Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnikai és Információs Rendszerek Tanszék

Király Tamás

IP-XACT alapú regisztermenedzsment és generátor megvalósítása

Tanszéki konzulens: Krébesz Tamás István, tanársegéd

Külső konzulens: Olaszi Tamás (ARM)

BUDAPEST, 2017

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Király Tamás**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzé tegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2017. 05. 15.

...…….

Király Tamás

Összefoglaló

A beágyazott rendszerek világának változása (például az IoT - Internet Of Things robbanásszerű fejlődése) a rendszerek fejlesztési módszereinek változásával is jár. A modern rendszerek fejlesztési módszereinek a kiindulási terv fejlesztés közbeni változtatásával is számolniuk kell. Az ilyen, akár az egész munkafolyamaton végig terjedő változtatások gyors és hatékony véghezvitelének egy módszere az egyetlen forrás használata kiegészítve az automatizálással. Minden változtatás a közös forrásban történik, és a további lépések automatikusan végrehajtódnak. Egy rendszer regisztereit sokféle fájlformátumban tárolhatjuk. Egy fejlesztés lépései mind más és más leírási módot igényelhetnek. Amennyiben megvan a közös forrás leírási módja, a fejlesztés lépései számára ebből elő kell állítani az aktuálisan használt fájlokat.

A szakdolgozat bemutatja a regiszterek leírásának néhány elterjedt módját. Értékeli az IP-XACT szabványt és bemutat egy IP-XACT alapú fejlesztőkörnyezetet, az ARM Socrates DE-t. A feladatban megvalósított generátorok ezt a fejlesztőkörnyezetet egészítik ki a későbbi fejlesztési lépések fájljainak elkészítésével.

A kész generátorok több mérnökcsapat munkáját pontosítják és gyorsítják fel.

Abstract

System design methods evolve around solving the constantly developing challenges embedded system designers face. Modern system design methods are required to handle changes in the specification during the later stages of the design process as well. These changes can affect multiple steps of or even the entire workflow. Manual changes in all these steps and the synchronization of the several source documents are prone to errors. Using a single source file and high level of automatization is a good solution to this problem. Every change is made on this single source file and the other steps are done automatically. The registers of a system can be described in multiple formats, and different tasks require different formats. These formats can be generated from a single source file.

The thesis work introduces a couple of register description methods. It details the IP-XACT standard and an IP-XACT-based development environment, the ARM Socrates DE. This design environment is complemented by my contribution, i.e., by the developed generators, which generate the output files required by the other stages of the design process.

The completed generators improve the quality and speed of the work of different engineering teams.

1. Bevezetés

Napjaink System on Chip (SoC, magyarul egylapkás chip) tervezeteiben sok periféria, így több ezer regiszter található. Már a tervezési fázisban a rendszertervező csapatnak elő kell állnia a regiszterek és a memóriatérkép specifikációjával, majd a hardvermérnök, a szoftvermérnök és a verifikációs mérnök csapat mindegyike elkezd dolgozni a hozzájuk tartozó területen. Amennyiben tesztelés közben változtatni kell (például hibát találtak tesztelés során vagy új funkciót kell hozzáadni menet közben), lehet, hogy a kezdeti specifikációt meg kell változtatni, ami akár az összes utána következő lépés frissítésével is járhat. A változás természetétől függően akár a rendszer összes különféle leírási módját kézzel kell javítani. Ez teljességgel a különböző csapatok közti kommunikáció minőségén múlik, amely elég komoly hibalehetőséget hordoz magában. Egy másik probléma, hogy a mérnök a kódjában bekövetkezett változtatást nem dokumentálja, így okozva hibákat a rendszerben. Ezen problémák kiküszöböléséről szól a regisztermenedzsment. Ebbe beletartozik mind a csapatok közötti szinkronizáció, mind a tervezési folyamathoz tartozó repetitív feladatok automatizálásának biztosítása. Tegyük fel, hogy egy regiszter- és memórialeíró nyelvben (például IP-XACT, SystemRDL) vagy egy egyszerű táblázatban megfogalmazott forrásból dolgozunk. Ilyenkor, ha minden dokumentáció, terv vagy verifikációs leírás automatikusan ebből a leírásból generálódik, akkor a projekt komplexitása csökken és eltűnik több hibaforrás is. Például elég ezen a ponton a forrást megváltoztatni, nem az egész folyamatot egy-egy változtatás után. Azonban nem csak a hibák elkerülése miatt előnyös az automatizálás. A mérnökök valós feladataikkal tudnak foglalkozni a kézi javítás helyett a regisztermenedzsment automatizálása következtében. A piacon több szoftver is elérhető, amelyek megoldást nyújtanak erre azáltal, hogy képesek ezekből a regiszterleírásokból megfelelő kimenetet biztosítani (Verilog, C header fájl a firmware számára, dokumentáció - Word, HTML stb.).

Az én feladatom az ARM Socrates DE szoftverhez olyan generátorokat írni, amely ezen funkciókat a piacon lévő egyéb megoldásokhoz, illetve felmerülő igényekhez igazítja. A dolgozat elején, a 2. fejezetben bemutatok a regisztermenedzsmentet támogató szabványok és szoftverek közül párat részletesebben, majd a 3. fejezetben a feladatom pontosítását és bővítését. Ezután a 4. fejezetben a

generátorok elkészítését tárgyalom. Végül az 5. fejezetben az elkészült generátorokat értékelem és a továbbfejlesztési lehetőségekről ejtek pár szót.

2. Irodalomkutatás, technológiák, hasonló alkotások bemutatása

2.1 IP-XACT

A regiszterleírás egy elterjedt módja az IP-XACT. Ez egy eXtensible Markup Language-et (XML) használó formátum, amely elektronikus rendszerek fejlesztésében, implementációjában és verifikációjában használatos szellemi tulajdonok (angolul Intellectual Property, IP) dokumentálására szolgál [1].

2.1.1 XML

Az XML-t eredetileg a Világháló új formátumainak definiálására találták ki, ez egy általános jelölőnyelv. Szöveg alapú formátum, amiben a jelölőrendszer számára a jelölések (angolul tag) szintaktikája van csak megszabva, jelentésük azonban nincs. Például:

```
<szám>123-4567</szám>
```

A „< >” jelek között található a jelölés, jelen példában a „<szám>”, a kezdő és a „/” jellelt kiegészített záró jelölés (itt a „</szám>”) között pedig az adott jelöléshez tartozó tartalom. Azt, hogy az adott jelölés hogyan van értelmezve mindig az adott alkalmazástól függ. Ez a szám jelölés és a benne lévő adat, lehet például egy telefonszám vagy egy vásárlóhoz tartozó egyedi azonosító stb. Azonban a megadott szintaktika segítségével általános célú értelmezővel (angolul parser) egyszerűen olvashatjuk, írhatjuk vagy formailag ellenőrizhetjük az XML fájlokat. Ahogyan terjedt az XML használata világossá vált, hogy nem csak új dokumentumformátumok leírására alkalmas, hanem strukturált adatleírásra is. Az alábbi kódban látszik, hogy egy ügyfél jelöléshez milyen további jelölések tartoznak. A vásárlás jelölésekből például több is van, további jelöléseket is tartalmazva, így létrejön egy sajátos adatstruktúra.

```
<ügyfél>
     <azonosító>12</azonosító>
     <vásárlás>
          <dátum>2017.01.04.<dátum>
          <szám>123-4567</szám>
          <mennyiség>50</mennyiség>
     </vásárlás>
     <vásárlás>
          <dátum>2017.03.30.<dátum>
          <szám>123-4568</szám>
     <mennyiség>64</mennyiség>
     </vásárlás>
</ügyfél>
```

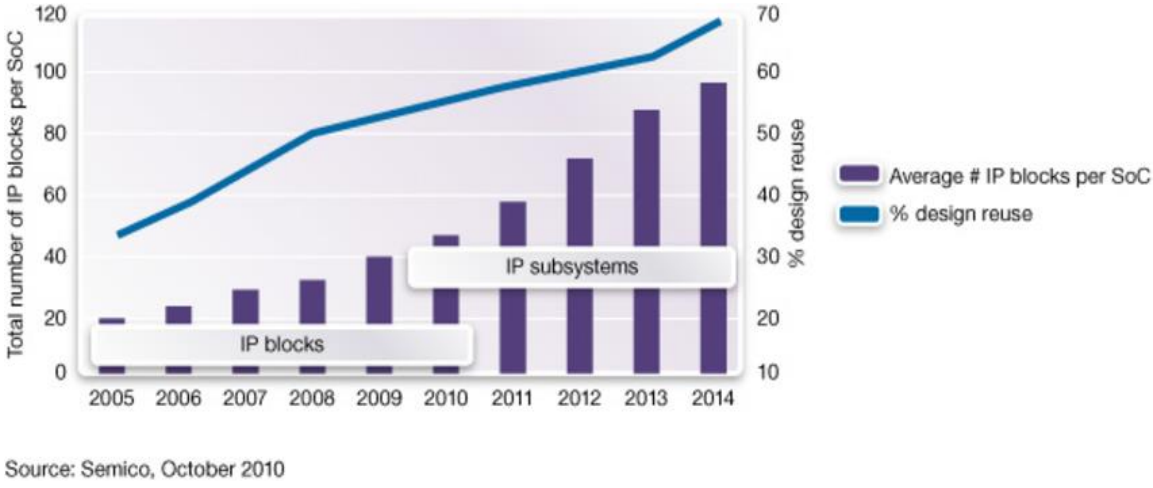
Ilyen adatokkal általában táblázatokban, programkonfigurációs fájlokban vagy hálózati protokollokban találkozhatunk. A korábbi adatformátumokhoz képest az XML mind tabuláris adat (mint például táblázatok vagy adatbázisok kapcsolatai), mind félig strukturált adat (mint például egy weboldal vagy üzleti dokumentum) leírására is alkalmas. A korábbi népszerű formátumok, mint például a Comma Separated Value (CSV, magyarul vesszővel elválasztott érték) vagy jól működnek tabuláris adatok kezelésére és kevésbé jól félig strukturált adatéra, vagy mint például a Rich Text Format (RTF, magyarul gazdag szöveg formátum), túlságosan specializáltak a félig strukturált adatokra. Emellett még az XML bővíthetősége, platformfüggetlensége, illetve az Unicode szabvány támogatása utat nyitott adatleíró formátumként történő széleskörű elterjedéséhez [2]. Fontos megjegyeznünk, hogy magában az XML nem csinál semmit sem, csak egy adattárolási mód, ezért a megfelelő sémára és az ezt felhasználó fejlesztőkörnyezetre lesz szükségünk.

2.1.2 Az IP-XACT előnyei

Az IP-XACT szabvány fő célkitűzése, hogy a különféle IP forgalmazó cégektől származó leírások kompatibilisek legyenek egymással, ezáltal is egyszerűsítve összetett IP csomagok importálását és exportálását különböző fejlesztőkörnyezetek között. Ez a szabvány kiegészítésnek lett kialakítva a hardverleíró nyelvekkel (hardware description language, HDL) való közös használatra. Az IP-XACT szabványt a SPIRIT Consortium cégcsoport alkotta 2003-ban és jelenleg az Accellera által alapított IP-XACT Schema Working Group fejleszti. A Working Group-ban több IP-vel, elektronikai rendszerekkel, illetve elektronikai tervezésautomatizálással (angolul electronic design

automation, EDA) foglalkozó cég is részt vesz, mint például az ARM, a Magillem vagy a Synopsys. A dolgozatomban használt IP-XACT verziót az IEEE 1685-2009 szabvány írja le [3]. Ebben a sémán kívül egy alkalmazásprogramozási felület (angolul application programming interface, API) leírása is található, ami egyéb eszközök számára könnyíti meg a metaadatok elérését. A szabvány definiálja milyen elemek és tulajdonságok jelenhetnek meg az XML dokumentumban, az elemek hierarchiáját és ennek szabályait, valamint adattípusait és ezek alapértelmezett értékeit.

Az IP-XACT megkönnyíti a rendszerintegrációt, verifikációt, automatizálást és igy komoly előnyt jelent a piacra kerülési idő (angolul Time-To-Market, TTM) tekintetében. A tervek újrafelhasználása vitathatatlan fontosságú a dizájnok folyton növekedő komplexitása és egyre rövidebb piacra kerülési idő-elvárások mellett [5].



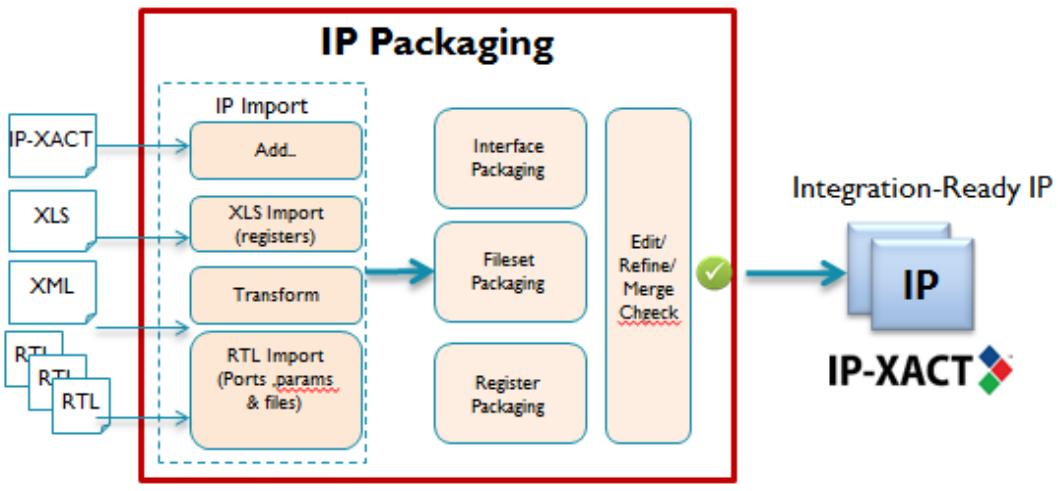
Source: Semico, October 2010

1. ábra. SoC-k fejlődése és az újrafelhasználás

Az 1. ábra jól mutatja azt a tendenciát, hogy az egyre komplexebb SoC-k egyre nagyobb újrafelhasználást vonnak maguk után. Az ábra bal oldali y tengelyén a sávdiagramhoz tartozó átlagos IP blokkok számát SoC-ként láthatjuk, a jobb oldalin a vonaldiagramhoz tartozó dizájn újrafelhasználást százalékban.

Azonban az interfészek standardizálásának, a folyamatok, az automatizálás és a verifikáció minőségének hiányában, a tervek újrafelhasználása nem tudja biztosítani a kívánt előnyöket. Az IP-XACT ezen problémák figyelembevételével készült olyan standardizált formátum, amely egyszerre elég rugalmas, hogy több cég igényeit is kielégítse, mégis elég szigorú ahhoz, hogy automatizált folyamatokban és verifikációban is használható legyen. A folyamatautomatizálás egy nagyon előnyös

megoldás jól körül határolt, és repetitív feladatok esetében (mint például a regisztermenedzsment). A komplex automatizált rendszerek problémája, hogy mikor valami hiba történik, akkor sokszor nehéz megmondani a keletkezésének helyét a sok összetett, egymástól is függő paraméter miatt. Néha a hiba csak egy sokkal későbbi lépés során derül ki, mint ahol keletkezik. Egy másik komoly probléma, amivel automatikus rendszereknek meg kell birkózniuk az a Garbage In Garbage Out (GIGO, magyarul szemét be, szemét ki), azaz mikor rossz bemeneti adatokból dolgozunk és ezért rossz kimenetet is fogunk kapni. Ezért kiemelten fontos szerepe van a kiindulási IP-XACT minőségének (pontos legyen, teljes és az IP-vel konzisztens), illetve az integrációs folyamat során több lépésnél előforduló ellenőrzéseknek. A minél jobb minőség eléréséhez több módszer is létezik. Egy komponens IP-XACT leírása sok adatot tartalmazhat, amik sokszor más adatokkal, fájlokkal vannak összefüggésben így problémás pontos IP-XACT fájlokat kézzel készíteni és karbantartani. Egy átlagos IP könyvtár körülbelül 150 komponensből, egyenként 200-10000 soros IP-XACT fájlból áll [5]. Az IP-XACT leírás tartalmazhat buszinterfészleírásokat, regiszterleírásokat, beágyazott szoftver és hardver fájlcsomagokat, szintézis időzítéssel kapcsolatos fájlokat stb., így egy komplexebb IP leírása igen nagy IP-XACT fájlt eredményezhet. Ezeket a nagy fájlokat kézzel írni nem igazán praktikus, így a kívánt minőségű IP-XACT elérésének első lépése magának a kiindulási IP-XACT fájlok automatikus és/vagy szoftverrel segített előállítása.



2. ábra. Packaging

Ez az úgynevezett packaging (magyarul csomagolás), amelyet a 2. ábra mutat. Packaging során a releváns adatok összeszedése után létrejön a pontos és konzisztens komponensleírás. A Függelék 1. ábráján láthatjuk egy komponens fő építőelemeit.

Ahhoz, hogy meg lehessen bizonyosodni az IP-XACT helyességéről és konzisztenciájáról az IP-hez, validációt végzünk. Legelső lépésben megbizonyosodunk arról, hogy az XML betartja-e az IP-XACT sémát. Ez a séma írja le, hogy mik a megengedett elemek, illetve milyen struktúrában kell lenniük egy IP-XACT fájlban.

```
<?xml version="1.0" encoding="UTF-8"?>
<spirit:design
xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009"
xmlns:soc="http://www.duolog.com/2011/05/socrates"
xmlns:soc2009="http://www.duolog.com/2009/02/socrates">
  <spirit:vendor>arm.com</spirit:vendor>
  <spirit:library>builder</spirit:library>
  <spirit:name>cssys_m3_v6m_XjY_design.csv</spirit:name>
```

3. ábra. Egy IP-XACT fájl eleje

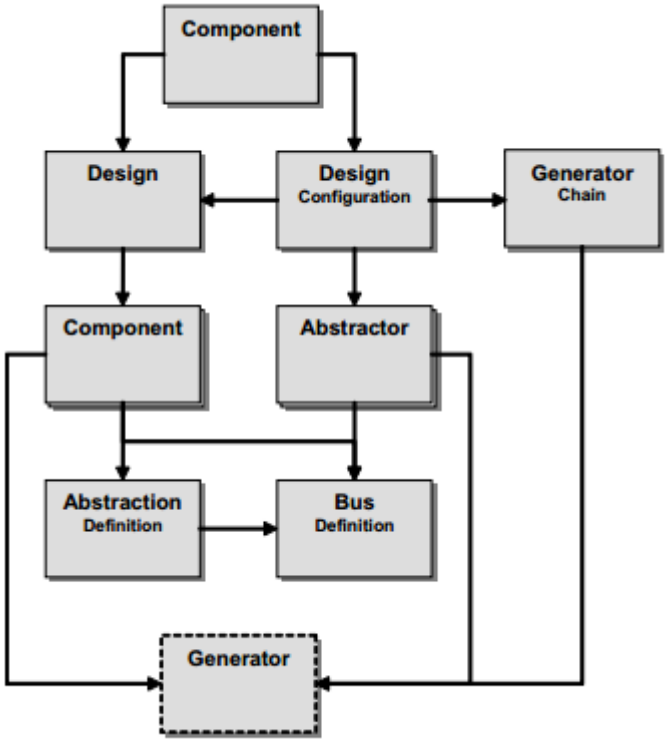
A 3. ábrán látható, hogy az XML dokumentum első jelölése tartalmazza az XML verzióját és a karakterek kódolásáról az információt, azonban a második már az IP-XACT sémákra mutató linkeket. Az IP-XACT séma ugyanis bővíthető úgynevezett gyártói bővítményekkel (angolul Vendor Extension, VE), amik segítségével az eredeti sémában nem definiált adatok is hozzáadhatóak. Validáció során nem csak az eredeti, hanem a gyártói bővítmény sémákban meghatározott szabályokat is ellenőrizzük. Természetesen a sémával csak a szintaxis és a struktúra vizsgálható, ahhoz, hogy egy IP-XACT teljességét és szemantikáját is ellenőrizzük a következő lépésben a szemantikai konzisztencia szabályok (angolul Semantic Consistency Rule, SCR) betartását kell vizsgálnunk. Ezek a szabályok is a szabványban vannak definiálva, és többek között kitérnek a csatlakozásokra, interfészek térképezésére és regiszterek ütközésére stb. A 4. ábra mutat egy példát egy ilyen szabályra:

Rule number	Rule	Single doc check	Notes
SCR 8.8	A **write-only register** shall only contain **write-only** or **writeOnce field**s.	Yes	See also: 6.10.2.2.

4. ábra. SCR példa

Ezeken kívül az IP-XACT helyességét további vizsgálatokkal is igazolhatjuk. Ezeknek a szabályok vizsgálatainak összefoglaló neve a Design Rule Checks (DRC, magyarul dizájn szabály vizsgálatok).

A szabvány lehetőséget ad komponensek, rendszerek, buszinterfészek, és az ezekhez tartozó absztrakciók, csatlakozások, paraméterek leírására is. Amikor több komponenst szeretnénk összekötni, szükségünk lesz egy tervre, ami leírja a létrehozandó kapcsolatokat, illetve a felhasznált komponensek példányait. Ez lesz az IP-XACT dizájn fájl. Az 5. ábrán az IP-XACT objektumok közötti kapcsolatokat láthatjuk részletesen.



5. ábra. IP-XACT objektumok interakciói

Feladatom szempontjából a legérdekesebbek a komponensek részletei, legfőképpen a címtérképek, regisztermezők és regiszterek, mivel ezeket fogjuk felhasználni a regiszterekkel kapcsolatos generátorokhoz. Ezeket láthatjuk a Függelék 2. ábráján.

2.1.3 Az IP-XACT hátrányai

Mivel az IP-XACT túlságosan nagy területet foglal magába és bonyolult, sokszor nem egyértelmű, hogy érdemes-e használni. Láthatjuk majd később a CMSIS-SVD-nél, hogy ha kevesebb adatra van szükségünk, sokszor egyszerűbb más formátumokat használni. Érdemes megjegyezni, hogy részint ezért, a regiszterleírás egyik legelterjedtebb módja még mindig a táblázatos/Excel formátum.

A szabvány alakításáért felelős bizottság nem az optimális munkapontban dolgozik. Sokféle ember sokféle háttérrel és nézőponttal többször csak nagyon lassan vagy egyáltalán nem is képes lépést tartani ennek a gyorsan fejlődő iparágnak az igényeivel.

A gyártói bővítményekkel is akadnak problémák. Használatuk során például sérül az IP-XACT egyik legnagyobb előnye, a sokféle fejlesztőkörnyezet és eszköz közötti átjárás. Ezt lehet valamennyire kompenzálni azzal, hogy az IP-XACT fájlba vagy mellé megfelelő részletességgel leírjuk a gyártói bővítmények specifikációját.

2.1.4 IP-XACT alternatívák

A regiszterleírás egyik legegyszerűbb módja a táblázatos (CSV, XLS) formátum. Erre láthatunk egy példát a 6. ábrán. A Microsoft Excel alkalmas az így tárolt adatok manipulálására, így bizonyos regisztermenedzsment feladatok el is végezhetőek vele. Probléma azonban egy ilyen regiszterleírás esetén magának a formátumnak a kialakítása. Mivel nincsen ipari standard ezért cégek vagy programok között nehézkes az átjárás. Több száz sor Excel karbantartása, formázása kényelmetlen. A regiszterleírások specifikációját mégis általában még mindig ebben adják meg a projekt elején, és ezért a legtöbb regisztermenedzsmenttel foglalkozó program rendelkezik Excel import funkcióval.

Register				Fields			
Offset	Name	Access	Reset	Offset	Name	Access	Width
0x000	SPCSECCTRL	read-write	0x0000_0000	0	SPCSECCFGLOCK	W1S	1
				1	Reserved	read-only	31
0x004	BUSWAIT	read-write	Parameterized	0	ACC_WAITN	read-write	1
				1	Reserved	read-only	15
				16	ACC_WAITN_STAT	read-only	1
				17	Reserved	read-only	15

6. ábra. Regiszterleírás Excel-ben

Az IP-XACT mintájára az Accellera egy direkt regiszterleírásra specializálódott formátumot is létrehozott. Ez a SystemRDL (System Register Description Language, magyarul rendszer regiszter leíró nyelv). Jelenleg a legújabb (2009-ben jelent meg) 1.0-s verziója már nagyon elavultnak számít, csak régi rendszerekkel való kompatibilitás miatt használják. Több hiányossága is van, nincsenek benne a verifikációs környezetek által használt memória objektumok, vagy hiába direkt RTL generálásra lett kialakítva mégsem tud eleget ez a szabvány ennek a feladatnak az elvégzéséhez.

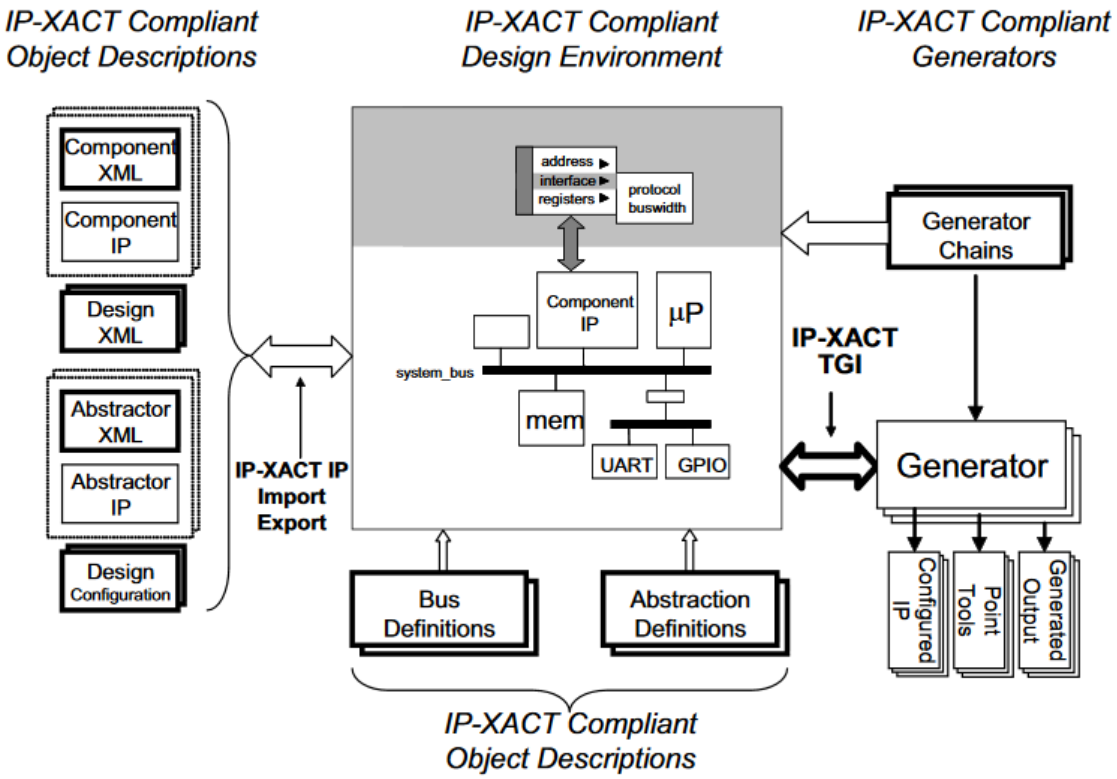
Az IP-XACT komplexitása és egyéb problémái következtében sok cég próbálkozik a saját regiszterleíró nyelvének használatával. Ezek problémája mind az, hogy nem lehet fejlesztőkörnyezetek között megfelelően váltani, csak az adott programmal vagy programcsomaggal működik.

2.1.5 IP-XACT fejlesztőkörnyezet

Egy IP-XACT fejlesztőkörnyezet (angolul Design Environment, DE) feladata olyan rendszerleírásokat készíteni és karbantartani, hogy a fejlesztési és implementációs folyamatokat hatásosan, az újrafelhasználhatóságot is előtérbe helyezve segítse azáltal, hogy képes összehangolni a különböző segédeszközöket és az IP-ket.

A 7. ábrán is látható, hogy egy IP-XACT alapú fejlesztőkörnyezet többféle módon biztosít lehetőséget közvetlen integrációra alkalmas IP-XACT fájl létrehozására. Ennek egyik útja, hogy a fejlesztőkörnyezet kinyeri a szükséges információkat a

különféle forrásfájlokból (általában RTL, Excel), majd létrehoz egy új fájlt (Packaging). Másik lehetőség, hogy egy már rendelkezésre álló IP-XACT fájlt importál be a fejlesztőkörnyezet, adott esetben annak konfigurációját is elvégezve.



7. ábra. Egy IP-XACT fejlesztőkörnyezet felépítése

Amint megvan az integrációra alkalmas IP-XACT fájl, a következő lépés az integráció. Ez az a folyamat, ami során létrejön(nek) a rendszerleíró dizájn fájl(ok). A fejlesztőkörnyezet feladata, hogy biztosítsa az ehhez szükséges szerkesztési módokat, tervrajzi vagy egyéb nézeteket, hogy egyszerű legyen navigálni és pontosítani a dizájnon.

A fejlesztőkörnyezetbe betöltött IP-XACT fájlokon végzett dizájn szabály vizsgálatok elvégzése is a DE feladata. Egyes fejlesztőkörnyezetek saját dizájn szabály vizsgálatok hozzáadásával is bővíthetik az IP-XACT fájlok minőségének ellenőrzését a szabványban leírt szabályok mellett.

2.1.5.1 IP-XACT generátorok

A generátorok futtatható objektumok (szkriptek vagy bináris programok), amik lehetnek belsők azaz a fejlesztőkörnyezet integráns elemei, vagy külsők, azaz a fejlesztőkörnyezettől függetlenül állnak rendelkezésre. Generátorokat tartalmazhatnak az IP csomagok is, például egy konfigurálható IP számára (mint például egy busz-mátrix generátor). Ekkor a konfigurálást a generátor végzi el az IP-n, a bemeneti paraméterek alapján.

Egy belső generátor sokféle feladatot el tud látni és hozzáférhet bármely a fejlesztőkörnyezet által biztosított módon az IP-XACT adatokhoz. Az IP-XACT szabvány nem írja le ezeket a protokollokat.

Egy külső generátor (sokszor TGI generátorként hivatkoznak rá, Tight Generator Interface – magyarul tömör generátor interfész) egy futtatható program vagy szkript amit a fejlesztőkörnyezetből lehet futtatni, hogy lekérdezze vagy konfigurálja a dizájnok és komponensek leírásait. Ezt a TGI segítségével teszik (innen a név), ezen keresztül érik el és változtatják a fejlesztőkörnyezetbe éppen betöltött IP-XACT adatokat. A szabvány kiköti, hogy a külső generátorok csak és kizárólag a TGI-n keresztül férhetnek hozzá az IP-XACT adatokhoz és magát a TGI-t is definiálja. Ezáltal eléri, hogy a külső generátorok fejlesztőkörnyezettől függetlenek lesznek.

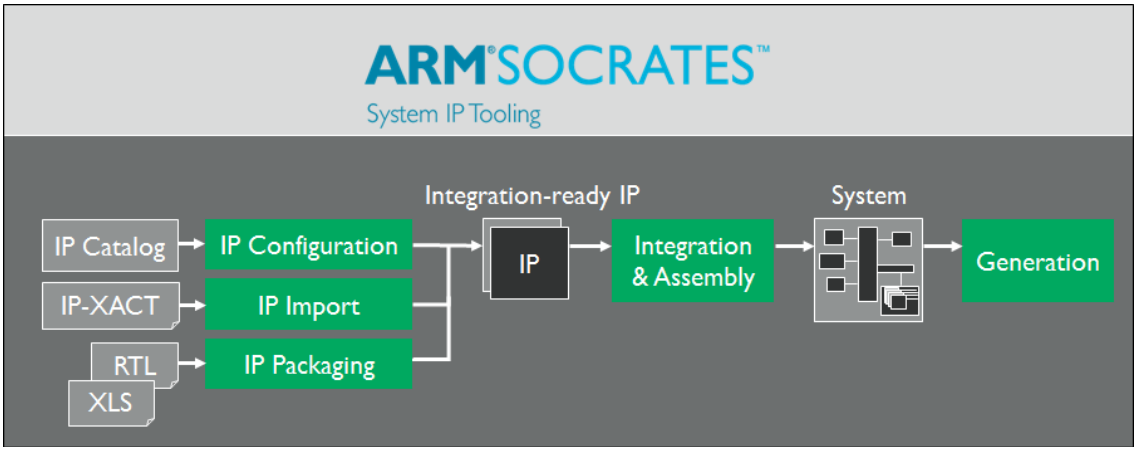
2.2 ARM Socrates DE

Az ARM Socrates DE egy IP-XACT fejlesztőkörnyezet, ami segítségével könnyen és gyorsan lehet ARM alapú rendszereket intelligensen konfigurálni és összerakni.

Az intelligens konfigurálás azt takarja, hogy konzisztens, egyszerűen használható, könnyen érthető, automatizálható és ellenőrzéseket is tartalmaz. Ezek során felhasználja az ARM által felhalmozott rendszerdizájnnal kapcsolatos tudást.

Grafikus interfészen (angolul Graphic User Interface, GUI) vagy parancssori interfészen keresztül (angolul Command Line Interfész, CLI) lehet használni az ARM

Socrates DE-t. A grafikus interfészről a Függelék 3. és Függelék 4. ábrája mutat képet, a parancssori interfész használatára pedig a Függelék 5. elemének, a generátor használati utasításának második felében található példa.



8. ábra. Egy tipikus ARM Socrates DE munkafolyamat

A 8. ábra mutatja, az ARM Socrates DE munkafolyamatának lépéseit, zöld színnel jelölve az elvégzendő feladatokat. A grafikus felület két különböző nézettel rendelkezik, amik az éppen elvégzendő feladathoz vannak szabva. Az IP Config nézet az ábra bal oldalán lévő első 3 feladatra van specializálódva, az IP Integration nézet pedig az Integration & Assembly (magyarul integrálás és összerakás) feladathoz. Más ablakokat tartalmaz a grafikus interfész a nézetektől függően. A Függelék 4. ábráján látható, hogy az IP Config nézetben látható az IP Catalog, ami tartalmazza a konfigurálható ARM IP-k katalógusát. A Függelék 3. ábrája azt mutatja, hogy az IP Integration nézetben látható a Data Explorer, aminek az ablakában navigálhatunk a projektünkben található adatok között.

A Packaging ebben az IP-XACT fejlesztőkörnyezetben RTL (Register Transfer Level, magyarul regiszter átviteli szint) vagy XLS (a Microsoft Excel formátuma) fájlokat használ forrásnak. Az RTL importálás Verilog vagy VHDL fájlokkal tud dolgozni. Elsődlegesen a peremen lévő portokból, illetve a hozzájuk tartozó paraméterekből hozza létre az IP-XACT fájlokat. Az XLS fájlokban a regiszterekre és a memóriatérképekre vonatkozó információk tárolhatóak, így egy komponens ezen adatait lehet importálni ezekből.

Az ARM Socrates DE biztosítja, hogy szabályosan legyenek beállítva a konfigurált komponensek, és hogy az elkészült rendszerek is érvényesek. Ezt a dizájn szabályok ellenőrzésével teszi, mint például a szabványban előírt SCR vagy az

úgynevezett Verilog Netlisting Checks (magyarul Verilog hálólista ellenőrzések). Az utóbbiak biztosítják, hogy az IP-XACT adatból helyes Verilog legyen generálható. A grafikus interfész lehetőséget nyújt a dizájn szabály vizsgálatok megjelenítésére és beállítására is.

A 8. ábra tartalmaz még a fejlesztőkörnyezet által megvalósítandó feladatot, amit csak megemlítek, ez a Generation. A fejlesztőkörnyezet kimeneti fájlok előállítására is képes generátorok segítségével. Az ARM Socrates DE-ben előállítható kimenet például a Verilog vagy az Excel Connectivity Table, ami nem más, mint az IP-XACT fájlban leírt kapcsolódások Excel formátumban történő leírása.

Az ARM Socrates DE szkript alapú alkalmazásprogramozási interfésze lehetőséget biztosít arra, hogy a grafikus felületen kézzel elvégzett feladatokat szkripteken keresztül végezzük el. A támogatott nyelvek a Ruby, a TCL és a Python. Ezek közül a Ruby nyelven létezik egy alkalmazásprogramozási interfész, ami megkönnyíti a különféle dizájnelemek elérését, létrehozását és változtatását. Erre az interfészre a továbbiakban Ruby API-ként fogok hivatkozni [6].

2.3 A Ruby programozási nyelv

Mivel a Ruby API már rendelkezik több számomra is szükséges függvénnyel, illetve a szkriptnyelvek közül ebben otthonosan mozgok, ezzel fog történni a feladat megvalósítása.

A Ruby egy dinamikus programozási nyelv, komplex, de egyúttal kifejező nyelvtannal és rendkívül bő alapkönyvtárral. Kölcsönvesz megoldásokat a Lisp, Smalltalk és Perl nyelvekből, és emellett nyelvtana könnyen megtanulható C vagy Java programozók számára. Az alapító Yukihiro „Matz” Matsumoto szavaival élve: „A programozók boldogságára lett a Ruby tervezve.”. A Ruby egy tisztán objektum-orientált programozási nyelv, azonban alkalmas procedurális és funkcionális programozásra is [7].

2.3.1 Embedded Ruby

Az Embedded Ruby (ERB) egy standard Ruby modul, aminek a segítségével valósítom meg a generátorokat. Ez a modul képes bármilyen szöveg előállítására mintákból (angolul template). A template-ek maguk egyszerű szövegből és Ruby kód kombinációjából állnak.

Template:

```
<%- 3.times do |number| -%>
This is line number <%= number + 1 -%>.
<%- end -%>
```

Kimenet:

```
This is line number 1.
This is line number 2.
This is line number 3.
```

A Ruby kód részletek változók kiértékelésére és a felépítés irányítására is szolgálnak, így sok különféle fájl előállítására alkalmas az ERB. Leginkább sok hasonló ismétlődő struktúrából álló fájlok esetén a leghasznosabb, mint például XML dokumentumok, weboldalak, hasonló tesztek stb.

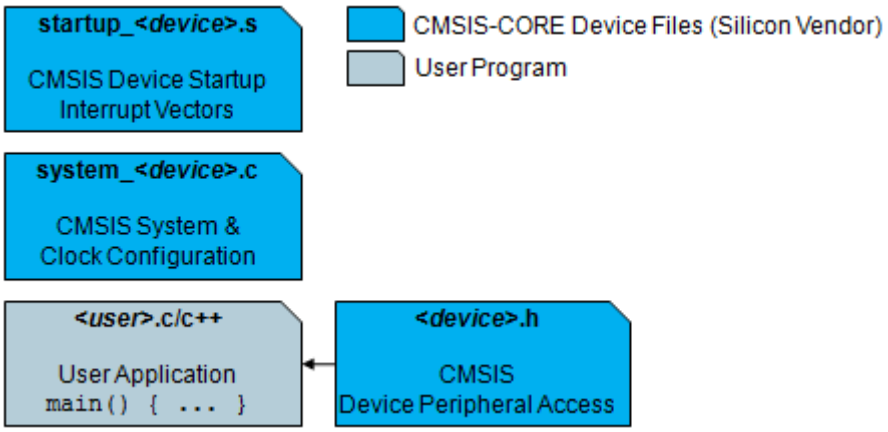
2.4 Cortex Microcontroller Software Interface Standard

Az ARM Cortex Microcontroller Software Interface Standard (CMSIS) nem más, mint egy olyan projektcsokor, aminek a feladata az ARM Cortex-M processzorokra történő szoftverfejlesztés segítése. Az első ilyen projekt a CMSIS-CORE (magyarul mag) volt, ami egy gyártófüggetlen hardverabsztrakciós réteg a Cortex-M processzorcsalád részére. Azóta a projekt több területre is kiterjedt, mint például a CMSIS-RTOS API (Real Time Operating System API, magyarul valós idejű operációs rendszer alkalmazásprogramozási interfésze) vagy a későbbiekben bemutatandó CMSIS-SVD (System View Description, magyarul rendszernézeti leírás). A beágyazott rendszerek világában a szoftver előállítása a költségek egy jelentős része.

Az összes Cortex-M termék szoftverinterfészeinek standardizálása ezt a költséget csökkenti, legfőképpen új projektek létrehozásánál vagy meglévő szoftver új készülékre történő átültetésénél. A CMSIS konzisztens és egyszerű szoftverinterfészt biztosít a processzor felé, ezzel egyszerűsíti a szoftverújrafelhasználást, így csökkenti a beágyazott rendszerfejlesztőknek szükséges ismereteket, és jelentősen rövidíti a piacra kerülési időt [8].

2.4.1 CMSIS-CORE

A 9. ábra mutatja, hogy a CMSIS-CORE használatához az alkalmazásunkhoz kell adnunk a következő fájlokat: startup_<device>.s (az újraindítást és kivételeket leíró táblákat tartalmazza), system_<device>.c és system_<device>.h (általános készülékkonfigurációs beállításokat tartalmaznak), illetve a <device.h> (a processzormag és a perifériák elérését biztosítja, továbbiakban ezzel foglalkozunk, sokszor egyszerűen header fájlként fogok rá hivatkozni).



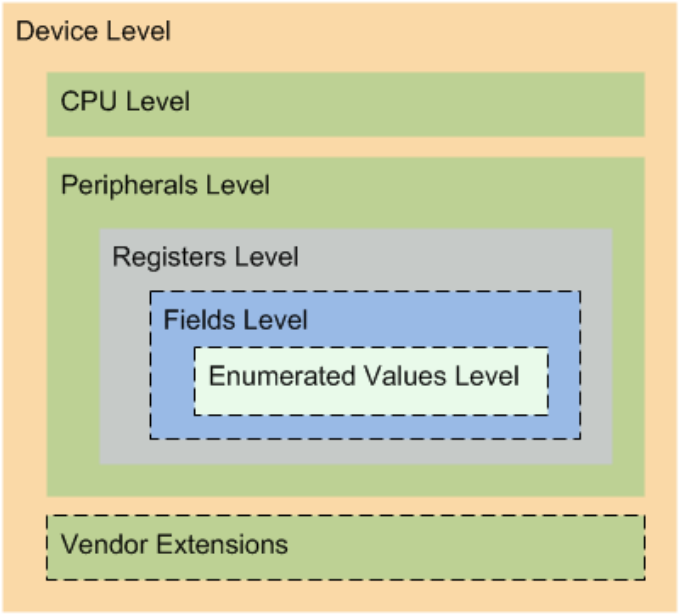
9. ábra. CMSIS-CORE használatához szükséges fájlok

Ezt a header fájlt használja az alkalmazást fejlesztő programozó a C forráskódban. Funkciói közül a perifériák elérését standardizált regiszterleírással valósítja meg. Minden perifériához tartozik egy regiszterelrendezés, egy báziscím és az elérési módok leírása. Ezen kívül megszakítás- és kivételdefiníciók, illetve bizonyos processzor beállítások találhatóak ebben a header fájlban. Ezt a fájlt a szilíciumlapkák gyártójának kell létrehoznia.

2.4.2 CMSIS-SVD

A CMSIS projektcsomag egy másik tagja a CMSIS-SVD formátum, ami az ARM Cortex-M processzor alapú beágyazott rendszerek leírását önti formába, különös tekintettel a perifériák memóriára térképezett regisztereire. A CMSIS-SVD fájlokban található leírások részletessége összevethető az eszközök kézikönyveiben találhatókkal. A tárolt információk a perifériák magas szintű funkcionális leírásától, egészen az egyes regisztermezők részletes definíciójukig terjednek. Ezeket a CMSIS-SVD fájlokat egyes fejlesztési segédeszközök eszközspecifikus debuggolásra használják, mivel a debuggerből elérhetőek részletes információk a perifériákról, regiszterekről, regiszter mezőkről és megszakításokról anélkül, hogy az eszköz-dokumentációra kellene hagyatkozni. A CMSIS-SVD fájlok egy másik felhasználása a CMSIS-CORE kompatibilis header fájlok generálása. A közös forrás biztosítja, hogy a header fájl és a debuggerben látható információ konzisztens egymással.

A CMSIS-SVD egy XML formátum, amit az IP-XACT inspirált. azonban az IP-XACT sokkal tágabb területe és komplexitása miatt létrehozták ezt a külön formátumot, ami direkt az eszköz programozói nézetének leírására fókuszál. Egy CMSIS-SVD fájl egy eszköz leírását tartalmazza.



10. ábra: CMSIS-SVD hierarchiaszintjei

A 10. ábrán láthatjuk az építőelemek kapcsolatait. Egy eszköz egy processzorból és legalább egy perifériából áll. Minden periféria tartalmaz legalább egy regisztert, minden regiszter felépülhet egy vagy több regisztermezőből. A regisztermezők értékeit tovább lehet magyarázni, úgynevezett felsorolt konstansokkal. A CMSIS-SVD fájl felső szintjén (Device Level) találhatóak az egész eszközre vonatkozó információk, mint például az eszköz neve, rövid leírása vagy verziója. A regisztertulajdonságokra vonatkozó alapértelmezett értékek (például a regisztertméret, újraindítási érték vagy elérési mód) beállíthatóak ezen a szinten és az alacsonyabb szinten lévő elemek megörökölik őket. Azonban, ha az alacsonyabb szinten specifikálva van az érték, az felülírja a felsőbb szinten beállított alapértelmezett értékeket. A processzor szint (CPU Level) az eszközben található processzor leírását tartalmazza, ami nélkül nem lehet CMSIS-CORE device header fájlt generálni. A periféria szinten (Peripheral Level) a perifériához tartozó információk, mint például az eszköz címterületén definiált báziscím, a hozzá tartozó megszakítások, illetve a címblokkok találhatóak. Ezek a címblokkok úgy vannak kialakítva, hogy a regiszter szinten (Register Level) leírt regiszterek mind elférjenek bennük. Minden, a regiszterek által nem felhasznált címterület automatikusan foglaltnak (angolul reserved) tekintett. A CMSIS-SVD fájl tartalmazhat még egy az IP-XACT formátumhoz hasonló gyártói bővítményeket, amik segítségével az eredeti sémában nem definiált információk is hozzáadhatóak [9].

A CMSIS-SVD támogatja a többszörös példányosítását a perifériáknak, a regisztereknek vagy a regiszter mezöknek. A „derivedFrom” tulajdonság megmutatja a másolandó forrást, és a kívánt tulajdonságok újradefiniálhatóak a megfelelő jelölések segítségével.

CMSIS-SVD formátum sikeres használatának kulcsfontosságú része a leírás minősége, aminek több eleme is van: szintaktikailag és strukturálisan is meg kell felelnie a specifikált CMSIS-SVD formátumnak, konzisztensnek, teljesnek, pontosnak és megfelelően részletesnek kell lennie. Ezt két lépésben lehet ellenőrizni: részint az XML eszközök lehetőséget adnak az adott CMSIS-SVD sémafájllal való összevetésre, illetve létezik egy CMSIS-SVD átalakító eszköz az SVDConv, ami ezen felül a tárolt adatok szemantikáját és konzisztenciáját is ellenőrzi. Ez az átalakító használható még a CMSIS-CORE kompatibilis header fájlok generálására is.

2.5 Regisztertesztelés

A regiszterek tesztelése egy rendkívül fontos része egy dizájn verifikációjának. Az egyik legelső, amit tesztelni kell, ugyanis a többi funkcionalitás a regiszterimplementáció pontosságától is függ, mivel a regiszterekben találhatóak a hardver konfigurációs beállításai és a hardver/szoftver interfész alapja.

A regiszterek verifikációját sokszor processzor alapú tesztekkel egyszerű elvégezni. Itt C nyelven írt regiszter tesztekkel történik a tesztelés. A legegyszerűbbek azok a tesztek, amelyek a hardver (újra)indítása utáni értékeket ellenőrzik. Itt egyszerűen újraindítják a rendszert, kiolvassák a regiszter értékét, majd összehasonlítják a specifikációban megadott újraindítási értékkel.

Az egyszerű írható-olvasható regisztereknél más teszteket is lehet végezni. Amennyiben lehetséges, véletlenszerű értékeket írnak a regiszterbe, majd ellenőrzésképpen visszaolvassák azokat. Csak olvasható regisztereknél hasonlóan járnak el, csak itt a sikertelenséget várják, azaz azt, hogy a visszaolvasott érték független a beírt értéktől. Szükséges még a csak írható és a direkt foglaltnak jelölt regisztereket is vizsgálni, amely esetekben a helyes működést a sikertelen olvasási kísérlet jelenti.

Egyes regiszterek a beléjük írt adatot nem egy az egyben tárolják. Ilyen lehet például a „zeroToToggle” (magyarul nullával vált), amely viselkedése olyan, hogyha nullás bitet írnak valamelyik regisztermezőbe, akkor az a bit értéket vált.

Léteznek speciális elérésű regiszterek is, például két címen egy fizikai regiszter érhető el, egyiken olvasható, másikan írható, vagy olyan, aminek elérhetősége egy másik regiszter beállításától függ.

Egyes konfigurációs regiszterek tesztelése ennél még egy fokkal bonyolultabb is lehet, például egy időzítőt beállító regiszter funkcionalitása az alap órajel segítségével történhet.

A bonyolultabb regiszterek automatikus teszteléséhez a kapcsolatok leírására is szükség van, amelyet az IP-XACT szabvány nem definiál egyértelműen.

2.6 HyperText Markup Language

A HyperText Markup Language (HTML) a svájci CERN-ben jött létre a nyolcvanas években, mikor a kutatók azzal szembesültek hogy a korabeli dokumentum formátumok nem felelnek meg a fizikusok által előállított kutatási eredmények megjelenítésére. A két alapötlet, ami meghatározza a HTML nyelvet azóta sem változott azaz, hogy a dokumentum ne csak egyszerű szöveg legyen hanem a tartalmat hordozó részt lássuk el címkékkel, amik kiegészítő információkat (metaadatokat) adnak a szöveghez, illetve, hogy a dokumentumok mutassanak túl önmagukon, azaz olyan hiperszövegeket tartalmazzanak, amelyek egy másik dokumentumra hivatkozhat (mint például weboldalakon a linkek). A HTML nyelv jelölőrendszere elemekből és attribútumokból épük fel [10]. Az elemekkel tudjuk megjelölni a szöveg egyes részeit, és az attribútumokkal ezen jelölések tulajdonságait tudjuk meghatározni, mint ahogy a 11. ábrán is láthatjuk.

```
<h1 style="font-size:300%;">Ez egy címsor</h1>
<p style="font-size:160%;">Ez egy paragrafus</p>
```

11. ábra. A HTML kód és mit látunk a böngészőben (font-size, magyarul betűméret)

Az XML-hez képest a legnagyobb különbség a HTML-ben az, hogy a jelölések mind előre definiálva vannak és nem lehet újakkal kiegészíteni őket. A két formátum együtt kiválóan működik, a HTML az előre definiált jelölései segítségével az adatok megjelenítésére kiváló, míg az XML-t rugalmassága miatt maguknak az adatoknak a tárolására érdemes használni.

3. A generátor tervezése

3.1 CMSIS-SVD generátor

A feladatom egy olyan generátor megvalósítása, ami IP-XACT fájlból a megfelelő információkat kinyerve elkészíti a készülékhez tartozó C header fájlt. Az SVDConv egy olyan szoftver, ami át tudja alakítani megfelelő C header fájllá a CMSIS-SVD fájlokat. Az én generátorom feladata a CMSIS-SVD fájl előállítása lesz. A generálási folyamatot mutatja a 12. ábra.

12. ábra. A munkafolyamat vizualizálva

Megfontolás tárgyát képezte, hogy a generátor egyből a header fájlt generálja. Az ARM-on belül már létezik a CMSIS-SVD fájlból a C header fájlt előállító program, a már említett SVDConv. Éppen ezért nem lenne szerencsés ugyanannak a feladatnak az elvégzésére mellé másikat is készíteni, mert ekkor, ha például a CMSIS-CORE-t frissítjük akkor ez a változtatás több generátor módosítását is maga után vonná.

A CMSIS-SVD szintén egy XML formátum vannak, mint az IP-XACT. Elviekben lehetőség lenne eXtensible Stylesheet Language-t (XSLT) használni az XML-XML átalakításra. Ennek a nyelvnek bonyolultsága, és a létrehozandó függvények egyéb generátorokban, általánosabb felhasználásának lehetősége miatt nem ezt használtam. Ehelyett az ARM Socrates DE-t és a hozzá tartozó Ruby API-t használtam fel az IP-XACT fájl beolvasására és a CMSIS-SVD fájl létrehozását elvégző generátor futtatására.

A generátort megvalósító Ruby szkriptet, segédmodult és az ERB template-et Notepad++ szövegszerkesztőben hoztam létre. Az ARM Socrates DE-ben is lenne lehetőség a létrehozásra, azonban a Notepad++ kényelmi funkciói nagyon előnyösek számomra. Ilyen például az automatikus kiegészítés, a blokk kiemelés, a keresés és a

csere több dokumentumon keresztül, továbba az ERB szerkesztésénél különösen fontos a személyre szabható szintaxiskiemelés.

A bemeneti IP-XACT fájlal kapcsolatban bizonyos feltételeket szabtam. A leírás helyességét elvárom, nem vizsgálom, erre az ARM Socrates DE tartalmaz eszközöket, a beépített dizájn szabály vizsgálatokat. A generátor bemeneti IP-XACT fájlja lehet egy levélkomponens (egy hierarchia ág végén, egyedül található komponens) vagy pedig felső komponens, amiben az összes alkotóelem címblokkja, regisztereinek leírása össze lett gyűjtve. Erre egy segítő szkriptet is írtam, ami összegyűjti egy projektben lévő komponensekből a regiszterleírásokat és az általam elvárt formátumban adja őket vissza.

Csak az IP-XACT fájlban szereplő információk alapján nagyon hiányos lenne a generált CMSIS-SVD fájl, mivel a regiszterleíráson kívül egyéb, IP-XACT fájlból nem kinyerhető adatot is tárol. A megfelelő C header fájl generálásához ezeket az adatokat hozzá kellett valahogy adnom. A generátor ezt konfigurációs paramétereken keresztül valósítja meg, ahol ez lehetséges. Egyes esetekben, például a megszakításoknál ez nehézkes lenne, ezért ezeket a felhasználóra hagyom, hogy kézzel adja hozzá a generált fájlhoz.

A kimeneti CMSIS-SVD fájl nem tartalmazhat hibákat, hogy az SVDConv át tudja alakítani megfelelő C header fájllá. Ehhez fontos, hogy a kimenet megfelelő struktúrát vegyen fel (pontos template), az IP-XACT fájlból kinyert adatok megfelelő formátumban legyenek átadva és a kötelező jelölések mindenképp legyenek kitöltve a generált CMSIS-SVD fájlban.

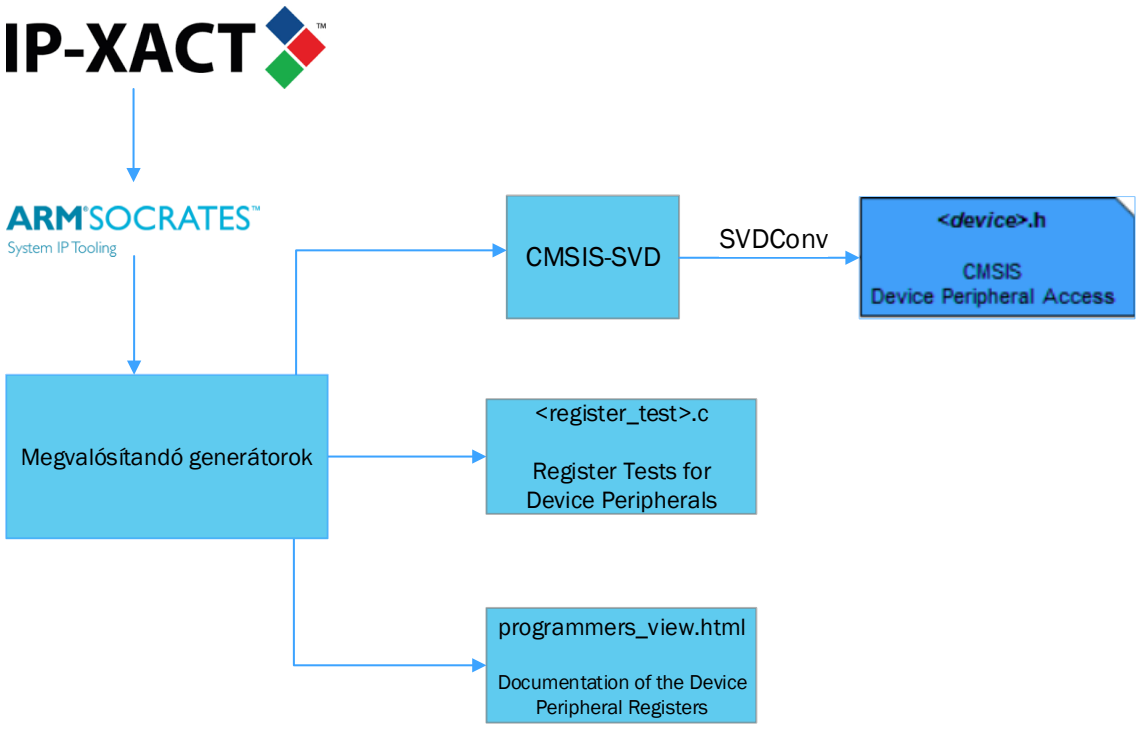
3.2 A feladat bővítése

A generátor megvalósítása közben egy mérnökcsapat munkafolyamatába is betekintést nyertem, ami ötleteket adott egyéb generátorok megvalósítására is.

A CMSIS-SVD fájlból generált C header fájl tartalmazza a perifériák leírását, többek között a regisztereket, amikhez bizonyos tesztek automatikusan előállíthatóak. A cél, hogy miután elkészítettem az IP-XACT fájlból a CMSIS-SVD fájlt, abból a C header-t, az IP-XACT fájlból olyan regisztereket generálok, amiket a C header

mellé téve működnek. Ezt úgy teszem, hogy az eredeti C header-ben található regiszterleírásokat megvizsgálom, hogy hogyan jön létre az IP-XACT fájlból, majd mikor már rendelkezésre áll, hogy hogyan tudok az egyes regiszterekre hivatkozni akkor ez alapján már tudok teszteket is generálni. A regisztertesztek automatikus volta miatt csak a kezdeti értékek vizsgálatából, írható-olvasható regiszterekbe való véletlenszerű adat írásából, csak olvasható regiszterek nem írhatóságából és a fenntartott regiszterek nem használhatóságából állnak. A változtatott írási értékű regiszterek tesztjeire is van lehetőség. Jelenleg is vizsgálom, hogy az egyéb speciális regiszterek tesztjeinek automatizálásának van-e valamilyen módja ebben a munkafolyamatban. A CMSIS-SVD fájlból generált C header fájlban a regisztermezők lehetnek makróval, struktúrával vagy enumerációval leírva, ezért a generátornak szükséges a paraméter, hogy miként fogja tudni elérni a regisztermezőket.

A regisztermenedzsment egy másik problémája a dokumentáció szinkronizációja az adott változtatásokkal. Mivel a regiszterek információi mind rendelkezésünkre állnak, a dokumentáció elkészítése ezeken az adatokon alapulhat. A kimeneti formátumnak a HTML-t választottam. Az így kialakult pontosított feladat összefoglalását a 13. ábra mutatja.



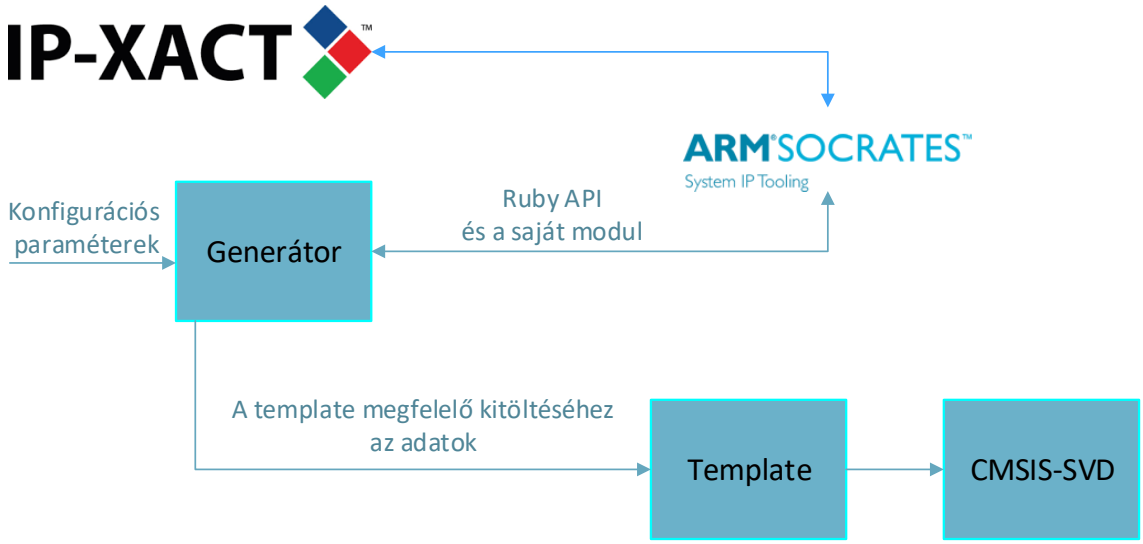
13. ábra. A teljes feladat összefoglalása

A feladathoz hozzátartozik az elkészített generátorok megírásán kívül a generátorok tesztelése, amelyet az 5. fejezetben ismeretetek részletesen, valamint a használati utasítások elkészítése is. Az utóbbinál szem előtt kell tartanom, hogy a felhasználó, aki futtatni szeretné a generátorokat nem biztos, hogy ismeri az ARM Socrates DE-t, ezért részletes leírást kell nyújtanom az egész folyamat elejétől (például az IP-XACT beolvasásáról vagy egy új ARM Socrates DE projekt létrehozásáról) részletezve, hogy parancssori vagy grafikus módban milyen különböző lépéseket kell megtenni, hogy a generátorok megfelelően működjenek. A használati utasítás része még a különböző konfigurációs paraméterek és a generátorokat érintő limitációk részletezése is. A Függelék 5. eleme a használati utasítás egy részletét tartalmazza.

4. A generátorok bemutatása

4.1 CMSIS-SVD generátor

A CMSIS-SVD generátor feladata az adott IP-XACT fájlból kinyerni a regiszterekkel kapcsolatos információkat, majd a megfelelő CMSIS-SVD fájlt generálni. Ennek a folyamatát látjuk a 14. ábrán.



14. ábra. CMSIS-SVD generátor működése

A generátor egy Ruby szkriptből, egy template (ERB) fájlból és pár kiegészítő könyvtárból áll. A szkriptet a későbbiekben felhasznált könyvtárak betöltésével kezdem: erb, fileutils, rexml, IPXACT2009 és cmsis_svd_helper_library. Az erb a template felhasználásával, a fileutils az általános fájlrendszerrel, rexml az XML-lel kapcsolatos függvényeket tartalmazza, míg az IPXACT2009 az ARM Socrates DE-hez tartozó Ruby API és az cmsis_svd_helper_library pedig a saját kiegészítésem, amibe tipikusan a generátorhoz írt függvényeket raktam. Az utolsó kettő segítségével az IP-XACT

adatokhoz tudok hozzájutni az ARM Socrates DE-be betöltött IP-XACT fájlból. Erre azért van szükség, hogy a template-et megfelelően tölthessem ki.

A generátort úgy terveztem, hogy több konfigurációs paraméterrel legyen meghívható, például a szkriptnek át tudom adni, hogy melyik könyvtárból vegye fel a template-et, hogy milyen könyvtárba mentse a kimenetet, illetve több IP-XACT fájlból hiányzó, de CMSIS-SVD fájlban tárolható információt is megadhatok ezek segítségével. Jellemző példa erre a CPU-val kapcsolatos információk, mert az IP-XACT fájlban erről semmi sincs tárolva:

```
cpu_name = getConfigItem("cpu_name", :default=>'other')
cpu_icache = getConfigItem("cpu_icache")
```

Egyes paramétereknek alapértelmezett értékeket is adunk, mivel a CMSIS-SVD kötelezően elvárhat olyan adatokat, amik nincsenek számunkra leírva az IP-XACT fájlokban.

4.1.1 A generátor működése

Ezen paraméterek felvétele után a generátor megkeresi a dizájnt, ha létezik és a legfelső komponenst. Ezeket már a megfelelő formátumban várom (nem foglalkozom azzal hogyha valaki nem megfelelő/elvárt IP-XACT fájlból indul ki), mivel az ARM-Socrates DE-ben található dizájn szabály vizsgálatok elvégzik a kiindulási fájlok minőségét. Ezek után az erb könyvtár segítségével elindítom a generálást.

```
erb = ERB.new(File.read(template_dir + "/Copyright.erb"), nil, '-')

copyright = erb.result()

erb = ERB.new(File.read(template_dir + "/cmsis_svd_device.erb"), nil, '-')

output = copyright + erb.result(binding)
```

Az ERB.new első paramétere kijelöli a template-et, amit felhasználunk, a második pedig, ha nem nil-re van állítva akkor az ERB kód egy külön szálon fut, a megadott értékre beállítva Ruby biztonsági szintjét és az utolsó paramétere állítja be a trim_mode-t (magyarul rövidre vágás mód). A vágási módok közül az itt is használt ’-’ azt jelenti, hogy minden ’-%>’-re végződő üres sort elhagyunk, így a template-ben lévő logikai jelölések nem okoznak üres sorokat. A fenti kódrészletben láthatjuk, hogy az

első template a ’Copyright’ amiben a kész fájl elé berakott szerzői joggal kapcsolatos szöveg található. A második template pedig magához a CMSIS-SVD fájl elkészítéséhez szükséges. A template-be felváltva pakoltam a CMSIS-SVD fájlba kiírandó jelöléseket, és az ebbe beágyazott kiértékelt, illetve a felépítést irányító logikai ERB jelöléseket. Egy CMSIS-SVD fájlt az XML jelöléssel kezdek mindig, majd a sémát leíró jelölés jön. A CMSIS-SVD fájl specifikációjának megfelelően a készülékkel kapcsolatos jelölésekkel folytatom. Már itt is sok a logikai ERB jelölés, az opcionális adatoknál megvizsgálom, hogy a generátor meghívásakor a szkriptnek át lett-e adva az adott adat vagy ha nem akkor kihagyjuk a jelölést. Például:

```
<%- if !licence.empty? -%>
  <licenseText>
    <%= licence -%>
  </licenseText>
<%- end -%>
```

Az általános módszer a template kitöltésére, ha opcionális a jelölés megvizsgálom, hogy van-e a megfelelő adat, általában egy függvény segítségével, ami az IP-XACT fájl megfelelő mezőit az ARM Socrates DE segítségével vizsgálja, majd a CMSIS-SVD séma által meghatározott jelölések közé beillesztem a megfelelő adatot. Egy egyszerű példa:

IP-XACT:

```
<spirit:access>read-write</spirit:access>
```

Template:

```
<%- if getAccessType_SVD(register) -%>
    <access><%= getAccessType_SVD(register) -%></access>
<%- end -%>
```

SVD:

```
<access>read-write</access>
```

A CPU-val kapcsolatos és az alapértelmezett értékek jelölései után a perifériák következnek. Minden perifériához egy-egy címblokk tartozik a felső szintű komponensben, ezeken lépkedek végig. A perifériák regiszterfájljain a foreachRegisterFile függvényem megy végig. A regiszterfájlnak megfelelő egység a CMSIS-SVD sémában a cluster. A feladatkiírás pontosításában megfogalmazott problémák miatt csak egy mély szerveződést támogatok jelenleg, tehát minden regiszterfájlon végig megyünk, létrehozzuk a hozzá tartozó cluster-t, illetve a benne

lévő regiszterleírásokat, majd a legvégén a felső szinten lévő regisztereket is. Ezt úgy éri el a függvény, hogy a regiszterfájlokat tároló tömb utolsó elemének beilleszti a felső szintet azaz a címblokkot és van egy opcionális paramétere a show_last (magyarul mutasd az utolsót), amit, ha igaznak állítunk be az iteráció során a második változó akkor lesz igaz, ha az utolsó elemnél járunk, ami ugye a felső szint lesz.

```
  <%-     foreachRegisterFile(address_block,     :show_last=>true)     do
|registerfile, last| -%>
    <%- if !last -%>
       <cluster>
...
    <%- end -%>
    <%- foreachRegister_Alternate(registerfile) do |register| -%>
```

Az utolsó regiszterfájl igazából nem is az hanem a címblokk, tehát kihagyjuk a cluster-rel kapcsolatos részeket mikor rá kerül a sor.

A regisztereken a foreachRegister_Alternate függvény iterál végig, amely kódja megtalálható a Függelék 6. elemében.

4.1.2 A megvalósítás közben felmerült problémák

A perifériák verziója egy kötelezően megjelenítendő adat egy CMSIS-SVD fájlban, azonban, ha az IP-XACT fájl nem egy levélkomponenst ír le vagy nincs mellette egy dizájn fájl és a perifériák komponenseinek is az IP-XACT fájljai akkor a verziót nem tudjuk meghatározni, és ezt konfigurációs paraméterként is nehézkes lenne átadni.

Szintén problémát okozhat, ha nincsenek megadva a perifériák komponensei az IP-XACT fájlban, csupán a regisztereik leírása egy felső komponensben. Ilyenkor nem tudjuk, hogy a létrehozandó CMSIS-SVD és C header fájlokba azt, hogy két periféria igazából egy komponens kétszer példányosítva. Ezt kézzel viszonylag könnyű átírni.

Több rész leírás (Description) jelöléseinél is találkoztam egy olyan problémával, hogy bizonyos XML jelölések karakterei nem voltak megfelelően feloldva, például: „... The parameter APBPPCEXP_DIS<N> defines if each bit within …”, ha ez bekerül így egy XML fájlba, a kisebb-nagyobb jel kitüntetett szerepe miatt úgy lesz értelmezve mintha egy jelölés lenne, pedig csak egy szöveg. Erre megoldás bizonyos karakterekre

(például itt <>-re) cserélni ezeket, amiket utána egy XML olvasó felismer, hogy ott miket kell megjeleníteni. Szerencsére a rexml könyvtárban erre találtam egy egyszerű megoldást, a Text.new(string) automatikusan feloldja az XML számára kitüntetett karaktereket. Erre még egy to_s metódust (to string, a Ruby által használt szöveges adattá alakító függvény) is meg kell hívnunk, hogy string formátumot kapjunk vissza, hogy kiírhassuk. Az átláthatóság kedvéért ezeket egy escapeXML függvénybe csomagoltam.

Címek, ofszetek, újraindítási értékek és maszkok esetében általánosságban hexadecimális értékekkel szoktunk találkozni (esetleg binárissal kisebb bitszámok esetén), azonban az IP-XACT fájlban tárolt számértékek lehetnek más számrendszerben is, illetve tartalmazhatnak számvégi szorzókat, mint a K (1024) vagy az M (1024*1024), amiket egy CMSIS-SVD fájl nem tartalmazhat. Ezeket a problémákat egy getHexaDecimal függvénnyel oldottam meg, amelyet minden problémás számra meghívok. Működése során először megszerzi az értéket decimálisban, majd ezt átváltja hexadecimálisra, amihez a CMSIS-SVD fájlban is használatos “0x” prefixumot teszi. Azért nem egyből a hexadecimális értéket szerzi meg, mivel már az ARM Socrates Ruby API-ja tartalmazott egy decimális értéket kiszámoló függvényt, és az átváltás Ruby-ban egyszerű.

A regisztermezők tartalmazhatnak egy vagy több enumerated value-t (magyarul felsorolt konstans). Ez nem más, mint egy szöveg-szám megfeleltetés, azaz a mezők értékének nevet adunk. Például:

```
<enumeratedValue>
    <name>disabled</name>
    <description>The clock source clk0 is turned off.</description>
    <value>0</value>
</enumeratedValue>
```

Ez azt írja le, hogy a nullás érték azt jelenti, hogy „disabled”, azaz kikapcsolt. Problémát az okozott, hogy egy IP-XACT fájlban meg lehet fogalmazni minden egyes értéknél, hogy milyen hozzáférési módhoz (írás, olvasás, írás-olvasás) tartozik, míg a CMSIS-SVD séma kettőben maximalizálja az ezen konstansokat összefoglaló enumeratedValue**s** jelölések számát mezőnként és egy ilyen jelöléshez csak egy hozzáférési mód tartozhat. Ez akkor okozhat gondot, ha például van egy bites mezőnk, aminek a nullás értékéhez olvasás és írás során más jelentést írunk, de az egyes értékéhez ugyanazt szeretnénk. Ez egy IP-XACT fájlban nagyon egyszerű, a fent

leírtaknak megfelelően kerül eltárolásra, de egy CMSIS-SVD fájlban nem lehet egy regisztermezőhöz három különböző hozzáférésű konstans. Ezt úgy oldottam meg, hogy amennyiben ilyen eset előfordul az írás-olvasás hozzáférési módhoz tartozó konstans értéket mind az írás, mind az olvasás hozzáférésű enumeratedValues jelölésbe teszem.

A template UNIX alapú rendszeren feltöltötte a kész CMSIS-SVD fájlt extra üres sorokkal, ugyanis mivel Windows-on történt a template írása és a sorvégi ’\r\n’ újsor karakter a UNIX rendszeren extra karaktert jelentett. Mivel ez a karakter az ERB trim_mode jele után volt, nem lett elhagyva, így lett egy extra üres sorunk a CMSIS-SVD fájlban. A sorvégi karakterek ’\n’-re történő cseréjével oldottam meg ezt a problémát.

4.2 C regiszter teszt generátor

A C regiszter teszt generátor feladata az adott IP-XACT fájlból kinyerni a regiszterekkel kapcsolatos információkat majd a megfelelő teszteket generálni. A generált tesztek legfontosabb tulajdonsága az, hogy a CMSIS-SVD generátorral létrejött CMSIS-SVD fájlból C header fájl generálható az SVDConv.exe segítségével és az ebben létrejött struktúrák és makrók egyezzenek meg a regisztertesztеknél felhasználtakkal, így a két generátor együtt használható.

Hasonlóan a CMSIS-SVD generátorhoz egy Ruby szkriptből, egy template fájlból és pár kiegészítő könyvtárból áll. A szkriptet a későbbiekben felhasznált könyvtárak betöltésével kezdem megint csak: erb, fileutils, IPXACT2009 és cmsis_svd_helper_library.

Ez a generátor azért szerényebb mennyiségű konfigurációs paraméterrel rendelkezik, a kimeneti könyvtárat, a template helyét és a készülékspecifikus névprefixumot lehet megadni.

4.2.1 A generátor működése

Ezen paraméterek felvétele után a generátor megkeresi a dizájnt, ha létezik, és a legfelső komponenst. Hasonlóan, mint a CMSIS-SVD generátornál, ezeket már a megfelelő formátumban várom, mivel a beépített dizájn szabály vizsgálatok megfelelőek az IP-XACT minőségének ellenőrzésére. Szintén ugyanúgy, az erb könyvtár segítségével indítom el a célfájl generálását, ami egy szerzői joggal kapcsolatos kommentblokkból és magából a C tesztekből áll.

A tesztek perifériánként vannak csoportosítva, tehát a felső komponens címblokkjain lépkedek végig.

```
<%- foreachAddressBlock(@top_component) do |address_block| -%>
  <%- if device_header -%>
    <%- peripheral_name = device_header + "_" + getName(address_block) -%>
  <%- else peripheral_name = getName(address_block) -%>
  <%- end -%>
  <%- peripheral_type_name = peripheral_name + "_type" -%>
// Tests for <%= peripheral_name -%>
```

A perifériák neve, illetve a típusának a neve akkor fog megegyezni a generált C device header-ben találhatóval, ha a paraméterként felvett device header is megegyezik a CMSIS-SVD generálás során használttal, tehát erre figyelni kell.

A tesztelés a reset értékek vizsgálatával kezdődik. A rendszer (újra)indítása után végigmegyek a regisztereken egyesével és megvizsgálom, hogy olvasható-e, ha igen akkor megvizsgálom az értékét. Ha ez az érték nem egyenlő az IP-XACT fájlban tárolt újraindítási értékkel akkor hibakódot generálok egy futóváltozó segítségével.

Template:

```
  <%- i = 0 -%>
  <%- foreachRegister(address_block).each do |register| -%>
  <%- if getAccessType_SVD(register) == ('read-only' || 'read-write') -%>
if(<%=  peripheral_name  -%>-><%=  getName(register)  -%>  !=  <%=
getHexaDecimalValue(getReset_SVD(register)) -%>) {err_code += (1<<<%= i -
%>)); }
    <%- i += 1 -%>
    <%- end -%>
  <%- end -%>
```

A kész C fájlban:

```
  if(CMSDK_u_cpu_sec_cfg_0->PIDR0         !=  0x59){err_code += (1<<1)); }
```

A csak olvasható regiszterek nem írhatóságának tesztelése következik, beleírok 0xFFFFFFFF-t és utána kiolvasom mit tárol, ha nem a resetértéket akkor hibakódot generálok.

Az írható és olvasható regisztereket a továbbiakban véletlenszerű adatok beírásával tesztelem. Az adatot a rand() beépített Ruby függvénnyel hozzuk létre, aminek a paramétere határozza meg a szám lehetséges nagyságát, így megadjuk neki a regiszter méretét. Beírjuk a regiszterbe, majd kiolvassuk a regiszter értékét. Amennyiben nem egyezik meg a kettő hibakódot generálunk.

A foglalt címek tesztje úgy történik, hogy megyünk egyesével végig a regisztereken és egy current_offset változóba mentjük, hogy a jelenlegi regiszter meddig tart. Amennyiben a következő regiszter és e között a cím között van üres hely, akkor ellenőriznünk kell, hogy ezeken a helyeken nincsenek regiszterek. Ezt a címre történő olvasással és írással tesszük. Amennyiben bármelyik hibát jelez, hibakódot generálunk.

```
  // - Testing reserved fields
  <%- foreachRegister(address_block).each_with_index do |register, index|
-%>
    <%-       if       (current_offset       <       getOffset(register,
:absolute_address=>false)) -%>
  for(i=<%=    getHexaDecimalValue(current_offset)    -%>;    i    <<    <%=
getHexaDecimalValue(getOffset(register,   :absolute_address=>false))   -%>;
i+=4){
    if (HW_REG_WORD(<%= peripheral_name + "_BASE" -%>, i) != 0x00000000) {
err_code4 += (1<<i); }
    HW_REG_WORD(<%= peripheral_name + "_BASE" -%>, i) = 0xFFFFFFFF;
    if (HW_REG_WORD(<%= peripheral_name + "_BASE" -%>, i) != 0x00000000) {
err_code4 += (1<<i); }
  }
    <%- end -%>
    <%- current_offset = getOffset(register, :absolute_address=>false) +
getWidth(register) -%>
  <%- end -%>
```

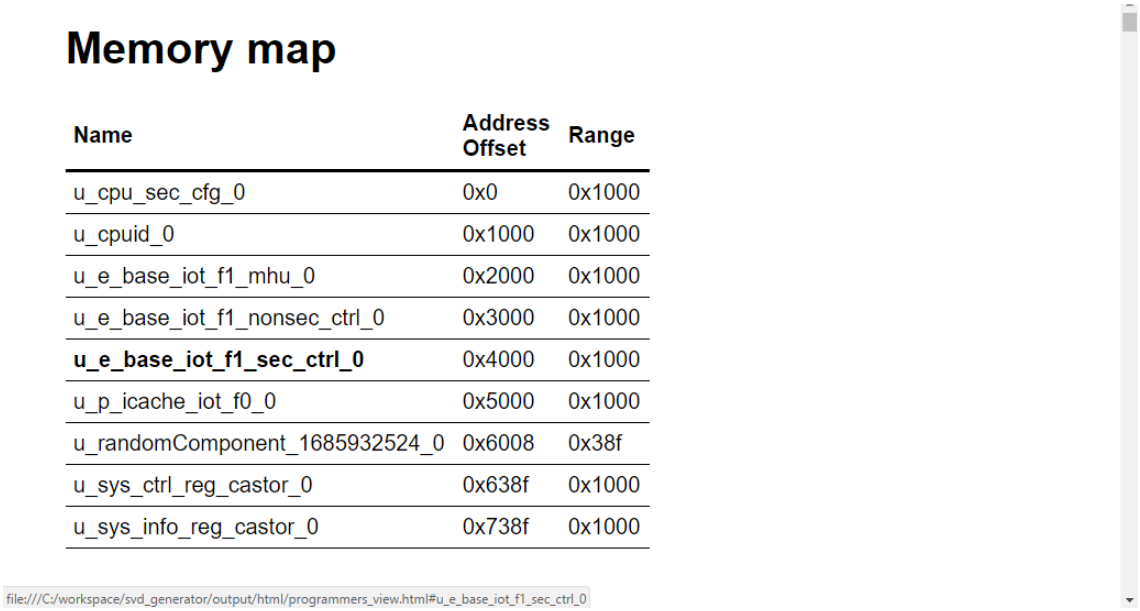
4.3 HTML generátor

A következő lépés az automatikus dokumentáció generálása volt. A HTML generátor a többihez hasonlóan egy Ruby szkriptből és egy erb template fájlból áll. A generátor kinyeri az információkat az IP-XACT fájlból a Ruby API és a saját

kiegészítéseim segítségével, majd a template fájl alapján megszerkeszti a HTML kimenetet.

A HTML fájl elején egy összefoglaló memóriatérképet csináltam, amin fel vannak sorolva a címblokkok/perifériák, amikre kattintva az adott címblokk részleteihez ugrik, egy HTML referencia segítségével. Ezt láthatjuk a 15. ábrán.

```
 <a href="#<%= getName(address_block) %>"><%= getName(address_block) %></a>
...
 <h2   id="<%=   getName(address_block)-%>"><%=   getName(address_block)   -%>
programmers view</h2>
```



Memory map

Name	Address Offset	Range
u_cpu_sec_cfg_0	0x0	0x1000
u_cpuid_0	0x1000	0x1000
u_e_base_iot_f1_mhu_0	0x2000	0x1000
u_e_base_iot_f1_nonsec_ctrl_0	0x3000	0x1000
u_e_base_iot_f1_sec_ctrl_0	0x4000	0x1000
u_p_icache_iot_f0_0	0x5000	0x1000
u_randomComponent_1685932524_0	0x6008	0x38f
u_sys_ctrl_reg_castor_0	0x638f	0x1000
u_sys_info_reg_castor_0	0x738f	0x1000

15. ábra. Memóriatérkép

A memóriatérképre (a HTML elejére) bármikor visszaugorhatunk a bal alsó sarokban található „Back to the Memory Map” gombbal. Ez a gomb mindig látható, ha nem a memóriatérképnél vagyunk és mindig a jobb alsó sarokban. Ezt egy jQuery szkript segítségével valósítottam meg:

```
<script>
            jQuery(document).ready(function() {
                var offset = 0;
                var duration = 0;
                jQuery(window).scroll(function() {
                    if (jQuery(this).scrollTop() > offset) {
                        jQuery('.back-to-top').fadeIn(duration);
                    } else {
                        jQuery('.back-to-top').fadeOut(duration);
                    }
                });
                jQuery('.back-to-top').click(function(event) {
                    event.preventDefault();
```

```
                        jQuery('html,    body').animate({scrollTop:    0},
duration);
                        return false;
                    })
                });
    </script>
```

A memóriatérkép után az egyes címblokkok részletei jönnek. Először a regiszterek és regiszterfájlok nevei, címofszettjei, hozzáférési módjai, újraindítási értékei és rövid leírásai vannak felsorolva egy táblázatban a címek alapján sorba rendezve. Ezek nevei is mind hivatkozások, ugyanis rájuk kattintva a regiszterfájlok esetében a bennük található regiszterek, regiszterek esetén pedig a regisztermezőik részletei érhetőek el. Regisztermezők esetén a foglalt mezőket szürke háttérrel jelölöm ezekben a táblázatokban.

Több problémával találkoztam a HTML kinézetének megtervezése során. Az egyik elvárása a mérnökcsapatnak a generátor felé az, hogy egy TRM-be (Technical Reference Manual, magyarul műszaki kézikönyv) beilleszthető legyen egyszerűen, mindenféle formázás nélkül. Azonban a cégen belül nincsenek lefektetve szabályok a regiszterek ilyen módú dokumentálásáról egyelőre, éppen most dolgoznak rajta, vagyis amint megszületik a formátum, a generátornak át kell vennie. Addig egy elég egyszerű kinézetet választottam. Egyes regiszter-dokumentációkban színkóddal vannak jelölve a regiszterek írhatósága/olvashatósága, azonban a cégen belül a fekete fehér dokumentáció elvárás, így ennek megfelelően járok el. Másik problémát a regiszterek bitjeinek számossága okozta, ezért olyan módszer kell, ami a használatban lévő bitszélességek mindegyikét olvashatóan jeleníti meg.

5. A generátorok tesztelése, értékelése és továbbfejlesztési lehetőségei

A generátorok tesztelése két fronton történik: megbirkóznak-e különböző bemenetekkel, illetve az elkészült fájlok megfelelnek-e az elvártaknak.

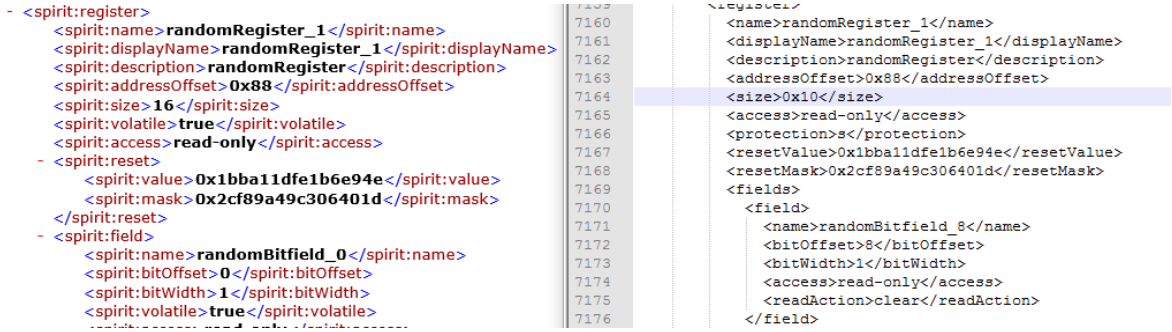
A bemeneti IP-XACT fájlok generálásához egy már meglévő generátort használtam. Ez egy Ruby szkriptet állít elő, ami az ARM Socrates DE-ben található Ruby API-t használja. Ezt futtatva létrehoz egy nekünk megfelelő IP-XACT fájlt, amiben a regiszterleírások véletlenszerűek, bizonyos beállítható paraméterek szerint. A generátort módosítanom kellett, ugyanis a regiszterek rövid megjelenítendő nevénél nem véletlenszerű, hanem konstans szöveg volt beállítva, aminek a hatására az SVDConv program figyelmeztetéseket küldött. Ezt úgy javítottam ki, hogy a regiszterek megjelenítendő nevét beállítottam a regiszterek nevére (ami alapértelmezésben egyedi).

Mindegyik generátor csak egy mély hierarchiát képes kezelni. Némi mérlegelés után arra jutottam, hogy felesleges erre erőforrást pazarolni. Először is azért, mert a bejárás nagy mértékben bonyolódna egy nagyon ritkán használt részlet bővítése érdekében. Másodjára pedig azért, mert a bejárás módja valószínűleg meg lesz változtatva, mivel a kiindulási IP-XACT fájlok formátuma változni fog a közeljövőben.

5.1 CMSIS-SVD

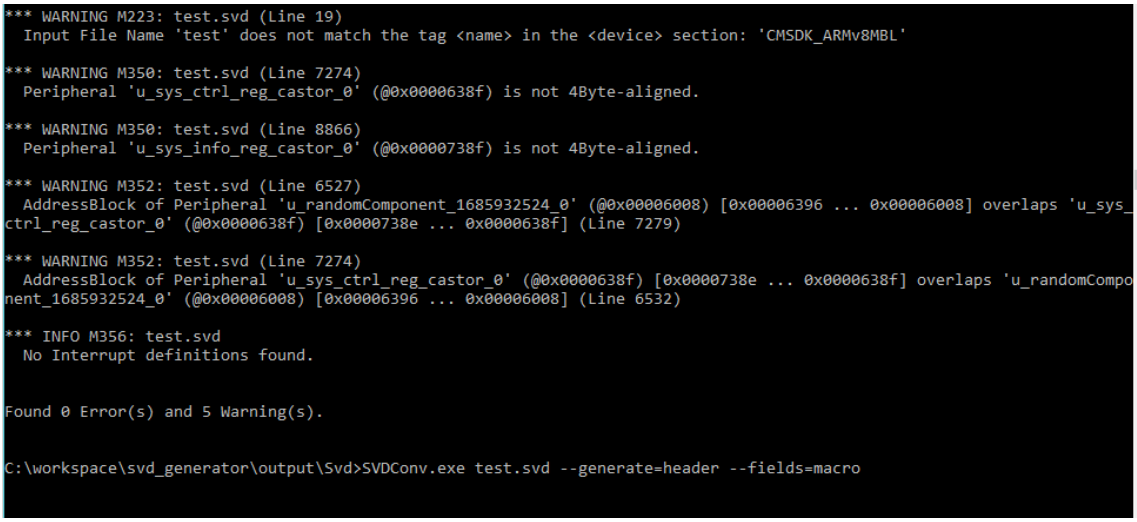
A CMSIS-SVD generátor tesztelése első körben ezen véletlenszerű IP-XACT fájlok átalakításából állt. Hamar kiderült, hogy a Ruby API függvényei nem lesznek megfelelőek. Problémát okozott például az eredeti foreachRegister hierarchiától függetlenül kigyűjtötte az összes regisztert. A CMSIS-SVD és az IP-XACT is támogatja a regiszterek fájlokba történő csoportosítását és a regiszterekhez alternatív definíciók hozzárendelését. Ahhoz, hogy a regiszterfájlok és az alternatív regiszterek által kialakított rendszer a CMSIS-SVD fájlban is úgy jelenjen meg, mint ahogyan az egy IP-XACT fájlban le van írva, új függvényt kellett írnom. A Ruby API függvényei mindig a kapott paraméterek ellenőrzésével kezdődnek, ahol szintén merültek fel problémák,

ugyanis a beépített függvények nagy része nem engedte az alternatív regisztereket. Ezekkel a problémákkal több függvény esetén is találkoztam, kiküszöbölésük után a CMSIS-SVD fájlban tárolt adatok megfeleltek az IP-XACT fájlban tároltakkal, amelyet a 16. ábra mutat.



16. ábra. Bal oldalon a kiindulási IP-XACT jobb oldalon a létrehozott CMSIS-SVD

Ezen felül az ellenőrzésre használtam az SVDConv programot is, ahogy látható a 17. ábrán. Ez sokféleképpen vizsgálja a bemenetének helyességét (szintaktikai, strukturális, konzisztencia, teljesség tesztek) így a feladatom ellenőrzése szempontjából fontos volt, hogy ne jelenezzen hibát.



17. ábra. SVDConv futtatása során üzeneteket kapunk, ha a bemeneti CMSIS-SVD fájl nem megfelelő

Második körben rendelkezésemre álltak egy mérnökcsapat által kézzel írt CMSIS header fájl mellett a kiindulási IP-XACT fájlok is. A CMSIS-SVD generátort a kiindulási IP-XACT fájlokra futtatva létrejött egy CMSIS-SVD fájl, amiből az SVDConv program segítségével generáltam egy C header fájlt.

A generált header fájlt ezek után összehasonlítottam a kézzel írt fájllal, és azt találtam, hogy a kettő hasonló, amelyet a 18. ábra mutat.

```
/*---------------- MHU ----------------*/
typedef struct
{
  __IM   uint32_t CPU0INTR_STAT;                /* Offset: 0x0 (read-only) CPU0INTR_STAT Register    */
  __OM   uint32_t CPU0INTR_SET;                 /* Offset: 0x4 (write-only) CPU0INTR_SET Register    */
  __OM   uint32_t CPU0INTR_CLR;                 /* Offset: 0x8 (write-only) CPU0INTR_CLR Register    */
         uint32_t RESERVED0[1];
  __IM   uint32_t CPU1INTR_STAT;                /* Offset: 0x10 (read-only) CPU1INTR_STAT Register    */
  __OM   uint32_t CPU1INTR_SET;                 /* Offset: 0x14 (write-only) CPU1INTR_SET Register   */
  __OM   uint32_t CPU1INTR_CLR;                 /* Offset: 0x18 (write-only) CPU1INTR_CLR Register    */
         uint32_t RESERVED1[1005];
  __IM   uint32_t PIDR4;                    /* Offset: 0xFD0 (read-only) PIDR4 Register    */
         uint32_t RESERVED2[3];
  __IM   uint32_t PIDR0;                    /* Offset: 0xFE0 (read-only) PIDR0 Register    */
  __IM   uint32_t PIDR1;                    /* Offset: 0xFE4 (read-only) PIDR1 Register    */
  __IM   uint32_t PIDR2;                    /* Offset: 0xFE8 (read-only) PIDR2 Register    */
  __IM   uint32_t PIDR3;                    /* Offset: 0xFEC (read-only) PIDR3 Register    */
  __IM   uint32_t CIDR0;                    /* Offset: 0xFF0 (read-only) CIDR0 Register    */
  __IM   uint32_t CIDR1;                    /* Offset: 0xFF4 (read-only) CIDR1 Register    */
  __IM   uint32_t CIDR2;                    /* Offset: 0xFF8 (read-only) CIDR2 Register    */
  __IM   uint32_t CIDR3;                    /* Offset: 0xFFC (read-only) CIDR3 Register    */
}MHU_TypeDef;
```

```
typedef struct {                                /*!< (@ 0x00002000) u_e_base_iot_f1_mhu_0 Structure */
  __IM  uint32_t  CPU0INTR_STAT;                /*!< (@ 0x00000000) CPU 0 Interrupt Status Register */
  __OM  uint32_t  CPU0INTR_SET;                 /*!< (@ 0x00000004) CPU 0 Interrupt Set Register */
  __OM  uint32_t  CPU0INTR_CLR;                 /*!< (@ 0x00000008) CPU 0 Interrupt Clear Register */
  __IM  uint32_t  RESERVED;
  __IM  uint32_t  CPU1INTR_STAT;                /*!< (@ 0x00000010) CPU 1 Interrupt Status Register */
  __OM  uint32_t  CPU1INTR_SET;                 /*!< (@ 0x00000014) CPU 1 Interrupt Set Register */
  __OM  uint32_t  CPU1INTR_CLR;                 /*!< (@ 0x00000018) CPU 1 Interrupt Clear Register */
  __IM  uint32_t  RESERVED1[1005];
  __IM  uint32_t  PIDR4;                        /*!< (@ 0x00000FD0) Peripheral ID 4 */
  __IM  uint32_t  RESERVED2[3];
  __IM  uint32_t  PIDR0;                        /*!< (@ 0x00000FE0) Peripheral ID 0 */
  __IM  uint32_t  PIDR1;                        /*!< (@ 0x00000FE4) Peripheral ID 1 */
  __IM  uint32_t  PIDR2;                        /*!< (@ 0x00000FE8) Peripheral ID 2 */
  __IM  uint32_t  PIDR3;                        /*!< (@ 0x00000FEC) Peripheral ID 3 */
  __IM  uint32_t  CIDR0;                        /*!< (@ 0x00000FF0) Component ID 0 */
  __IM  uint32_t  CIDR1;                        /*!< (@ 0x00000FF4) Component ID 1 */
  __IM  uint32_t  CIDR2;                        /*!< (@ 0x00000FF8) Component ID 2 */
  __IM  uint32_t  CIDR3;                        /*!< (@ 0x00000FFC) Component ID 3 */
} CMSDK_u_e_base_iot_f1_mhu_0_Type;             /*!< Size = 4096 (0x1000) */
```

18. ábra: A kézzel írt, aztán általam generált C header fájlból ugyanaz a struktúra

Látszik, hogy a generált C struktúra funkcionálisan megegyezik, azonban a kommentek részletei nem vesztek el, illetve a periféria neve sem változott az IP-XACT fájlhoz képest.

A szakdolgozat kiírásában megfogalmazott feladat tehát elkészült. Természetesen a projekt még nincsen kész. A következő lépés a generátor felhasználása egy tényleges munkafolyamatban lesz, ahonnan biztosan lesz visszajelzés. Jelenleg erre várok, elméletileg nemsokára meg fog történni. További fejlesztés legnyilvánvalóbb lépése az IP-XACT-ből hiányzó, de CMSIS-SVD-be leírható adatok maradékának a hozzáadása. Ami hiányzik ezek közül azok olyanok, hogy konfigurációs paraméterekkel nehézkesen adhatóak meg (például a különböző megszakításokról az adatok). Így ezek hozzáadásához a generátor számára új módon kellene ezeket biztosítani.

5.2 C regiszter teszt

A C regiszter teszt generátor esetén a legfontosabb az, hogy a generált tesztekben ugyanúgy hivatkozzunk a regiszterekre, mint ahogyan a CMSIS-SVD fájlból generált C header-ben vannak, hogy működjenek egymás mellett. A 19. ábrán láthatjuk, hogy a teszteket tartalmazó fájl valóban a C header-ben definiált struktúrát használja.

```
// Tests for CMSDK_u_cpuid_0
// - Checking initial values


  printf("CMSDK_u_cpuid_0 test\n");


  if(CMSDK_u_cpuid_0->CPU_IDENTITY
  if(CMSDK_u_cpuid_0->PIDR4
  if(CMSDK_u_cpuid_0->PIDR0
  if(CMSDK_u_cpuid_0->PIDR1
  if(CMSDK_u_cpuid_0->PIDR2
  if(CMSDK_u_cpuid_0->PIDR3
  if(CMSDK_u_cpuid_0->CIDR0
  if(CMSDK_u_cpuid_0->CIDR1
  if(CMSDK_u_cpuid_0->CIDR2
  if(CMSDK_u_cpuid_0->CIDR3
```

```
204 /**
205   * @brief Importing registers for COXA (u_cpuid_0)
206   */
207
208 typedef struct {                                    /*!<
209   __IM  uint32_t  CPU_IDENTITY;                     /*!<
210   __IM  uint32_t  RESERVED[1011];
211   __IM  uint32_t  PIDR4;                            /*!<
212   __IM  uint32_t  RESERVED1[3];
213   __IM  uint32_t  PIDR0;                            /*!<
214   __IM  uint32_t  PIDR1;                            /*!<
215   __IM  uint32_t  PIDR2;                            /*!<
216   __IM  uint32_t  PIDR3;                            /*!<
217   __IM  uint32_t  CIDR0;                            /*!<
218   __IM  uint32_t  CIDR1;                            /*!<
219   __IM  uint32_t  CIDR2;                            /*!<
220   __IM  uint32_t  CIDR3;                            /*!<
221 } CMSDK_u_cpuid_0_Type;                             /*!<
```

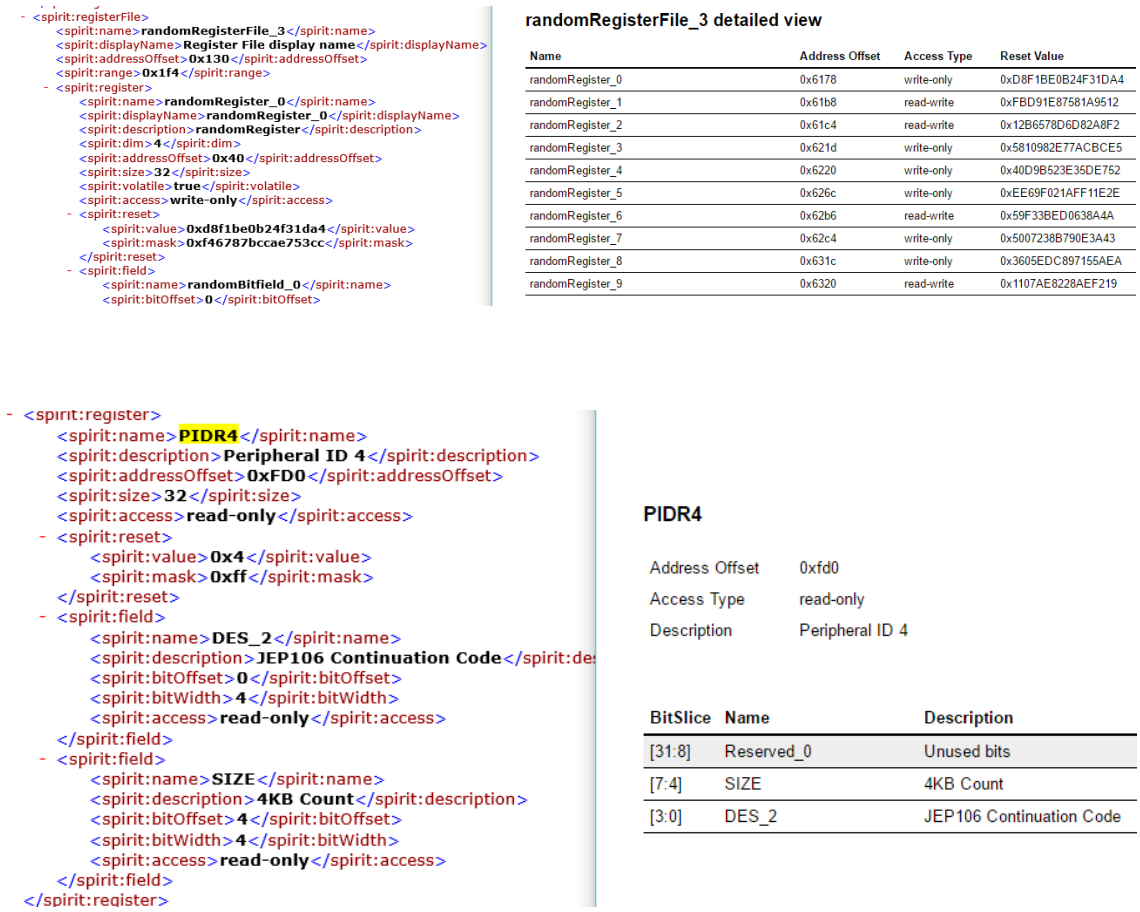
19. ábra. Bal oldalt a generált C regiszter teszt részlete, jobb oldalt a hozzá tartozó generált C header részlete

A tesztek jelenleg csak a kezdeti értékek vizsgálatából, írható-olvasható regiszterekbe véletlenszerű adat írásából, csak olvasható regiszterek nem írhatóságából és a fenntartott regiszterek nem használhatóságából állnak. Ezekről egy példa a Függelék 7. részében látható.

A generátor további funkciókkal történő bővítése a jövőbeli feladatok között szerepel. Például a regiszter teszt generátor egyelőre nem képes a regisztermezők kezelésére. Ehhez meg kell oldanom, hogy a generált C header fájl által használt 4 különböző regisztermezőleírási mód mindegyikével működjön. Miután ez sikerült a változtatott írási módú tesztek hozzáadása következik. Lehetséges, hogy a bonyolultabb (IP-XACT fájlban nem leírható) regiszterek tesztjeihez nem az IP-XACT fájlt fogom forrásnak használni, hanem a kiindulási Excel táblázatot, amiben valamilyen feldolgozható formátumban le lesznek írva az extra kapcsolatok, tulajdonságok.

5.3 HTML

A HTML generátor által létrehozott dokumentációt a műszaki kézikönyvbe illesztésre terveztem. Egy ilyen műszaki kézikönyv pontossága rendkívül fontos, ezért a generátorunk tesztelése a kiindulási adatok és az elkészült fájlban lévő adatok összehasonlításából állt, amelyet a 20. ábra mutat.



20. ábra. Bal oldalt a kiindulási IP-XACT, jobb oldalt a generált HTML fájl látható

A HTML generátornál a kinézet valószínűleg változni fog, amint kialakul a cégen belüli regisztermegjelenítési standard. Egy továbbfejlesztési lehetőség a rendkívül nagy regiszterek megjelenítése több sorban a könnyebb olvashatóság elérésére.

Irodalomjegyzék

[1] Accellera.org: *IP-XACT Working Group* http://accellera.org/activities/working-groups/ip-xact/

[2] Dare Obasanjo: *Understanding XML* https://msdn.microsoft.com/en-us/library/aa468558.aspx

[3] IEEE Standards: 1685-2009 - *IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows* https://standards.ieee.org/findstds/standard/1685-2009.html

[4] Lanka, T. R. (2014). *Designing with ARM Cortex-M based System-On-Chips.* http://www.embedded.com/design/mcus-processors-and-socs/4429701/Designing-with-ARM-Cortex-M-based-System-On-Chips--SoCs----Part-I--The-basics.

[5] Marc van Hintum, Paul Williams: *The Value of High Quality IP-XACT XML* https://www.design-reuse.com/articles/19895/ip-xact-xml.html

[6] *ARM Socrates DE User Guide*

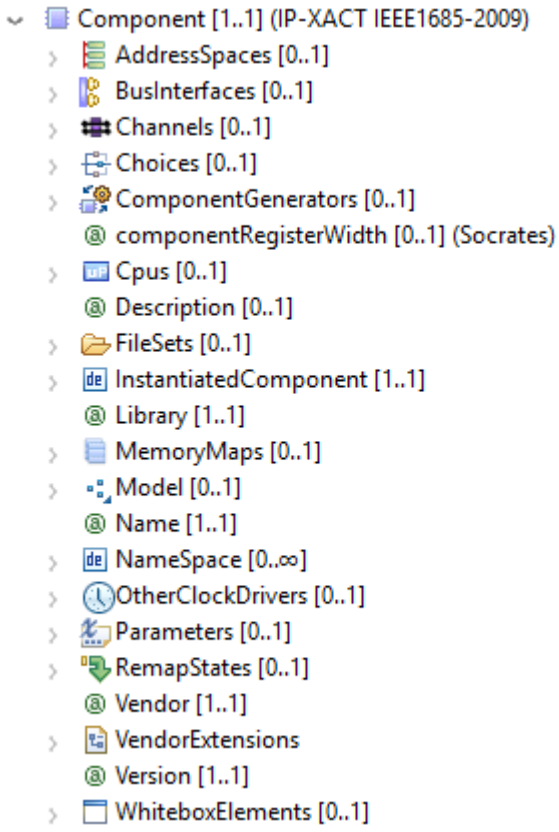
[7] David Flanagan, Yukihiro Matsumoto(2008): *The Ruby Programming Language*

[8] ARM.com: *CMSIS* https://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

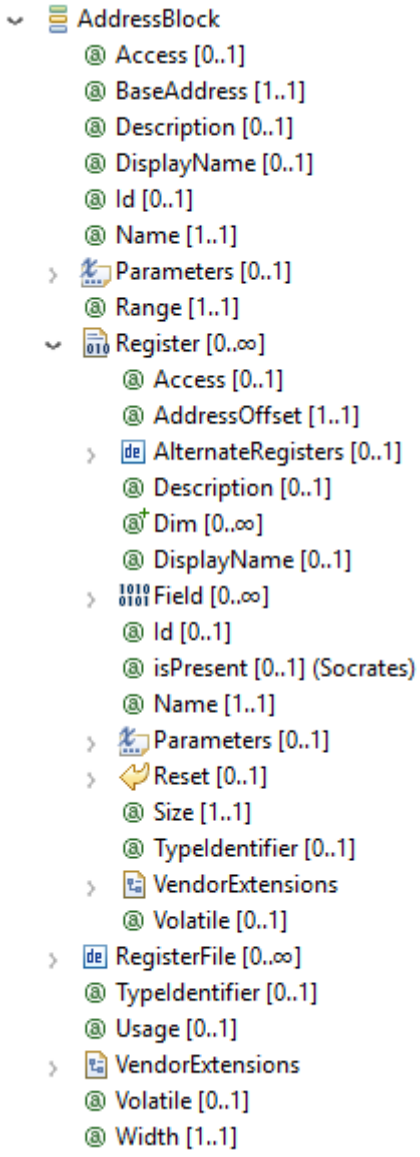
[9] Keil.com: *CMSIS-SVD* http://www.keil.com/pack/doc/CMSIS/SVD/html/svd_Format_pg.html

[10] Jedlik Oktatási Stúdió (2011): *Honlapépítés a XXI. Században*

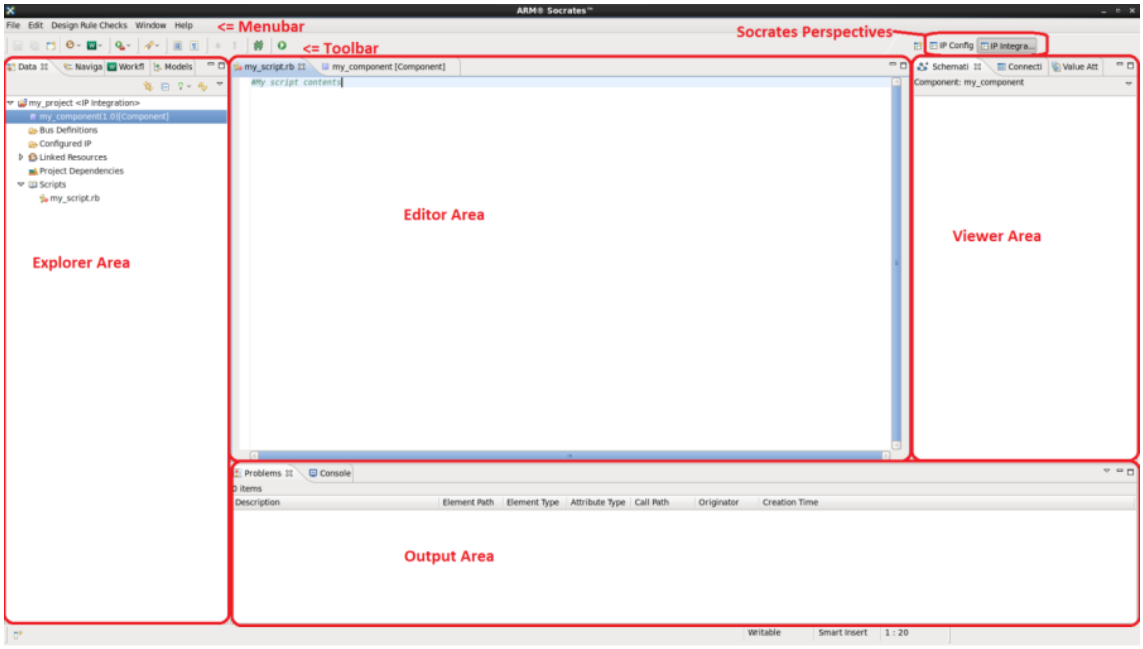
Függelék



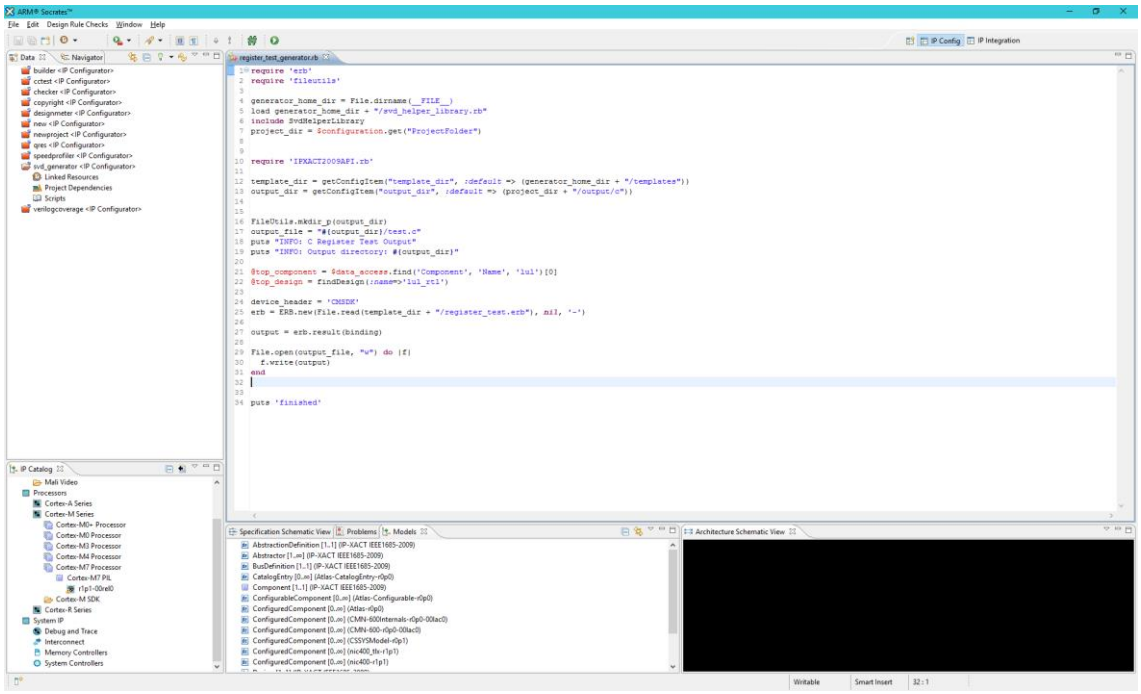
Függelék 1: Egy IP-XACT komponens részei



Függelék 2: Egy komponens címblokkja



Függelék 3: ARM Socrates DE grafikus interfésze, IP Integration nézet



Függelék 4: ARM Socrates DE grafikus interfésze, IP Config nézet

User guide for the CMSIS-SVD generator

This guide describes how to generate a SVD file from IP-XACT via Socrates DE, and how to create a CMSIS device header file.

Things you need before

1. Check out the generator from git for example to your Flows_GIT/Generators/ directory
   $ git clone sipt/flows/Generators/register_CMSIS_SVD
   $ cd register_CMSIS_SVD
   $ git checkout tags/17.1.0.1-Dev02
2. SVDConv: Linux, Windows
3. Socrates DE 17.1

Usage in GUI

- Launch ARM-Socrates
- Create a Socrates Project via File -> New -> Socrates Project (please note that it has to be Socrates Project, not Project)
- Right-click on the Linked Resources item under the newly created project and select Add Linked Folder or Add Linked File (if you want to add a single file) and select the IP-XACT component file containing the register descriptions.
- If you are only doing a leaf component or you have a top component containing a memory map with all the registers mapped skip the next step
- Make sure you have linked all the IP-XACT component files containing the register descriptions to the project. Right-click on Scripts under the project and select New Script. In the dialog window change the script name to 'RegisterBuilder', select 'Link to file in file system', browse to 'Flows_GIT/Generators/register_CMSIS_SVD/Scripts/register_builder.rb' and click Finish. Right-click on the newly created script file and select Run Script.
- Right-click on Scripts under the project and select New Script. In the dialog window change the script name to 'RegisterCMSISSVDGenerator', select 'Link to file in file system', browse to 'Flows_GIT/Generators/register_CMSIS_SVD/Scripts/register_cmsis_svd_generator.rb' and click Finish.
- Right-click on the newly created script file and select Run Script if you want to execute it with default configurations.
- If you want to run the script with your configurations, right-click in the Workflows tab and create select New Workflow. This tab should be in the lower left corner of the GUI. If you don’t see it, you may need to set the Perspective to IP Integration or IP Config in the upper right corner (or alternatively enable it via Window -> Show View -> Workflows)
- Double click on the newly created workflow if it didn’t open automatically, select your project from the drop-down in the editor and press the Add button on the right-hand side to add a workflow step.

- In the appearing dialog select Script from the (rather long) list in the middle, change the scripting language to Ruby and point to the CMSIS SVD generator file at 'Flows_GIT/Generators/register_CMSIS_SVD/Scripts/register_cmsis_svd_generator.rb'
- If you need to change some of the generator configuration options (see below), you can do so in the Script Settings part of this dialog, otherwise click OK, save the workflow and run it via the Run button in the upper right corner of the Workflow editor.
- Run SVDConv (with parameters detailed here)
    - e.g: SVDConv test.svd --generate=header --fields=macro

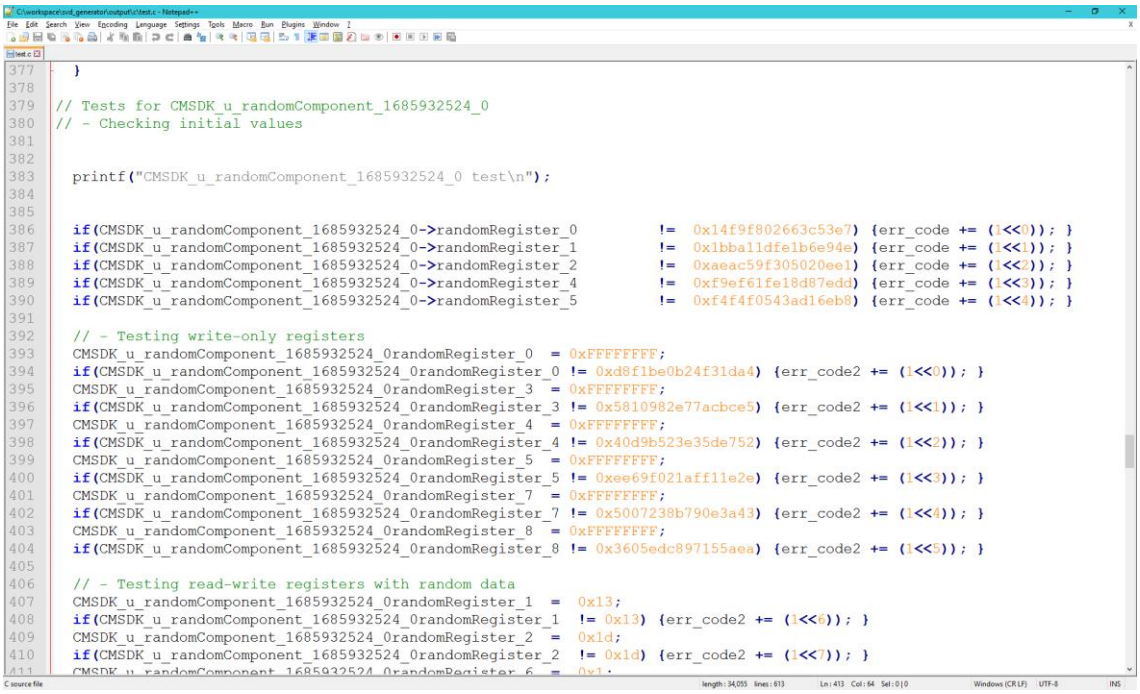
Usage in CLI

- Create a Socrates Project:
    - socrates_cli -data <WORKSPACE> --project <PROJECTNAME> --flow !AddNewProject
    - e.g.: socrates_cli -data /home/user01/armSocrates/workspace/ --project test_project --flow !AddNewProject
- If you are doing a leaf component or you have a top component containing a memory map with all the registers mapped skip the next step
- Run Builder script:
    - socrates_cli -data <WORKSPACE> --project <PROJECTNAME> --flow !RunScript -DScriptFile="Flows_GIT/Generators/register_CMSIS_SVD/Scripts/register_builder.rb?"
    - e.g.:socrates_cli -data /home/user01/armSocrates/workspace/ --project test_project --flow !RunScript -DScriptFile="'Flows_GIT/Generators/register_CMSIS_SVD/Scripts/register_builder.rb"
- Run Generator script:
    - socrates_cli -data <WORKSPACE> --project <PROJECTNAME> --flow !RunScript -DScriptFile="Flows_GIT/Generators/register_CMSIS_SVD/Scripts/register_cmsis_svd_generator.rb?<CONFIG_PARAM1>=<USER_VALUE1>&<CONFIG_PARAM2>=<USER_VALUE2>"
    - e.g.:socrates_cli -data /home/user01/armSocrates/workspace/ --project test_project --flow !RunScript -DScriptFile="Flows_GIT/Generators/register_CMSIS_SVD/Scripts/register_cmsis_svd_generator.rb?output_dir=/home/user01/output&top_component=UART"
- Run SVDConv (with parameters detailed here)
    - e.g: SVDConv test.svd --generate=header --fields=macro

Függelék 5: Részlet a CMSIS-SVD generátor felhasználói útmutatójából

```
def foreachRegister_Alternate(design_element, options = {}, &block)
  de_received = defined?(design_element) ? design_element : :none
  arguments_received = defined?(options) ? options : nil
  supported_element_types = ["RegisterFile", "AddressBlock"]
  valid_options = [:show_index, :show_last]
  design_element, element_type, error_messages = commonChecks(de_received,
arguments_received, supported_element_types, valid_options)

  defaults = {:show_index => false, :show_last => false}
  options = defaults.merge(options)

  show_index = options[:show_index]
  show_last = options[:show_last]

  if !(error_messages.empty?)
    raiseError(__method__,       design_element,       arguments_received,
error_messages)
  end
  # Body

  register_list = design_element.elements("Register")
  register_list  +=  design_element.find("AlternateRegister",  nil,  nil,
false).to_a

  register_list.sort!{|x,y| getOffset_SVD(x)<=>getOffset_SVD(y)}

  register_list_last = register_list.length - 1
  last = false
  if block_given?
    register_list.each_with_index do |register, register_index|
      last = true if register_list_last == register_index
      if block.arity == 1
        yield register
      elsif block.arity == 2
        if show_index and not show_last
          yield register, register_index
        elsif show_last and not show_index
          yield register, last
        else
          raiseError(__method__,    design_element,    arguments_received,
"block has 2 parameters while both show_index and show_last are true")
        end
      elsif block.arity == 3
        yield register, register_index, last
      end
    end
  end

  return register_list
end
```

Függelék 6: foreachRegister_Alternate



```
}

// Tests for CMSDK_u_randomComponent_1685932524_0
// - Checking initial values


printf("CMSDK_u_randomComponent_1685932524_0 test\n");


if(CMSDK_u_randomComponent_1685932524_0->randomRegister_0         !=  0x14f9f802663c53e7) {err_code += (1<<0); }
if(CMSDK_u_randomComponent_1685932524_0->randomRegister_1         !=  0x1bba11dfe1b6e94e) {err_code += (1<<1); }
if(CMSDK_u_randomComponent_1685932524_0->randomRegister_2         !=  0xaeac59f305020ee1) {err_code += (1<<2); }
if(CMSDK_u_randomComponent_1685932524_0->randomRegister_4         !=  0xf9ef61fe18d87edd) {err_code += (1<<3); }
if(CMSDK_u_randomComponent_1685932524_0->randomRegister_5         !=  0xf4f4f0543ad16eb0) {err_code += (1<<4); }

// - Testing write-only registers
CMSDK_u_randomComponent_1685932524_0randomRegister_0  = 0xFFFFFFFF;
if(CMSDK_u_randomComponent_1685932524_0randomRegister_0 != 0xd8f1be0b24f31da4) {err_code2 += (1<<0); }
CMSDK_u_randomComponent_1685932524_0randomRegister_3  = 0xFFFFFFFF;
if(CMSDK_u_randomComponent_1685932524_0randomRegister_3 != 0x5810982e77acbce5) {err_code2 += (1<<1); }
CMSDK_u_randomComponent_1685932524_0randomRegister_4  = 0xFFFFFFFF;
if(CMSDK_u_randomComponent_1685932524_0randomRegister_4 != 0x40d9b523e35de752) {err_code2 += (1<<2); }
CMSDK_u_randomComponent_1685932524_0randomRegister_5  = 0xFFFFFFFF;
if(CMSDK_u_randomComponent_1685932524_0randomRegister_5 != 0xee69f021aff11e2e) {err_code2 += (1<<3); }
CMSDK_u_randomComponent_1685932524_0randomRegister_7  = 0xFFFFFFFF;
if(CMSDK_u_randomComponent_1685932524_0randomRegister_7 != 0x5007238b790e3a43) {err_code2 += (1<<4); }
CMSDK_u_randomComponent_1685932524_0randomRegister_8  = 0xFFFFFFFF;
if(CMSDK_u_randomComponent_1685932524_0randomRegister_8 != 0x3605edc897155aea) {err_code2 += (1<<5); }

// - Testing read-write registers with random data
CMSDK_u_randomComponent_1685932524_0randomRegister_1  =  0x13;
if(CMSDK_u_randomComponent_1685932524_0randomRegister_1  != 0x13) {err_code2 += (1<<6); }
CMSDK_u_randomComponent_1685932524_0randomRegister_2  =  0x1d;
if(CMSDK_u_randomComponent_1685932524_0randomRegister_2  != 0x1d) {err_code2 += (1<<7); }
```

Függelék 7: C regiszter teszt generátor kimenetéből részlet: