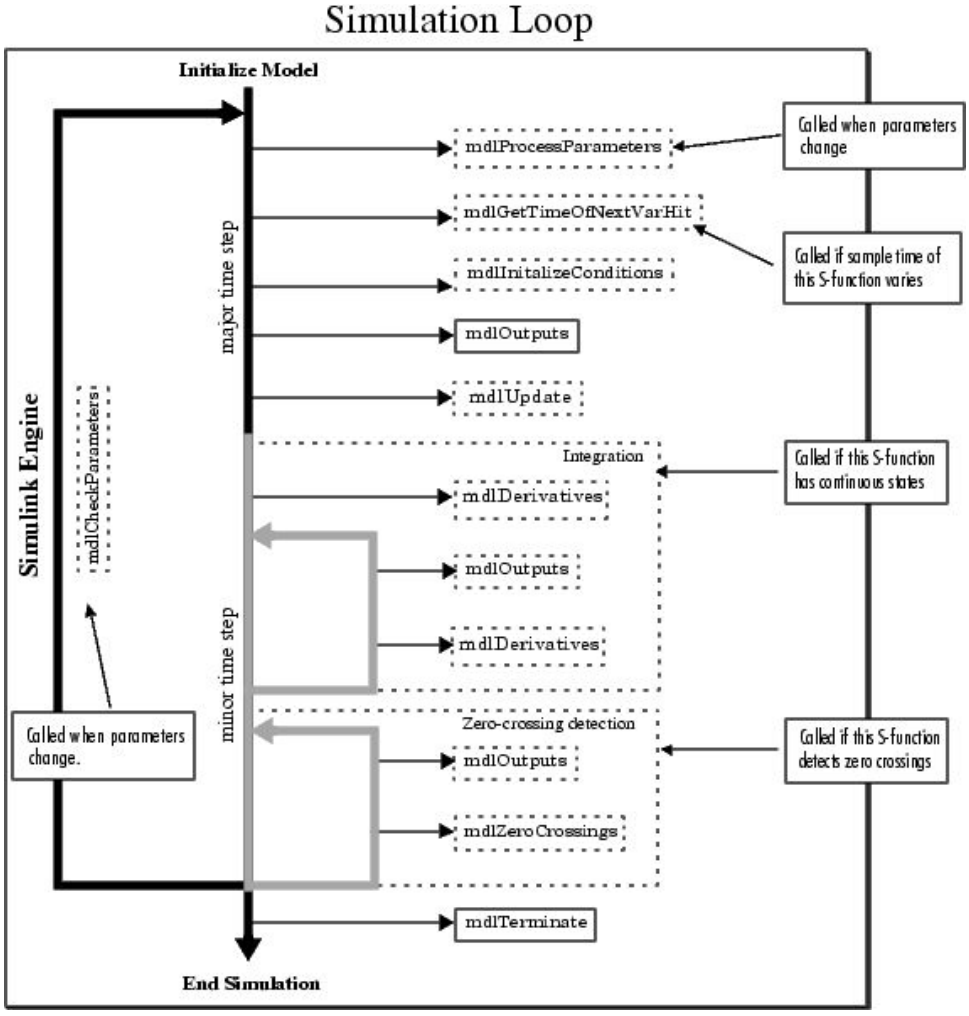
A felsorolt általános funkciókat számos kisebb, specifikus Callback eljárásra lehet lebontani, amelyeket az adott modell igényeit figyelembe véve kell alkalmazni. Azonban van pár olyan Callback eljárás, amelyeket kötelező minden S-függvényben implemetálni, ezek az alábbiak:

mdlInitializeSizes Itt történik a bemenő és kimenő portok számának, az egyes portok méretének és egyéb paraméterek meghatározása, mint például a diszkrét illetve folytonos állapotok száma stb.

mdlInitializeSampleTimes A blokk mintavételezési idejének kijelölése, több opció közül lehet választani, egy ilyen például az öröklött (INHERITED_SAMPLE_TIME) a folytonos (CONTINOUS_SAMPLE_TIME) és a diszkrét (DISCRETE_SAMPLE-_TIME).

mdlOutputs Minden szimulációs lépésnél meghívódik, itt kerülnek kiszámításra a blokk-kimenetek értékei.

mdlTerminate Ez a függvény a szimuláció végeztét követő funkciók helye, amennyiben nincs ilyen funkció, üresen hagyható.



4.3. ábra. S-függvény Callback eljárások a szimuláció során [50]

Ezen kötelező Callback eljárásokon kívül a diplomatémából adódóan, a Real-Time Workshop felé biztosított paraméter-értékátadás és a szimuláció alatti, futási időben történő paraméter-frissítés miatt, még a következő függvényeket kellett használni:

mdlRTW A Real-TimeWorkshop kódgenerálási folyamatához nyújt információkat az S-függvényről (inlined S-függvény esetén, ld. később). Ezt úgy teszi, hogy a modellt leíró model.rtw fájlba egy paraméter rekordot helyez el az S-függvény a szimuláció során nem változtatható (nontunable) paramétereiből, amiket azután a beágyazó (inline) TLC fájl fel tud használni.

mdlCheckParameters Minden esetben meghívódik, ha a felhasználó a szimuláció futása során változtatható tulajdonságú paramétert megváltoztat, itt nyílik lehetőség a változtatás következményeinek meghatározására.

A függvényeknél említett nem változtatható/változtatható (nontunable/tunable) paramétertulajdonság azt jelenti, hogy a szimuláció közben változtatható-e az adott paraméter értéke vagy sem. Ezek alapján, egy paraméter lehet változtatható (tunable) és nem változtatható.

Minden S-függvény kódjának záró része azonos, ez kötött sablon, ami a Simulink/Real-Time Workshop interfészt adja. Ez a záró kódsor:

```
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

Amennyiben egy S-függvényt megpróbálunk enélkül a végső kódrész nélkül használni, a függvény fordításánál hibajelzést kapunk. Ez a lezárás választja ki az aktuális alkalmazásnak megfelelő kódot, a simulink.c csatolódik, ha a fájlt mex formátumra fordítjuk, a cg_sfun.h pedig akkor linkelődik be, ha a Real-Time Workshop használja fel az S-függvényt.

Ugyanígy, minden C-MEX S-függvény első két sora is kötelezően azonos (a harmadik is általában szerepel), ez a séma pedig az alábbi:

```
#define S_FUNCTION_NAME your_sfunction_name_here
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
```

A simstruc.h fájl (matlabroot/simulink/include/simstruc.h) egy C nyelven írott header fájl, ami definiálja a Simulink adatstukturáját, a korábbiakban már említésre került SimStruct-ot, valamint a hozzá kapcsolódó, a SimStruct adatszerkezet elemeihez hozzáférést biztosító makrókat. A SimStruct magában foglalja a Simulink modellhez vagy az S-függvényhez tartozó összes adatot, a blokk paraméterektől kezdve a kimeneteken át.

Egyetlen SimStruct adatstruktúra allokálódik a Simulink modellhez, és ehhez minden, a modellben szereplő S-függvény saját SimStruct-ja kapcsolódik. A SimStruct-ok kapcsolata leginkább egy fa struktúrájú könyvtárszerkezethez hasonlít, ahol a Simulink modellhez tartozó SimStruct a fa gyökere, és az egyes S-függvényekhez tartozó SimStruct-ok a gyerekek. A Simulink számos makrót biztosít, hogy az S-függvények egyszerűen elérhessék a SimStruct adatstruktúra mezőit.

Egy S-függvény leegyszerűsített vázlata (nem tartalmazza az összes kötelezően használandó callback eljárást):

```
#define S_FUNCTION_NAME your_sfunction_name_here
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
```

```
static void mdlInitializeSizes(SimStruct *S)
{
}
<additional S-function routines/code>
static void mdlTerminate(SimStruct *S)
{
}
#ifdef MATLAB_MEX_FILE /* Is this file being compiled as a
MEX-file? */
#include "simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfun.h" /* Code generation registration
function */
#endif
```

C-MEX S-függvényt háromféle módon lehet készíteni:

- A Mathworks által rendelkezésre bocsátott C-MEX S-függvény template fájlok és a Simulink könyvtári blokkjainak S-függvénye alapján kézzel megírni.
- A *S-függvény Építő Varázslót* használva (S-Function Builder), amely a felhasználó által megadott specifikációk (például, hogy 3 bemeneti port legyen, integer típusúak stb.) és kódtöredékek alapján (pl.: u = u+1) automatikusan legenerálja az S-függvény kódját.
- A *Saját Kód Beillesztése* (Legacy Code Tool) funkció segítségével. Ekkor a már meglevő C kódokból, hozzájuk kiegészítő MATLAB M-kódokat és specifikációt csatolva C-MEX S-függvény készíthető egyszerűen.

Természetesen mindhárom megközelítésnek megvan a saját előnye. Minél több beavatkozást igényel egy S-függvény létrehozása, annál inkább testreszabható a függvény, azaz egy kézi kódolású S-függvénnyel funkciók sokkal szélesebb körét lehet megvalósítani, mint egy az *S-függvény Építő Varázslóval* készült verzióval. A sorban a *Saját Kód Beillesztése* az utolsó, vagyis ezzel a módszerrel a legkönnyebb saját S-függvényt írni, de egyben itt lesz a legszűkebb a megvalósítható funkciók listája [50].

A diplomához csak kézzel írott S-függvény készült.

4.2. A TLC nyelv

A TLC (Target Language Compiler, Célkörnyezet Nyelv Fordító) a Real-Time Workshop beépített része. TLC fájlok segítségével lehet a Real-Time Workshop által egy Simulink modellhez generált kódot az egyéni igényekhez igazítani. Ilyen egyéni változtatásokkal platform-specifikus kód generálható, beilleszthető saját algoritmus, például a teljesítmény

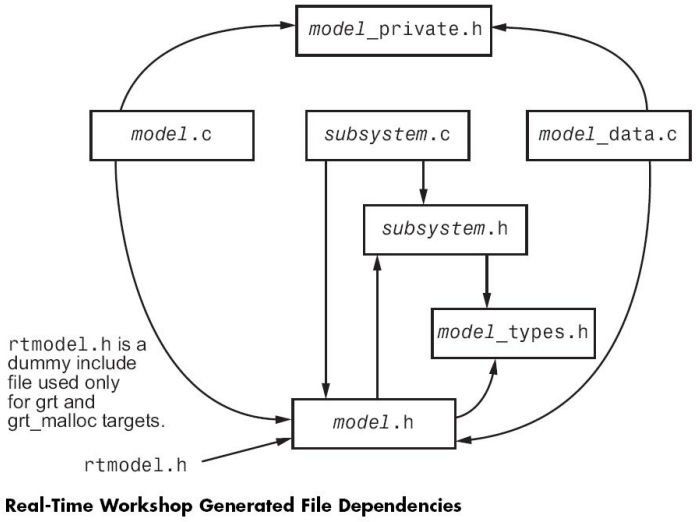
és kódméret optimalizálására, vagy akár a már meglevő eljárásokkal való kompatibilitás is biztosítható.

A TLC-hez kapcsolódóan a MATLAB könyvtárstruktúrában elérhető:

- Minden egyes Simulink blokkhoz tartozó TLC fájl,
- A teljes modell alapján generált kódra ható TLC fájlok.

A TLC fájlok olyan ASCII fájlok, amelyek explicit vezérlik, hogy a Real-Time Workshop az adott Simulink modellből hogyan generáljon kódot. Egy TLC kód vonatkozhat egy adott blokkra vagy a teljes modellre – például egy Simulink blokk TLC fájljának módosításával változtatható az adott blokkhoz generált kód.

A MathWorks által rendelkezésre bocsátott Simulink könyvtári blokkok mindegyikéhez tartozik TLC fájl is, de a TLC fájl létezése nem alapfeltétel egy Simulink blokk létezéséhez, a TLC fájl jelentősége csak és kizárólag a kódgeneráláshoz kapcsolódik.



4.4. ábra. A Real-Time Workshop generálta fájlok függőségi kapcsolata [40]

Mindamellett, a TLC egyetlen célra lett tervezve: hogy a modell leíró model.rtw fájlt (lásd következő alfejezet) target-specifikus kóddá vagy szöveggé konvertálja. A model.rtw fájl a Simulink blokk diagram egy köztes állapota, már valamiféle fordított reprezentációja a modellnek, ami egy nagyon magasszintű nyelven leírja a blokkdiagram végrehajtási szemantikáját. Ebből a köztes leírásból csinál a TLC C vagy C++ kódot [51]. A Real-Time Workshop generálta kódokról és azok egymástól való függőségéről mutat a 4.4. ábra egy összefoglalást.

A TLC nevében a Target (cél) szó is szerepel, ennek a jelentése sem elhanyagolható. A TLC esetén a target nem csak azt a magasszintű nyelvet jelenti, ami a kimenet lesz, hanem egyben azt a valósidejű rendszerkörnyezetet is, ahol a kódot majd alkalmazni fog-

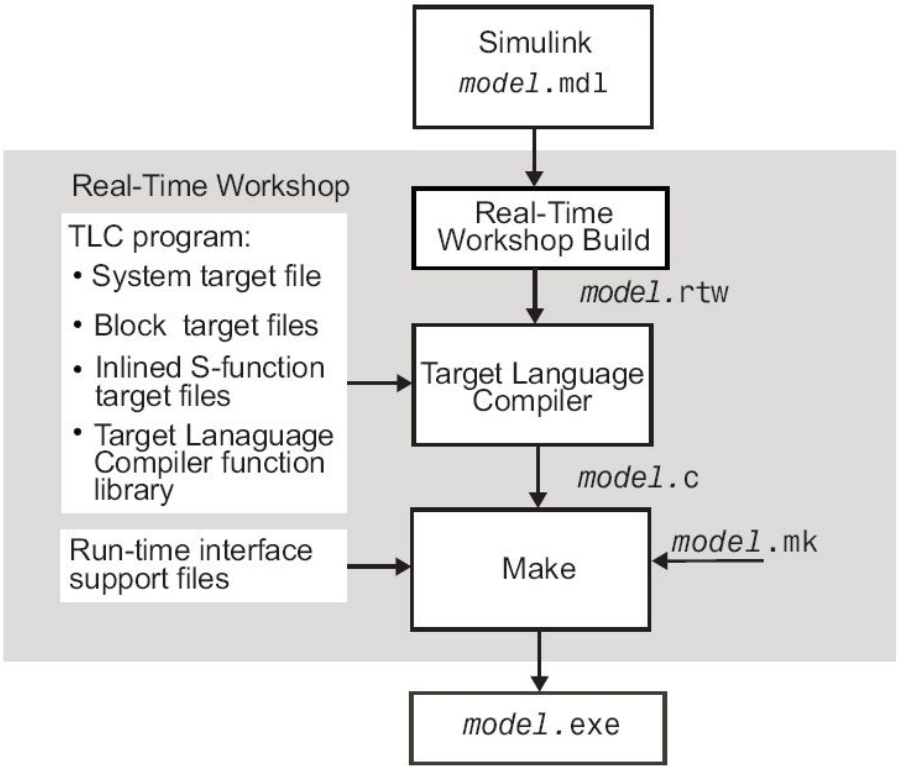
ják. A TLC-fájlok segítségével generált kód képes figyelembe venni a speciális processzor architektúrák korlátait és kihasználni annak lehetőségeit.

A Real-Time Workshop a kódot (a model.rtw beolvasása után) a célkörnyezet-specifikus (target) fájlok és a teljes modellre vonatkozó TLC fájlok alapján generálja. A target fájlok határozzák meg minden blokkhoz az egyéni kódot, a modell hatáskörű fájlok pedig az átfogó kódstílust.

A TLC nyelvnek bizonyos hasonlóságai vannak a HTML, Perl és MATLAB nyelvekkel is. A markup szintaxis hasonló a HTML-hez, a teljesítmény és rugalmasság kapcsán a Perl és egyéb script nyelvekhez hasonlít, adatkezelés terén pedig a MATLAB erejével bír (a TLC meghívhat MATLAB függvényeket). A TLC-vel generált kód erőteljesen optimizált és jól kommentezett.

Függetlenül attól, hogy lineáris, nemlineáris, folytonos, diszkrét vagy hibrid idejű, bármilyen Simulink modellből lehet TLC fájlok használatával kódot generálni. Minden Simulink blokk, a MATLAB függvény blokkok és az M-fájlokat meghívó S-függvényeket kivéve, automatikusan kóddá konvertálódik.

A Real-Time Workshop kódgenerálási folyamatában a TLC szerepét jól mutatja a 4.5. ábra.



4.5. ábra. A Target Language Compiler helye a kódgenerálási folyamatban [51]

4.2.1. A modell.rtw fájl

A Real-Time Workshop kódgenerálási folyamata során a TLC fájlok a model.rtw fájl alapján dolgoznak. A model.rtw fájl a Simulink modell (model.mdl) fájl köztes fordítása, ami leírja a modellben szereplő blokkokat, azok ki- és bemeneteit, paramétereit, állapotait és egyéb, a modellhez kapcsolódó jellegzetességeket.

A Simulink modellből a model.rtw-t a Real-Time Workshop generálja, a fájl tulajdonképpen egy adatbázis, aminek a tartalma lényegi információkat hordoz a Simulink modell minden egyes blokkjáról. A modellhez generált model.rtw fájl megtekintésére van lehetőség: a *Configuration Parameters* párbeszéd ablak *Real-Time Workshop* felületén a *TLC debugging category* fül alatt be lehet állítani, hogy a modellhez generált model.rtw fájlt is megkapjuk a kódgenerálás végén.

A modell.rtw fájl alapstruktúrája a következő:

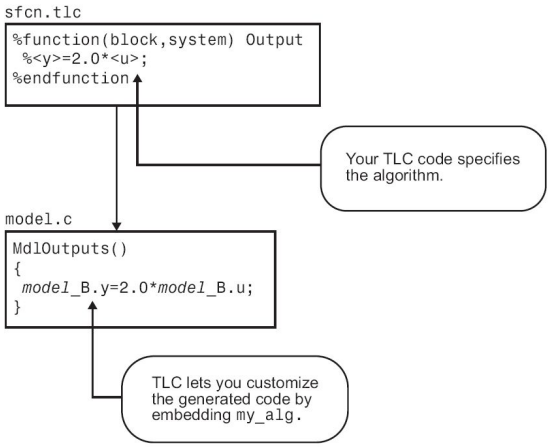
```
CompiledModel {
  System {
    Block {
      DataInputPort {
      ...
      }
      DataOutputPort{
      ...
      }
      ParamSettings {
      ...
      }
      Parameter {
      ...
      }
    }
  }
}
```

Egy blokk a modellben való minden előfordulásáról egy hozzá tartozó blokk-rekord kerül a model.rtw fájlba. A system target fájl TLC kód, amely a teljes rendszer kódjáért felel, végighalad az összes blokk-rekordon és meghívja az adott blokkhoz tartozó blokk-target fájlban szereplő függvényeket.

4.2.2. A TLC és az S-függvények összekapcsolása

Amikor a Real-Time Workshop-pal egy modellhez kódot kell generálni, az automatizált eljárás első lépése a model.rtw fájl létrehozása. Ez a fájl tartalmazza az összes, a Simulink modellből való kódgeneráláshoz szükséges modell-specifikus információt. Ezt követően

a model.rtw fájl a Target Language Compiler-hez kerül, ami ezt és egy sor hivatkozott rendszer- és blokk-target fájlt feldolgozva generálja a kódot.



4.6. ábra. Inlined S-függvény [51]

Az a Simulink modell, amiből kódot akarunk generálni, nem feltétlenül építhető fel kizárólag alap Simulink blokkokból. Amennyiben valamilyen speciális feladatot is modellezni kell, az előzőekben bemutatott módon S-függvényekkel saját blokkok létrehozása válik szükségessé. A saját blokk jelen esetben C-MEX S-függvényt jelent.

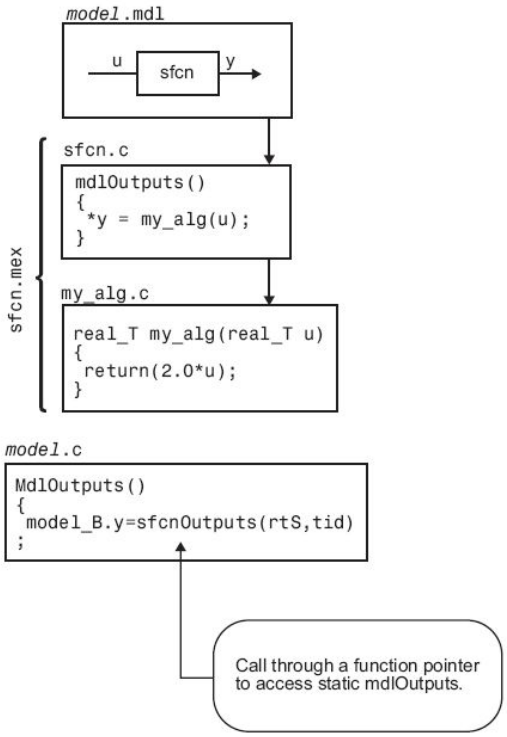
A C-MEX S-függvényekhez lehet blokk-target TLC fáljt is írni, az S-függvényből generált kód testreszabásához. A blokk target TLC fájl típusától függően beszélünk „beágyazott" (inlined, 4.6. ábra) vagy „nem beágyazott" (noninlined, 4.7. ábra) S-függvényről.

A Real-Time Workshop egy generikus API-t biztosít, amin keresztül hozzá lehet férni a felhasználó által írt algoritmusokhoz és driverekhez. Ez az API-elérés a különböző Callback függvények használatával valósul meg.

Egy általános valós-idejű alkalmazás esetén, főképp, ha sok végrehajtási lépésből áll, az API hívásának plusz költsége megengedhetetlen. Ennek áthidalására született meg az S-függvény beágyazási (inline) módszer, ekkor ugyanis az S-függvény kódja közvetlenül bekerül a generált kódba, nem kellenek API hívások az eléréséhez. A beágyazott S-függvény így gyorsabb és optimálisabb kódhoz vezet.

Tehát az S-függvény inline lehetővé teszi, hogy az S-függvény blokkjának kódját egy az egyben be lehessen ágyazni a TLC-generált kódba. Enélkül a megközelítés nélkül a generált kódban az S-függvény funkcióit csak a forrás C-MEX fájl függvénymutatókon keresztüli hívásaival lehet elérni, és emiatt az S-függvény teljes saját adatszerkezetét (SimStruct) is fenn kell tartani hozzá.

Természetesen az S-függvényből akkor is lehet kódot generálni, ha nincs hozzá TLC fájl. Ekkor, a noninlined esethez hasonlóan az S-függvény C vagy C++ kódját függvényhívásokon keresztül éri el a generált kód, ami jelentős overhead. A noninlined eset annyiban más, mint a TLC kód nélküli, hogy a noninlined S-függvény TLC kódját használva azért



4.7. ábra. Noninlined S-függvény [51]

valamilyen szintű beágyazás megtörténik, csak továbbra is maradnak más fájl felé irányuló függvényhívások.

Összefoglalva, az S-függvény beágyazásával növelhető a teljesítmény, mivel a generált kódban megszünteti az S-függvény kódja felé irányuló függvényhívásokat és az S-függvényhez tartozó SimStruct felesleges memóriafoglalását.

A TLC minden esetben beágyazza az S-függvényt, ha az S-függvény fájl nevével megegyező nevű .tlc fájlt talál (4.8 ábra). Amennyiben a tlc fájl helyesen formázott, az S-függvényhez elkészül a beágyazott kód [51].

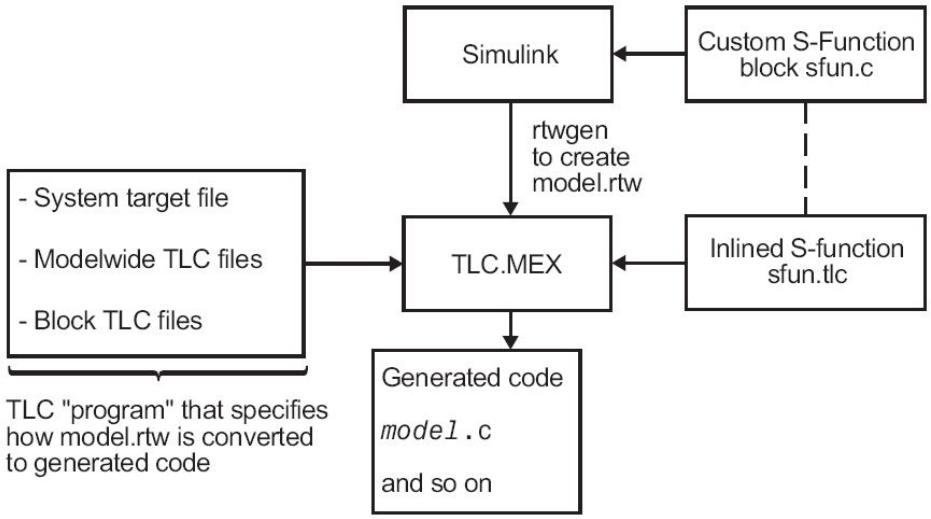
4.2.3. A TLC nyelv szintaktikája

A TLC nyelv szintaktikájának fő jellegzetessége, hogy minden TLC parancs előtt % jel áll. A TLC nyelv részletes bemutatása előtt pár, a szintaxisra és kódolási konvenciókra vonatkozó hasznos alapismeret:

%% Comment TLC komment, ami nem kerül ki semmilyen kimenetre

/* comment */ Olyan komment, ami a kimenetre kerül

%keyword minden TLC direktíva (kulcsszó) "%" jellel kezdődik



4.8. ábra. TLC [51]

%<expr> TLC token operátor

. (pont két változónév között) érvényességi tartományt kijelölő operátor, például Top.Level2.Level3

... (sor végén) a soron túllógó állítás folytatása, a kimeneten a helyére nem kerül sortörés

\(sor végén) a soron túllógó állítás folytatása, a kimeneten a helyén sortörés lesz

localvarIdentifier a lokális változók nevei kisbetűvel kezdődnek

GlobalvarIdentifier a globális változók nevének első betűje mindig nagy

RecordIdentifier rekord azonosítók mindig nagybetűvel kezdődnek

EXISTS() a beépített TLC függvények nevei csupa nagybetűvel íródnak

Egy további fontos tulajdonsága a nyelvnek, hogy a kis- és nagybetű különbségre érzékeny (case-sensitive).

A következőkben a TLC nyelv kicsit részletesebb bemutatása a cél, a kiemelten fontos parancsok ismertetésével. Ezen parancsok használata a diplomafeladat megvalósításához nélkülözhetetlen volt.

A fontosabb parancsok:

%addincludepath Ezen paranccsal lehet a TLC fájlnak megmondani a belinkelendő fájlok elérési útvonalát.

%assign Változó létrehozáshoz és módosításhoz.

%foreach/%endforeach Lépteti a ciklusváltozót 0 és a felső határként megadott érték között.

%if/%endif Jelentése megegyező a C nyelvben használt if paranccsal, azaz eldönti, hogy egy adott kódrészlet végrehajtódjon vagy sem.

%include Belinkel egy fájlt a meghívó fájlba, a C nyelvtől eltérően a TLC include egyben végrehajtást is jelent.

%selectfile Direkt kimenet kijelölése, a kimenet lehet fájl vagy buffer.

%with/%endwith Az érvényességi tartomány módosítása, például root.blokk.változó helyett elég lesz a változóra hivatkozni.

%implements Ez a direktíva kötelező minden TLC fájl esetén, a Target Language Compiler ez alapján ellenőrzi a pontos blokk típust és a blokk által támogatott nyelvet. Egy MITMÓT-os kapcsolót megvalósító TLC fájl első nem komment sora például a következő:

```
%implements mmSwitch "C"
```

%function/%endfunction Függvény deklarációnál használandó, minden TLC függvény %function direktívával kell kezdődjön, és a végén az %endfunction kulcsszónak kell szerepelnie. A két parancs között van a függvény törzse, ide kerül a függvény által megvalósítandó funkciók kódja. A %function direktíva után a függvény neve következik, utána zárójelben az átadott paraméterek, végül a kimenet kijelölése történik. Ha kimenetnek az Output van kijelölve, minden, ami nem TLC direktíva, a függvény kimenete lesz - a TLC kódban aktuálisan kijelölt fájlba. Ez azt jelenti, hogy minden nem TLC parancssor a függvény által generált kód lesz [51]. A direktíva használatára példa:

```
%function Outputs(block, system) Output
%endfunction
```

4.2.4. Blokk target TLC függvények

A rendszer-target fájl az egyes blokk-target fájlokban deklarált függvényeket hívja meg kódgenerálásnál. A blokk-target fájlok függvényei adottak, a Simulink C-MEX S-függvény Callback eljárásokkal párhuzamba állíthatók. Az egyes függvények argumentumai minden esetben block és system. A block a Simulink blokk nevére utal (például a Gain nevű blokk esetén a block argumentum *gain* lesz) a system meg a Simulink modell azon rendszere/alrendszere, ahol a blokk van.

Az alább felsorolandó függvények közül az első kettő alapesetben nem generál semmiféle kódot, ezek az előfeldolgozáshoz és alapvető beállításokhoz szükségesek. Az utánuk következő függvények már mind aktívan részt vesznek a blokkhoz generált kód kialakításában, amit majd aztán a Real-Time Workshop a megfelelő helyen elhelyez.

BlockInstanceSetup(block, system) Ez a függvény egy blokk-típusnak a modellben levő minden példányára meghívódik, amennyiben az adott blokk-típus tartalmazza a függvényt a blokk-target fájljában. Azaz például ha az aktuális Simulink modellünkben van három LED blokk, és van a LED blokk blokk-target fájljában *BlockInstanceSetup(block, system)* függvény definiálva, akkor az abban foglaltak mind a három blokkra egyenként végrehajtódnak.

BlockTypeSetup(block, system) Minden blokk típusra egyszer hajtódik végre, azaz ha ugyanabból a típusú blokkból több is szerepel a modellben (pl. 5 darab LED blokk van), csak egyszer hívódik meg.

Enable(block, system) A Real-Time Workshop minden esetben, amikor egy nemvirtuális Simulink alrendszerben Enable függvényt tartalmazó blokk van. Ha a blokk target fájljában van Enable függvény, akkor az itt keletkező blokk/specifikus engedélyezés kódja automatikusan bekerül a blokkot magában foglaló alrendszer Enable függvényének kódjába.

Disable(block, system) Egy nemvirtuális alrendszernek minden olyan esetben lesz Disable függvénye, amikor egy Simulink alrendszerben Disable függvényt tartalmazó blokk van. Ha a blokk target fájljában van Disable függvény, akkor az itt keletkező blokk specifikus letiltó kódja automatikusan bekerül a blokkot magában foglaló alrendszer Disable függvényének kódjába.

Start(block, system) Erre a függvényre akkor van szükség, ha a Start függvényben kell kódot elhelyezni. Az ide kerülő kód egyetlen egyszer hajtódik végre. Általában olyan funkcióknak van itt a helye, amelyeket a szimuláció legelején egyszer végre kell hajtani (például a munkaváltozók értékeinek inicializálása) vagy amit az alrendszerre csak egyszer kell végrehajtani, és nem kell újra lefuttatni minden engedélyezésnél.

InitializeConditions(block, system) Az innen generált kód két helyre kerülhet: ha a blokk olyan nemvirtuális alrendszerben van, ami úgy van konfigurálva, hogy minden újraengedélyezés esetén végre kell az inicializálást hajtania, akkor a InitializeConditions kódja az alrendszer Initialize függvényében kap helyet, és a Start függvény majd ezt az Initialize függvényt fogja meghívni. Egyébként, ha a blokk a modell gyökerében van, vagy olyan nemvirtuális alrendszerben, ami nem használja az initialize függvényt, a blokk függvényéből generált kód a Start függvénybe kerül be (inline). A Start és az InitializeConditions függvények szerepe hasonló, a különbség abban rejlik, hogy a Start függvénybe azok a funkciók kerülnek, amiket a blokkot magába foglaló alrendszer újraengedélyezésénél nem kell ismét végrehajtani. Ezzel ellentétben, az InitializeConditions függvénybe kerül az a kódrész, amit ilyen esetben is végre kell hajtani.

Outputs(block, system) Ez a függvény az, amit általában minden blokk target fájl tartalmaz. A függvény végrehajtása során generált kód a Simulink modell Outputs függvényébe kerül, illetve ha a blokk nemvirtuális alrendszer tagja, akkor ezen alrendszer Output függvényébe.

Update(block, system) Ha a blokknak van olyan kódja, amit minden fő szimulációs lépésben frissíteni kell, akkor annak ide kell kerülnie. Az itt generált kód a modell vagy a blokkot tartalmazó alrendszer Update függvényébe kerül, attól függően, hogy a blokk nemvirtuális alrendszerben van-e.

Derivatives(block, system) A Derivate függvényre akkor van szükség, ha a blokk folytonos állapotát ki kell számítani. A kód a modell, vagy ha a blokk alrendszer tagja, az alrendszer Derivatives függvényének kódjába kerül.

Terminate(block, system) Az itt helyet kapó kód a hozzá tartozó S-függvény MdlTerminate függvényébe kerül. Hasznos lehet, ha az S-függvényhez kapcsolódóan szükség van adatmentésre, memória felszabadításra vagy hardver reset-re.

A teljes kódgenerálási rendszert a 4.9 ábra mutatja.

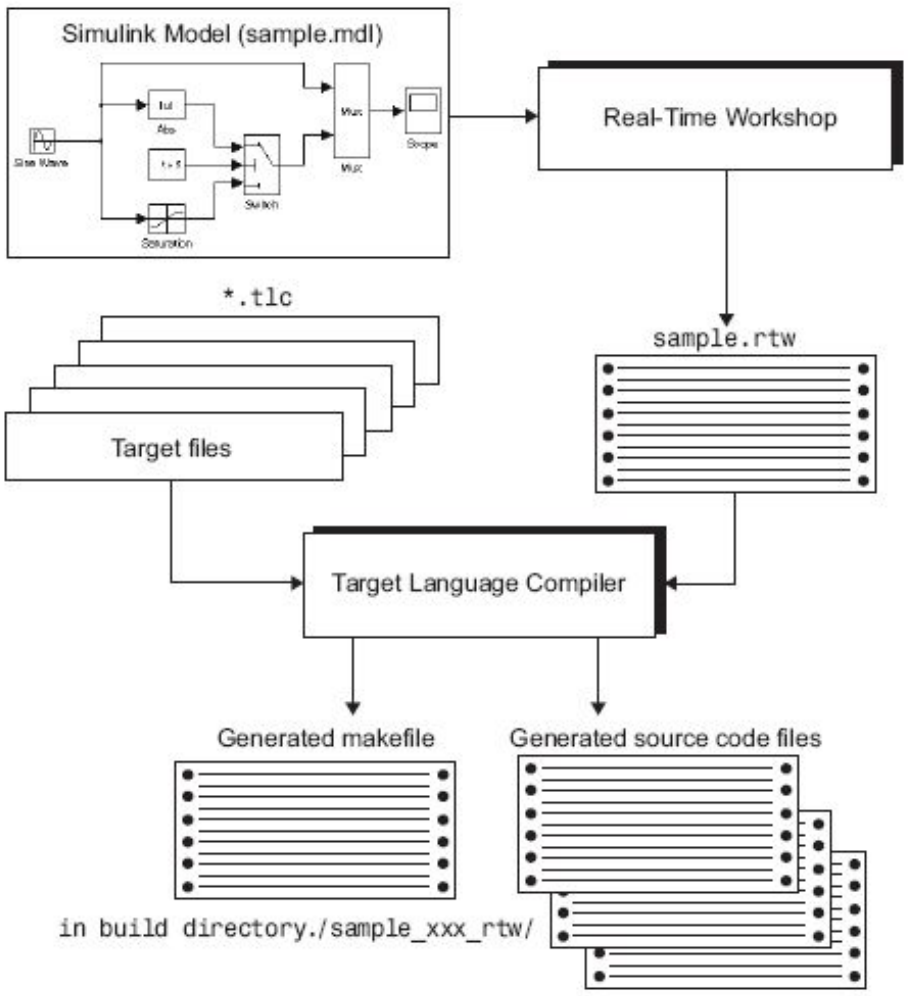
Egy beágyazott device driver TLC kódjához az előbb felsorolt függvényeknél általában lényegesen kevesebb kell. A diplomafeladat megoldásához az alábbi TLC block target függvényeket használtam:

- Start(block, system): az I/O perifériák inicializálásához kapcsolódó műveletek kerültek ide, például a hőmérő egység esetén a hőmérő eszköz inicializálását végző kód.
- Outputs(block, system): a blokk kimeneteinek kiértékelése itt történik. Ez a rész függ a megvalósított driver típusától, azaz egy bemeneti eszköz, mint például a kapcsoló, általában egy I/O eszköz kimenetének értékét olvassa be, majd ezt képezi le a saját y kimeneti vektorába. Kimeneti driver esetén, mint például a DAC, a helyzet pont fordított, itt a blokk a bemeneti u vektorának értékét írja ki egy I/O perifériára.

Ezeken kívül még felmerült a lehetősége a Terminate(block, system) függvény használatának, de mivel a tervezett alkalmazások futási ideje végtelenített, erre nem volt szükség [51].

4.2.5. System target fájlok

A TLC program ASCII fájlok összessége, amiket más néven scripteknek is hívnak. Az a target fájl, ami meghívja az %include direktívával az összes többi, a programhoz szükséges target fájlt, a belépési pont (entry point). A modell-hatókörű target fájlokat a teljes modell vonatkozásában használjuk, alap információkat szolgáltatnak a TLC-nek a model.rtw fájl target-specifikus kóddá való fordításához.



4.9. ábra. Kódgenerálás a Target Language Compiler használatával [51]

A system target fájl (System Target File, STF) a TLC belépési pontja. Leginkább a C program main() függvényéhez hasonlítható, bár felépítése és jellege egészen más, csak a funkció hasonló. A system target fájl az, ami a teljes kódgenerációs folyamatot felügyeli, meghatározza, hogy az egyes blokkok mikor kerülnek végrehajtásra, hogyan valósuljon meg adatgyűjtés és még sok minden más.

Egy system target fájl általános felépítése (amit az idézett kódrészlet után részletesen elemezni fogunk):

```
%%----------------------------
%% Header Comments Section
%%----------------------------
%% SYSTLC: Example Real-Time Target
%% TMF: my_target.tmf MAKE: make_rtw EXTMODE: ext_comm
%% Inital comments contain directives for STF Browser.
%% Documentation, date, copyright, and other info may follow.
```

```
...
%selectfile NULL_FILE
...
%%-----------------------------
%% TLC Configuration Variables Section
%%-----------------------------
%% Assign code format, language, target type.
%%
%assign CodeFormat = "Embedded-C"
%assign TargetType = "RT"
%assign Language = "C"
%%
%%-----------------------------
%% (OPTIONAL) Import Target Settings
%%-----------------------------
%include "mytarget_settings.tlc"
%%
%%-----------------------------
%% TLC Program Entry Point
%%-----------------------------
%% Call entry point function.
%include "codegenentry.tlc"
%%
%%-----------------------------
%% (OPTIONAL) Generate Files for Build Process
%%-----------------------------
%include "mytarget_genfiles.tlc"
%%-----------------------------
%% RTW_OPTIONS Section
%%-----------------------------
/%
BEGIN_RTW_OPTIONS
%% Define rtwoptions structure array. This array defines target-specific
%% code generation variables, and controls how they are displayed.
rtwoptions(1).prompt = 'example code generation options';
...
rtwoptions(6).prompt = 'Show eliminated blocks';
rtwoptions(6).type = 'Checkbox';
...
%---------------------------------------%
% Configure RTW code generation settings %
%---------------------------------------%
...
%%-----------------------------
%% rtwgensettings Structure
%%-----------------------------
%% Define suffix string for naming build directory here.
rtwgensettings.BuildDirSuffix = '_mytarget_rtw'
```

```
%% (OPTIONAL) target inheritance declaration
rtwgensettings.DerivedFrom = 'ert.tlc';
%% (OPTIONAL) r14 callback compatibility declaration
rtwgensettings.Version = '1';
%% (OPTIONAL) other rtwGenSettings fields...
...
END_RTW_OPTIONS
%/
%%----------------------------
%% targetComponentClass - MATHWORKS INTERNAL USE ONLY
%% REMOVE NEXT SECTION FROM USER_DEFINED CUSTOM TARGETS
%%----------------------------
/%
BEGIN_CONFIGSET_TARGET_COMPONENT
targetComponentClass = 'Simulink.ERTTargetCC';
END_CONFIGSET_TARGET_COMPONENT
%/
```

Ha egy már létező system target fájl alapján akarjuk a sajátunkat elkészíteni, az első lépés a sablon utolsó három sorának törlése, mert ezt csak a Mathworks cég saját belső fejlesztésű targetjei esetén lehet használni. Erre minden Mathworks által rendelkezésre bocsátott STF fájl figyelmeztet is, kommentben szerepel, hogy az adott részt törölni kell [51, 52].

Az STF fejléce (az előző kód első 7 sora) TLC kommentnek tűnik, de valójában ez a fájl igen fontos része, mert ebből a szabványos headerből fogja a System Target File Browser tudni, hogy egy system target fájlról van szó. A kódgenerálási folyamatnál mindig ki kell egy STF-et jelölni, ellenben nem jelölhetünk ki akármilyen fájlt, csak a System Target File Browser által mutatottakat, ezért a helyes fejléc nélkülözhetetlen. A saját STF-et a Browser csak akkor fogja megtalálni, ha a fájl elérési útja a MATLAB Path-hoz lett adva.

A system target fájl fejlécének elemei:

SYSTLC Ez az a leírás, ami a System Target File Browserben megjelenik.

TMF A build folyamat során alkalmazandó template makefile (TMF). Amennyiben szerepel itt valami, az az STF kijelölése után automatikusan a *Configuration Parameters Real-Time Workshop* paneljének *Template makefile* mezőjébe kerül.

MAKE A build processzhez használt make parancs. Az STF kiválasztása esetén automatikusan bekerül a *Configuration Parameters* ablak *Real-Time Workshop* paneljének *make command* mezőjébe.

EXTMODE A külső módhoz (external mode) – amikor a Simulinkben futó modell és a célrendszeren futó kód között kommunikációs kapcsolat van – használt fájl neve (amennyiben van ilyen). Ha a target nem támogatja, a no_ext_comm leíró használata javasolt.

A MITMOT ARM target fájlhoz nem tartozik template makefile, és az external mód sem támogatott, így ezek a mezők üresen maradtak a fejlécben.

Általában bármilyen beágyazott targethez jó kiindulási alap a Real-Time Workshop Embedded Coder system target fájlja (ert.tlc). Az alap konfigurációs változók kijelölésénél az alábbi szempontokat kell figyelembe venni:

CodeFormat Ezzel a paraméterrel lehet az alkalmazandó kódstílust kiválasztani. Beágyazott target esetén az Embedded-C a javasolt opció, mert ez a kódstílus kifejezetten a beágyazott rendszerek szükségleteit figyelembe véve alakítja a kódot: minimális memóriafelhasználás, statikus memóriafoglalás, egyszerű interfész a generált kódhoz.

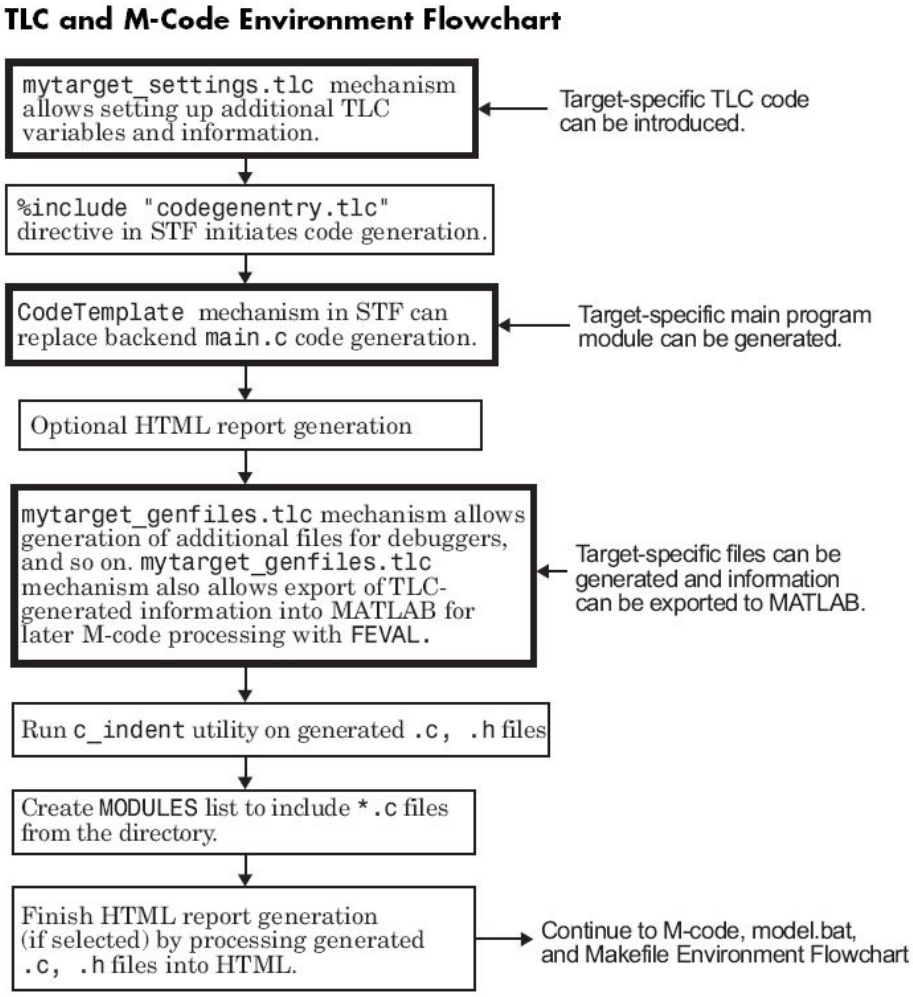
Language Az egyetlen helyes érték ide a C, ami a C és C++ nyelvhez is támogatja a kódgenerálást, a kettő közül TargetLang konfigurációs paraméter értékétől függően választ.

TargetType RT (Real-Time) vagy NRT (Non Real-Time) lehet, előfeldolgozáshoz szükséges változó, a generálandó kód típusát hivatott eldönteni. Beágyazott target esetén az RT a helyes választás.

A kódgenerálási folyamat normális esetben a codegenentry.tlc fájl belinkelésével kezdődik, azaz ahol a system target fájlban a következő utasításhoz érünk:

```
%include "codegenentry.tlc"
```

Ha egyéb target-specifikus kódgenerálási funkciókra is szükség van, azt általában a mytarget_settings.tlc és mytarget_genfiles.tlc target fájlok system fájlba való belinkelésével lehet megvalósítani. Természetesen bármilyen más nevű tlc fájl is alkalmas a feladatra, ez csak a Mathworks által ajánlott struktúra [53] (4.10. ábra).



4.10. ábra. TLC kódgenerálási folyamat [53]

5. fejezet

A MITMÓT ARM target

A MITMÓT ARM target a MITMÓT-specifikus kódgeneráláshoz készült Simulink blokkok (C-MEX S-függvények) és TLC fájlok összességét jelenti, azaz mindazon eszközöket, amelyek a fejlesztői környezetet alapját képezik.

A MITMÓT ARM Simulink blokk-könyvtár blokkjait használva a felhasználó által készített Simulink modellből automatikusan a MITMÓT ARM kártyán futó, eCOS operációs rendszert használó C-kódot lehet generálni.

A TLC kódgenerálás és saját target fejlesztésének MathWorks dokumentációja igen szerteágazó és bőséges, ám valós használatban nehezen alkalmazható. A dokumentáció készítésénél, valószínűleg a bőség zavarában, nem tudtak egy jól felépített hierarchiát kialakítani, és a végeredmény inkább a részletekben elvesző leírás lett. Emiatt a már meglévő és működő targetek fájljainak hosszas tanulmányozása elengedhetetlen volt.

Eredetileg a VxWorks Tornado targetből való kiindulás tűnt a leglogikusabbnak, mivel ez az egyetlen jelenleg elérhető olyan target, amelyik operációs rendszert használ. Végül a Tornado target, felépítése folytán nem volt alkalmas a MITMÓT target fejlesztésének segítéséhez, így egy másik, használható targetet kellett keresni, ami végül a TI C6000-es DSP lett.

Ehhez a targethez a MathWorks teljes blokk-könyvtárat biztosít, TLC forrásfájlokkal együtt, melyek tanulmányozása sokat segített a feladat megoldásában.

A TI 6000C target operációs rendszer nélküli használathoz készült, így a MITMÓT ARM - eCOS target esetén jelentős struktúra módosításokra volt szükség.

5.1. A MITMÓT ARM system target fájl

A MITMÓT ARM system target fájl az ERT (Embedded Real-Time) STF mintájára készült, azaz az ERT system target fájlt választva látott opciók jelennek meg a MITMÓT STF választása esetén is.

A MITMÓT ARM modul system target fájlja a mm_arm.tlc. A target fájl fejléce:

```
%% SYSTLC: MITMOT ARM Real-Time Target TMF: - MAKE: make_rtw \
%%   EXTMODE: no_ext_comm
%%
%% $RCSfile: mm_arm.tlc,v $
%%
%% $Date: 2007/04/25 19:48:09 $
%%
%% by Laura Fabian
%% Abstract: MITMOT ARM + eCOS real-time system target file.
%%
```

Ez a szükséges template, hogy az mm_arm.tlc megjelenjen a System Target File Browserben, mint választható system target fájl. Ezután a kódgenerálás nyelvének és típusának kiválasztása történik:

```
%selectfile NULL_FILE

%assign TargetName = "MITMOT ARM"
%assign TargetType = "RT"
%assign Language   = "C"
%assign CodeFormat = "Embedded-C"
```

Ebből kiderül, hogy a target neve MITMOT ARM, a target Real-Time környezet, C (vagy C++) nyelvű kód legyen a végeredmény, és a kód Embedded-C formában készüljön, azaz beágyazott rendszerekre optimalizált módon.

A MITMÓT-specifikus kódgenerálás igazi kezdete egy egyszerű include utasítás, ami szinte elveszik a többi sor között:

```
%assign ERTCustomFileTemplate = "mm_arm_template.tlc"
```

Az mm_arm_template.tlc meghív további MITMÓT ARM – eCOS specifikus TLC fájlokat, amelyek a Real-Time Workshop által generált kódot a MITMÓT környezet igényeihez igazítják. Ennek részletes kifejtése a következő alfejezetben található.

A tényleges kódgenerálási folyamat kezdetét az alábbi sor jelzi:

```
%include "codegenentry.tlc"
```

A codegenentry.tlc három másik TLC fájlt include-ol, amelyek a MATLAB - Simulink - Real-Time Workshop kódgeneráláshoz általános környezeti és elérési beállításokért felelnek.

A MITMÓT ARM STF záró sorai pedig a kötelező formula:

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

  rtwgensettings.BuildDirSuffix = '_mm_arm_rtw';
```

```
  rtwgensettings.DerivedFrom = 'ert.tlc';
  rtwgensettings.Version = '1';
  rtwgensettings.UsingMalloc    = 'if_RealTimeMalloc';

  END_RTW_OPTIONS
 %/

%%[EOF] mm_arm.tlc
```

5.2. Segédfüggvények a kódgeneráláshoz

A kódgeneráláshoz három segédfüggvény készült, ezek: mm_arm_ertmain.tlc, mm _arm _template.tlc és mm_arm_main_common.tlc. Most ezen három segédfüggvény bemutatása következik.

5.2.1. mm_arm_template.tlc

Ez a fájl a MITMÓT ARM – eCOS specifikus main függvény generálását indítja el. Mindezt az mm_arm_ertmain.tlc fájl include-jával, illetve az include-olt fálj *SLibCreateMMARMERTMain(fname)* függvényének kiértékelésével teszi, amely függvény a szabályos eCOS main függvény létrehozásáért felelős.

Az mm_arm_template.tlc kódja igen egyszerű, mindössze pár sor:

```
%% $RCSfile: mm_arm_template.tlc,v $
%%
%% $Date: 2007/03/30 11:12:09 $
%%
%% by Laura Fabian
%%
%% Abstract: MITMOT ARM + eCOS real-time template file
%%

%selectfile NULL_FILE

%% We will use mm_arm_ertmain.tlc
%%
%include "mm_arm_ertmain.tlc"

%% The name of the model-specific main function
%%
%assign fname = "%<CompiledModel.Name>" + "_main"

%% The model-specific main function will be created
```

```
%% with a function call to the
%% SLibCreateMMARMERTMain(fname) function
%% which is declared in mm_arm_ertmain.tlc
%%
%<SLibCreateMMARMERTMain(fname)>

%%[EOF] mm_arm_template.tlc
```

5.2.2. mm_arm_main_common.tlc

Ez a fájl négy segédfüggvény definícióját tartalmazza, melyek a kódgenerálás átláthatóbbá tétele miatt ebbe a külön fájlba kerültek. A függvények közül az egyik például a generált kód olvashatóságát segíti elő a szabványos kimenetre kitett utasítások platformfüggőségének elrejtésével:

```
%% ================================================================
%% Function: debugMsgPrint
%% Abstract:  Pring a message to the appropriate debugging output
%%            medium.
%%

%% Print debug messages with the
%% target-specific print function
%%
%function debugMsgPrint(msg) void

%% Opens a buffer with name buff, writes there
%% eCOS-specific C-commands
%% and then closes the buffer and returns
%% the contents which was written to it
%%
%openfile buff
//eCOS printf function
    diag_printf("%<msg>");
%closefile buff
%return buff
%endfunction %% debugMsgPrint
```

Van egy függvény az összes LED kigyújtására és eloltására, ennek célja az esetleges hibák előfordulása esetén a figyelemfelkeltés, egy másik függvény pedig ezen hibák közül egyet meg is fogalmaz, azaz az OverrunFlag bebillenése esetén kigyújtja az összes LED-et.

A fájl legfontosabb függvénye a main_ecos_defines(). Ez igazából nem is függvény, mert feladata mindössze annyi, hogy az éppen aktuálisan nyitott fájlba kiírja az eCOS main függvényének alap define-jait.

Egy részlet a függvényből:

```
%% ================================================
%% Function: main_ecos_defines
%% Abstract: defines for main func
%%
%function main_ecos_defines() void
%openfile buff
/*******************/
/*** Threads ***/
/*******************/
#define A_THREAD_STACK_SIZE (2048 / sizeof(int))
int A_thread_stack[A_THREAD_STACK_SIZE];
cyg_handle_t A_thread_handle;
cyg_thread A_thread_obj;

.....


%closefile buff
%return buff
%endfunction %% main_ecos_defines
```

5.2.3. mm_arm_ertmain.tlc

Ez a fájl a tényleges main függvény generálásért felel. Itt található a szimulációs lépést kód szinten megvalósító rt_OneStep függvény leírása, az eCOS main függvényhez szükséges include fájlok include-ként való beillesztése, valamint a Real-Time Workshop kódgenerálás eredményeként kapott kód egyedi fejlécének meghatározása.

Részlet a kódból:

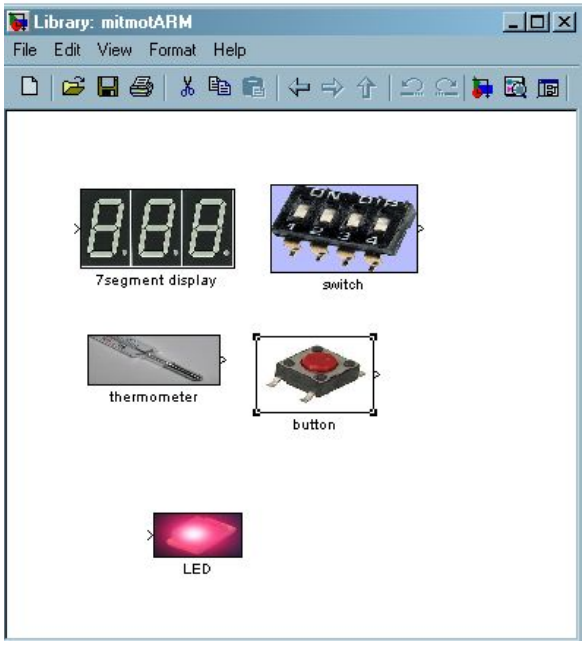
```
%% mm_arm_ertmain.tlc
%%
%% $Date: 2007/04/30 12:18:02 $
%%
%%
%% Abstract:
%%   Generation of ERT-based main.c file for
%%   Embedded Target for MITMOT ARM.
%%   External mode, mat-file logging, and continuous states
%%   are not supported by this target.


%selectfile NULL_FILE

%% The mm_arm_main_common.tlc file contains four functions,
%% from which one writes
%% the eCOS-specific defines to a buffer and returns it.
```

```
%%
%include "mm_arm_main_common.tlc"

....

%% The initialize function for non-Tornado, single-tasking
%% main function creation.
%%
%% ====================================================
%% Function: FcnSimpleNonOSMain
%%
%function FcnSimpleNonOSMain() Output

%assign rootSystem = System[NumSystems-1]
%assign reqInsts = LibGetSystemField(rootSystem,
"ReqRootPrmHdrDataInsts")

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

....


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

%function SLibCreateMMARMERTMain(fName) void

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


%openfile tmpFcnBuf
%<SLibDeclareModelFcnArgs(TLC_TRUE)>\

%% The eCOS specific defines will be included
%%
%<main_ecos_defines()>

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


%% EOF  mm_arm_ertmain.tlc
```

5.3. A MITMÓT blokk-könyvtár

Simulink alá elkészült egy külön blokk-könyvtár, MITMOT ARM néven (5.1. ábra). A könyvtár a MITMOT ARM kártya alap érzékelő/beavatkozó DPY moduljához készült. A könyvtár blokkjai maszkolt S-függvények. Van blokk a LED, kapcsoló, nyomógomb, hétszegmenses kijelző és hőmérő funkcióinak eléréséhez.

Az egyes blokkok a felhasználó számára egyszerűen konfigurálható, pop-up menüválasztásos illetve editálható felülettel rendelkeznek.



5.1. ábra. A MITMÓT ARM Simulink könyvtár

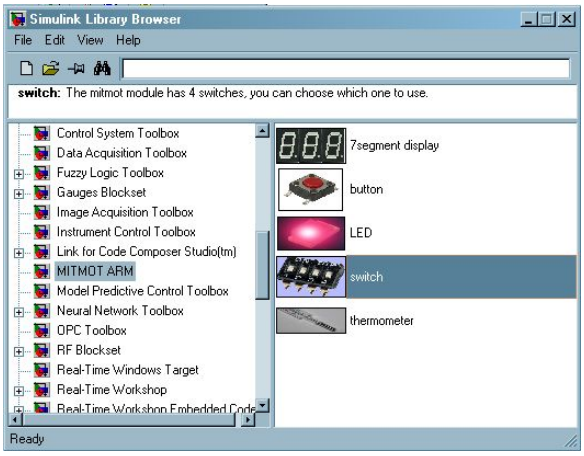
A MITMÓT ARM könyvtár Simulink Library Browser-ben való megjelenítéséhez (5.2. ábra) a könyvtár elérési útját a MATLAB Path-hoz kell adni.

Maga a MITMÓT ARM könyvtár tartalmazza egyrészt a könyvtár megjelenését leíró M-fájlt, másrészt az egyes blokkok TLC, C és MEX kódjait (a MEX kód a C kódból a mex parancsot használva készül) és végül a blokkok maszkolásához használt képfájlokat.

A továbbiakban az egyes blokkok megvalósításának részletes bemutatása következik.

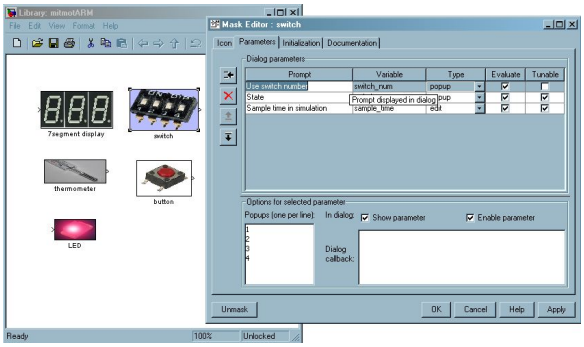
5.3.1. A MITMÓT blokk-könyvtár S-függvényei

Ahogy a korábbiakban már szerepelt, a Simulink alap blokk-készlethez lehet saját blokkokat adni, ehhez kellenek az S-függvények. A MITMÓT ARM blokk-könyvtárhoz készült S-függvények mind C-MEX S-függvények, azaz C nyelven íródtak, majd a MATLAB mex paranccsal MEX formátumra lettek fordítva, és ezt a MEX fájlt használja a Simulink a szimulációk során.



5.2. ábra. A MITMÓT ARM könyvtár a Simulink Library Browser-ben

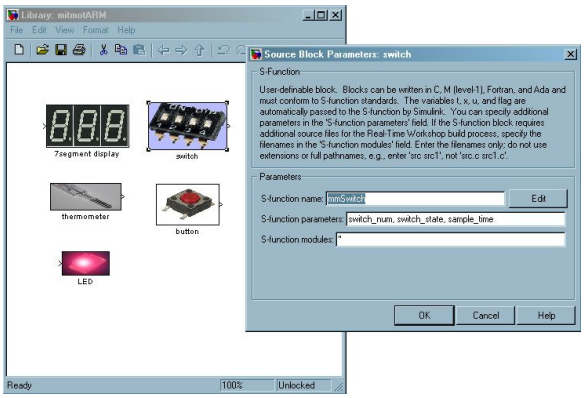
Minden MITMÓT blokk maszkolt S-függvény. Ez az jelenti egyrészt, hogy minden blokk grafikus kinézete tetszés szerint alakítható. Továbbá, a legfőbb szempont, ami miatt a maszkolás felelt meg leginkább a feladat megvalósításához: tetszőleges párbeszédablak alakítható ki egy blokkhoz, ami lehetővé teszi a felhasználó számára az egyes blokkparaméterek (a megadott keretek közötti) tetszés szerinti változtatását, ezzel személyreszabott használatot biztosítva.



5.3. ábra. A Switch blokk Mask Editor panelje

Egy maszkolt S-függvény esetén két párbeszédablak áll a fejlesztő rendelkezésére:

- Az Edit Mask felület, ahol a blokkhoz tartózó, a felhasználónak készülő párbeszédablakot lehet kialakítani (5.3. ábra),
- A Look Under Mask panel, ez az alap felhasználó számára rejtett. Az Edit Mask felületen beállított paramétereket az S-function parameters mezőben lehet az S-függvény számára elérhetővé tenni, az adott blokkhoz tartozó S-függvény nevét pedig az S-function name mezőben kell megadni (5.4. ábra).



5.4. ábra. Look Under Mask panel a Switch blokknál

Nyomógomb (button)

A MITMÓT ARM blokk könyvtár nyomógomb eleménél három szimulációs paramétert tud a felhasználó módosítani (5.5. ábra):

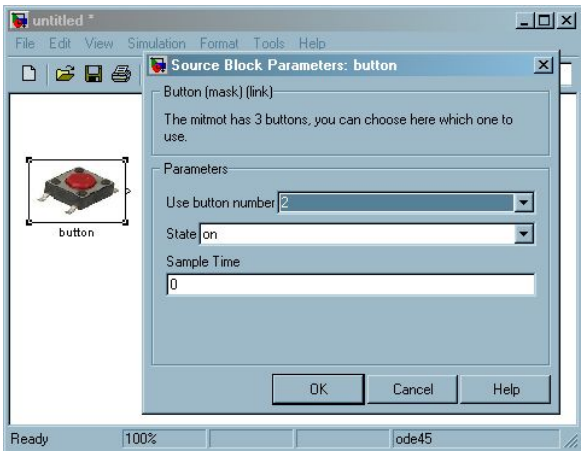
1. Hányas nyomógombot használjuk - *Use button number* legördülő menü, egytől háromig lehet választani.

2. A kijelölt nyomógomb lenyomva vagy elengedve van - *State* legördülő menü, értéke on vagy off lehet.

3. A nyomógomb mintavételezési ideje, azaz hogy milyen gyakran kerüljön kiértékelésre a gomb állapota, editálható felület.

A három paraméter közül a nyomógomb száma és mintavételezési ideje a szimuláció közben nem változtatható, azaz úgynevezett nontunable paraméterek. A gomb állapota paraméter esetén természetesen más a helyzet, hiszen szimuláció közben szükség lehet a gombot lenyomni-elengedni, így ez a változó tunable. Minden MITMÓT ARM blokk esetén van mintavételezési idő kijelölés, jelenleg azonban csak akkor helyes a működés, ha minden blokk ugyanarra az értékre van állítva, mert csak a single-rate üzemmód támogatott. A mező a multi-rate működés felé való esetleges továbblépés miatt került be, továbbá a blokkok virtuális voltát kiküszöbölendő.

Egy blokk virtuálisnak számít, ha a szimulációban aktívan nem vesz részt, azaz semmi olyat nem csinál, ami a szimuláció lefutását lényegileg módosítja. Emiatt minden device driver (jelen esetben MITMÓT periféria) virtuálisnak számít, mert például a LED csak annyit csinál, hogy beolvassa a bemenetét, a nyomógomb csak kitesz a kimenetére valamilyen értéket. A Real-Time Workshop kódgenerátora ellenben még a kódgenerálás megkezdése előtt optimalizálja a modellt, a virtuális blokkokat eliminálva. Emiatt szintén szükséges volt a mintavételezési idő direkt kijelölésének bevezetése.



5.5. ábra. A nyomógomb (button) blokk paraméterei

És most következzen a nyomógomb-blokk S-függvényének részletes bemutatása. Minden, a MITMÓT ARM könyvtárban levő blokk S-függvénye igen hasonló a megegyező funkciókból adódóan, így a többi blokk kapcsán majd csak a nyomógomb-blokk S-függvényétől való jelentősebb eltérések kerülnek bemutatásra.

Minden C-MEX S-függvény kötelező fejléce:

```
#define S_FUNCTION_NAME  mmButton
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
```

Az S-függvény paramétereinek nevesítése, a párbeszédablakban megadott értékek lekérése makrók segítségével valósul meg, melyek rögtön a kötelező fejléc után következnek. A paraméternevesítés nem kötelező, csak az átláthatóság miatt javasolt, mert a meghívott függvények a paraméterek Look Under Mask panelen meghatározott sorrendjét használják, ezért a Look Under Mask panel S-function parameters mezőjében megadott, az S-függvény által használandó paraméterek sorrendjét a későbbi lekérdezések miatt kiemelkedően fontos megjegyezni. Az S-függvényben a párbeszédablak paramétereinek lekérése ekkor az alábbi módon történik:

```
enum {E_BUTTON_NUM=0, E_BUTTON_STATE, E_SAMPLE_TIME};

#define N_PAR 3

#define BUTTONNUM(S) (mxGetScalar(ssGetSFcnParam(S,E_BUTTON_NUM)))
#define BUTTONSTATE(S) (mxGetScalar(ssGetSFcnParam(S,E_BUTTON_STATE)))
#define SAMPLETIME(S) (mxGetScalar(ssGetSFcnParam(S,E_SAMPLE_TIME)))
```

A következő lépés a mdlInitializeSizes függvénymegírása. Ebben először deklarálni kell, hogy hány bemeneti paramétert várunk. Ez az a szám kell legyen, ahány paramétert az adott blokk Look Under Mask *S-function parameters* mezőjében megadtunk, különben

hibajelzést kapunk. Miután ki lett jelölve a várt paraméterek száma, ez ellenőrzésre is kerül, azaz kiderül, hogy a kellő számú paraméter vajon rendelkezésre áll-e. Ha a paraméterek száma körül minden rendben, következhet a szimuláció közbeni viselkedésük beállítása - ez lehet tunable és nontunable. A nyomógomb esetén csak az állapota lesz tunable, ez az SS_PRM_SIM_ONLY_TUNABLE értéket kapja, azaz az érték szimuláció közben változtatható.

A paraméterek beállítása után a ki- és bemeneti portok számának és adattípusának megadása jön, majd hogy a blokk hány mintavételezési lépésközt használ - ez a MITMÓT blokkok esetén mindig egy. Legvégül a Simulink-nek szóló egyéb, az S-függvény kezelésére vonatkozó parancsokat lehet adni.

```
/* Function: mdlInitializeSizes ===========================================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-function
 *    block's characteristics (number of inputs, outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct *S)
{
ssSetNumSFcnParams(S, N_PAR); //Set the number of input parameters

//If the number of input parameters not equals with the actual
  //number of inputs, return
  if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) return;

  //Set the parameter nontunable
  ssSetSFcnParamNotTunable(S, E_BUTTON_NUM);

  //Set the parameter tunable, for simulation only
ssSetSFcnParamTunable(S, E_BUTTON_STATE, SS_PRM_SIM_ONLY_TUNABLE);

//Set the parameter nontunable
ssSetSFcnParamNotTunable(S, E_SAMPLE_TIME);

//Set the number of input ports to 0
if (!ssSetNumInputPorts(S, 0)) return;

//Set the number of output ports to 1
  if (!ssSetNumOutputPorts(S, 1)) return;

  //The output is a 1-width signal
  ssSetOutputPortWidth(S, 0, 1);

  //The type of the output is integer
  ssSetOutputPortDataType(S, 0, SS_INT32);

//The block is with 1 sample-time
  ssSetNumSampleTimes(S, 1);
```

```
//Other options
  ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
```

A mintavételezési idő párbeszédablakból állítható. Mivel az S-függvény mdlInitializeSampleTimes függvényének feladata a mintavételezési idő beállítása, ezért ebben a függvényben le kell kérdezni a párbeszédablakban megadott értéket, ami az alábbi direktívával tehető meg:

```
/* Function: mdlInitializeSampleTimes =============================
 * Abstract:
 *    This function is used to specify the sample time(s) for your
 *    S-function. You must register the same number of sample
 *    times as specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime( S, 0, SAMPLETIME(S) );
}
```

A nyomógomb kimenete a szimuláció során az állapotától függően 0 vagy 1, ennek beállítására az mdlOutputs függvényben kerül sor:

```
/* Function: mdlOutputs ===========================================
 * Abstract:
 *    In this function, you compute the outputs of your S-function
 *    block.
 * Generally outputs are placed in the output vector, ssGetY(S).
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
int32_T *y = (int32_T *)ssGetOutputPortSignal(S,0);
if (BUTTONSTATE(S)==1)
*y = 1;
else
*y = 0;
}
```

A nyomógomb blokkhoz TLC fájl is tartozik, és a kódgeneráláshoz tudni kell, hogy a MITMÓT ARM DPY kártya pontosan melyik nyomógombját kell használni, amit pedig a felhasználó állít be a szimulációs modell párbeszédablakán. Az S-függvény és a blokkhoz tartozó TLC fájl közti paraméterátadást az mdlRTW függvény valósítja meg.

```
/* Function: mdlRTW ===============================================
 * Abstract:
 *    Writes out the value of switch to the variable P1 in the RTW file.
```

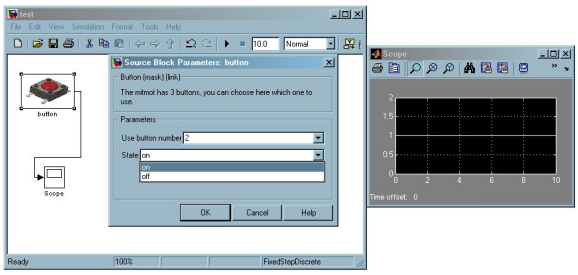
```
 */
#define MDL_RTW
static void mdlRTW(SimStruct *S)
{
int32_T button_num = (int32_T)BUTTONNUM(S);
    ssWriteRTWParamSettings(S, 1, SSWRITE_VALUE_DTYPE_NUM, "button_num",
     &button_num, DTINFO(SS_INT32,0));
}
```

Mind a kódgenerálás, mind a szimuláció szempontjából szükségtelen lezáró feladatok elvégzése, így az mdlTerminate függvény üresen maradhat.

```
/* Function: mdlTerminate =====================================
 * Abstract:
 *    In this function, you should perform any actions
 *    that are necessary at the termination of a simulation.
 *    For example, if memory was allocated in mdlStart,
 * this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
}
```

Az S-függvény zárása pedig a korábban bemutatott kötelező formula:

```
#ifdef MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"     /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"      /* Code generation registration
                             function */
#endif
```

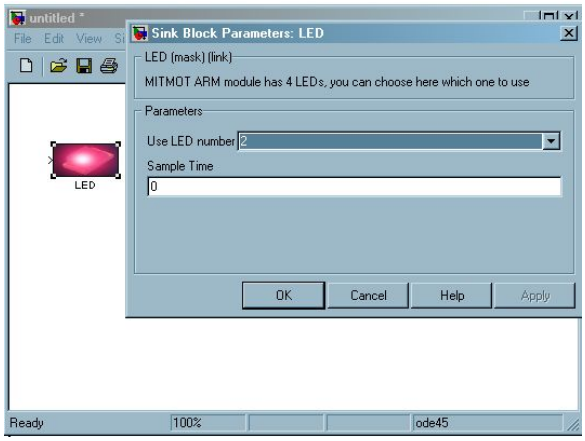


5.6. ábra. A Button S-függvény szimulációs működése

A nyomógomb S-függvény szimulációs viselkedése látható az 5.6. ábrán.

LED

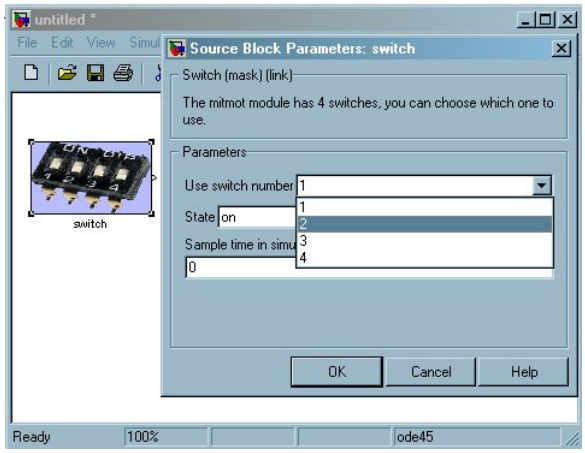
A LED-nél csak két paramétert állíthatunk: hányast használjuk és milyen mintavételezési idővel. A LED esetén az állapot manuális állításának nincs értelme, hiszen ez kizárólag a bemenetétől függ. A blokknak egy bemeneti portja van, ami integer értéket vár (int32_T Simulink típus). A LED blokk paraméter ablakát mutatja az 5.7. ábra.



5.7. ábra. A LED blokk paraméterei

Kapcsoló (switch)

A kapcsoló működése a gomb blokkhoz teljesen hasonló, lehet számot, állapotot és mintavételezési időt kijelölni (5.8. ábra).



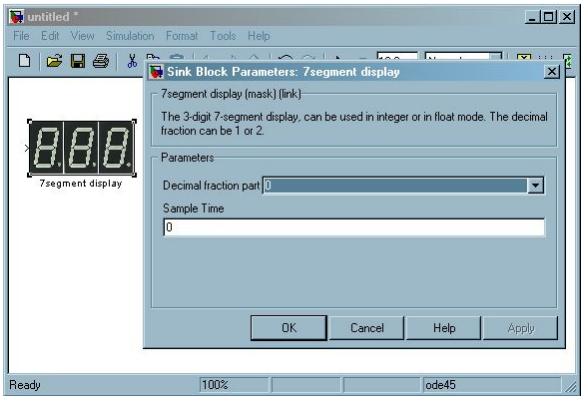
5.8. ábra. A kapcsoló (switch) blokk paraméterei

Kijelző (display)

A kijelző S-függvényébe több újdonság is van. Egyrészt bekerül egy eddig nem használt függvény, az mdlStart, amely a szimuláció indításakor hívódik meg :

```
#define MDL_START
#if defined(MDL_START)
static void mdlStart(SimStruct *S)
{
/* During simulation, just print a message */
  if (ssGetSimMode(S) == SS_SIMMODE_NORMAL) {
    mexPrintf("\n 7-segment display initalize \n");
    }
}
#endif /*  MDL_START */
```

Másrészt ebben a függvényben egy eddig ismeretlen, mex kezdetű parancs is helyet kapott. Az mdlStart függvény a szimuláció kezdetekor hívódik meg, jelen esetben annyi a feladata, hogy kiírja a MATLAB promptra, hogy „7-segment display initalize”, ehhez kell a mexPrintf függvény. SS_SIMMODE_NORMAL értéket csak akkor kapunk, ha normál Simulink szimuláció fut, azaz csak ekkor ír a promptra, más esetben nem lenne értelme.



5.9. ábra. A kijelző (disp) blokk paraméterei

A kijelző mdlOutputs függvényében is a mexPrintf függvény hívódik meg, minden kiértékelésnél a MATLAB parancssorra ír üzenetet, így követhető, hogy a kijelző blokk a szimuláció során valóban működik. Az 5.9. ábrán látható a kijelző modul paraméterablaka.

Hőmérő (temp)

A hőmérő modul működése a kapcsoló és nyomógomb működéséhez hasonló, nincs bemenete és egy kimenete van. Mivel a valós hőmérsékletmérést nem feladat szimulálni, a szimuláció során a hőmérő kimenete a felhasználó által megadott értéket veszi fel.

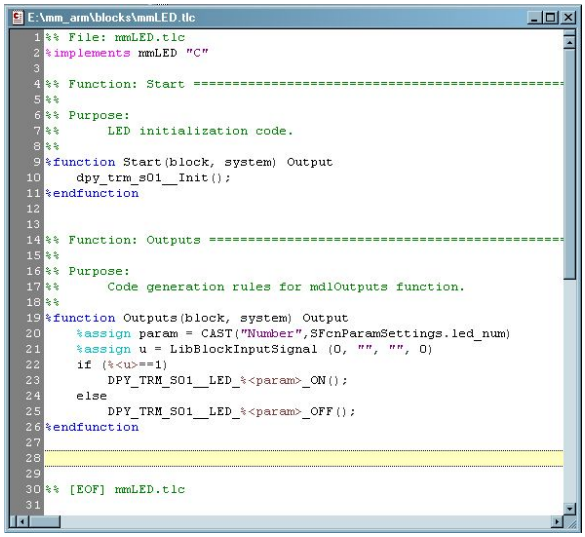
5.3.2. A MITMÓT blokk-könyvtár TLC kódjai

TLC kód készült a LED, kapcsoló, gomb, hétszegmenses kijelző és hőmérő funkcióinak eléréséhez. Ezek a TLC kódok felelősek az adott blokk funkcióinak kódban való megjelenítéséért.

Minden perifériafunkció eléréséhez a MITMÓT ARM modulhoz és a DPY kártyához készült API-k [44, 45] használata biztosítja az alapot.

LED

A LED perifériához két dolgot kellett megvalósítani: egyrészt a LED használatához inicializálni kell a DPY kártyát, ez a kódrész a TLC kód Start függvényébe kapott helyet. Másrészt, maga a LED funkció, azaz a LED kigyújtása vagy lekapcsolása a kódgenerálásnál lesz a fő kimenet, így ezek az Outputs függvénybe kerültek.



5.10. ábra. A LED blokk TLC kódja

A TLC kód Outputs függvénye először átveszi a LED blokk maszkján a felhasználó által beállított, a kódgenerálás szempontjából számunkra fontos paramétert, azaz, hogy hányas sorszámú LED-et használjuk:

```
%assign param = CAST("Number",SFcnParamSettings.led_num)
```

Ezután a bemenet értékét kell megnézni, hiszen annak függvényében kell kigyújtani vagy lekapcsolni a LED-et. A bemeneti változó lekérdezése az alábbi módon történik:

```
%assign u = LibBlockInputSignal (0, "", "", 0)
```

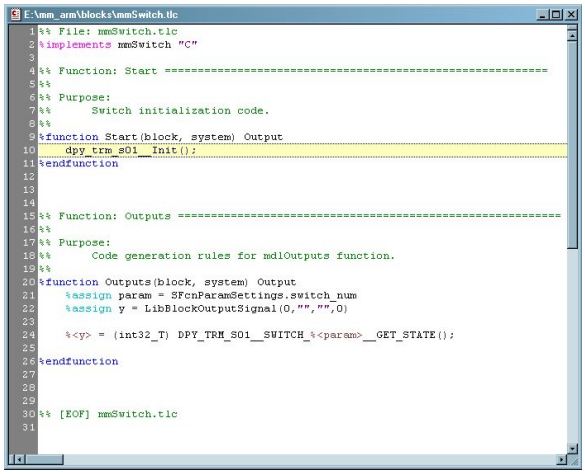
Végül a kimenetre egy az egyben kikerülő C kód a bemenet függvényében:

```
if (%<u>==1)
DPY_TRM_S01__LED_%<param>_ON();
```

```
else
DPY_TRM_S01__LED_%<param>_OFF();
```

Mint látható, maga a blokk TLC kódja (5.10. ábra) nem különösebben bonyolult, ezekből az egyszerű blokk leírásokból majd a Real-Time Workshop kódgenerátora fogja összerakni az alkalmazásokhoz szükséges bonyolult kódot.

Kapcsoló (switch)



5.11. ábra. A kapcsoló (switch) blokk TLC kódja

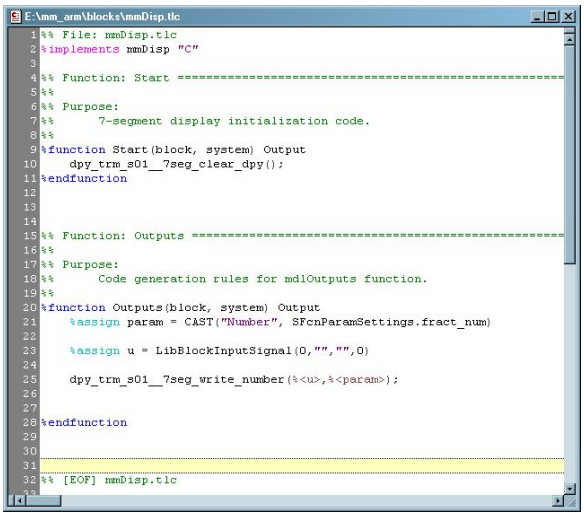
A Start függvényben itt is a DPY kártya alap inicializálása történik. Az Outputs függvénybe, az előbbiekben bemutatott LED-től eltérően, nem a bemenet kiértékelése történik, hanem kimenetre kerül a Switch aktuális állapota. Ehhez le kell kérdezni a felhasználó által megadott kapcsoló állapotát, majd a változóként beállított kimenetnek át kell adni ezt az értéket. A blokk TLC kódjából mutat részletet az 5.11. ábra.

Kijelző (display)

A kijelző blokk inicializálásakor, a TLC Start függvényben a kijelző törlése történik meg. A kijelző kimeneti függvénye, az Outputs pedig a felhasználótól paraméterként megkapott tizedes pontosságot figyelembe véve a bemeneti porton megkapott számérték.

Ahogy minden TLC kód esetén látszik, itt már nincs a változóknál típusbesorolás. Ezt a blokk S-függvénye határozza meg, a TLC már csak az onnan érkező típust veszi át. A kódban ugyan van egy CAST() utasítás, de az csak a „Number” értéket jelöli. Vannak egyéb TLC típusok is, mint például a Complex, de a MITMÓT esetén double, float és integer típusokra volt szükség, ezek pedig mind a TLC Number kategóriába esnek.

A kijelző blokk TLC kódjának részlete látható az 5.12. ábrán.



5.12. ábra. A kijelző (disp) blokk TLC kódja

Hőmérő (temp)

A hőmérő esetén a tényleges működés megkezdése előtt szükséges a periféria inicializálása, így ez kapott helyet a TLC Start függvényben. Ezek után, a tényleges működés abból áll, hogy az aktuális mérési eredményt a lekérdezési időpillanatokban a blokk a kimenetére teszi, azaz a blokk kimeneti változója megkapja az adott lekérdezési pillanatban aktuális hőmérsékleti értéket.

Nyomógomb (button)

A nyomógomb TLC kódja a kapcsoló kódjának szinte tökéletes másolata. Ez abból adódik, hogy a két periféria szerepe és működése közel azonos. Itt is annyi a megvalósítandó feladat, hogy a DPY kártya inicializálása után a blokk kimenetére kell tenni a nyomógomb aktuális állapotát jelző számértéket.

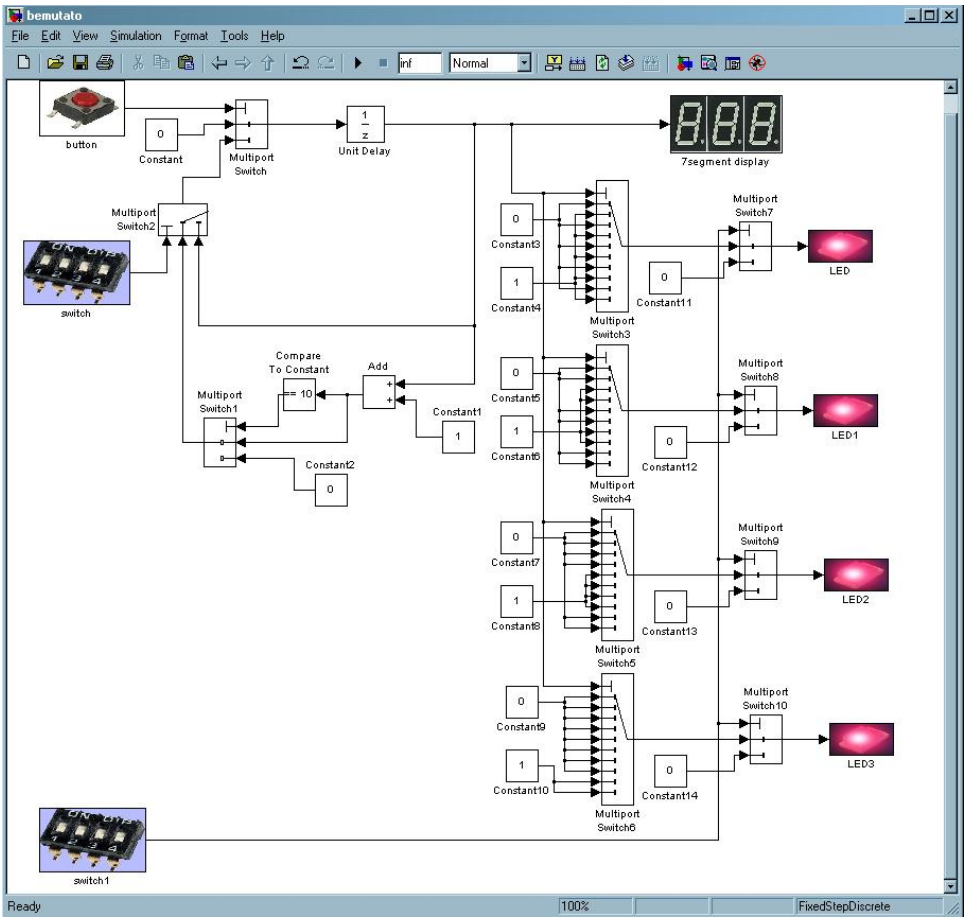
5.4. Egy kódgenerálási példa bemutatása

A kódgenerálási környezet bemutatásaként most egy konkrét példa következik, amely egy stopperóra-szerű funkciót valósít meg, amihez a MITMÓT ARM kártya DPY moduljának LED, kijelző, kapcsoló és nyomógomb elemeit használjuk.

A bemutatandó példaalkalmazás:

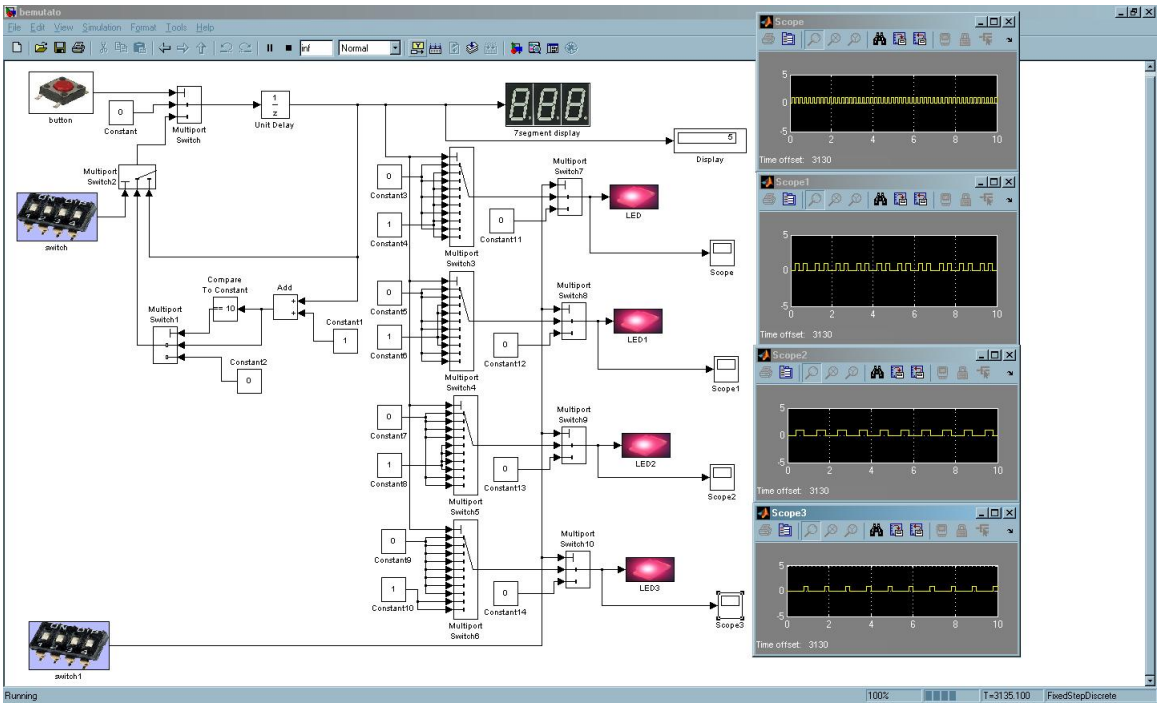
- A kijelzőn egy 0-tól 9-ig futó számláló értékét látjuk, az aktuális érték frissítése a megadott mintavételi frekvenciával történik.

- A DPY modul 2-es nyomógombja a reset, lenyomásának hatására a számláló nullázódik.
- A 2-es kapcsoló állásától függően a kártyán levő 4 LED-en bináris számként kijelezzük a számláló aktuális értékét, azaz például ha a kettes kapcsoló ON állásban van és a számláló épp 3-at mutat, az első és a második LED fog világítani. A kapcsoló OFF állásában a LED-ek nem világítanak.
- Az 1-es kapcsoló a START-STOP funkció, ha ON állásban van, megállítja a számlálót, OFF állásban pedig folytatódik a számláló léptetése.



5.13. ábra. A példaalkalmazás Simulink modellje - bemutato.mdl

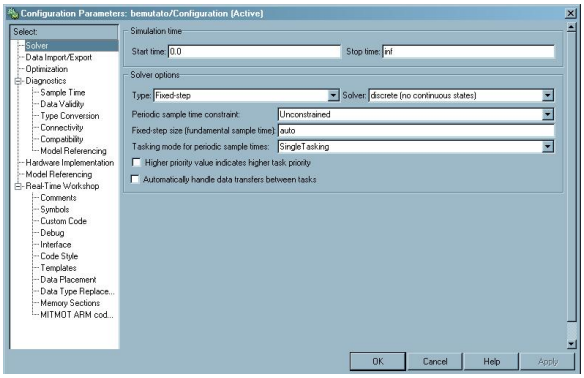
A bemutato.mdl fájl a példaalkalmazás Simulink modellje (5.13. ábra). A modell szimulációs viselkedését az egyes jelek értékeit megjelenítő blokkok beillesztésével lehet megfigyelni. A működést ellenőrizendő, Display és Scope blokkok kerültek a modellbe, amelyeken követhetők a szimulációs lépések során az egyes jelek aktuális értékei (5.14. ábra).



5.14. ábra. A példaalkalmazás Simulink modelljének működése

Az alkalmazás eCOS alatt futtatható C kódját a modellből a Real-Time Workshop kódgenerátorával kapjuk meg. A Simulink *Configuration Parameters* panelen a következő beállítások szükségesek:

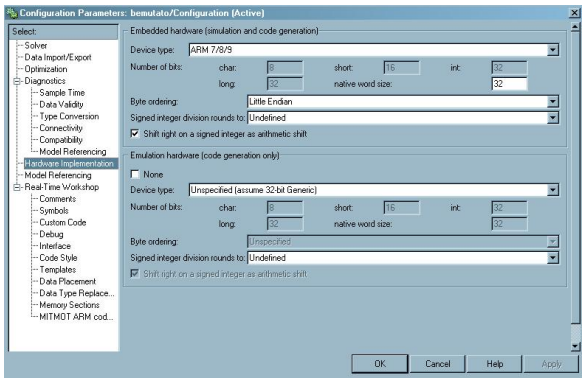
Solver panel Type: *Fixed-step* és Solver: *discrete*, Tasking mode for periodic sample times: *Single Tasking* (5.15. ábra)



5.15. ábra. Solver panel

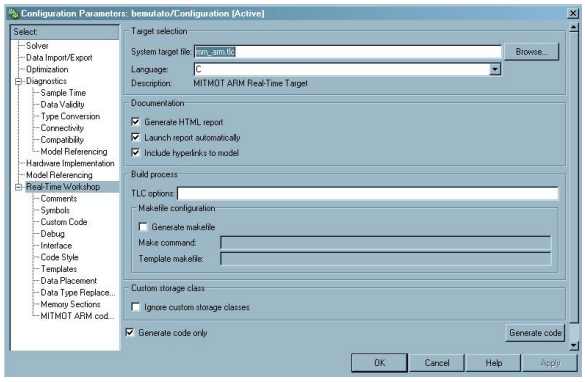
Hardware Implementation panel Device type: *ARM 7/8/9* (5.16. ábra)

Real-Time Workshop panel System target file: *mm_arm.tlc*, Language: *C*, a *Generate HTML report* kijelölése a generált kód megtekintése végett praktikus, a *TLC options-*



5.16. ábra. Hardware Implementation panel

t és *Generate makefile* checkbox-ot üresen kell hagyni, a *Generate code only* opciót ki kell jelölni (5.17. ábra)



5.17. ábra. Real-Time Workshop panel

A generált fájlok:

bemutato.c Ebben a fájlban a modell szimulációnak megfelelő léptetését megvalósító bemutato_step(), az alkalmazás-specifikus inicializálásokat végző bemutato_initialize() és az alkalmazás lefutása utáni feladatokat végző bemutato_terminate() függvények vannak. A függvények közül a bemutato_terminate() üres (stub), mivel az alkalmazás végtelen idejű futásra lett tervezve, így nem lesz soha befejező állapotban. A bemutato_step() függvényben a minden léptetésnél bekövetkező interruptot követő állapotkiértékelés van: a kapcsolók állásának lekérése, a kijelző értékének frissítése, a kapcsolóktól függően a megfelelő LED-ek kigyújtása-lekapcsolása. A bemutato_initialize() a MITMÓT DPY kártya és a hozzá tartozó, inicializálást igénylő perifériák inicializálását valósítja meg.

bemutato_data.c A Simulink modellben szereplő konstansok értékei és az egyéb blokkok inicializálási paraméterei ide vannak összegyűjtve.

bemutato_main.c Az alkalmazás eCOS specifikus main függvénye, ide kerülnek a szükséges include-ok, valamint ez az eCOS program fő szála, a garantált válaszidejű léptetést megvalósító periodikus interrupt és az interrupthoz kapcsolódó DSR rutin inicializálásának helye. Az eCOS cyg_user_start függvényben kerül meghívásra a példaalkalmazáshoz szükséges inicializálást végző bemutato_initialize() függvény, a főszálban pedig az rt_OneStep() a bemutato_step() függvény meghívásával lépteti a modellt. Az egyes lépések közti időközt az eCOS interrupt rutinjának periódusa határozza meg, azaz minden megszakítást követően a modell egyet léptetődik.

bemutato.h A fájl a modellhez kapcsolódó adatstruktúrákat definiálja, ilyen például, hogy a *delay* blokk milyen struktúrában tárolja az aktuális értékét, vagy hogy a modellben szereplő konstansoknak mi az adattípusa.

bemutato_private.h Itt van a modellhez tartozó private típusú adatok és definíciók helye.

bemutato_types.h Ez a header fájl deklarálja a modellhez kapcsolódó új típusokat.

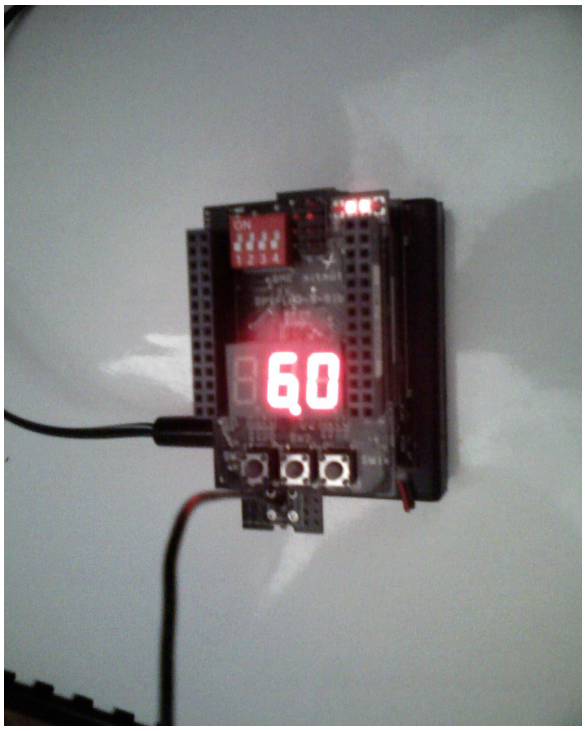
rtwtypes.h A Simulink és Real-Time Workshophoz tartozó speciális adattípusok feloldása, úgymint például:

```
typedef int int32_T;
```

Azaz az S-függvények C kódjában az int típust az int32_T jelentette.

A generált C kódot a MITMÓT-on való futtatáshoz először a kártyához készült eCOS – Eclipse szoftverkörnyezetet használva *hex* formátumra kell fordítani, majd ezt a hex fájlt kell a Philips Flash Utility programot használva a MITMÓT ARM kártyára tölteni. A szoftverkörnyezet használatáról az [54] irodalomban lehet részletesen olvasni.

A Simulink modellből automatikusan generált kód működése az elvártnak megfelelő, ahogy ez az 5.18. ábrán látható.



5.18. ábra. A példaalkalmazás futtatása MITMÓT-on

6. fejezet

Összefoglalás

6.1. Eredmények

A diplomatervben bemutatásra került a Modell-alapú tervezési eljárás, megismerhettünk ipari biztonságkritikus szabványokat. Hangsúlyozva az automatikus kódgenerálás egyre növekvő szerepét, néhány kereskedelmi forgalomban levő automatikus kódgenerálásra képes szoftver ismertetése is helyet kapott.

A dolgozat fő témája egy, a MITMÓT ARM kártyához MATLAB-Simulink alatt készült Modell-alapú tervezési környezet fejlesztése. A diplomamunkában a fejlesztési folyamat minden főbb lépése nyomon követhető.

A fejlesztés eredményeként elkészült a Simulink blokk-könyvtárakhoz hasonlóan egy MITMÓT ARM blokk-könyvtár, mely a MITMÓT ARM kártya DPY moduljának perifériáit modellező blokkokból áll. A MITMÓT ARM blokk-könyvtár elemeivel lehetségessé vált a MITMÓT ARM DPY modul egyszerű érzékelő és beavatkozó egységeinek szimulációja (úgymint például LED, kapcsoló) Simulink alatt.

Továbbá, a MITMÓT ARM blokk-könyvtár blokkjait használva, a Simulink modellből Real-Time Workshop és Real-Time Workshop Embedded Coder használatával automatikusan generálható a MITMÓT ARM kártyára, eCOS RTOS alatt futtatható C kód.

Ezáltal, mindenféle kézi kódolást kiiktatva, a MITMÓT ARM kártyára lehetségessé vált programot Simulink modell formájában tervezni, majd a modell, a diplomában bemutatott fejlesztői környezetet használva, automatikusan a MITMÓT ARM kártyán futó kóddá konvertálható.

6.2. Továbbfejlesztési lehetőségek

További fejlesztési lehetőség a jelenlegi implementáció kiterjesztése multi-rate esetre, azaz amikor a modell több mintavételi frekvenciát is kezel, illetve a multi-tasking bevezetése.

A MITMÓT ARM platform DPY modulján kívül rendelkezésre áll egy kommunikációs modul is, illetve egy robot mozgató egység, ezekhez szintén lehetséges lenne saját szimulációs és kódgenerációs modulok fejlesztése.

Érdekes összevetést eredményezhetne egy olyan fejlesztés is, ahol a kódgenerálás Simulink modell helyett Stateflow folyamatábrából kiindulva valósulna meg.

Irodalomjegyzék

[1] A. Mulpur: *„Model-Based Design Begins Paying Off For Signal Processing Design”*, CommsDesign, 2004, *http://www.commsdesign.com/showArticle.jhtml?articleID=17602732*

[2] F. Vahid, T. Givargis: *„Embedded System Design: A Unified Hardware/Software Introduction”*, John Wiley & Sons, ISBN: 0471386782, 2002

[3] P. J. Mosterman: *„Automatic Code Generation: Facilitating New Teaching Opportunities in Engineering Education”*, 36th ASEE/IEEE Frontiers in Education Conference, San Diego, CA, October 28 - 31, 2006

[4] I. Stürmer: *„Automatic Code Generation (Model-based Implementation)”*, 2007, *http://www.ichmaschine.de/autocode.html*

[5] E. De Wille, D. Vede: *„Software Process Models”*, *http://www.the-software-experts.de/e_dta-sw-process.htm*

[6] B. Marick: *„New Models for Test Development”*, 12th International Software Quality Week, 24-28 May 1999, San Jose / Silicon Valley, California USA

[7] E. Liversidge: *„Death of the V-Model”*, Harmonic Software Systems Ltd., 2005, *http://www.harmonicss.co.uk/docs/Death%20of%20the%20V-Model.pdf*

[8] *„Shortening the Embedded Design Cycle with Model-Based Design”*, National Instruments, 2006, *http://zone.ni.com/devzone/cda/tut/p/id/4074*

[9] G. Reed: *„Model-based design aids test and verification”*, Test & Measurement World, 12/1/2006, *http://www.reed-electronics.com/tmworld/article/CA6401653.html?industryid=21385*

[10] A. Behboodian: *„Model-Based Design”*, DSP Magazine, May 2006, pp. 52-55, *http://www.xilinx.com/publications/magazines/dsp_02/xc_pdf/p52-55-2dsp-mbd.pdf*

[11] T. A. Henzinger, J. Sifakis: *„The Embedded Systems Design Challenge"*, LECTURE NOTES IN COMPUTER SCIENCE, NUMB 4085, pages 1-15, 2006

[12] Q. Li, C. Yao: *„Real-Time Concepts for Embedded Systems"*, CMP Books, 2003, ISBN: 1-57820-124-1

[13] P. Barnard: *„Software Development Principles Applied to Graphical Model Development"*, AIAA Modeling and Simulation Technologies Conference and Exhibit, San Francisco, California, 15 - 18 August 2005

[14] Jad El-khoury, DeJiu Chen, M. Törngren: *„A Survey of Modelling Approaches for Embedded Computer Control Systems"*, Technical Report, Royal Institute of Technology, Stockholm, 2003 *http://www.md.kth.se/download/publications /2003/damek/Trita2003_ 36.pdf*

[15] T. Erkkinen: *„Model Style Guidelines for Flight Code Generation"*, AIAA Modeling and Simulation Technologies Conference and Exhibit, San Francisco, California, 15 - 18 August 2005

[16] T. Erkkinen: *„Model Style Guidelines for Flight Code Generation"*, American Institute of Aeronautics and Astronautics, *https://tagteamdbserver.mathworks.com/ttserverroot /Download/28430_ AIAA-2005-6216Erkkinen.pdf*

[17] N. Jones: *„Introduction to MISRA C"*, Embedded.com 07/01/02 *http://www.embedded.com/showArticle.jhtml?articleID=9900659*

[18] *„MISRA Home Page"*, MIRA Ltd, 2007, *www.misra.org.uk*

[19] S. Lehman: *„Controller Style Guidelines for Production Intent Development using MATLAB, Simulink, and Stateflow"*, 2001, *http://www.mathworks.com/matlabcentral/fileexchange /loadFile.do?objectId=4280&objectType=file*

[20] MathWorks Automotive Advisory Board (MAAB): *„Controller Style Guidelines for Production Intent Using MATLAB, Simulink and Stateflow"*, 2001, *www.mathworks.com/industries/auto/maab.html*

[21] P.Leprovost: *„Assessment of existing certification practices"*, WP2 Final Report, ARIBA, 1999, *http://www.aribaproject.org/rapport2/titre.htm*

[22] *„DO-178B"*, Wikipedia Encyclopedia, 2007, *http://en.wikipedia.org/wiki/DO-178B*

[23] *„The DO-178B Level A Safety-Critical Guideline for Avionics Systems”*, ALT Software Inc., 2007, *http://www.altsoftware.com/products/do-178b/safety/*

[24] *„A Modeling Tool resumes”*, DsysD Ltd, 2007, *http://www.dsysd.com/modellingtools.html*

[25] *„The Reference for DO-178B Resources”*, Esterel Technologies, Inc., 2007 *http://www.esterel-technologies.com/do-178b/*

[26] *„LabVIEW”*, Wikipedia Encyclopedia, 2007, *http://en.wikipedia.org/wiki/LabVIEW*

[27] B. MacCleery: *„Accelerating Embedded System Design”*, National Instruments News, Jan. 24, 2006, *http://zone.ni.com/devzone/cda/tut/p/id/3806*

[28] *„dSPACE Home Page”*, dSPACE GmbH., 2007, *http://www.dspaceinc.com/ww/en/inc/home.cfm*

[29] S. Davies: *„Real-life experience of using a modelling subset for TargetLink in safety-related work”*, MISRA Autocode Forum, 2005, *http://www.misra.org.uk/papers/F0505_Davies_Reallife.pdf*

[30] *„dSPACE: production code generator TargetLink supports AUTOSAR”*, Embedded Control Europe, 2007, *http://www.embedded-control-europe.com/prodnews?cat=1&pid=9807*

[31] Dr. A. Schönhoff, K. Harth: *„Barracuda's Maiden Flight”*, dSPACE NEws, 3/2006,

[32] B. Schätz, T. Hain, F. Houdek, W. Prenninger, M. Rappl, J. Romberg, O. Slotosch, M. Strecker, A. Wisspeintner: *„CASE Tools for Embedded Systems”*, 73-100 pp., Technische Universität München, 2003 *http://citeseer.ist.psu.edu/654225.html*

[33] *„ASCET Overview”*, ETAS Group, 2007 *http://en.etasgroup.com/products/ascet/*

[34] *„ASCET Autocode Ensures MISRA Conformance”*, ETAS/MKT61_Sar/01.2007, *http://www.qa-systems.de/media/File/MISRA_EmbeddedWorld2007.pdf*

[35] *„MATLAB”*, Wikipedia Encyclopedia, 2007, *http://en.wikipedia.org/wiki/MATLAB*

[36] *„Hajime Robots Home Page”*, Hajime Research Institute, Ltd., 2007, *http://www.hajimerobot.co.jp/robot/robot.htm*

[37] *„MATLAB Home Page”*, The MathWorks Inc., 2007, *http://www.mathworks.com/*

[38] The MathWorks, Inc: *„Matlab and Simulink: the platform for Model-Based Design”*, 2007, *www.mathworks.com/mbd*

[39] The MathWorks, Inc: *„Simulink 6 Using Simulink (sl_using.pdf)”*, version 6.6, 2007 *www.mathworks.com*

[40] The MathWorks, Inc: *„Real-Time Workshop 6 User's Guide (rtw_ug.pdf)”*, version 6.6, Márc. 2007, *www.mathworks.com*

[41] The MathWorks, Inc: *„Real Time Workshop Embedded Coder 4 User's Guide (ecodder_ug.pdf)”*, version 4.6, 2007, *www.mathworks.com*

[42] The MathWorks, Inc: *„Matlab 7 Desktop Tool and Development Environment (matlab_env.pdf)”*, version 7.4, 2007, *www.mathworks.com*

[43] Cs. Tóth et al: *„The modular MITMOT system”*, technical report, 2006, *http://bri.mit.bme.hu/?l=mitmot&p=what%20is*

[44] Csordás Péter, Scherer Balázs: *„DPY kijelző kártya C API”*, Budapesti Műszaki és Gazdaságtudományi Egyetem Méréstechnika és Információs Rendszerek Tanszék Beágyazott Információs Rendszerek csoport, 2005. augusztus, *http://bri.mit.bme.hu/*

[45] Scherer Balázs: *„Az eCos használata”*, Budapesti Műszaki és Gazdaságtudományi Egyetem Méréstechnika és Információs Rendszerek Tanszék Beágyazott Információs Rendszerek csoport, 2006. február, *http://bri.mit.bme.hu/*

[46] *„eCOS Home Page”*, 2007, *http://ecos.sourceware.org/*

[47] Hercog, D.; Curkovic, M.; Edelbaher, G.; Urlep, E.: *„Programming of the DSP2 board with the Matlab/Simulink”*, Industrial Technology, 2003 IEEE International Conference on Volume 2, Issue , 10-12 Dec. 2003 Page(s): 709 - 713 Vol.2

[48] D. Herzog: *„DSP-2 Library for Simulink - User's Manual”*, University of Maribor, 2006, *http://www.ro.feri.uni-mb.si/projekti/dsp2/documentation/dsp2_library_for_simulink_matlab_7.pdf*

[49] V. Huszár: *„Gyors prototípus fejlesztés Mitmót platform és Matlab-Simulink segítségével”*, TDK dolgozat, Budapesti Műszaki és Gazdaságtudományi Egyetem, Méréstechnika és Információs Rendszerek Tanszék, 2005

[50] The MathWorks, Inc: *„Simulink 6 Writing S-Functions (sfunctions.pdf)”*, version 6.6, 2007, *www.mathworks.com*

[51] The MathWorks, Inc: *„Real-Time Workshop 6 Target Language Compiler (rtw_ tlc.pdf)”*, version 6.6, 2007, *www.mathworks.com*

[52] The MathWorks, Inc: *„Real-Time Workshop Embedded Coder 4 Reference (ecoder_ ref.pdf)”*, version 4.6, 2007, *www.mathworks.com*

[53] The MathWorks, Inc: *„Real-Time Workshop Embedded Coder 4 Developing Embedded Targets (ecoder_ det.pdf)”*, version 4.6, 2007, *www.mathworks.com*

[54] Scherer Balázs: *„MCU ARM szoftverkörnyezet telepítési útmutatója”*, Budapesti Műszaki és Gazdaságtudományi Egyetem Méréstechnika és Információs Rendszerek Tanszék Beágyazott Információs Rendszerek csoport, 2005. március, *http://bri.mit.bme.hu/*

Függelék

F.1. A CD-melléklet tartalma

F.1.1. MITMÓT ARM target könyvtár

Az *mm_arm* könyvtár tartalmazza a MITMÓT-specifikus MATLAB-Simulink fejlesztői környezet forrásfájljait. A könyvtáron belül két alkönyvtár található. Az egyik szintén *mm_arm* nevű – ez a MITMÓT ARM target modell-hatáskörű TLC fájlainak helye; a *blocks* alkönyvtárban pedig a MITMÓT ARM Simulink blokkok C-MEX és TLC kódjai találhatóak, a Simulink Library Browserbe illesztő *slblocks.m* fájllal együtt.

F.1.2. MITMÓT ARM szoftverkörnyezet

- Cygwin
- eCOS
- Eclipse
- Philips Flash Utility

A felsorolt programok install fájljai a *mitmot_sw* könyvtárban találhatóak, telepítési útmutatóval együtt (mcu_arm_software_telepitesi_utmutato_d01a.pdf).

Ábrák jegyzéke

1.1. Egy ipari fejlesztés állomásai [3] 4
1.2. A V-modell [7] 6
1.3. Az automatikus kódgenerálás alapelve (autocode tool-chain) [4] 7
1.4. A Modell-alapú tervezés elemei [10] 9

2.1. DO-178B logó [22] 20
2.2. Az A380 első repülése (2005. április 27.) [25] 23
2.3. Kódgenerálás SCADE módra [25] 24
2.4. LabVIEW 8 targetek [27] 25
2.5. A V-modell ASCET-SD által támogatott fázisai [32] 27

3.1. MATLAB-Simulink környezet [36] 30
3.2. Simulink és MBD [3] 31
3.3. A Real-Time Workshop helye a fejlesztési folyamatban [40] 33
3.4. A MITMÓT [43] 36

4.1. A Simulink blokk matematikája [50] 39
4.2. A Simulink szimuláció menete [50] 40
4.3. S-függvény Callback eljárások a szimuláció során [50] 42
4.4. A Real-Time Workshop generálta fájlok függőségi kapcsolata [40] 45
4.5. A Target Language Compiler helye a kódgenerálási folyamatban [51] 46
4.6. Inlined S-függvény [51] 48
4.7. Noninlined S-függvény [51] 49
4.8. TLC [51] 50
4.9. Kódgenerálás a Target Language Compiler használatával [51] 54
4.10. TLC kódgenerálási folyamat [53] 58

5.1. A MITMÓT ARM Simulink könyvtár 65
5.2. A MITMÓT ARM könyvtár a Simulink Library Browser-ben 66
5.3. A Switch blokk Mask Editor panelje 66
5.4. Look Under Mask panel a Switch blokknál 67

5.5. A nyomógomb (button) blokk paraméterei 68
5.6. A Button S-függvény szimulációs működése 71
5.7. A LED blokk paraméterei 72
5.8. A kapcsoló (switch) blokk paraméterei 72
5.9. A kijelző (disp) blokk paraméterei 73
5.10. A LED blokk TLC kódja 74
5.11. A kapcsoló (switch) blokk TLC kódja 75
5.12. A kijelző (disp) blokk TLC kódja 76
5.13. A példaalkalmazás Simulink modellje - bemutato.mdl 77
5.14. A példaalkalmazás Simulink modelljének működése 78
5.15. Solver panel 78
5.16. Hardware Implementation panel 79
5.17. Real-Time Workshop panel 79
5.18. A példaalkalmazás futtatása MITMÓT-on 81

Rövidítések

ADC	Analog to Digital Converter
ANSI	American National Standards Institute
ARIBA	ATM system safety criticality Raises Issues in Balancing Actors responsibility
ARM	Advanced RISC Machine
ASAM	Association of Standardization for Automation and Measuring Systems
ASCET	Advanced Simulation and Control Engineering Tool
ASCET-MD	ASCET-Modeling and Design
ASCET-MIP	ASCET-MATLAB Integration Package
ASCET-RP	ASCET-Rapid Prototyping
ASCET-SE	ASCET-Software Engineering
AUTOSAR	AUTomotive Open System ARchitecture
AVR	Advanced Virtual RISC
BME	Budapesti Műszaki Egyetem
CD	Compact Disk
DAC	Digital to Analog Converter
eCOS	embedded Configurable Operating System
ECU	Electronic Control Unit
EUROCAE	European Organization for Civil Aviation Equipment
FAA	Federal Aviation Administration
FDA	Food and Drug Administration
FORTRAN	The IBM Mathematical Formula Translating System
FPGA	Field Programmable Gate Array
GNU	GNU's Not Unix
IC	Integrated Circuit
ISM	Industrial, Scientific and Medical
ISO	International Standards Organization

JAA	Joint Aviation Authorities
LabVIEW	Laboratory Virtual Instrumentation Engineering Workbench
MAAB	MathWorks Automotive Advisory Board
MATLAB	MATrix LABoratory
MBD	Model Based Design
MISRA	Motor Industry Software Reliability Association
PC	Personal Computer
POSIX	Portable Operating System Interface
PSAC	Plan for Software Aspects for Certification
RTCA	Radio Technical Commission for Aeronautics
RTW	Real-Time Workshop
S-function	System-function
SCMP	Software Configuration Management Plan
SDP	Software Development Plan
SCADE	Safety Critical Application Development Environment
SCS	Software Coding Standard
SDS	Software Design Standards
SRS	Software Requirements Standards
STF	System Target File
SVP	Software Verification Plan
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
TMF	Template Makefile
UAV	Unmanned Aerial Vehicle
VHDL	Very high speed integrated circuit Hardware Description Language
VI	Virtual Instrument
TLC	Target Language Compiler
ĺITRON	Micro Industrial The Real-time Operating system Nucleus

Köszönetnyílvánítás

Köszönöm konzulensemnek, dr. Márkus Jánosnak a félév során nyújtott segítséget, tanácsokat és javításokat.

Köszönöm Somlay Gergelynek hasznos javaslatait és az átolvasások során mutatott végtelen türelmét.