FELADATKIÍRÁS

A feladatkiírást a **tanszék saját előírása szerint** vagy a tanszéki adminisztrációban lehet átvenni, és a tanszéki pecséttel ellátott, a tanszékvezető által aláírt lapot kell belefűzni a leadott munkába, vagy a tanszékvezető által elektronikusan jóváhagyott feladatkiírást kell a Diplomaterv Portálról letölteni és a leadott munkába belefűzni (ezen oldal HELYETT, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell megismételni a feladatkiírást.



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Automatizálási és Alkalmazott Informatikai Tanszék

Kovácsvölgyi Dávid

Adatfolyamgráfból hardverleírás automatizált generálása

Konzulens

Suba Gergely

BUDAPEST, 2018

Tartalomjegyzék

[Összefoglaló 5](#_Toc529902045)

[Abstract 6](#_Toc529902046)

[1 Bevezetés 7](#_Toc529902047)

[1.1 Formázási tudnivalók 7](#_Toc529902048)

[1.1.1 Címsorok 7](#_Toc529902049)

[1.1.2 Képek 7](#_Toc529902050)

[1.1.3 Kódrészletek 7](#_Toc529902051)

[1.1.4 Irodalomjegyzék 7](#_Toc529902052)

[2 Utolsó simítások 9](#_Toc529902053)

[Irodalomjegyzék 10](#_Toc529902054)

[Függelék 11](#_Toc529902055)

Hallgatói nyilatkozat

Alulírott **Kovácsvölgyi Dávid**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/ diplomatervet (nem kívánt törlendő) 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) közzétegye. 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, 2018. 11. 15.

...…………………………………………….

Kovácsvölgyi Dávid

Összefoglaló

Egyre nagyobb az igény, hogy általános célú, magas szintű programozási nyelvekből is lehessen hardvert szintetizálni, minimális beavatkozással. A magas szintű szintézis (HLS, High-level synthesis) sok esetben gyorsabb és kevesebb költségekkel járó hardverfejlesztést tesz lehetővé.

Ilyen rendszer az Irányítástechnika és Informatika Tanszéken fejlesztett PipeComp, amely magas szintű nyelvekből hardverleírást generál. A PipeComp programozási nyelveket feldolgozó frontendekből és hardverleírást generáló backendekből áll. A PipeComp köztes reprezentációja a HIG adatfolyamgráf.

A dolgozat témája a PipeComp részeként egy VHDL backend fejlesztése, ami HIG adatfolyamról fordít. A szoftver Java nyelven készült.

A dolgozatom ismerteti a VHDL hardver leírónyelv sajátosságait és kódgenerálásra alkalmas templatenyelveket továbbá bemutatja a HIG adatfolyamgráf modellezésére használt frameworkot. Leírja a fejlesztendő szoftver pontos specifikációját, a szoftver tervezésének lépéseit, az implementáció fontosabb részleteit, és a szoftver tesztelésének lépéseit is bemutatja.

Abstract

There is a growing need for applications that can synthesis harware from high level programming languages. The HLS (High-level Synthesis) is faster and cheaper than the actual hardwaredevelopment in most cases.

One of the implementation of these applications is the PipeComp system which was developed by the Irányítástechnika és Informatika Tanszék department of our university. The PipeComp has frontends which processes high level lenguages into the common HIG dataflow graph, and it has backends that creates hardware definition (e.g. VHDL, Verilog…etc) from HIG.

In this paper i would like to document an implementation of a PipeComp backend which creates VHDL from dataflow graph. The software will be written in Java (and use technologies based on java).

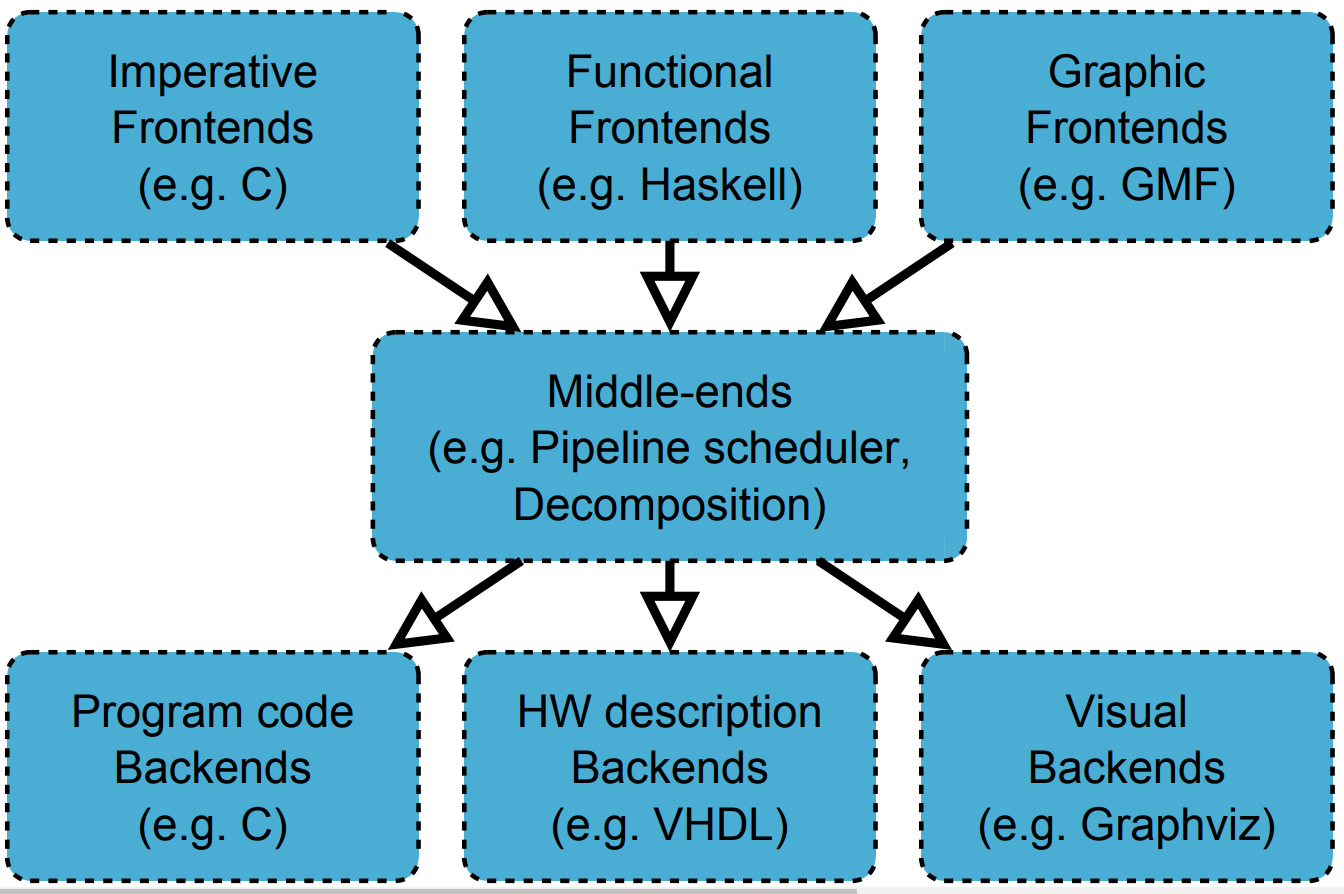
My paper will review the VHDL hardware definition language and the template languages which can generate code easily. It will help understand the framework used for modelling HIG, and provides documentation about the architecture and implementation of the software. It will reveal how the testing and planning was done.

# Bevezetés

Mivel a kapuszintű tervezése egy áramkörnek lassú és nagy odafigyelést igényel, ezért rég óta foglalkoztatja a mérnököket, hogy a tervezést egy magasabb absztrakciós szinten tudják elvégezni, ahol a rendszer viselkedését legyenek képesek előírni, és ebből hardverleírást generálni egy arra alkalmas fordító segítségével. Az első generációt a Synopsys cég készítette el 1994-ben Behavioral Compiler néven, ami VHDL-t és Verilogot használt bemeneti nyelveként. Viszont az eszközök amik viselkedést leíró VHDL-t és Verilogot alkalmazták nem terjedtek el, mert a nyelvek nem voltak alkalmasak magasszinten való algoritmus leírásra, ezért 2004-ben leállították a Behavioral Compiler támogatsát és tovább fejlesztését. Eközben megjelent az igény egy ténylegesen magas szintű nyelvet (pl. C-t) használó szintézer kifejlesztésére, ezt hívják ma magas szintű szintézisnek.

## PipeComp keretrendszer

A BME IIT tanszékén folyó HLS projekt célja pont egy ilyen magasszintű szintézis keretrendszer megalkotása. Ennek neve a PipeComp[7], ami a sok bemeneti és kimeneti nyelv és formátum támogatása érdekében 3 rétegű architektúrát valósít meg (1‑1. ábra). A bemenet feldolgozásáért a frontend réteg felelős, ami egy köztes algoritmust leíró adatmodellbe fordítja ez az ún. HIG adatfolyamgráf. A middle-end végrehajt bizonyos nyelvfüggetlen optimalizációs lépéseket, illetve különböző transzformációkat hajt végre az adatfolyamgráfon. A backend réteg végzi az adatfolyamgráf feldolgozását, és kívánt nyelvre való fordítást.



1‑. ábra PipeComp keretrendszer architektúra

## HLS Backend

A feladatom egy ilyen VHDL-t generáló HLS Backend implementálása volt, amivel alapműveleteket (összeadás, szorzás, osztás …stb) és egyszerű irányítási konstrukciókat kellett tudnom kezelni (összetett művelet, elágazás, ismétlődés, memória). Az általam elkészített program bemenetként a HIG adatfolyamgráfot fogadja, bejárja és feldolgozza. Végezetül VHDL leírókat generál amik alkalmasak a HIG által leírt algoritmus végrehajtására.

# VHDL hardverleíró nyelv jellegzetességei

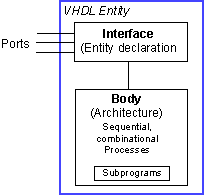
A VHDL(VHSIC Hardware Description Language) az egyik legelterjedtebb hardver leíró nyelv napjainkban. A hardver leíró nyelvek céljukat tekintve eltérnek más magasabb szintű szoftveres nyelvektől, mert utóbbiak a fordítás után a processzorban futnak ami sorosan hajtja végre a fordított kód utasításait. Míg a hardver leíró nyelvek esetén az általa definiált modulok és azok összeköttetései egymás mellett párhuzamosan léteznek. Tehát nem egy algoritmus lépésről lépésre történő végrehajtását, hanem annak elemi blokkok és azok összeköttetéseivel írják le azt.

## Felépítés

Főleg FPGA és ASIC áramkörök tervezésére és szimulálására használják a hardverleíró nyelveket, mivel a bonyolult és ebből fakadóan nehezen áttekinthető kapcsolásirajzokkal szemben a hardver leíró nyelvek gépek és ember által is jól olvasható kódot eredményeznek. Az áttekinthetőséget növeli az is, hogy a VHDL lehetőséget ad több modul definiálására és ezek egymásba ágyazására is.

### Entity és Architecture

A modulok két fő részből állnak (2‑1. ábra). Az egyik az ún. Entity ami a modul a környezete felé mutatott tulajdonságait tartalmazza, ilyenek a modul ki- és bemenetei (továbbiakban port) illetve azok típusa és bitszáma. Opcionálisan tartalmazhat generikus paramétereket is, ezeket felhasználhatjuk a modul funkcionális leírásában és a portjai bitszámát is paraméterezhetővé tehetjük velük. A modul példányosítása során megadhatjuk milyen értéket szeretnénk a generikus paraméterek helyére behelyetesíteni, így azonos logikájú modulokat nem szüskéges többször újra megírni. A másik fő rész az ún. Architecture, ez a modul működését és belső összeköttetéseit írja le, itt definiálhatók a modul almoduljai is, ezekre component-ként hivatkoznunk kell az almodulok Entity részében definiált interfész pontos másolatával. Ezután már többet is példányosíthatunk az adott modulból ki- és bemeneteire kötött vezetékek és generikus paramétereinek megadásával.



2‑. ábra A VHDL strukturális felépítése

### Példányosítás

A modulokban definiálhatunk más modulokat komponensként, amennyiben szükségünk van egy másik modul funkcionalitására, így a VHDL kód sokkal struktúráltabbá, áttekinthetőbbé válik, és lehetőségünk nyílik a egyes modulok újra felhasználására. Akár több példányt is létrehozhatunk ugyanabból a modulból miután felvettük a komponens listára az entity-jében megadott portokat és generikus paraméterek pontos másolatát. A példányosításkor meg kell adni a generikus paraméterek pontos értékét, továbbá a komponens példány portjaihoz kell rendelni a kívánt signalokat vagy portokat (ahogy a IDE HIVATKOZÁS ábrán is látszik).

### Főbb adattípusok és konstansok

A VHDL szabványban összesen 6 egyszerű adattípus létezik és ebből csak 3 szinteziháltahó a végleges áramkörbe (a többit inkább tesztelésnél használhatóak), ezek a következők:

* BIT : 0,1 értéket vehet fel
* BOOLEAN: true, false értéket vehet fel. Az előző típustól a hordozott információ tekintetében nem tér, az áttekinthetőség érdekében használják.
* INTEGER: 32 bites előjeles számot reprezentál

Ezeken felül általánosan használatos egy külön importálandó könyvtár ami a IEEE 1164-es szabványban [1] rögzítettek. Ez az std\_logic és std\_logic\_vectort tartalmazza az első egy adattípus a másik annak vektorba fűzött formája. A std\_logic és a bit között az a különbség, hogy a std\_logic 0 és 1 értékeken kívül más értékeket is felvehet: ’U' – Uninitialized, 'X' - Forcing Unknown, 'Z' - High Impedance, 'W' - Weak unknown, 'L' - Weak 0, 'H' - Weak '1', '-' - Don't care. Ezek közül csak a ’Z’ érték szintezálódik a végső áramkörbe, a többi a modulok szimulációjának és a tesztelésének könnyítése miatt van jelen.

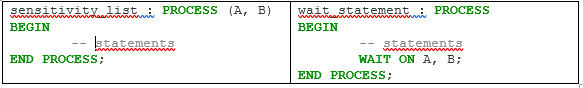
A modulokban definiálhatóak konstans értékek is a megfelelő adattípus megadásával(példa IDE HIVATKOZÁS). Ezek később értékül adhatóak portoknak és signaloknak.

### Signal

A signal a modulon belüli „vezetékeket” reprezentálja, a meghajtó és fogadó között. A vezetéket egyszerre csak egy forrás tudja meghajtani adattal, de több nyelőt is köthetünk az adatok fogadására. A forrás és nyelő bármi lehet ami a signal definiálásakor megadott adattípust ad eredményeként vagy vár bementeként (signal definiálásra példa: IDE HIVATKOZÁS). A signalt nem csak összeköttetés, hanem mint egy fajta pufferként is felfoghatjuk amiben a modul bizonyos műveleteinek eredményei tárolódnak. Bár egyszere csak egy forrás tudja meghajtani a signalt, de ez a forrás időben változtatható pl: process segítségével.

### Processek

A process a modul elvárt viselkedését írja le. A processben leírt viselkedést egy végtelen ciklus sorról sorra hajtja végre, ezt a wait kulcsszó használatával megállíthatjuk és újra indulását egy feltételhez köthetjük. Továbbá a ciklustörzs lefutását egy a modulban deklarált tag(signal, port) változásához is köthetjük, ha azt a process sensitivity listjébe felvesszük. Ezt mutatja a 2‑2. ábra ahol mindkét process ugyanazt a működést írja le, feltéve ha statements helyére ugyanazokat a kifejezéseket írjuk.



2‑. ábra Sensitivity list és wait utasítás példa

# Eclipse alapú modellezési technológiák

Az Eclipse Foundation[3] egy non profit, a tagjai által támogatott cég, amit 2004 januárjában hozták létre, hogy segítse és irányítsa az Eclipse közösség munkáját. Ez a közösség mára már több nyíltforráskódú project fejlesztéséért felelős, ilyen az Eclipse foundation megalakítása előtt már elindult Eclipse project is.

## Eclipse project

Az Eclipse project[4] egy nyíltforráskódú többnyire java alapú project, amit az IBM hozott létre 2001-ben az Eclipse SDK fejlesztése és karbantartása céljából. 2004-ben a projectvezetését a többek között erre a célra létrehozott Eclipse Foundation vette át, és a mai napig ez a szervezet egyengeti útjait. Mára a project jelentősen kinőte magát, és az idő előre haladtával már inkább mint "Eclipse top-level project"-ként hivatkoznak rá a projectek résztvevői mivel mostanra már több alprojectre bontották a nagy projectet. Jelen pillanatban öt alproject fut egymás mellett ezek a Platform, a Java development tools, a Plug-in Development Environment, E4 és az Orion.

### Eclipse platform

Az Eclipse platform[5] nem más mint olyan keretrendszerek és szolgáltatások összessége, amik alkalmazásaink elkészítését segítik, mert így minden ami a fejlesztés szükséges egy helyen elérhető vagy beszerezhető. A platform támogatja a külső szolgáltatások/keretrendszerek integrációját pluginnek formájában így minél jobban testre szabható és felhasználóbarát élményt nyújt. A pluginnek megoldást nyújthatnak különböző a fejlesztés közben felmerülő problémákra.

### Eclipse Modelling Framework

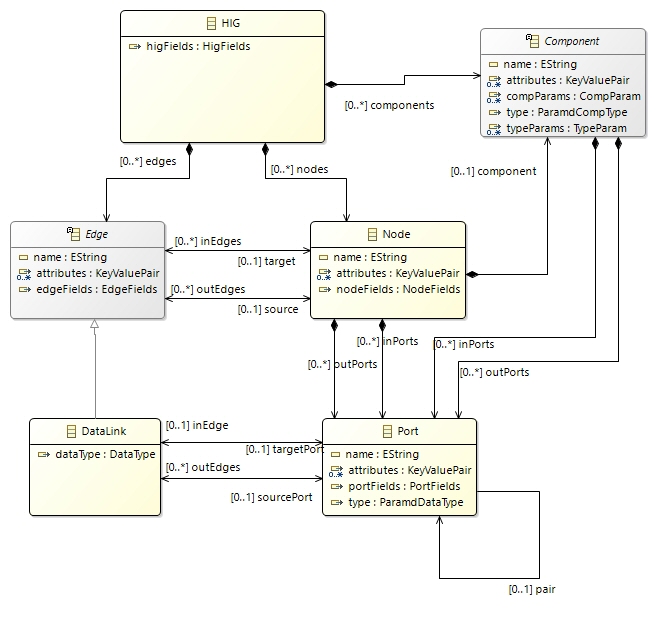
A rendszerek modellezésénél mindig is nagy problémát jelentett a modell megalkotása és módosítása után a modellt reprezentáló kód elkészítésnek időigénye és az a tény, hogy az így készülő boiler-plate kód megírása több lehetőséget adott az emberi hibázásra. Tehát a fejlesztési idő és a hiba lehetőségének csökkentésére szükségünk van egy olyan megoldásra ami mindezt magából a modellből kiindulva automatikusan elő tudja állítani. Erre több megoldás is létezik a piacon, az egyik ilyen az Eclipse Foundation által kínált Eclipse Modelling Framework(továbbiakban EMF). A modellünk definiálása után az EMF elkészíti az azt reprezentáló java osztályokat és hozzájuk tartozó utility osztályokat is, így nekünk erre nem kell időt szakítanunk.

Az EMF segítségével van lehetőségünk egy grafikus felület segítségével definiálni magas szintű modellünket, az így létrehozott modellek neve Ecore modellek. Az Ecore egy metamodell ami a strukturális felépítését és lehetséges tulajdonságait tartalmazza az ecore segítségével létrehozható példánymodellnek. Ezen példánymodellek létrehozását az EMF az Ecore alapján generálható szerkesztővel is segíti.

Az Ecore modell generálásához az alábbi komponenseket használhatjuk fel:

* EObject: Minden EMF által generált osztály az EObject leszármazottja. Igazából a java.lang.Object EMF reprezentációjaként tekinthetünk rá.
* EPackage: Ez tartalmazza a modelleket a legfelső szinten, a modellek elemei egy EPackage példányhoz tartoznak.
* EClass: Ezt tekinthetjük az osztályok reprezentációinak. Egy Java interfész és azt implementáló osztályként generálódik le modellünkből.
* EAttribute: egy attribútumot modellez, amelynek neve és típusa van.
* EReference: Két osztály közötti kapcsolatot reprezentálja, tartalmazza a kapcsolat multiplicitását és nevét. Ezen felül asszociációt és aggregáció is jelképezhet.

Ezeket a komponenseket szemlélteti a 3‑1. ábra amin a HIG adatfolyamgráf ecore reprezentációja látható. A példánymodelleket fa struktúrában tárolja el az EMF. Ezek bejárásának segítésére is generál utility osztályokat, a visitor patternt hívja segítségül és a bejáró



3‑1. ábra A HIG grafikus ecore modellje

# Template-nyelvek

A Java-ba beépített a szövegek dinamikus generálására nyújtott lehetőségei meglehetősen kezdetlegesek, mert nehezen olvasható kódot eredményeznek, és ebből fakadóan nehéz használni is őket erre a célra. Ezért szükséges volt egy erre a célra kitalált template-nyelvet keresnem, amiképes egy külső dokumentumba beágyazódni, így változtatni dinamikusan a dokumentum egyes részein. Erre több módszert kipróbáltam ezek leírása következik.

## Összehasonlítás szempontjai

### Java és Eclipse támogatás

Mivel a Javat választottam a program elkészítéséhez, a template-nyelvnek is szükséges támogatnia azt.

Továbbá a HIG adatmodell egy Eclipse-s keretrednszerben lett modellezve, ezért adott volt az Eclipse fejlesztőkörnyezet használata is. Így egy olyan template-nyelvet szerettem volna választani Eclipse-ben megfelelően támogatott.

### Vezérlő szerkezetek támogatása

A HIG modellben sok listás elem található, ezért a ciklusok megléte elengedhetetlenül fontos volt a választás során. Továbbá az elágazások segíthetnek megoldani bizonyos mapping problémákat.

A belső függvények és egy fajta szkriptnyelv támogatottságát is figyelembe szeretném venni, mivel sokszor kell majd a kapott adaton műveleteket végrehajtanom, és így nem kell korábban megtennem azt.

## Freemarker

A Freemarker[8] egy ingyenes Java alapú template motor, eredetileg dinamikus weboldalak generálására készítették, mivel független a webes technológiáktól gyakran használják forrás file-ok és e-mailek generálására is.

### Java és Eclipse támogatás

Található Eclipse plugin a FreeMarkerhez, ami szintaktikai kiemelést, ellenőrzést és kódkiegészítő funkcióval is rendelkezik. Használata viszonylag egyszerűbb, de a template változóit kívülről egy Map segítségével kell a template-be juttatni. Minden használatnál fel kell olvasni a háttértárolóról a tisztán szöveges template file-t ezzel nem elhanyagolható overheadet eredményezve más binárisra fordított template nyelvekkel szemben.

### Vezérlő szerkezetek támogatása

A template kifejezéseket elhatároló eleme a ${KIFEJEZÉS}, az ilyen szerkezetbe írt kifejezések a template kiértékelése során a kívülről adott adatok alapján kapnak értéket. Akár objetumokat is átadhatunk amelyek függvényeihez és belső adatstruktúráihoz is hozzáférhetünk.

#### Listák és ciklusok

A Freemarker támogatja a listák és ciklusok kezelését is, végig iterálhatunk rajtuk és az éppen soron következő elemhez mint az objektumokhoz férhetünk hozzá. A listákat a <#list *listaneve* as *objektumneve*> kifejezéssel kezelhetjük.

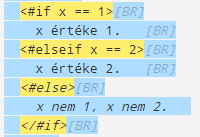


‑. ábra Freemarker lista kezelés

A ciklusos működést is <#list..> kifejezéssel tudjuk előhívni, itt valójában egy szám listán iterálunk végig ezzel imitálva az n ideig tartó ciklikus működést.

#### Elágazások

Lehetőség van elágazások használatára is, ezzel ha nincs szükségünk egy szövegrészre azt egy bizonyos feltételhez köthetjük a templaten belül. Az elágazás leírásához a <#if…> kifejezéssel használhatjuk, végezetül pedig </#if> záró kifejezéssel tudjuk megadni a végét. Az if kifejezés után használható a <#elseif…>,<#else>, mint *egyébként ha* és *egyébként* ágak kezelésére.



‑. ábra Freemarker elágazás kezelés

## Apache Velocity

## Xtend

## Konklúzió

# VHDL generálás

## HIG adatmodell

## Hardver szintézer specifikációja

## Implementáció

# Tesztelés

Irodalomjegyzék

1. Multivalue Logic System for VHDL Model Interoperability https://en.wikipedia.org/wiki/IEEE\_1164
2. Eclipse modelling framework: <https://eclipsesource.com/blogs/tutorials/emf-tutorial/>
3. Eclipse Foundation, http://www.eclipse.org/org/
4. Eclipse Project, <http://wiki.eclipse.org/Eclipse_Project>
5. Eclipse Platform, <http://wiki.eclipse.org/Platform>
6. High-level synthesis, <https://en.wikipedia.org/wiki/High-level_synthesis>
7. G. Suba, and P. Arató: Concept of the system-level synthesis framework PipeComp, <http://hls.iit.bme.hu/lib/exe/fetch.php/hu/pipecomp.pdf>
8. Freemarker, https://freemarker.apache.org/

Függelék