

Digitális rendszer modellezése

90 – wid

Konzulens:

Dr. László Zoltán



Csapattagok

Csomák Gábor	MFN5OW	csomakk@gmail.com
Jégh Tamás András	L1W80X	jeghtamas@gmail.com
Sziklay György	GABOLN	sziklaygy@gmail.com
Vad Zsolt Sándor	NOGXFT	zse.vad@gmail.com
Wiesner Péter Ádám	ARFFY9	peteee24@gmail.com

2011. május 9.

Tartalomjegyzék

1.	Követelmény, projekt, funkcionalitás.....	6
1.1	Követelmény definíció.....	6
1.1.1	A program célja, alapvető feladatai	6
1.1.2	A felhasználói felület	6
1.1.3	A program futtatásához szükséges követelmények.....	6
1.1.4	A szoftver fejlesztésével kapcsolatos alapkövetelmények, elvek, célok és egyéb megkötések	7
1.2	Projekt terv.....	8
1.2.1	A felhasznált fejlesztőeszközök	8
1.2.2	A fejlesztőcsapat tagjai, azok feladatkörei	8
1.2.3	Kommunikáció	8
1.2.4	Fejlesztési ütemterv	10
1.2.5	Dokumentációk beadási határideje.....	10
1.2.6	Ütemterv és konzultációk.....	10
1.2.7	Átadás.....	11
1.2.8	Kockázatok azonosítása és elemzése	11
1.3	Feladateleírás	12
1.4	Szótár.....	14
	Szó	14
	Jelentés.....	14
1.5	Essential use-case-ek.....	16
1.5.1	Use-case diagram	16
1.5.2	Use-case leírások	16
1.6	Napló	18
2.	Analízis modell kidolgozása	19
2.1	Objektum katalógus	19
2.1.1	DigitalBoard	19
2.1.2	Wire	19
2.1.3	DigitalObject	19
2.1.4	Input	19
2.1.5	Switch	19
2.1.6	Generator	19
2.1.7	Gate	19

2.1.8	andGate	19
2.1.9	orGate	20
2.1.10	Inverter	20
2.1.11	Output	20
2.1.12	LED	20
2.1	Osztályok leírása	21
2.1.1	DigitalBoard	21
2.1.2	Wire	21
2.1.3	DigitalObject	22
2.1.4	Input	23
2.1.5	Switch	23
2.1.6	Generator	23
2.1.7	Output	24
2.1.8	LED	24
2.1.9	Gate	25
2.1.10	andGate	25
2.1.11	orGate	26
2.1.12	Inverter	26
2.2	Statikus struktúra diagramok	28
2.3	Szekvencia diagramok	28
2.4	State-chart	35
2.5	Napló	35
3.	Szkeleton tervezése	36
3.1	A szkeleton modell valóságos use-case-ei	36
3.1.1	Use-case diagram	36
3.1.2	Use-case leírások	37
3.2	Architektúra	39
3.2.1	Leírás	39
3.2.2	Tesztáramkörök	39
3.3	A szkeleton kezelői felületének terve, dialógusok	39
3.4	Szekvencia diagramok a belső működésre	41
3.5	Napló	47
4.	Szkeleton beadás	48
4.1	Fordítási és futtatási útmutató	48

4.1.1	Fájllista.....	48
4.1.2	Fordítás.....	48
4.1.3	Futtatás.....	48
4.2	Értékelés	49
4.3	Napló	49
5.	Prototípus koncepciója.....	50
	Specifikáció hatása a projektre	50
5.1.1	Módosítás leírása	50
5.1.2	Analízis modell változásai	50
5.2	Prototípus interface definíciója.....	52
5.2.1	Az interfész általános leírása	52
5.2.2	Bemeneti nyelv.....	52
5.2.3	butaHDL [©] ismertetése, alapvető ötletek, technikai elgondolások a megvalósításáról	53
5.2.4	Kimeneti nyelv	55
5.3	Összes részletes use-case	56
5.4	Tesztelési terv.....	58
5.4.1	A tesztelés menete	58
5.4.2	A teszteléshez használható parancsok	59
5.4.3	A kimeneten megjelenő hibaüzenetek, figyelmeztetések	59
5.4.4	A kimeneten megjelenő egyéb események	59
5.4.5	Az áramkör formátuma	59
5.4.6	Tesztesetek.....	60
5.5	Tesztelést támogató segéd- és fordítóprogramok specifikálása.....	61
5.6	Napló	62
6.	Részletes tervek.....	63
6.1	Osztályok és metódusok tervei.	63
6.1.1	DigitalBoard	63
2.5.1	Wire	64
2.5.2	DigitalObject	64
6.1.2	Composit.....	65
6.1.3	Input	67
6.1.4	Switch	67
6.1.5	Generator	67
6.1.6	Output	68

6.1.7	LED	68
6.1.8	Oscilloscope	69
6.1.9	Gate	69
6.1.10	ANDGate	70
6.1.11	orGate	70
6.1.12	Inverter	70
6.1.13	bhdlParser	71
6.1.14	PIN	72
6.2	A tesztek részletes tervei, leírásuk a teszt nyelvén	72
6.3	A tesztelést támogató programok tervei	77
6.4	Napló	77
7.	Prototípus beadása	78
7.1	Fordítási és futtatási útmutató	78
7.1.1	Fájllista	78
7.1.2	Fordítás	79
7.1.3	Futtatás	79
7.2	Tesztek jegyzőkönyvei	79
7.2.1	Egyszerű áramkör	79
7.2.2	Visszacsatolt stabil áramkör	79
7.2.3	Instabil áramkör	79
7.2.4	Bonyolult áramkör	80
7.3	Értékelés	81
7.4	Napló	81
7.5	Változtatások	81
8.	Grafikus felület specifikációja	83
8.1	A grafikus interfész	83
8.2	A grafikus rendszer architektúrája	85
8.2.1	A felület működési elve	85
8.2.2	A felület osztály-struktúrája	86
8.3	A grafikus objektumok felsorolása	87
8.3.1	View	88
	Controller	88
8.3.2	Timer	90
8.3.3	WireView	91

8.3.4	DigitalObjectView	92
8.4	Kapcsolat az alkalmazói rendszerrel.....	93
8.5	Napló	95
9.	Grafikus változat beadása	96
9.1	Változások	96
9.1.1	A grafikus interfész (bővítés)	97
9.1.2	Szekvencia diagramok	98
9.1.3	Fájllista.....	109
9.1.4	Fordítás és telepítés	111
9.1.5	Futtatás.....	112
9.2	Értékelés	112
9.3	Napló	112
10.	Összegzés és értékelés	113
10.1	Összegzés.....	113
10.2	Értékelés	113
	Mit tanultak a projektből konkrétan és általában?	113

1. Követelmény, projekt, funkcionalitás

1.1 Követelmény definíció

1.1.1 A program célja, alapvető feladatai

A program egyszerű digitális áramkörök szimulációját, tesztelését valósítja meg, alapvető logikai építőelemek felhasználásával. Az áramkörben kapcsolók illetve jelgenerátorok vannak, melyeket tetszőlegesen lehet beállítani.

A fejlesztőcsapat célja egy olyan kész program előállítása, mely teljes mértékben kielégíti a specifikációban megkövetelteket, és ami minden olyan gépen lefordítható, futtatható, mely megfelel a későbbiekben megfogalmazott követelményeknek.

1.1.2 A felhasználói felület

A kész program végső változata billentyűzet és egér felhasználásával lesz használható és grafikus felhasználói felülettel fog rendelkezni.

1.1.3 A program futtatásához szükséges követelmények

A futáshoz szükséges, hogy a felhasználó számítógépére telepítve legyen a JRE¹ 1.6.

¹ Java Runtime Environment

A program hardverigénye: Pentium 1,6 GHz vagy annál gyorsabb processzor, minimum 512 MB RAM és 125 MB hely a háttértárolón.

1.1.4 A szoftver fejlesztésével kapcsolatos alapkövetelmények, elvek, célok és egyéb megkötések

Modellhűség:

Az elkészült program minőségét jelentős mértékben az határozza meg, hogy a megvalósítása mennyire pontosan követi a specifikációt, így azok követésére már a kezdetektől nagy hangsúlyt fektetünk.

Továbbfejleszthetőség:

A modell elkészítése során ügyelünk arra, hogy az könnyen kiegészíthető és továbbfejleszthető legyen.

Modularitás:

A moduláris programozás elvét követve a programot lehetőleg funkció szerinti részfeladatokra bontva készítjük el, amelyeket így külön-külön lehet tesztelni és kiértékelni, amely nagyban elősegíti a program fejlődését és az esetleges hibák felderítését.

Teljesítmény, optimalizálhatóság:

A legjobb teljesítmény érdekében törekszünk a lehető leghatékonyabb megoldásokat felhasználni, amelyek az adott környezetben a leghatékonyabban tudják kihasználni az erőforrásokat és így a szimuláció fennakadásmentes futását biztosítják.

Felhasználhatóság:

A fejlesztők célja egy könnyen használható, átlátható kezelőfelülettel rendelkező program készítése, melynek irányítása gyorsan elsajátítható akár a felhasználói kézikönyv nélkül is.

Modern technológiák használata:

Az UML és a RUP használata illetve követése lehetővé teszi a szoftver logikusan megszervezett fejlesztését. A létrehozott, szabványosan szervezett dokumentációk megkönnyítik a későbbi munkafázisokat, és egy átlátható képet nyújtanak a fejlesztési folyamatról.

Rugalmasság:

A programnak minden, a rendszerkövetelményeknek megfelelő rendszeren megfelelően kell futnia. Továbbá a grafikai kezelőfelület olyan különálló réteget képez a modell és az alap program felett, mely könnyen módosítható, anélkül, hogy a modellt bármilyen mértékben változtatni kellene.

1.2 Projekt terv

1.2.1 A felhasznált fejlesztőeszközök

A program fejlesztése során az Eclipse IDE²-t fogjuk használni. Az UML modellek elkészítéséhez az Enterprise Architect³-et használjuk, ezzel létrehozva az UML szabványoknak megfelelő diagramokat.

A dokumentációt a Microsoft Word⁴ programmal írjuk meg.

Verziókezelést a TortoiseGit⁵ grafikus kliens felhasználásával msysgit⁶ felületen oldjuk meg.

1.2.2 A fejlesztőcsapat tagjai, azok feladatkörei

Név	Neptun	Feladatkör
Csomák Gábor	MFN5OW	csapatvezetés, projektszervezés, dokumentáció, kódírás
Jégh Tamás	L1W80X	dokumentáció, tesztelés, kódírás
Sziklay György	GABOLN	dokumentáció, tesztelés, kódírás
Vad Zsolt	NOGXFT	végleges dokumentáció, tesztelés, kódírás
Wiesner Péter	ARFFY9	dokumentáció, tesztelés, kódírás

1.2.3 Kommunikáció

Csapatunkba kizárólag olyan emberek kerültek, akik már sokadik éve ismerik egymást illetve egymás képességeit. A válogatás alatt figyelembe vettük, hogy minden feladatkörhöz az arra legmegfelelőbb személy kerüljön. A csapatban 5-en veszünk részt, ezért a szinkron kommunikáció adta előnyöket sajnos nem tudjuk kellőképpen kihasználni, mivel nehezen tudunk olyan időpontot találni, amely mindenkinek alkalmas lenne erre. A csapattagok péntek délben konzultációt tartanak, amely a közös munka leghatékonyabb formáját biztosítja.

A kapcsolatot ezen kívül telefonon és levelezőlistán tartjuk. Mindenki rendelkezik Gmail-es postafiókkal is, ezért ritkán lehetőség nyílik arra, hogy rövid megbeszéléseket folytassunk a

² <http://eclipse.org>

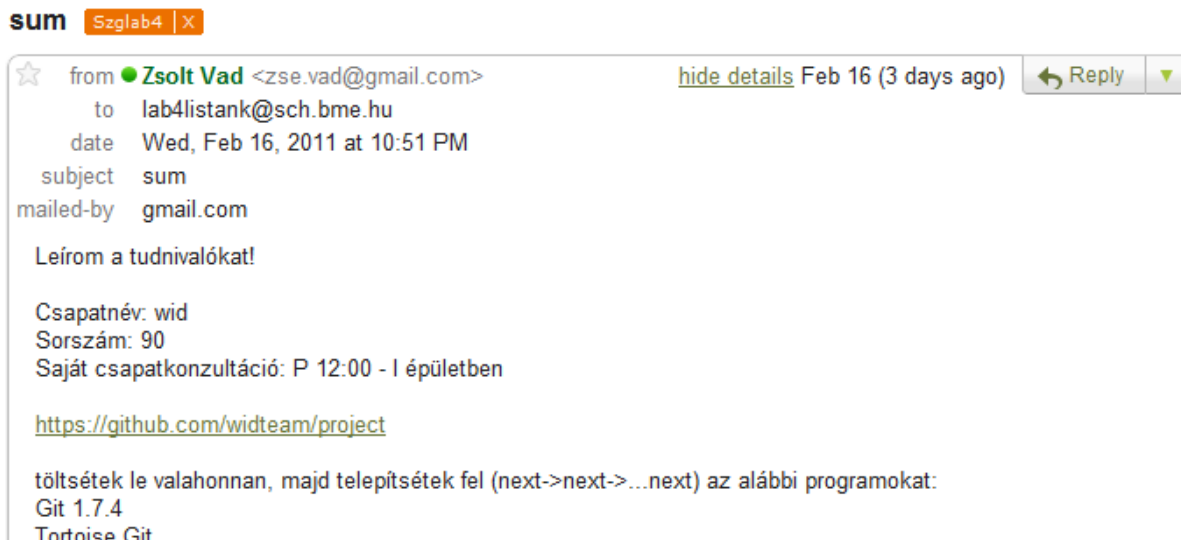
³ <http://www.sparxsystems.com.au/>

⁴ <http://office.microsoft.com/en-us/word/>

⁵ <http://code.google.com/p/tortoisegit/>

⁶ <http://code.google.com/p/msysgit/>

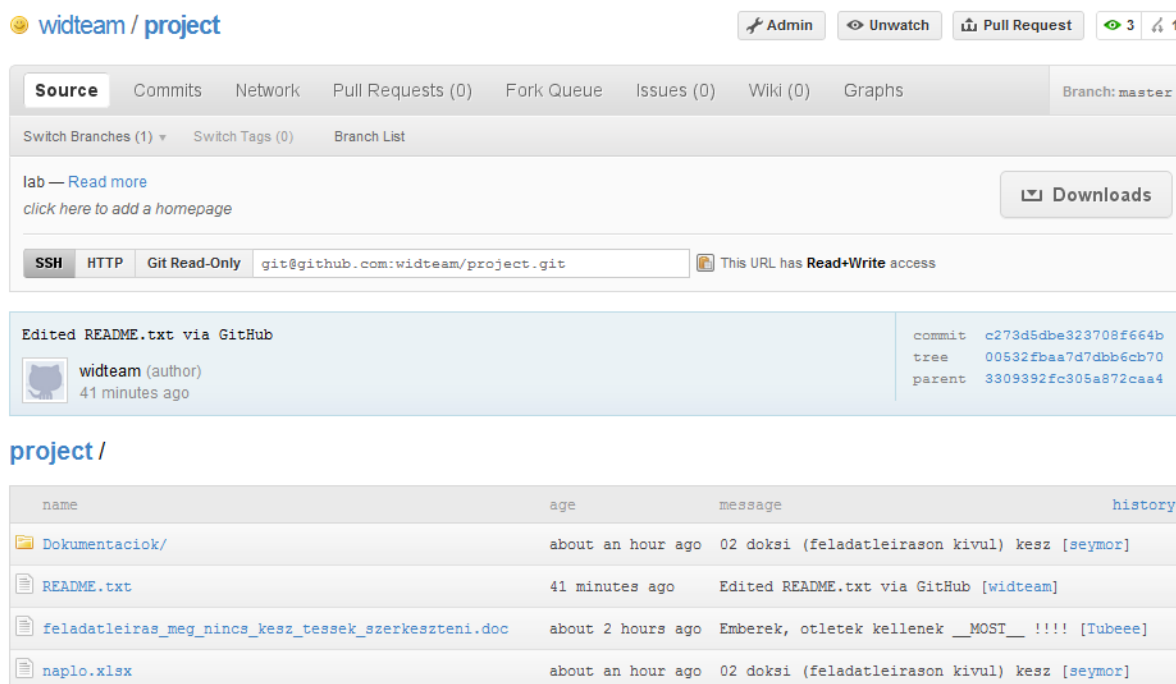
GTalk⁷ alkalmazás segítségével. A telefonos konzultációt csak rendkívüli helyzetben alkalmazzuk.



1. ábra - Levelező lista

GIT verziókezelés:

Ennek segítségével a forráskódot akár egymással párhuzamosan is szerkeszthetik a csapattagok, azokhoz megjegyzéseket fűzhetnek, nyomon követhetik a másik munkáját, illetve szükség esetén visszatérhetnek régebbi verziókhoz is. A GIT webes felületéhez a <http://github.com> oldalt használjuk.



2. ábra - GITHUB könyvtár

⁷

<http://www.google.com/talk/>

1.2.4 Fejlesztési ütemterv

A projekt fejlesztése során három mérföldkövet nevezhetünk meg, amelyek különleges jelentőséggel bírnak.

1. A szkeleton változat célja annak bemutatása, hogy az objektum és dinamikus modellek a követelményekben definiált feladat egy megfelelő modelljét alkotják. Ebben a programban már minden business objektum szerepel, de még csak az interfészük definiált. Szerepe, hogy a különböző forgatókönyvek és szekvenciadiagramok ellenőrizhetőek legyenek.
2. A prototípus egy olyan program, amely már az összes szükséges funkcionalitást megvalósítja és ezt demonstrálni is tudja, ám egyelőre még grafikus felület nélkül.
3. A harmadik lépésben készül el a teljes, grafikus program, mely a prototípustól csak a megjelenésben különbözik.

1.2.5 Dokumentációk beadási határideje

	Dátum	Feladat
1	febr. 21.	Követelmény, projekt, funkcionalitás
2	febr. 28.	Analízis modell kidolgozása 1.
3	márc. 7.	Analízis modell kidolgozása 2.
4	márc. 14.	Szkeleton tervezése
5	márc. 21.	Skeleton
6	márc. 28.	Prototípus koncepciója
7	ápr. 4.	Részletes tervek
8	ápr. 11.	
9	ápr. 18.	Prototípus
10	ápr. 26.	Grafikus felület specifikációja
11	máj. 2.	
12	máj. 9.	Grafikus változat
13	máj. 13.	Összefoglalás

1.2.6 Ütemterv és konzultációk

	Dátum	Feladat
2	febr. 16.	konzultáció
3	febr. 23.	konzultáció
4	márc. 2.	konzultáció
5	márc. 9.	konzultáció
6	márc. 16.	konzultáció
7	márc. 23.	szkeleton bemutató - konzultáció
8	márc. 30.	konzultáció
9	ápr. 6.	konzultáció
10	ápr. 13.	konzultáció
11	ápr. 20.	prototípus bemutató - konzultáció
12	ápr. 27.	konzultáció
13	máj. 4.	konzultáció
14	máj. 11.	grafikus változat bemutató - konzultáció

1.2.7 Átadás

A dokumentáció nyomtatott formában, fedőlappal ellátva a konzulensnek lesz átadva a megjelölt időpontokban. A fejlesztés során készített forráskódot a konzulens folyamatosan ellenőrizheti. A működő verzió mindig az IIT⁸ laboratóriumában kerül bemutatásra.

1.2.8 Kockázatok azonosítása és elemzése

Kockázatok azonosítása:

	Rövid leírás	Típus	Ok	Következmény
1	csapattag kiválás	projekt	önszántából kilépő tag	ha van olyan személy, akinek felbomlott a csapata, akkor őt megkeressük
2	csapattag távollét	projekt	betegség, külföldre utazott	feladatkörök újbóli kiosztása erre az időre
3	specifikáció megváltozása	termék	további funkció hozzárendelése a szoftverhez	külön konzultáció
4	dokumentáció visszautasítása	termék	hiányos vagy rossz leírások	rendkívüli konzultáció
5	dokumentáció javítása	termék	apró módosítások szükségesek	külön konzultáció
6	csapatkonzultáció elmaradása	projekt	nem sikerül az adott időpontban találkozni	másik konzultációs időpont megbeszélése
7	konzultációról való elmaradás	projekt	egyik csapattag sem tud részt venni a konzulensen való egyeztetésen	rendkívüli konzultáció
8	más szoftverekre való átállás	üzleti	a jelenlegi szoftvernek nincs olyan funkciója, amire szüksége van a csapattagoknak	új szoftver keresése
9	határidő túllépése	projekt	kicsúszik a csapat az időből	rendkívüli konzultáció

⁸ Irányítástechnika és Informatikai Tanszék

Kockázatok elemzése:

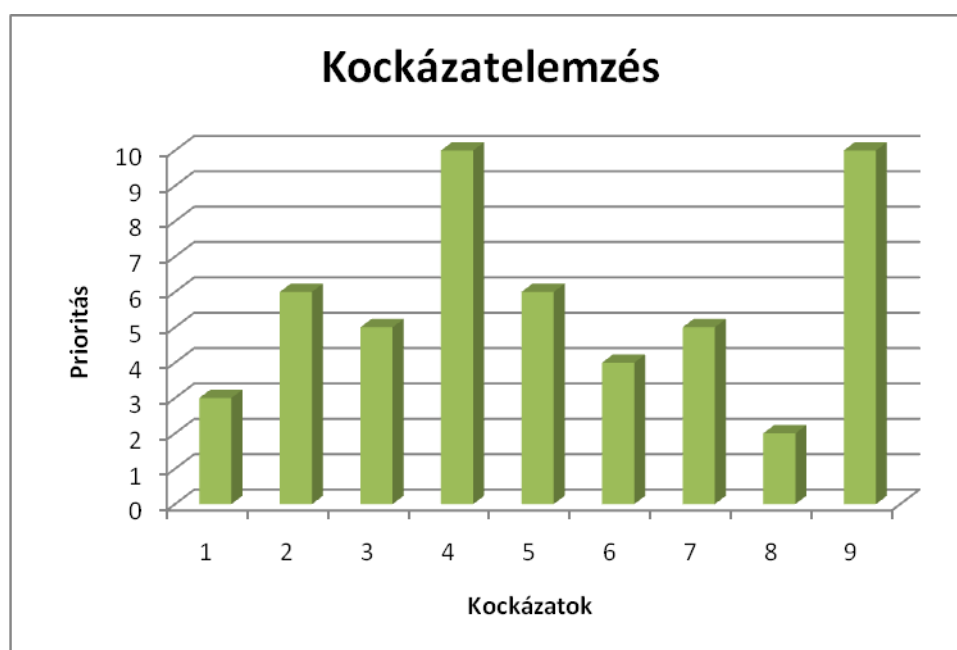
Az események valószínűsége és hatása az {1,2,3,4,5} halmazból veszik fel értéküket.

Valószínűség: 1 – Alacsony valószínűségű esemény, 5 – Biztos esemény

Hatás: 1 – Elhanyagolható következmény, 5 – Súlyos következmény

A kockázat prioritása a kockázatnak való kitettség mértéke, tehát a valószínűség és a hatás szorzata. Ennek alapján lehet sorba rendezni a kockázatokat.

	Rövid leírás	Valószínűség	Hatás	Prioritás (V*H)
1	csapattag kiválás	1	3	3
2	csapattag távollét	3	2	6
3	specifikáció megváltozása	5	1	5
4	dokumentáció visszautasítása	2	5	10
5	dokumentáció javítása	3	2	6
6	csapatkonzultáció elmaradása	1	4	4
7	konzultációról való elmaradás	1	5	5
8	más szoftverekre való átállás	1	2	2
9	határidő túllépése	2	5	10



3. ábra - Kockázatelemzés

1.3 Feladateleírás

A program lehetőséget biztosít egy logikai kapukból felépített digitális áramkör teljes körű szimulációjára, tesztelésére. Az áramkör működésének eredményeit kijelző(k) jeleníti(k) meg, de a program használója akár a program futása közben, az egyes építőelemek között is képes ellenőrizni munkájának egyes részleteit, így a felhasználó teljes képet kaphat az áramkör felépítéséről, működésének mikéntjéről, a különböző építőelemek kapcsolatáról. A tesztelni/szimulálni kívánt áramkör legalább egy jelforrásból, tetszőleges számú alapvető építőelemből és szintén tetszőleges számú kijelzőből áll, melyek drótokkal vannak összekötve. A rendszer bonyolultsága tetszőlegesen nagy lehet, segítségével bármilyen ma használatos elektronikai eszköz részletes, logikai mása tesztelhető.

A program indítása után egyből megjelenik a munkaterület és egy menü, mely segítségével könnyen elérhetőek az egyes lehetőségek. A munkaterület kezdetben egy üres felület, melyen később a megtervezett áramkör grafikus megjelenítése foglal helyet. Az áramkör

megjelenítéséhez szükséges a rendszert leíró fájl betöltése. Betöltés előtt nem lehet tesztelést folytatni, a felhasználó csak a menü által felkínált lehetőségek közül választhat. A szimuláció megkezdéséhez az elindítás gombra kell kattintani (a folyamat futását befolyásoló parancsokat lásd később), míg ez nem történik meg az áramkörnek csak a felépítése figyelhető meg.

Egy-egy építőelemet az elektronikában megszokott, konvencionális jelölésével reprezentál a program. Az elemek bemeneti oldalán a program minden csatlakozási pontot megjelenít egy rövid egyenes szakasszal, „drótkezdeménnyel” csak úgy, mint a kimeneti oldalon. Két összekötni kívánt elem között a kapcsolat – drót – egy egyszerű szakaszként jelenik meg. Egy kapcsolat több bementet is összeköthet egyetlen kimenettel, ilyenkor ez a kapcsolaton egy elágazásként jelenik meg.

Jelforrásnak minősül egy kapcsoló melyet a tervező az egérrel fel-le kapcsolhat, ezáltal adva neki értéket. A kapcsoló különböző állapotai egyértelműen megkülönböztethetőek, kinézetéből egyértelműen meghatározható annak értéke. Egy másik jelforrás a jelgenerátor, mely egy előre meghatározott, tetszőleges hosszú, mintájú (a tervező által meghatározott és billentyűzettel bevitt) jelsorozatot ad a kimenetén. A program egy négyzetként van reprezentálva, melynek kitöltöttsége attól függ, hogy a jelsorozat adott pillanatnyi értéke milyen.

A kijelző – LED – egyszerű kitöltött vagy kitöltetlen színes kör, a bemenetétől függően. Nem rendelkezik kimenettel, csakis a bemeneti értékének vizuális megjelenítésére szolgál. Amennyiben a kör valamely színnel kitöltött, a vezetéken folyik áram, ha üres, akkor nem.

Az INVERTER megjelenítése egy egyenlő szárú háromszög segítségével történik, a kapu egyetlen bemenete az alapnál található, egyetlen kimenete pedig a két szár által alkotott csúcsánál. A kapu maga minden jelet az ellentettjére változtat – ha a bemeneten van feszültség, a kimeneten nem lesz, és fordítva.

Az ÉS kapu az az építőelem mely a kimeneténél akkor szolgáltat feszültséget, ha az összes bemenetén van feszültség.

A VAGY kapu akkor ad feszültséget a kimenetén, ha valamelyik bemenetén van feszültség.

A kapcsolatokat reprezentáló drót tetszőleges hosszú lehet.

A felhasználó a szimulációt elindítva nyomon követheti a jel terjedését a vezetékeken a jelforrásoktól kezdve, ahogy a jel a kimenetek és bemenet között terjed, és lehetősége nyílik a rendszer viselkedésének megfigyelésére. Lekérdezheti, hogy milyen értékek jelennek meg az egyes építőelemek kimenetén, egészen a kijelző elemekig. A szimulációt bármikor „pillanat állj” állapotba helyezheti a felhasználó, melynek következtében az egyes elemek értékei ugyan nem módosulnak már, de leolvashatóak. A szimuláció ezek után bármikor folytatható. Lehetőség van még a futó folyamat megállítására, melynek következtében minden az alapértelmezett, eredeti értékre áll vissza.

A program nem engedélyezi a szabálytalan elrendezéseket, figyelmezteti a tervezőt minden helytelen elrendezésre, így nem tartalmazhat a rendszer bemenet nélküli építőelemet, összekötött kimeneteket, hibásan vagy egyáltalán nem értelmezhető összeköttetéseket, melyek során akár instabil működés is felléphet. Mindezek következménye, hogy a programban ábrázolt áramkör hibamentes és biztosan megvalósítható lesz. A program minden rendellenességről, érvénytelenségről, létező illetve esetleges hibákról üzenetekben azonnal tájékoztatja a felhasználót, segítve a munkáját.

Ilyen üzenetek például:

„Hiba! Kimenet-kimenet típusú összekapcsolás nem engedélyezett!”

„Hiba! Egy vagy több építőelem nem rendelkezik bemenettel!”

„Hiba! Érvénytelen visszacsatolás!”

„Vigyázat! Az áramkör visszacsatolást tartalmaz, mely instabilitást okozhat”

Stb.

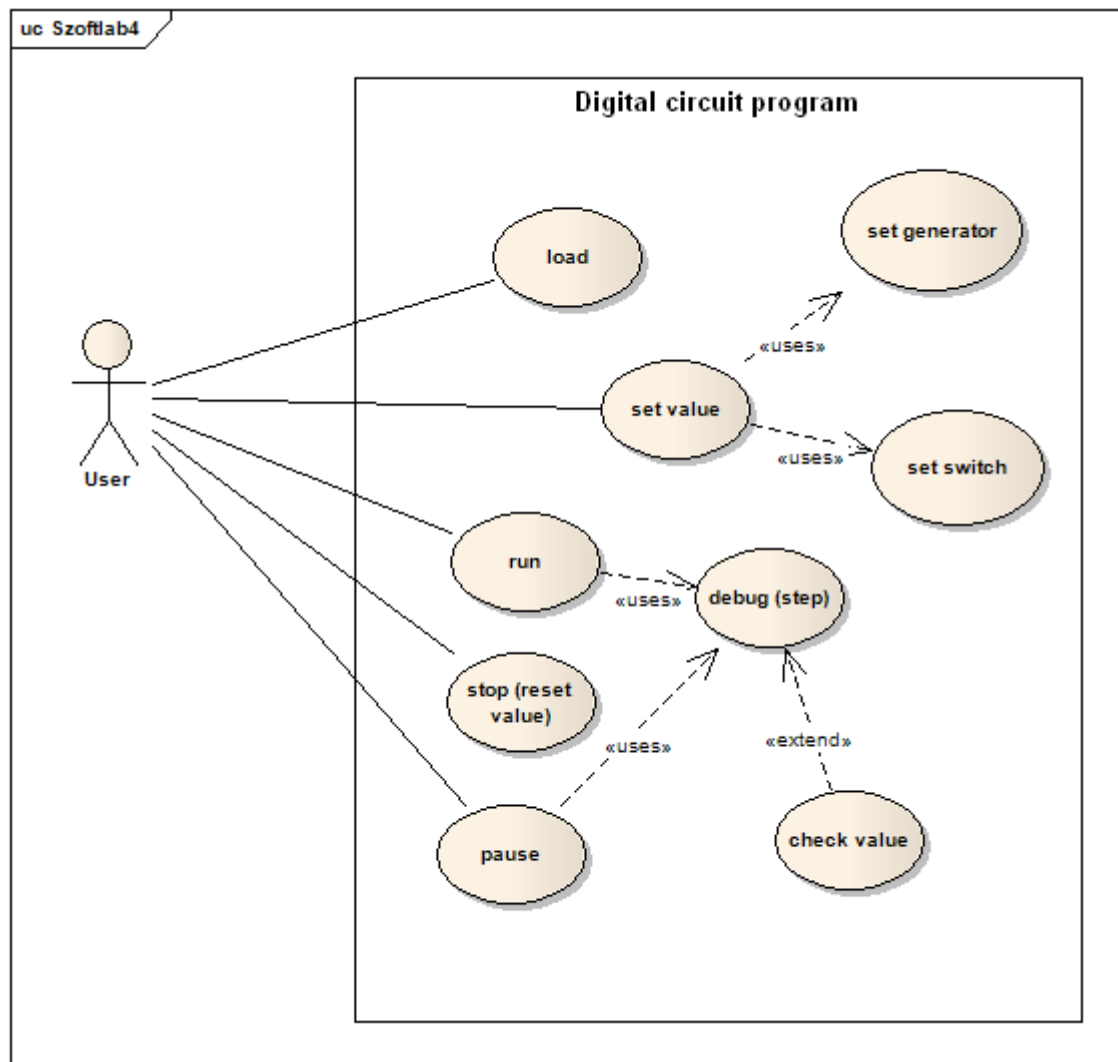
1.4 Szótár

Szó	Jelentés
digitális	Valamely változó jelenségnek, vagy fizikai mennyiségnek diszkrét (nem folytonos), megszámlálhatóan felaprózott, s így számokkal meghatározható, felírható értékeinek halmaza.
digitális áramkör	Elektromos hálózat, amely digitális információ tárolására, késleltetésére, számlálására, átalakítására, átvitelére, átkapcsolására szolgál.
elektromos hálózat	Elektromos árammal működő kapcsolatrendszer.
információ	Az informatika alapfogalma. Azt az adatot tekintjük információnak, amely számunkra releváns és ismerethiányt csökkent. Mértékegysége a bit.
bináris szám	Egy szám kettes számrendszerben kifejezett reprezentációját értjük.
bit	A digitális áramkörben található információ alapegysége. Értéke a $\{0,1\}$ kételemű halmazból kerül ki.
bitsorozat	Bitek véges listája, melyben a lista tagjainak sorrendje jól meghatározott. Egy elem többször is előfordulhat.
halmaz	Matematikai alapfogalom, ezért nem definiáljuk. (Elemek összessége)
logikai alapelem	A digitális áramkörbe beilleszthető fizikai elemek: inverter, logikai kapu, jelgenerátor, kapcsoló, kijelző, vezeték.
logikai kapu	Valamely logikai alpműveletet (és, vagy, nem), vagy ezek kombinációját megvalósító digitális áramköri alapelem, melynek tetszőleges számú bemenete és egy kimenete van.
logikai érték	Az $\{igaz, hamis\}$ kételemű halmazból vett érték. Digitális áramkörben: igaz=1, hamis=0.
logikai ÉS művelet	A művelet eredménye akkor igaz, ha minden bemenő logikai állítás értéke igaz.
logikai VAGY művelet	A művelet eredménye akkor igaz, ha legalább az egyik bemenő logikai állítás értéke igaz.
logikai NEM művelet	A bemenő állítás logikai értékét megváltoztatja az ellenkezőjére.
negáció	Logikai értelemben tagadást jelent. (igaz=>hamis, hamis=>igaz)
negált érték	Valamely logikai értéknek a negáció után felvett értéke.
inverter	A bemenetére érkező logikai igazra hamist, hamisra igazat ad ki a kimenetére.
ÉS kapu	Logikai igazat ad a kimenetére, ha az összes bemenete igaz, különben hamis.
VAGY kapu	Logikai igazat ad a kimenetére, ha legalább egy bemenete igaz, különben hamis.

kapu kimenet	bemenet,	A logikai kapuk csatlakozási pontjai a digitális áramkörhöz.
kapcsoló		2 állapotú jelforrás. Bekapcsolt állapotban a kimenete igaz, kikapcsolt állapotban hamis.
jelforrás		Olyan digitális építőelem, aminek csak egy kimenete van és a kimeneti érték a $\{0,1\}$ halmazból kerül ki.
jelgenerátor		Olyan jelforrás, amely ciklikusan ad ki magából egy előre beállított bitsorozatot.
ciklus		Ismétlődő tevékenység.
kijelző		A bemenetére érkező digitális információ tartalmát jeleníti meg.
vezeték		Két digitális áramköri elem között fizikai összeköttetést biztosít.

1.5 Essential use-case-ek

1.5.1 Use-case diagram



1.5.2 Use-case leírások

Use-case neve	<i>load</i>
Rövid leírás	A felhasználó betölt egy előre elkészített áramkört.
Aktorok	User
Forgatókönyv	Egy browser segítségével kiválasztja a felhasználó a megnyitni kívánt áramkör fájlját.

Use-case neve	<i>set value</i>
Rövid leírás	A felhasználó beállítja a kapcsolók, illetve a jelgenerátor értékeit.
Aktorok	User
Forgatókönyv	A felhasználó a beállítható elemtől függően, és annak beállításának a forgatókönyve szerint beállítja, hogy a futásnál milyen értékekkel dolgozzon az áramkör.

Use-case neve	<i>set generator</i>
Rövid leírás	A jelgenerátor beállítása.
Aktorok	User
Forgatókönyv	A jelgenerátor menüjében a felhasználó beállítja azt a bitsorozatot, amit a generátor ciklikusan fog a kimenetére adni.

Use-case neve	<i>set switch</i>
Rövid leírás	A kapcsoló beállítása.
Aktorok	User
Forgatókönyv	A felhasználó beállítja a kapcsoló állapotát (Be/Ki).

Use-case neve	<i>run</i>
Rövid leírás	Az áramkör működési szimulációjának indítása.
Aktorok	User
Forgatókönyv	A felhasználó elindítja az áramkör szimulációját, mely beállított értékek szerint reprezentálja a hálózat működését, folyamatosan frissítve az állapotokat, értékeket.

Use-case neve	<i>debug (step)</i>
Rövid leírás	Egy óraciklus változásai.
Aktorok	User
Forgatókönyv	Egy ciklus alatt végbemenő változások kiértékelése, új logikai értékek kiszámítása, illetve megjelenítése (a hálózat frissítése).

Use-case neve	<i>stop</i>
Rövid leírás	A szimuláció leállítása.
Aktorok	User
Forgatókönyv	A felhasználó leállítja a szimuláció futását, ekkor az áramkör alapértelmezett állapotba kerül.

Use-case neve	<i>pause</i>
Rövid leírás	A szimuláció szüneteltetése.
Aktorok	User
Forgatókönyv	A felhasználó megállítja a szimuláció futását, és egy pillanatképet kap az áramkör állapotairól.

Use-case neve	<i>check value</i>
Rövid leírás	Értékek ellenőrzése.
Aktorok	User
Forgatókönyv	A felhasználó lekérdezi az áramkör bármely részén futó jelnek a logikai értékét.

1.6 Napló

Kezdés	Vége	Munka megnevezése	Résztvevők	Időtartam (óra:perc)
2011. 2. 18. 12:15	2011. 2. 18. 12:55	Use-case diagramok szerkesztése	Csomák, Jéghe, Sziklay, Vad, Wiesner	0:40
2011. 2. 18. 13:00	2011. 2. 18. 14:30	Követelmények, projektterv megfogalmazása	Csomák, Jéghe, Sziklay, Vad, Wiesner	1:30
2011. 2. 18. 14:30	2011. 2. 18. 15:00	Szótár készítése	Csomák, Jéghe, Sziklay, Vad, Wiesner	0:30
2011. 2. 19. 15:00	2011. 2. 19. 15:40	Use-case leírás elkészítése	Sziklay	0:40
2011. 2. 19. 14:00	2011. 2. 19. 16:00	Feladatleírás	Jéghe	2:00
2011. 2. 19. 23:20	2011. 2. 20. 0:30	Szótár bővítése	Jéghe, Vad	1:10
2011. 2. 20. 0:40	2011. 2. 20. 1:20	Feladatleírás bővítése	Jéghe, Wiesner	0:40
2011. 2. 20. 2:00	2011. 2. 20. 3:00	Kockázatelemzés	Vad	1:00
2011. 2. 20. 23:50	2011. 2. 21. 0:10	Dokumentumjavítás	Wiesner	0:20

2. Analízis modell kidolgozása

2.1 Objektum katalógus

2.1.1 DigitalBoard

A digitális áramkört nyilvántartó és a vezérlést biztosító objektum. Az áramkör összes elemét és a köztük lévő kapcsolatokat létrehozza, kezeli és tárolja. Új áramkör megnyitásakor betölti az áramkört egy fájlból – eközben ellenőrzi a szintaktikai helyességét –, létrehoz minden digitális elemet, hierarchia szerint sorrendezi, felfedezi a visszacsatolásokat. További feladata az áramkör számításait vezérelni, így ha a felhasználó léptetést kér, a generátorokat lépteti, és újraszámolja az áramkör komponenseinek állapotát. Ha a felhasználó a futtatást választja, változtatható időközönként lépteti a jelgenerátort, és a komponensek értékeit újraszámolja.

2.1.2 Wire

A DigitalObject típusú elemeket összekötő objektum. Egy bemenettel és tetszőleges számú kimenettel rendelkezik. Az értékét a bemenetén lévő objektum állítja be, melyet továbbít minden kimenetére. Az érték mellett nyilvántartja a bemenetén és a kimenetein lévő objektumokat is. Minden Wire típusú objektum egyedi azonosítóval rendelkezik.

2.1.3 DigitalObject

A Wire objektum mellett ez a másik legfontosabb építőeleme a digitális áramköröknek. Meghatározzák, megjelenítik, vagy egy belső függvény szerint a kimenetükre csatlakozó Wire típusú objektumok értékeit megváltoztatják. Minden DigitalObject típusú objektum egyedi azonosítóval rendelkezik.

2.1.4 Input

Bemenet nélküli áramkörü elem, melynek kimenete vagy adott időközönként, vagy felhasználói interakció során változhat meg.

2.1.5 Switch

Olyan az Input objektum, melynek értékét a felhasználó képes megváltoztatni.

2.1.6 Generator

Speciális Input objektum, mely a kimenetét ciklikusan változtatja egy, felhasználó által szerkeszthető szekvencia alapján. Az objektum nyilvántartja a szekvenciát, illetve az aktuális pozíciót a mintában. A generátor Reset utasításának hívásával a pozíció a minta elejére állítható.

2.1.7 Gate

Olyan DigitalObject objektum, mely 1 vagy több bemenettel rendelkezik, és kimenete egy belső függvény (vagy igazságtábla) szerint alakul.

2.1.8 andGate

Logikai ÉS kaput megvalósító objektum. A bemeneteiről beolvasott értékekből kiszámolja és továbbadja a kimenetére az új értéket. Az új értékét az ÉS kapu igazságtáblája szerint számolja ki, mely két bemenet esetén a következő:

bemenet		kimenet
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1
X	0	0
X	1	X
0	X	0
1	X	X
X	X	X

Jelentések: 0: logikai HAMIS érték | 1: logikai IGAZ érték | X: don't care

2.1.9 orGate

Logikai VAGY kaput megvalósító objektum. A bemeneteiről beolvasott értékekből kiszámolja és továbbadja a kimenetére az új értéket. Az új értékét az VAGY kapu igazságtáblája szerint számolja ki, mely két bemenet esetén a következő:

bemenet		kimenet
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1
X	0	X
X	1	1
0	X	X
1	X	1
X	X	X

2.1.10 Inverter

Logikai invertert megvalósító objektum. Egyetlen egy bemenettel rendelkezik, és kimenetére ennek inverzét adja, ha annak van értelme. Igazságtáblája a következő:

bemenet	kimenet
A	NOT A
0	1
1	0
X	X

2.1.11 Output

Minden olyan DigitalObject objektum, amely a bemenet kiírására vagy megjelenítésére szolgál, például LED.

2.1.12 LED

A LED ösosztálya, az Output, definíciója szerint működik, bemenete értékét le lehet lekérdezni.

2.1 Osztályok leírása

2.1.1 DigitalBoard

- **Felelősség**

Olyan osztály, amelyben megjelennek a felhasználói interakcióhoz szükséges attribútumok és metódusok. Továbbá ez hozza létre és ez tárolja az áramkör objektumait.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**

Status SimStatus: Háromállapotú változó, amely a szimuláció aktuális állapotát tárolja. Lehetséges értékei: RUNNING, PAUSED, STOPPED.

- **Metódusok**

void setStatus(Status status): Feladata a belső Status változó értékének beállítása.

void Run(): A szimulációt indítja úgy, hogy elsőnek beállítja a státuszát RUNNING értékre, majd meghívja a StepComponents() metódust.

void Pause(): Lehetőséget ad a szimuláció lépésenkénti végrehajtására. Lehetővé teszi, hogy a felhasználó hívja meg a StepComponents() függvényt. Meghívja a setStatus(PAUSED) metódust.

void Stop(): A szimulációt megállítja. Az összes DigitalObject és Wire típusú objektum értékét az eredeti, betöltéskori értékre állítja vissza. Meghívja a setStatus(STOPPED) metódust.

void StepComponents(): Meghívja az összes iComponent interfészt megvalósító objektum Step() metódusát.

void LoadBoard(String strFilePath): A megfelelő paraméterrel meghívja a ParseFile(String strFilePath) metódust.

void ParseFile(String strFilePath): A megadott útvonalon található fájlt olvassa be és soronként értelmezi az állományt. A beolvasott sorban lévő parancsoktól függően hozza létre a Wire, DigitalObject típusú objektumokat, ezeket hozzáadja a megfelelő listákhoz, esetleg összeköt két már beolvasott DigitalObject objektumot a megfelelő Wire SetConnecion() függvényének meghívásával. A fájlban jelezve van, ha egy létrehozandó elem egy visszacsatolás része, ekkor a korábban létrehozott elemet hozzáadja az adott DigitalObject Feedbacks tömbjéhez.

2.1.2 Wire

- **Felelősség**

Az áramkörben található értékek tárolására szolgáló objektum. DigitalObject típusú objektumok között teremt kapcsolatot.

- **Ősosztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**

1. **String ID:** Egyedi karakteres azonosító, mely egyértelműen meghatároz egy Wire objektumot.
2. **int Value:** A Wire objektumok által tárolt érték. A Gate objektumok ezek alapján számolják ki kimenetüket.
3. **List <DigitalObject> wireIn:** DigitalObject objektum-referencia, amely a vezeték bemenetéhez kapcsolódik.
4. **List <DigitalObject> wireOut:** DigitalObject objektum-referenciákból álló lista azon objektumokról, melyek a vezeték kimeneteihez kapcsolódnak.

- **Metódusok**

5. **String GetID():** A Wire egyedi azonosítójának lekérdezésére szolgáló metódus.
6. **int GetValue():** A Wire objektum értékének lekérdezésére szolgáló metódus.
7. **void SetValue(int value):** A Wire objektum értékének módosítására szolgáló metódus.
8. **void SetConnection(DigitalObject, DigitalObject):** Kapcsolatot teremt két DigitalObject között. Beállítja a vezeték inputját, illetve hozzáadja az output tömbjéhez a DigitalObjectet.

2.1.3 DigitalObject

- **Felelősség**

Absztrakt osztály, amelyből az összes nem Wire típusú objektumot származtatjuk. Legnagyobb felelőssége az iComponent interfész megvalósítása, továbbá nyilvántartja a bemeneteit és kimeneteit.

- **Össztályok**

Nincs

- **Interfészek**

Nincs

- **Attribútumok**

9. **String ID:** Egyedi karakteres azonosító, mely egyértelműen meghatároz egy DigitalObject objektumot.
10. **List<Wire> wireIn:** Wire objektum-referenciákból álló lista azon Wire objektumokról, melyek az objektum bemeneteihez kapcsolódnak.
11. **List<Wire> wireOut:** Wire objektum-referenciákból álló lista azon Wire objektumokról, melyek az objektum kimeneteihez kapcsolódnak.

- **Metódusok**

1.2 void Count()

1.3 boolean Step()

1.4 String GetID()

Felelősségük megegyezik a 4.2.2 fejezetben leírtakkal.

1.5 **String GetMyType():** Visszaadja a DigitalObjectet megvalósító elem osztályát.

2.1.4 Input

- **Felelősség**

Absztrakt osztály a beviteli modulok (pl. jelgenerátor, kapcsoló) leszármaztatására.

- **Ősosztályok**

DigitalObject

- **Interfészek**

DigitalObject-en keresztül megvalósítja az iComponent interfészt.

- **Attribútumok**

12. **int Value**: Az adott Input objektum értékét tárolja.

- **Metódusok**

Az őosztály metódusain kívül nincs más szolgáltatása.

2.1.5 Switch

- **Felelősség**

Feladata minden egyes Step() hívásakor a kimenetéhez csatlakozó Wire objektumoknak értéket adni. A felhasználó a Toggle() metódus segítségével képes egy ilyen objektum értékének megváltoztatására.

- **Ősosztályok**

DigitalObject->Input

- **Interfészek**

DigitalObject-en keresztül megvalósítja az iComponent interfészt.

- **Attribútumok**

Az őosztályok attribútumain kívül nincs más tagváltozója.

- **Metódusok**

void Toggle(): A Value változó értékét állítja nullából egybe, egyből nullába.

2.1.6 Generator

- **Felelősség**

Speciális Input objektum, mely adott frekvencián változtatja a kimenetét. Feladata – adott idő elteltével – a kimenetéhez csatlakozó Wire objektumok értékének be-, ill. átállítása.

- **Ősosztályok**

DigitalObject->Input

- **Interfészek**

DigitalObject-en keresztül megvalósítja az iComponent interfészt.

- **Attribútumok**

int Frequency: A generátor léptetésének a gyakoriságát tároljuk ebben. Az itt megadott számú Step() hívás után fog csak lépni.

int Sequence: Ebben tároljuk a kimenetre kiküldendő mintát. Értelmezése bináris.

int SequencePos: Az attribútum tartja nyilván az aktuális pozíciót a szekvenciában. A Reset() hívás 0 értékre állítja.

- **Metódusok**

void Reset(): A SequencePos. értékét állítja alapértelmezettre, azaz a minta elejére.

void SetFrequency(int frequency): a Frequency értékét állítja be, a paraméterben megadott értékre.

void SetSequence(int sequence): a Sequence, azaz a minta értékét állítja be, a paraméterben megadott értékre.

2.1.7 Output

- **Felelősség**

Absztrakt osztály a megjelenítő modulok (pl. LED) leszármaztatására.

- **Ősosztályok**

DigitalObject

- **Interfészek**

DigitalObject-en keresztül megvalósítja az iComponent interfészt.

- **Attribútumok**

13. **int Value:** Ez tárolja a bemeneti Wire objektum értékét.

- **Metódusok**

Az őszosztály metódusain kívül nincs más szolgáltatása.

2.1.8 LED

- **Felelősség**

Output absztrakt osztályt valósítja meg. Lekérdezi a hozzá csatolt vezeték (wireIn) értékét, és megjeleníti azt a felhasználó számára.

- **Ősosztályok**

DigitalObject->Output

- **Interfészek**

DigitalObject-en keresztül megvalósítja az iComponent interfészt.

- **Attribútumok**

Az őszosztályok attribútumain kívül nincs más tagváltozója.

- **Metódusok**

Az őszosztály metódusain kívül nincs más szolgáltatása.

2.1.9 Gate

- **Felelősség**

Absztrakt osztály a logikai kapuk (pl. ÉS, VAGY, Inverter) leszármaztatására. A bemeneti vezeték(ek)nek lekérdezi az értékét, és kiszámolja a kimenet új értékét, és be is állítja azt, az leszármazott osztály igazságtáblája szerint.

- **Ősosztályok**

DigitalObject

- **Interfészek**

DigitalObject-en keresztül megvalósítja az iComponent interfészt.

- **Attribútumok**

List<DigitalObject> Feedbacks: Ha egy Gate egyik bemenete egy visszacsatolás kezdete, akkor tartalmaz egy Feedbacks tömböt, mely referenciát tárol az összes, az adott visszacsatolásban résztvevő DigitalObject-re.

int PreviousValue: A legutolsó ciklus (Step()) eredményét (kimeneti értékét) tárolja, a stabilitásellenőrzés céljából (Count() metódus).

- **Metódusok**

AddToFeedBacks: Hozzáadja a paraméterként kapott DigitalObjectet a Feedbacks tömbhöz.

AddOutput: A paraméterként kapott vezeték (Wire) a kimenetre köti.

2.1.10 andGate

- **Felelősség**

Gate absztrakt osztályt valósítja meg, bemeneti értékeit lekérdezi (wireIn) a GetValue() metódussal, majd előállítja a kimeneti értéket (wireOut) a logika táblája alapján és beállítja azt, a SetValue(int) segítségével.

- **Ősosztályok**

DigitalObject->Gate

- **Interfészek**

DigitalObject-en keresztül megvalósítja az iComponent interfészt.

- **Attribútumok**

Az őszosztályok attribútumain kívül nincs más tagváltozója.

- **Metódusok**

Az őszosztály metódusain kívül nincs más szolgáltatása.

2.1.11 orGate

- **Felelősség**

Gate absztrakt osztályt valósítja meg, bemeneti értékeiből (wireIn) előállítja a kimeneti értéket (wireOut) a logika táblája alapján.

- **Ősosztályok**

DigitalObject->Gate

- **Interfészek**

DigitalObject-en keresztül megvalósítja az iComponent interfészt.

- **Attribútumok**

Az őssosztályok attribútumain kívül nincs más tagváltozója.

- **Metódusok**

Az őssosztály metódusain kívül nincs más szolgáltatása.

2.1.12 Inverter

- **Felelősség**

Gate absztrakt osztályt valósítja meg, bemeneti értékét (wireIn) lekérdezi a GetValue() metódussal, és előállítja a kimeneti értéket (wireOut) a logika igazságtáblája alapján, melyet a kimenetére ad a SetValue(int) függvénnnyel. Csak egyetlen bemenete lehet.

- **Ősosztályok**

DigitalObject->Gate

- **Interfészek**

DigitalObject-en keresztül megvalósítja az iComponent interfészt.

- **Attribútumok**

Az őssosztályok attribútumain kívül nincs más tagváltozója.

- **Metódusok**

Az őssosztály metódusain kívül nincs más szolgáltatása.

4.2.14 Oscilloscope

- **Felelősség**

Output absztrakt osztályt valósítja meg. Egy bemenettel rendelkezik, melynek értékét a felhasználó által megadott ideig (lépésszámon keresztül) megjeleníti.

- **Őszosztályok**

Output

- **Interfészek**

Nincs

- **Attribútumok**

- **int Sample:** Itt tárolja a bemenetére érkező jelek értékét.
- **int SampleSize:** Meghatározza, hogy az időben később beérkezett jeleket meddig tárolja el az Oscilloscope (azaz a buffer mérete). A tárolás FIFO elv szerint működik.

- **Metódusok**

- **void SetSampleSize(int):** A metódus beállítja a SampleSize attribútum értékét.

4.2.15 Composit

- **Felelősség**

Legfontosabb feladata a kompozitot felépítő elemek tárolása, és azok irányítása.

- **Őszosztályok**

DigitalObject

- **Interfészek**

Nincs

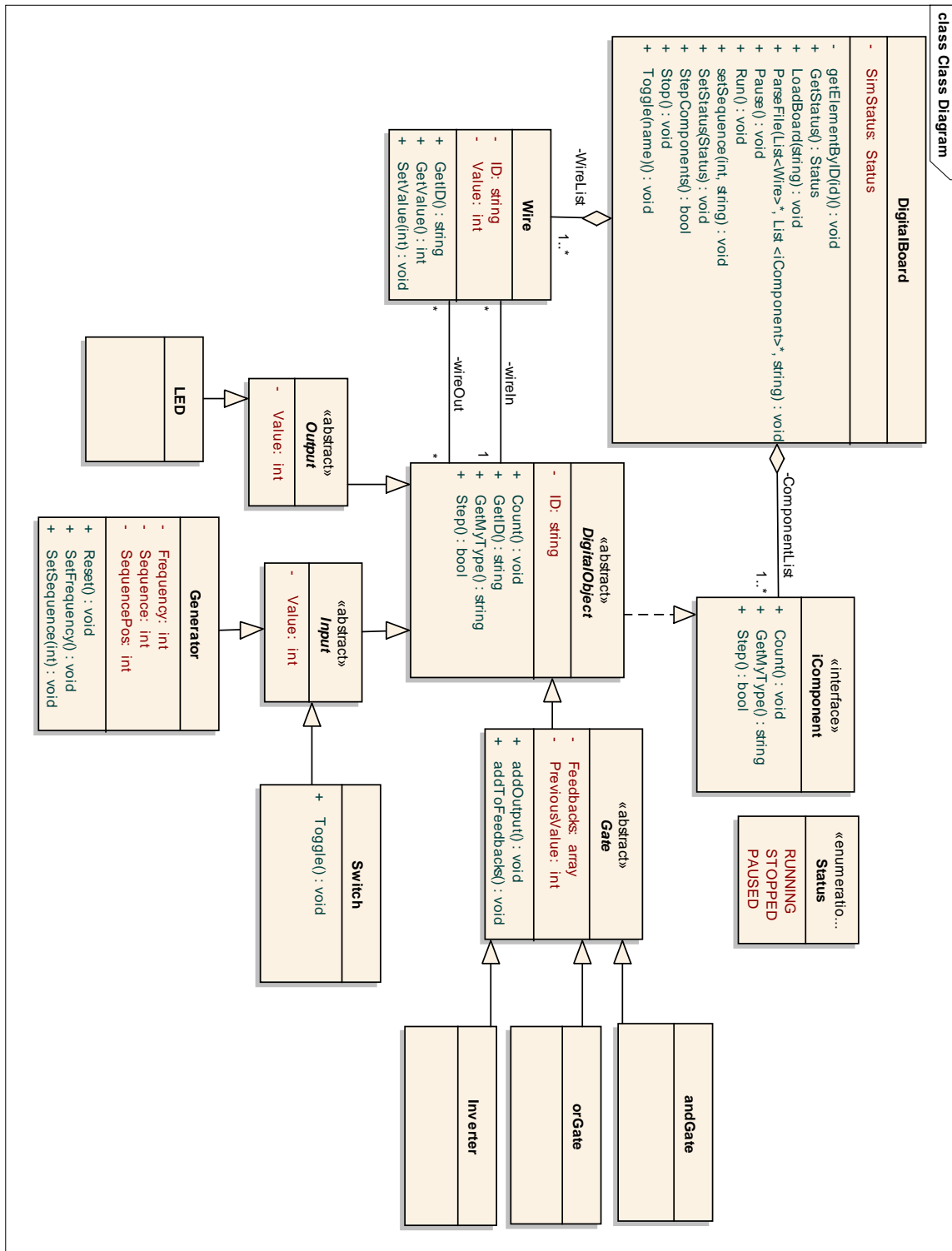
- **Attribútumok**

- **List<DigitalObject> ComponentList:** A kompozitot alkotó elemek listája.
- **List<Wire> WireList:** Lista a Wire objektumok tárolására.

- **Metódusok**

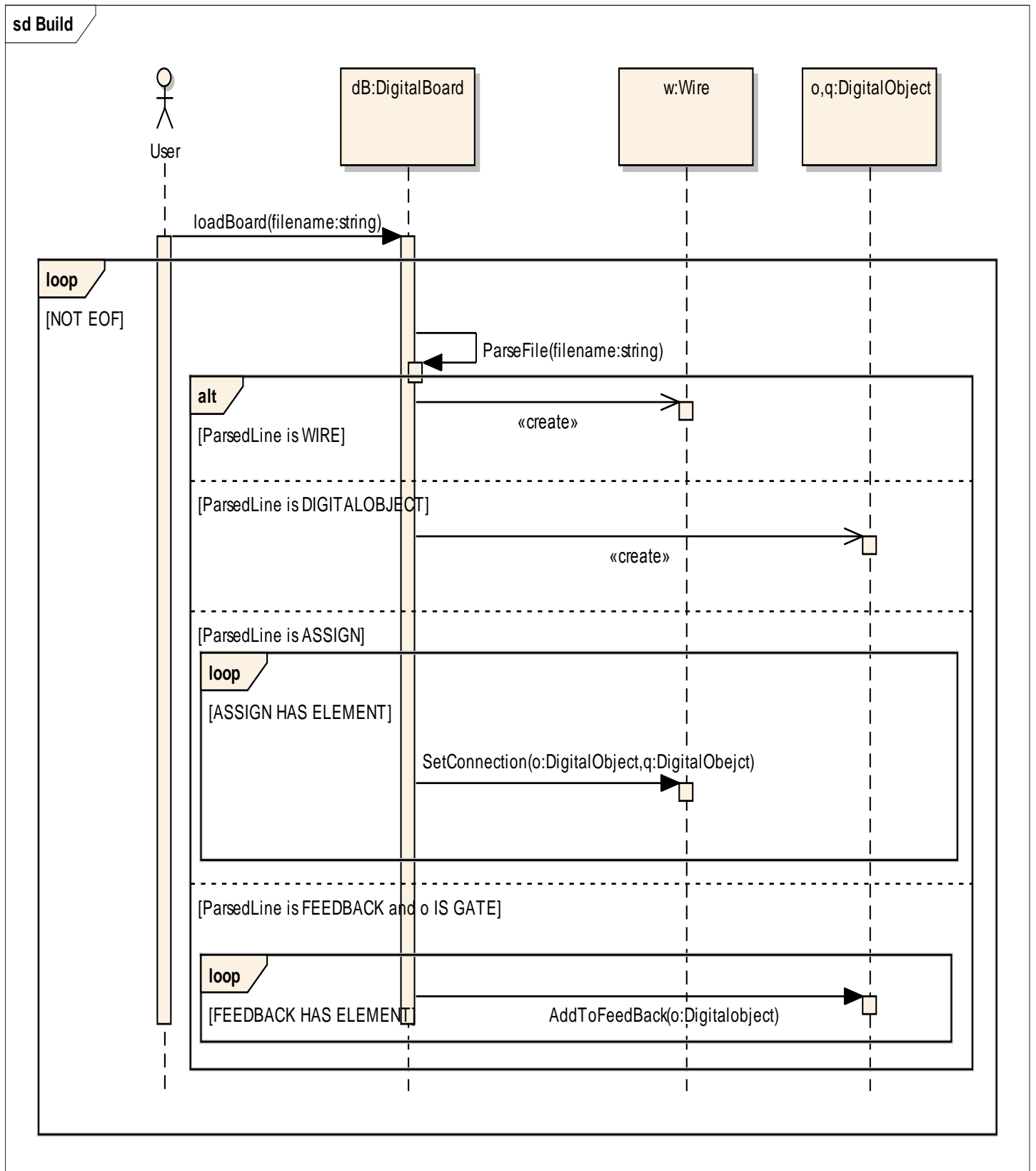
- **DigitalObject GetElementByID(String):** Feladata a paraméterben megkapott egyedi azonosítóval ellátott objektumot visszaadni.
- **void StepComponents():** Meghívja az összes DigitalObject osztályt megvalósító objektum Step() metódusát.
- **void SetFrequency(int, String):** Beállítja paraméterben megadott azonosítóval rendelkező jelgenerátor frekvenciájának az értékét a paraméterben kapott értékkel megegyezőre.
- **void SetSequence(int, String):** Beállítja a paraméterben megadott azonosítóval rendelkező jelgenerátor szekvenciáját.
- **void Toggle(String):** A paraméterként kapott azonosítóval megegyező kapcsoló értékét megváltoztatja logikai igazról hamisra, hamisról igazra.
- **void LoadComposit():** Feltölti a ComponentList és a WireList listát a kompozitot alkotó elemekkel, és vezetékekkel.

2.2 Statikus struktúra diagramok

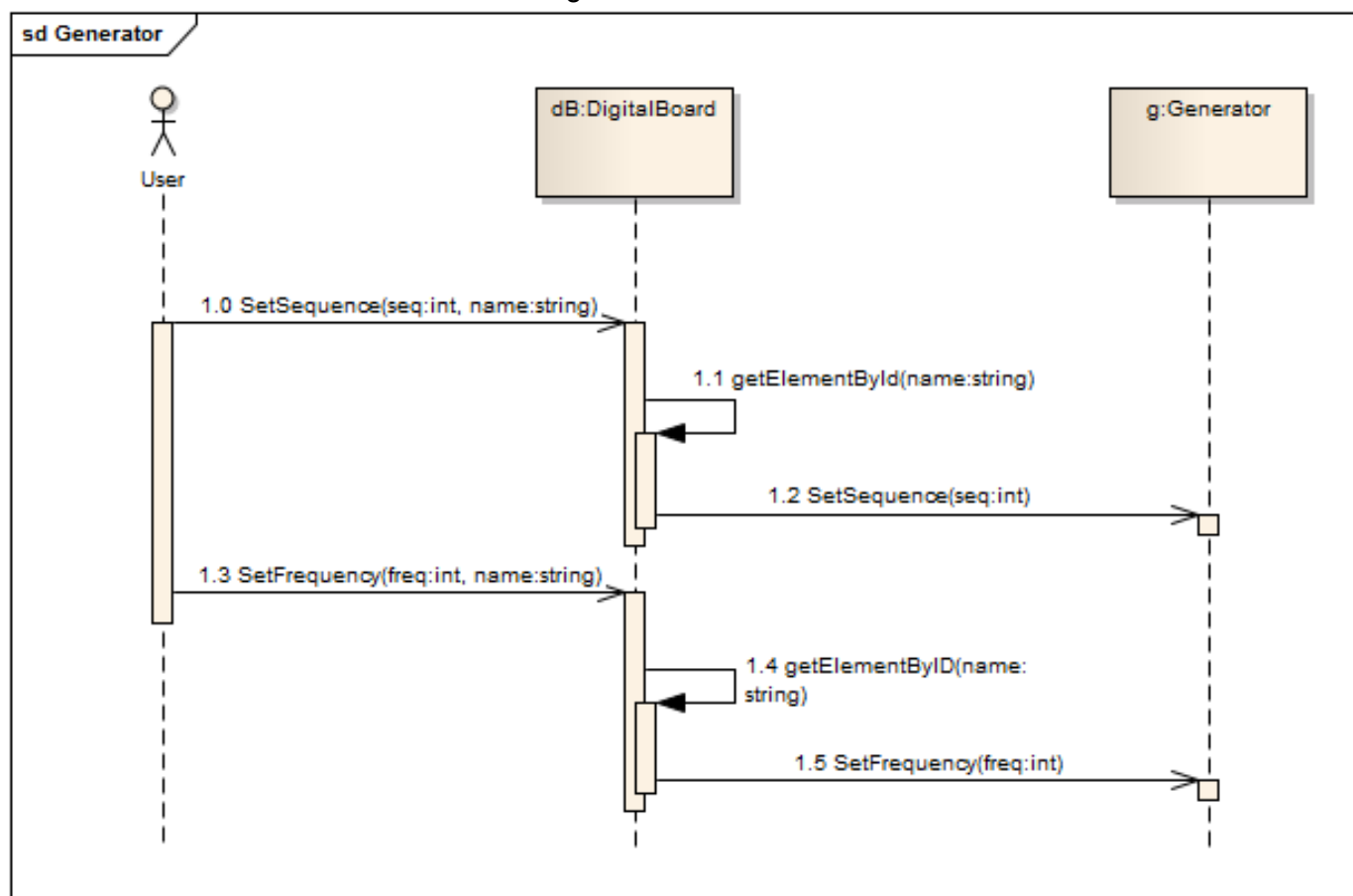


2.3 Szekvencia diagramok

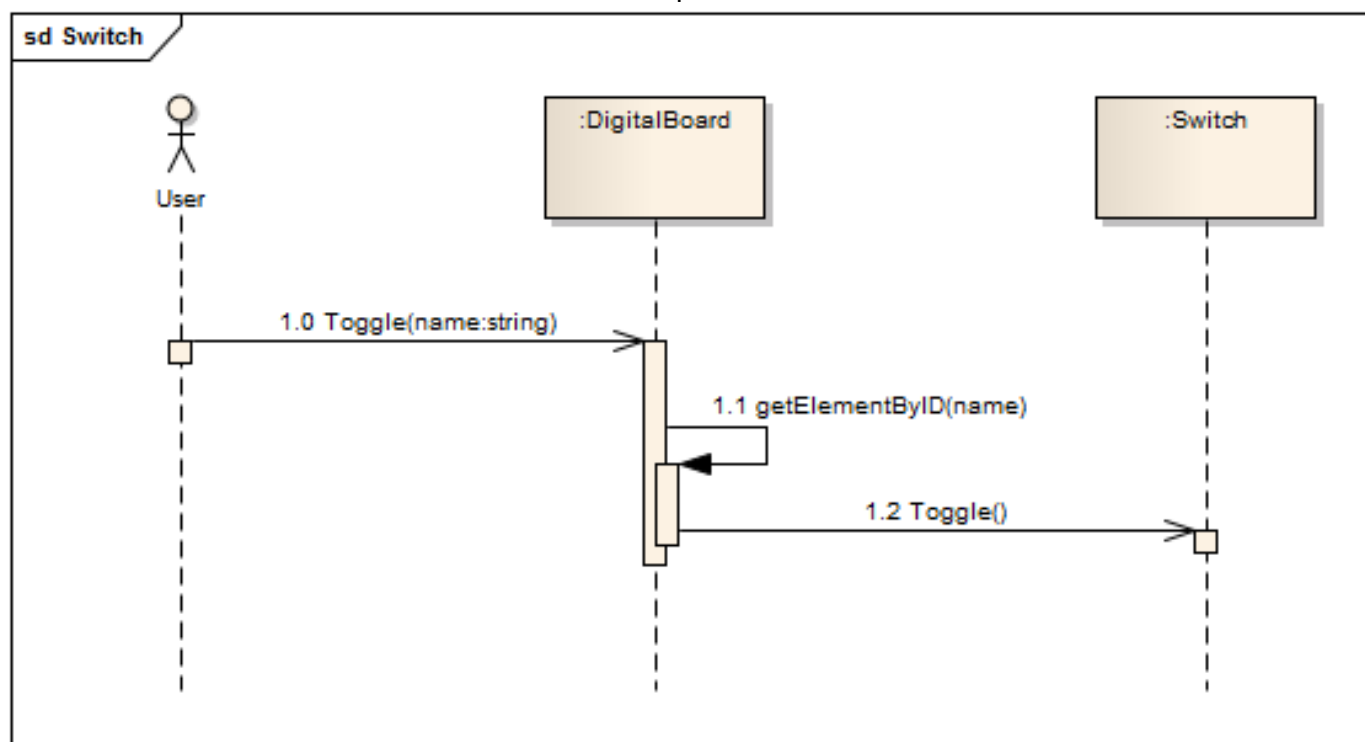
Kapcsolatok felépítése



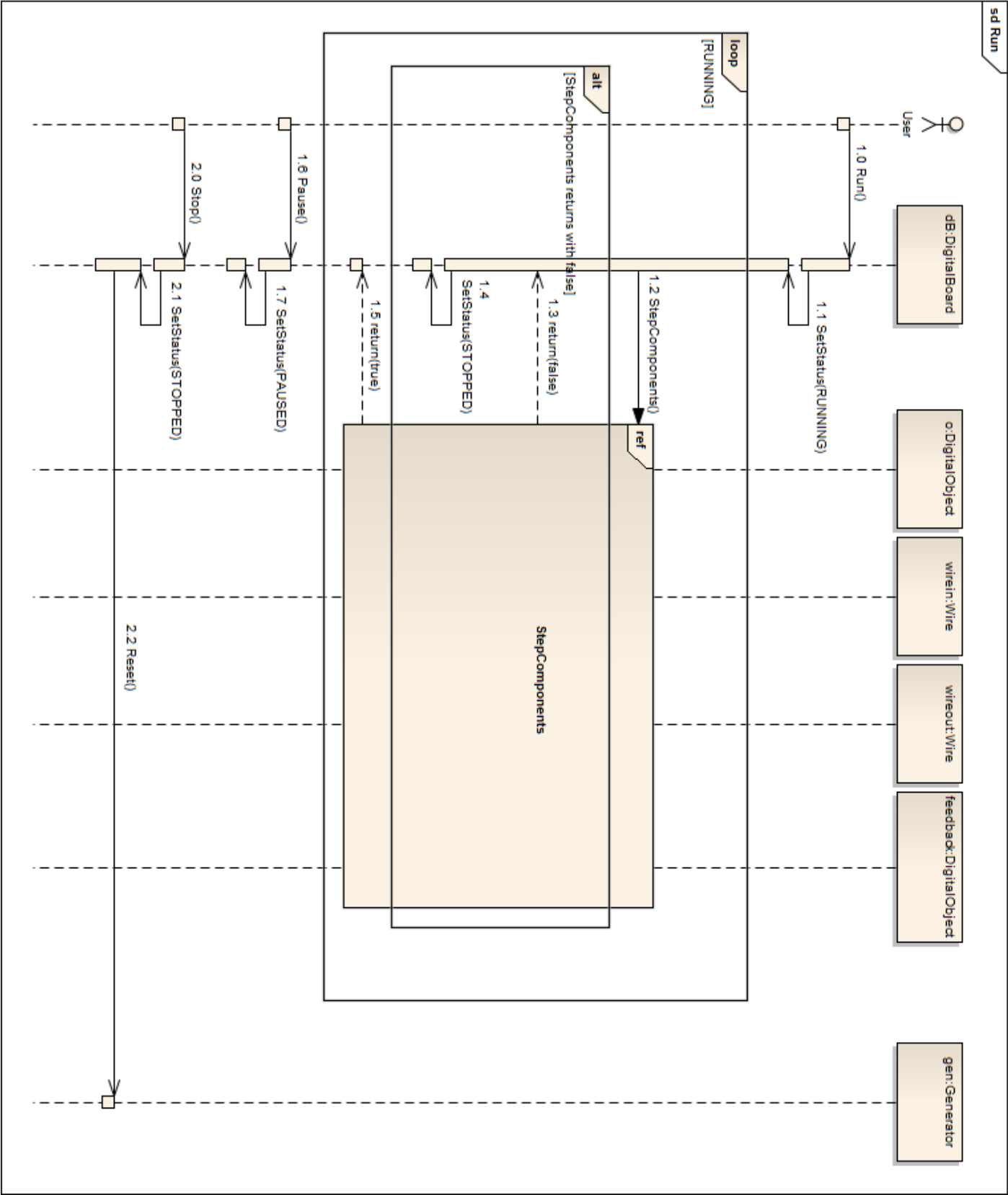
Jelgenerátor beállítása



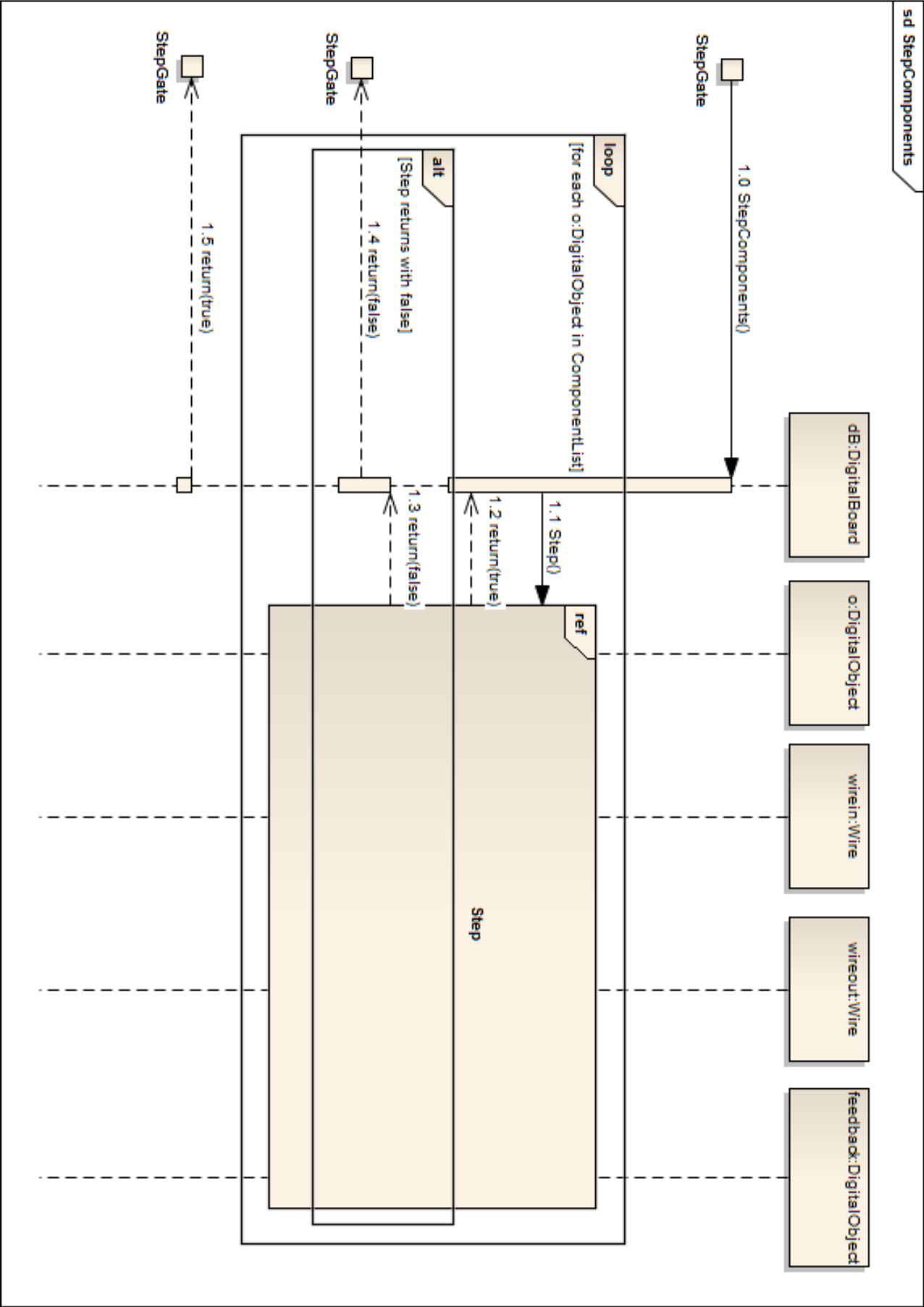
Switch kapcsolása



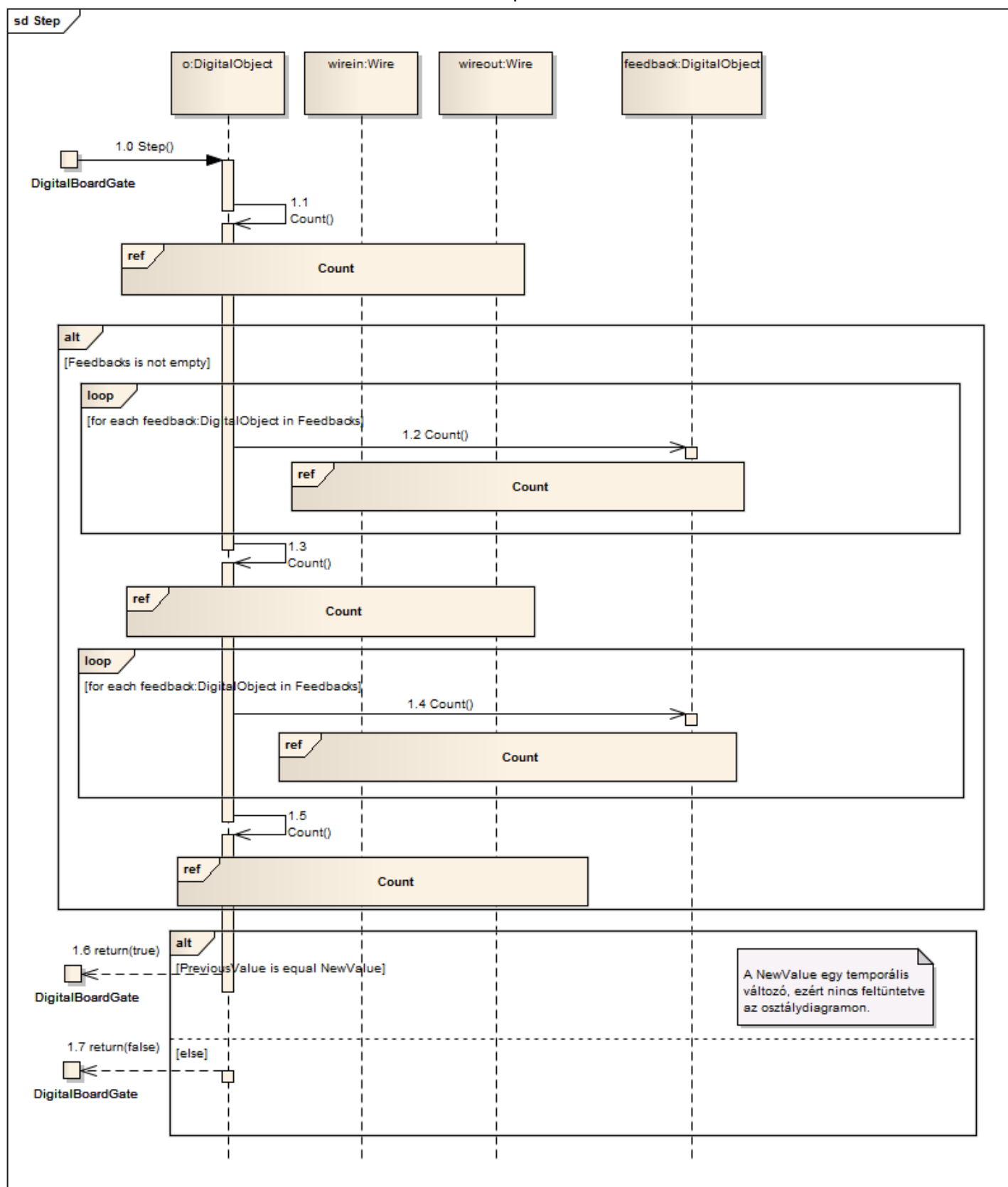
Állapotváltások



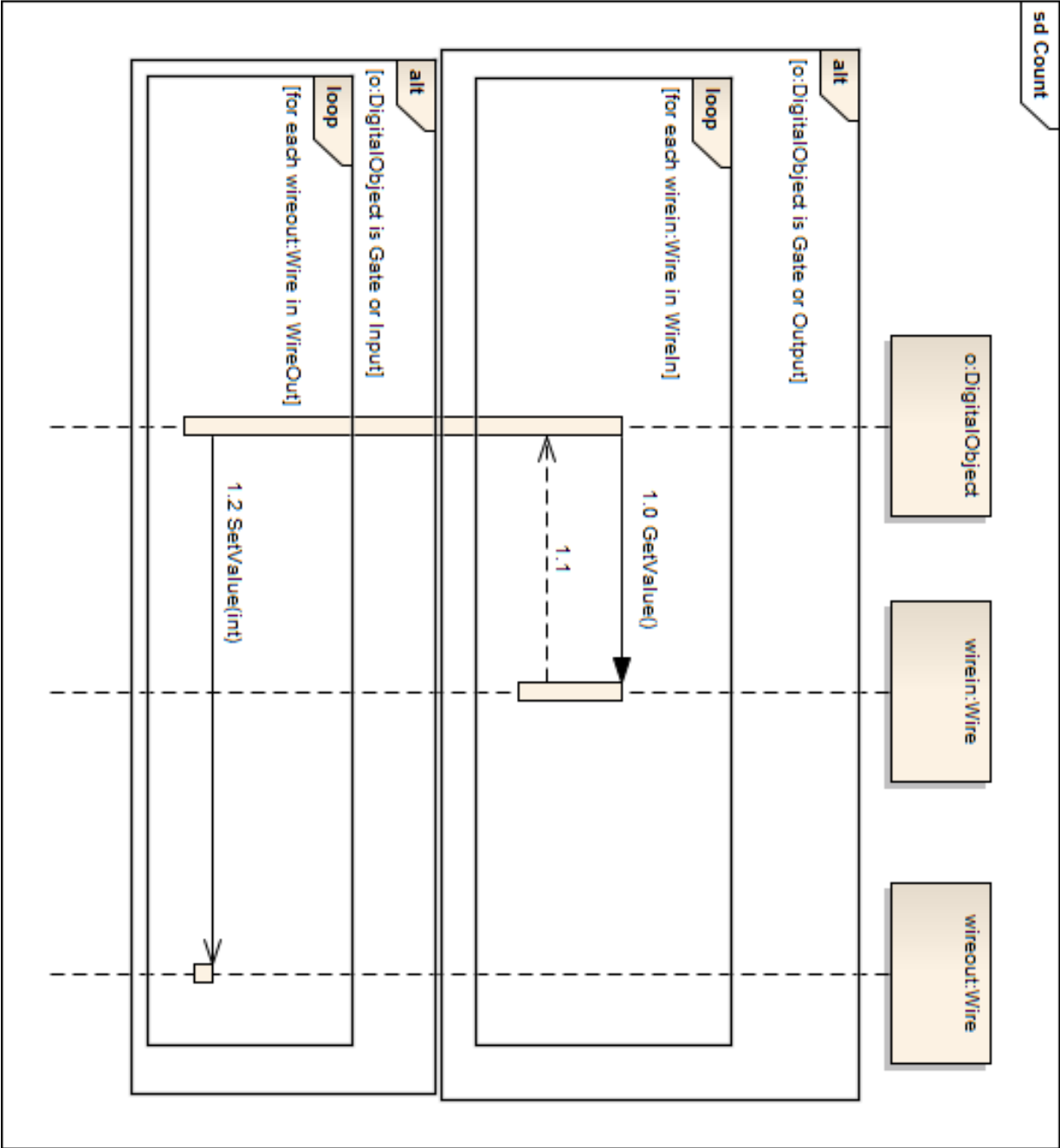
StepComponents



Step

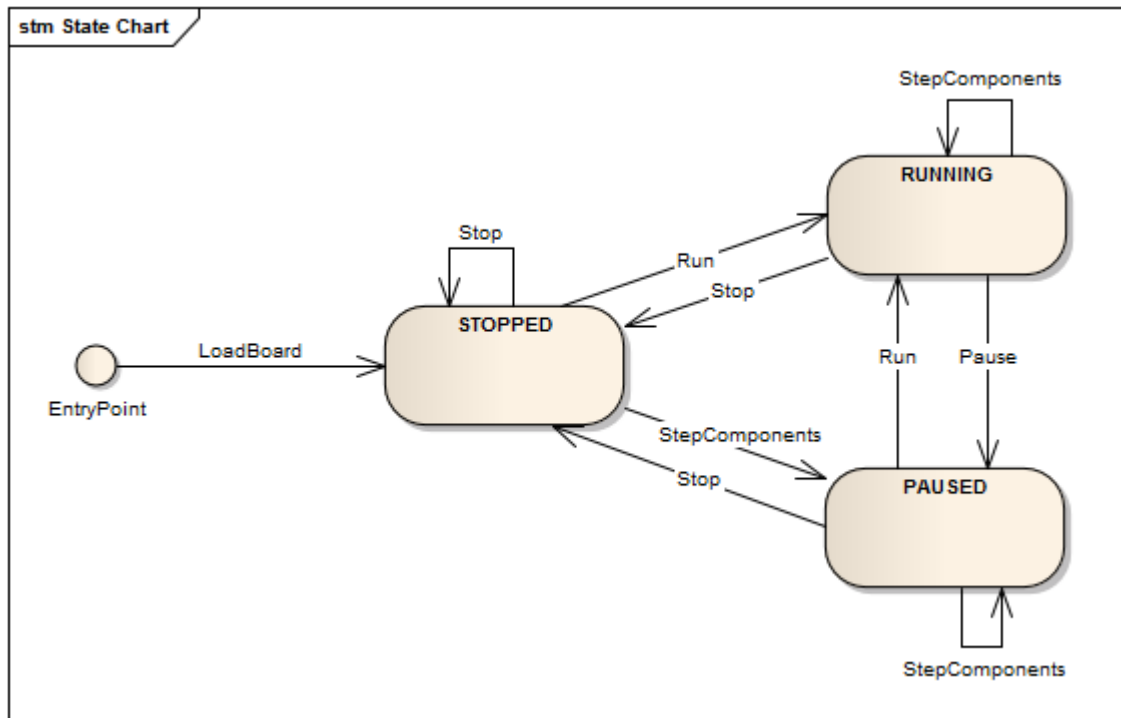


Count



2.4 State-chart

Szimuláció - állapot



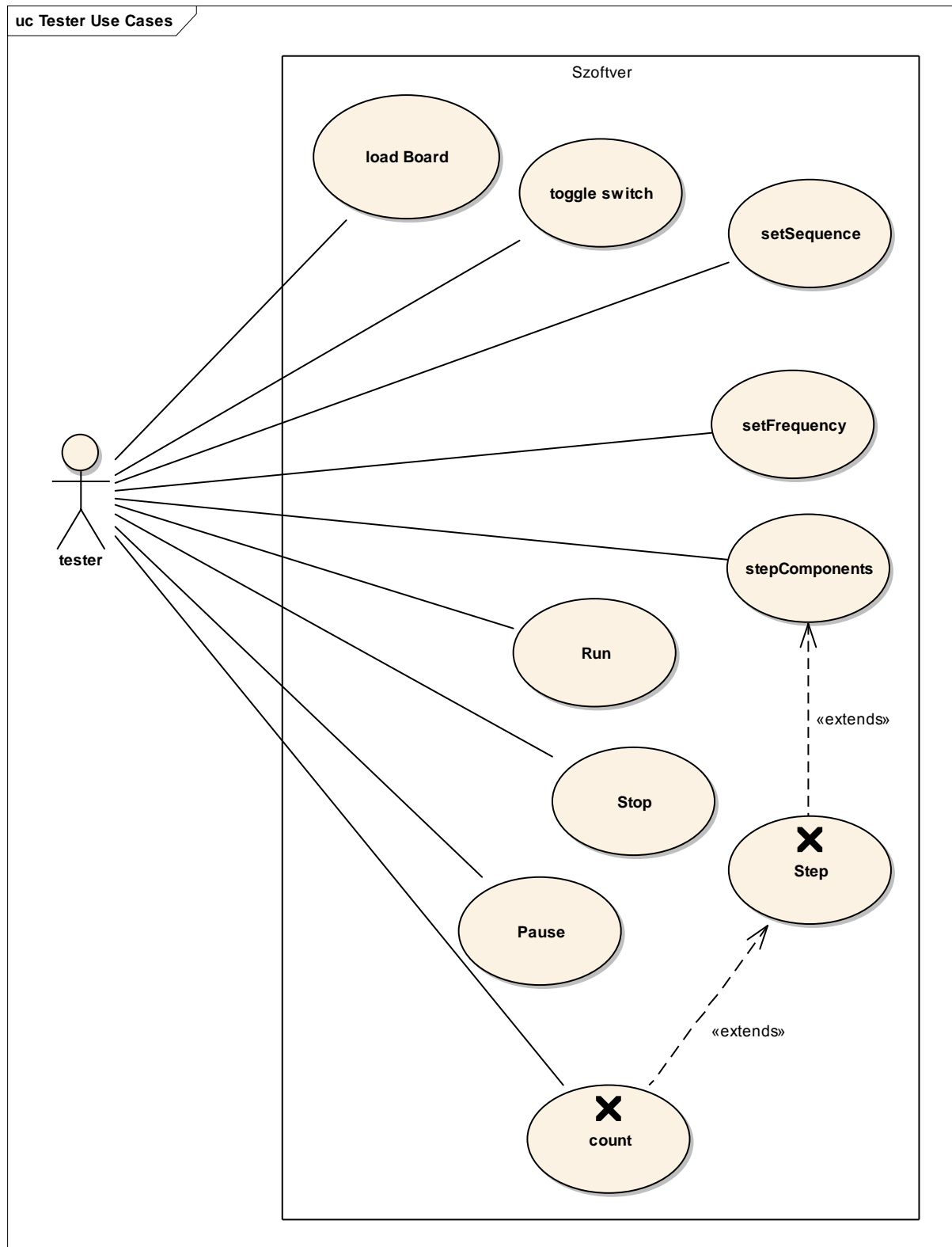
2.5 Napló

Kezdés	Vége	Megnevezés	Részrtvevők	Időtartam
03.02.2011 10:00	03.02.2011 12:00	Szekvencia diagramok módosítása, létrehozása: Generator, Step, Switch, Count	Csomák, Wiesner	02:00
03.04.2011 12:00	03.04.2011 14:00	Értekezlet Döntés: parseFile metódus átgondolása, Init szekvencia diagram átgondolása	Jégh, Wiesner, Vad	02:00
03.05.2011 22:00	03.06.2011 02:00	Szekvencia diagramok módosítása (Run, Step, StepComponents, Count)	Vad	04:00
03.06.2011 15:00	03.06.2011 17:00	Szekvenciadiagramok módosítása (Build, Run)	Wiesner	02:00
03.06.2011 21:00	03.07.2011 00:00	Dokumentáció véglegesítése	Sziklay	03:00
03.06.2011 21:00	03.07.2011 00:00	Osztálydiagram, Szekvenciadiagram módosítása (Build)	Wiesner	03:00
03.07.2011 01:30	03.07.2011 1:50	Osztályleírás javítása	Jégh	00:20:00

3. Szkeleton tervezése

3.1 A szkeleton modell valóságos use-case-ei

3.1.1 Use-case diagram



3.1.2 Use-case leírások

Use-case neve	<i>Loard Board</i>
Rövid leírás	Ez a use-case tartalmazza a teszt áramkörök betöltését.
Aktorok	Tester
Forgatókönyv	A felhasználót megkérdezzük, melyik teszt-áramkört töltsse be a program, majd betöltjük azt.

Use-case neve	<i>Toggle Switch</i>
Rövid leírás	Ez a use-case tartalmazza az áramkörben található switchek átállításának lehetőségét.
Aktorok	Tester
Forgatókönyv	Amennyiben az áramkörben található kapcsoló, megkérdezzük, melyik switchet szeretné átállítani. Amennyiben van olyan, beállítjuk a kívánt értékre.

Use-case neve	<i>setSequence</i>
Rövid leírás	Ez a use-case tartalmazza a generátorok szekvenciájának állítását.
Aktorok	Tester
Forgatókönyv	Amennyiben az áramkörben található generátor, megkérdezzük a tesztelőt, melyiket szeretné átállítani. Amennyiben létezik, megkérdezzük a beállítandó szekvenciát, és a generátornak átadjuk.

Use-case neve	<i>StepComponents</i>
Rövid leírás	Ez a use-case tartalmazza az áramkör egy lépését, azaz a generátorok értékváltását, majd a vezetékek értékeinek újraszámítását.
Aktorok	Tester
Forgatókönyv	A komponensek listájában sorrendben minden elemet felszólítunk a léptetésre, azaz generátor esetén a szekvencia következő elemét adjuk a kimenetre, belső változókkal nem rendelkező objektumoknál pedig a kimenetet frissítjük, a Count hívással. Amennyiben egy objektumnak nem üres a visszacsatolás listája, azokra meghívjuk sorrendben a Count függvényt.

Use-case neve	<i>Step</i>
Rövid leírás	Ebben a use-case-ben a tesztelő ellenőrizheti az egyes komponensek léptetésének működését.
Aktorok	Tester
Forgatókönyv	Kiírjuk a bemenetek értékét, majd léptetünk, majd kiszámoljuk a kimenetet, és azt is kiírjuk a tesztelés segítéséhez.

Use-case neve	<i>Count</i>
Rövid leírás	Ebben a use-case-ben a tesztelő ellenőrizheti az egyes

	komponensek kimenetszámításának működését.
Aktorok	Tester
Forgatókönyv	Kiírjuk a bemenetek értékét, majd kiszámítjuk a kimenetet, és azt is kiírjuk a tesztelés segítéséhez.

Use-case neve	<i>Check Value</i>
Rövid leírás	Ebben a use-case-ben a tesztelő tetszőleges komponensnek lekérdezheti az értékét.
Aktorok	Tester
Forgatókönyv	Megkérdezzük, melyik objektum értékére kíváncsi a tesztelő, majd – amennyiben van olyan elem – kiírjuk a kimenetre.

Use-case neve	<i>SetFrequency</i>
Rövid leírás	Ez a use-case tartalmazza a generátor szekvenciájának a beállítását.
Aktorok	Tester
Forgatókönyv	A felhasználó megadja a szekvencia sorozatot.

Use-case neve	<i>Run</i>
Rövid leírás	Ez a use-case tartalmazza a hálózat 'Run' állapotba kapcsolását.
Aktorok	Tester
Forgatókönyv	A felhasználó elindítja a hálózatot.

Use-case neve	<i>Stop</i>
Rövid leírás	Ez a use-case tartalmazza a hálózat 'Stop' állapotba kapcsolását.
Aktorok	Tester
Forgatókönyv	A felhasználó leállítja a hálózat futását.

Use-case neve	<i>Pause</i>
Rövid leírás	Ez a use-case tartalmazza a hálózat 'Pause' állapotba kapcsolását.
Aktorok	Tester
Forgatókönyv	A felhasználó megállítja a hálózat futását.

3.2 Architektúra

3.2.1 Leírás

A szkeleton célja a use case diagramokban megjelenő szituációk tesztelése. Ehhez egy olyan programvázat hozunk létre, amelyben a belső algoritmusok illetve adatszerkezetek még nincsenek teljesen implementálva. Létrejönnek azonban objektum példányok, és a közöttük zajló függvényhívásokat megjelenítjük a tesztelőnek. Cél, hogy az egyes függvényhívások sorrendje és módja megjelenjen, így ellenőrizve, hogy a szekvencia diagramoknak megfelelően történik-e a működés. A folyamatok egymás után, szekvenciálisan kerülnek meghívásra, tehát még nincs szükség szálakra, illetve azok ütemezésére és szinkronizálására sem. A csapat minden eset bemutatására a lehető legminimálisabb áramkör-reprezentációt igyekezett megtervezni, amely képes az objektumok közötti kommunikáció teljes értékű bemutatására.

3.2.2 Tesztáramkörök

Teszt neve	<i>Egyszerű áramkör</i>
Elemek	gen01:Generator, sw01:Switch, or01:orGate, led01:LED, w01:Wire, w02:Wire, w03:Wire
Leírás	A tesztáramkör azt mutatja meg, hogy hogyan alakul egy VAGY kapu kimenete a bemenetétől függően. A bemenetén található egy Generator és egy Switch, a kimenetét pedig egy LED-en jelenítjük meg.

Teszt neve	<i>Visszacsatolás tartalmazó, stabil áramkör</i>
Elemek	sw01:Switch, and01:andGate, led01:LED, w01:Wire, w02:Wire
Leírás	A tesztáramkör azt mutatja meg, hogy hogyan alakul egy ÉS kapu kimenete egy visszacsatolt és egy Switch-re kötött bemenetétől függően. A visszacsatolásban nem szerepel másik kapu. Az ÉS kapu kimenete a LED-en jelenik meg.

Teszt neve	<i>Visszacsatolást tartalmazó, instabil áramkör</i>
Elemek	sw01:Switch, and01:andGate, inv01:Inverter, led01:LED, w01:Wire, w02:Wire, w03:Wire
Leírás	A tesztáramkör azt mutatja meg, hogy hogyan alakul egy ÉS kapu kimenete egy visszacsatolást és egy Switch-re kötött bemenetétől függően. A visszacsatolásban szerepel egy Inverter is. Az ÉS kapu kimenete a LED-en jelenik meg.

3.3 A szkeleton kezelői felületének terve, dialógusok

A programváz célja a modell helyes működésének bizonyítása. A szkeletonban minden objektum létrejön, ám ezeknek csak az interfésze létezik, a belső működés egyiknél sincs implementálva. Ezen objektumok közötti folyamatok tesztelése történik a programban.

Minden egyes metódus meghívásakor megjeleníti a program kimenetén (System.out) az őt tartalmazó objektum típusát és azonosítóját, továbbá az eljárás paraméterlistáját a korábbi szekvencia diagramoknak megfelelően. Ezután a metódus meghívja a működéséhez szükséges további metódusokat.

Mivel az elágazások kezelésére szolgáló belső algoritmusok még nem kerülnek megírásra, az eldöntendő eseteknél a felhasználót kérdezzük meg. Ekkor a program felsorolja lehetséges választási lehetőségeket és az azok kiválasztásához szükséges azonosítókat, amelyeket a felhasználónak a billentyűzet segítségével kell a programnak átadnia. A további függvényhívás a felhasználó által bevitt adatok szerint történik.

A szkeletonnak alkalmasnak kell lennie szekvencia diagramok ellenőrzésére. Az egyszerű, karakteres képernyőkezelés biztosítja a rendszer egyszerűségét. Az átláthatóság és a könnyű tesztelhetőség érdekében a metódushívás a következő formában történik:

- Objektum neve
- Osztály neve
- Metódus neve és paraméterlistája (az esetlegesen átadott objektumok neve és típusa)
- Hívás esetén *CALLED* és a metódus neve előtt egy "→" áll; Visszatérésnél a metódus nevével "←" áll és egy *RETURNED* kulcsszó szerepel, zárójelben a visszatérési érték.

Általános

@ Felhasználói interakcióra várás (lehetőség1 – azonosító, lehetőség2 – azonosító, ...)

@ lehetőség1

O1:Osztály→O2:Osztály | MetódusNeve(oo:osztály, érték:típus,...) CALLED

O1:Osztály←O2:Osztály | MetódusNeve(oo:osztály, érték:típus,...) RETURNED
[visszatérési érték]

SetSequence()

dB:DigitalBoard→ dB:DigitalBoard | GetElemetByID(gen01:String) CALLED

dB:DigitalBoard← dB:DigitalBoard | GetElemetByID(gen01:String)
RETURNED[gen01:Generator]

dB:DigitalBoard→gen01:Generator | SetSequence(seq:int) CALLED

dB:DigitalBoard← gen01:Generator | SetSequence(seq:int) RETURNED[]

Egy Step() hívása

dB:DigitalBoard→and01:andGate | Step() CALLED

and01:andGate→a:andGate | Count() CALLED

and01:andGate→w01:Wire | GetValue() CALLED

and01:andGate←w01:Wire | GetValue() RETURNED [value:Int]

and01:andGate→w02:Wire | GetValue() CALLED

and01:andGate←w02:Wire | GetValue() RETURNED [value:Int]

and01:andGate→w03:Wire | SetValue(value: Int) CALLED

and01:andGate←w03:Wire | SetValue(value: Int) RETURNED

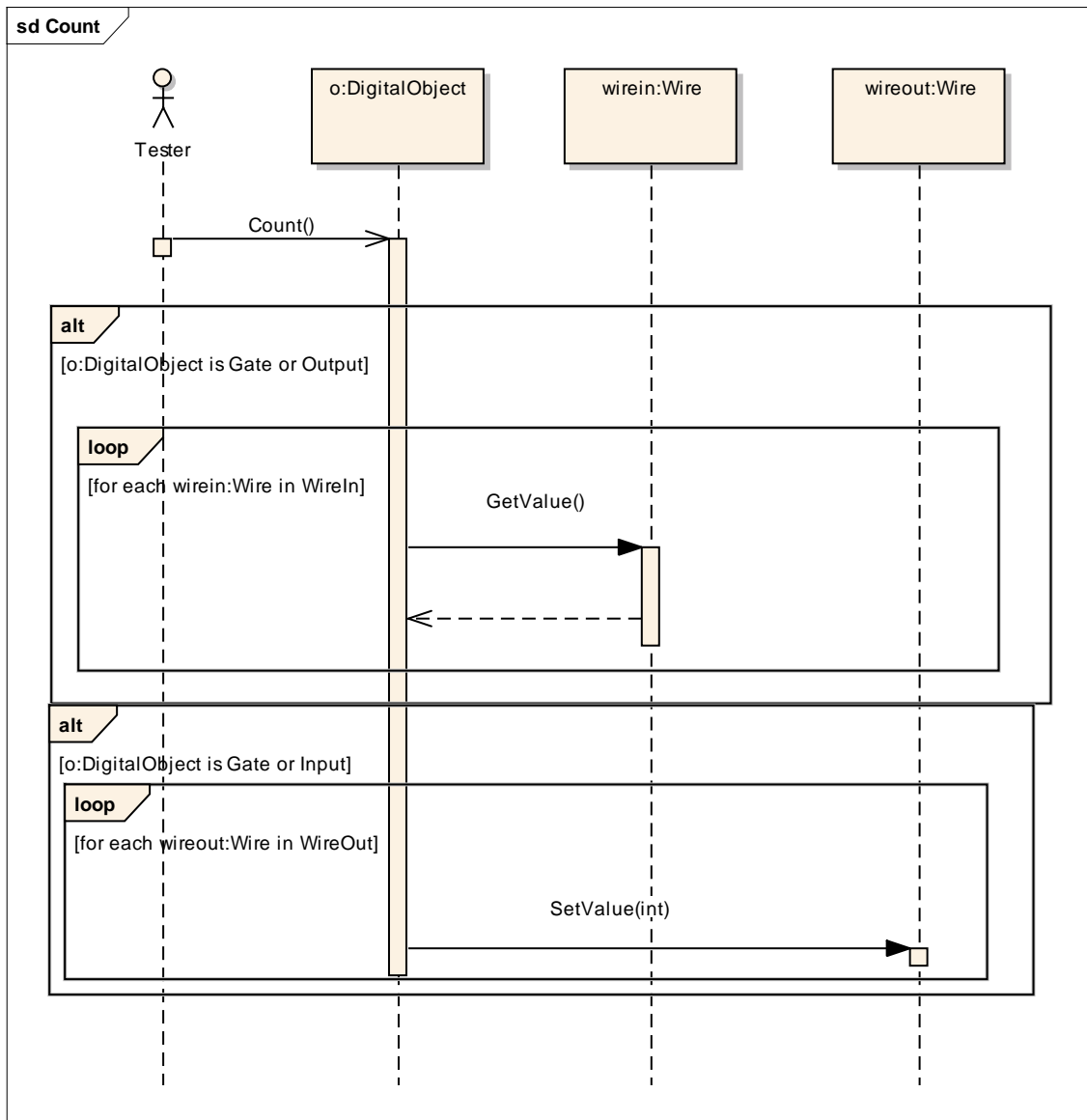
and01:andGate← and01:andGate | Count() RETURNED[]

dB:DigitalBoard← and01:andGate | Step() RETURNED

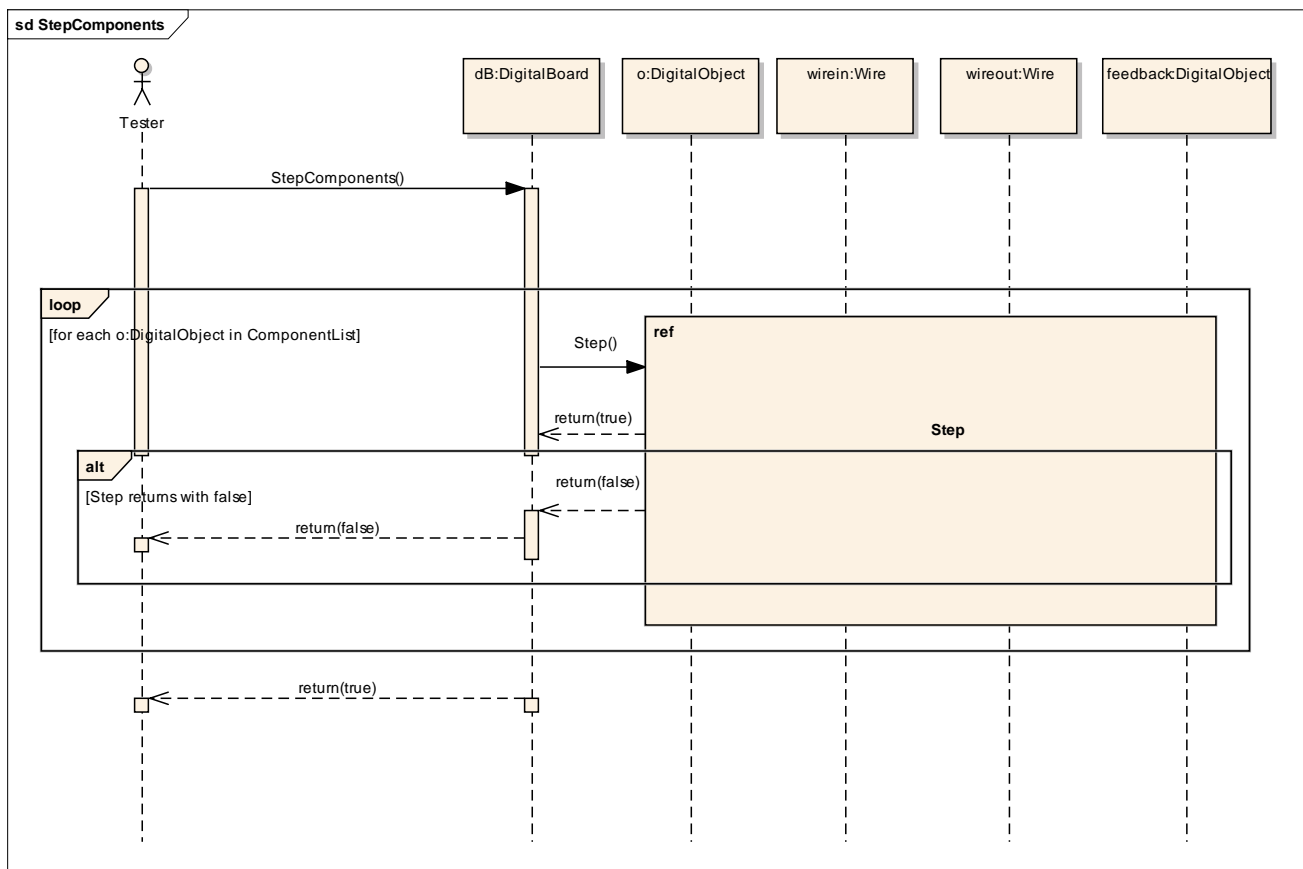
3.4 Szekvencia diagramok a belső működésre

A belső működést leíró use-case-ek szekvencia diagramjai elérhetőek a dokumentáció **4.4** fejezetében, a változtatásra szoruló diagramok új verziói alább találhatóak.

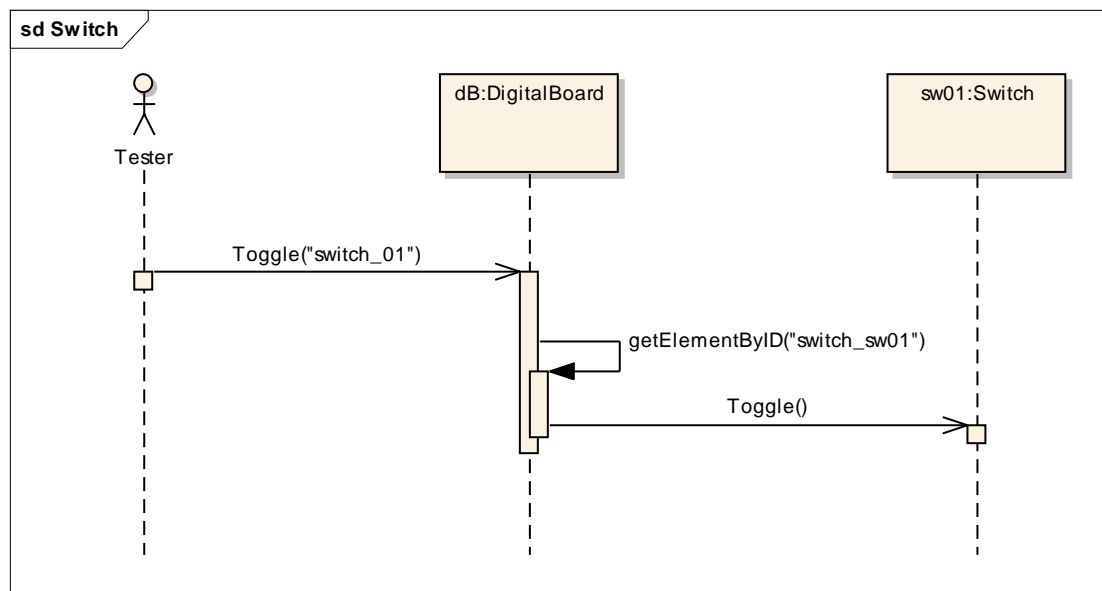
A Tesztelő meghívhatja az o:DigitalObject Count() metódusát, hogy ellenőrizze stabilitását:



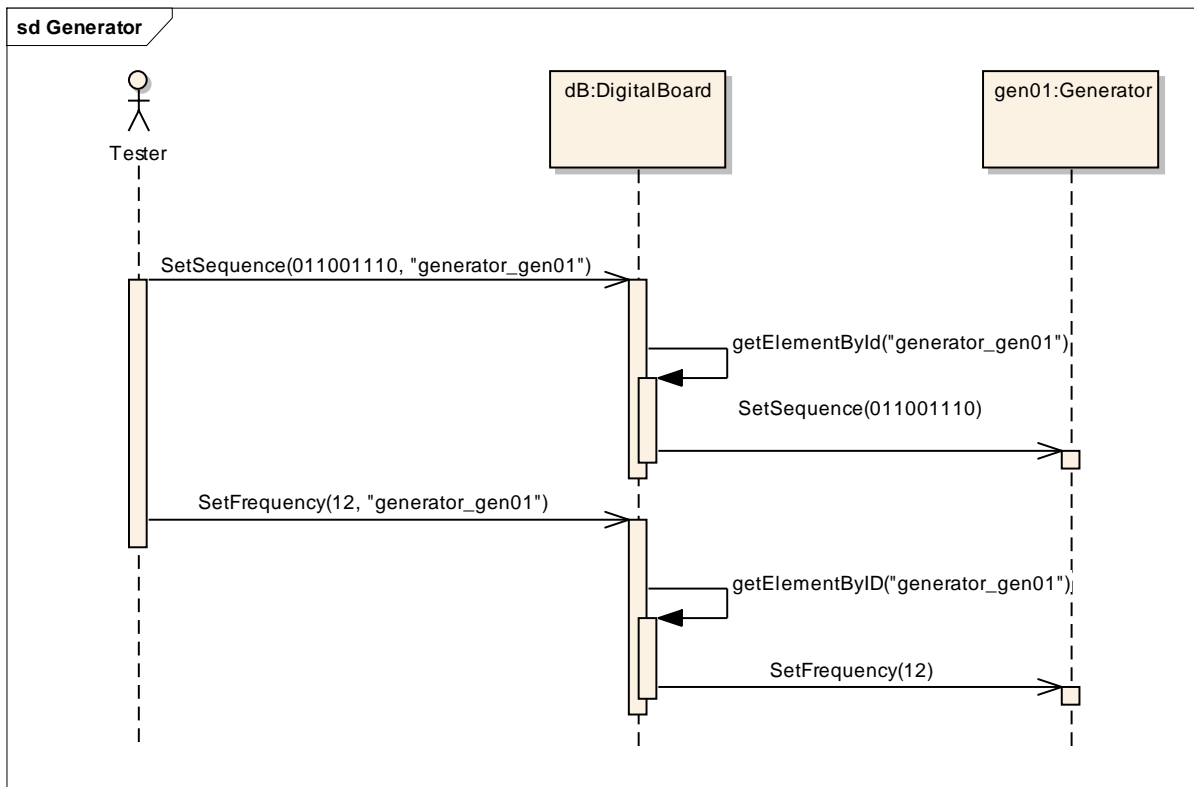
A Tesztelő léptetheti a hálózatot a dB:DigitalBoard StepComponents() metódusának hívásával:



A Tesztelő állíthatja az sw01:Switch kapcsoló állását Toggle() metódusának hívásával:

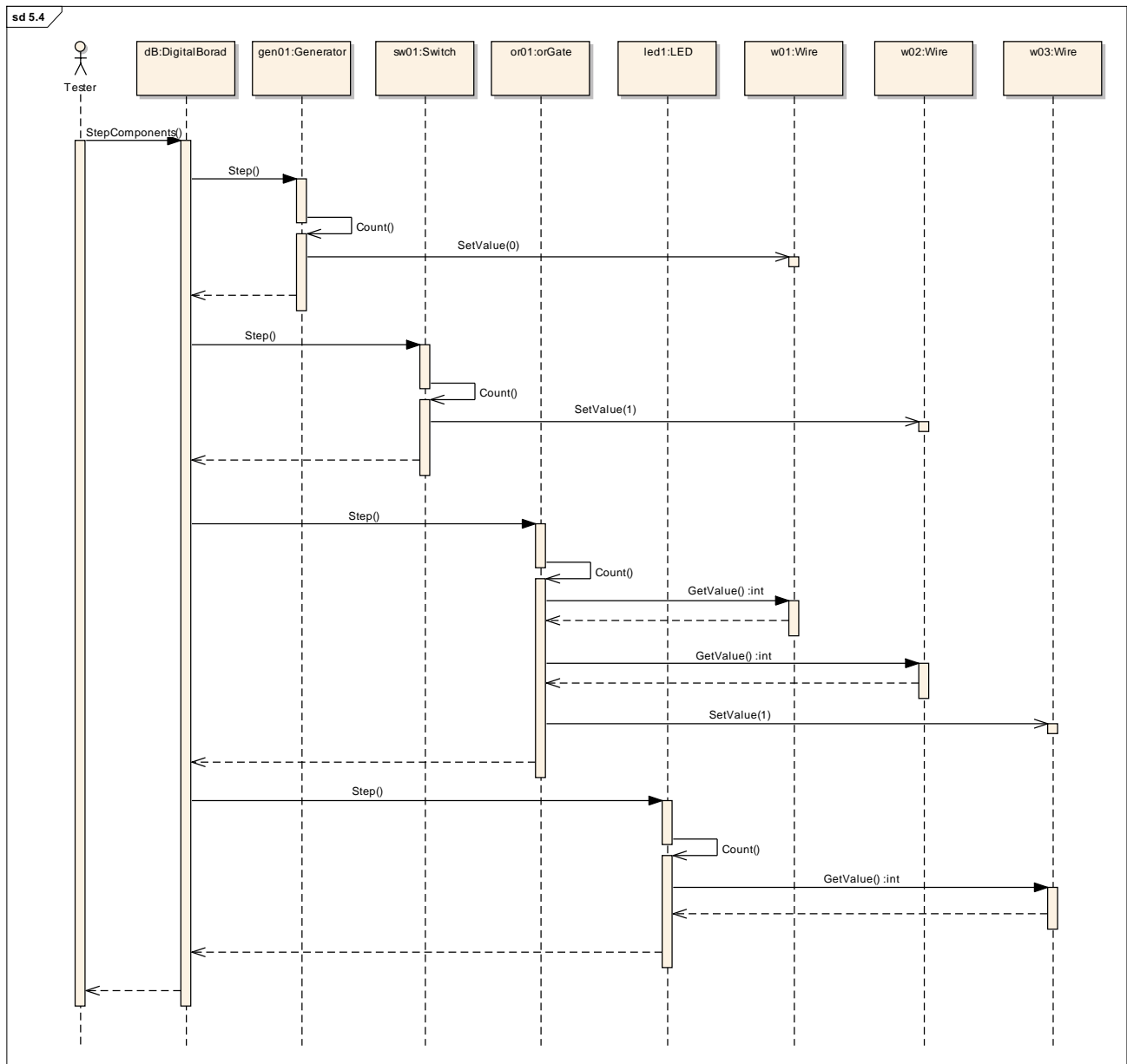


A Tesztelő állíthatja az gen01:Generator jelsorozatát és frekvenciáját a SetSequence() ill. SetFrequency metódusának hívásával:

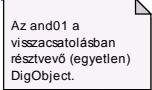


Egy konkrét hálózatra történő metódushívások dialógusai:

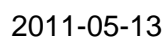
Egyszerű áramkör



sd 5.4 visszacsatolással stabil



sd 5.4 visszacsatolással instabil



3.5 Napló

Kezdés	Vége	Munka megnevezése	Résztevők	Időtartam (óra:perc)
2011. 3. 11. 12:00	2011. 3. 11. 13:30	Értekezlet Döntés: feladatok kiosztása	Csomák, Jéghe, Sziklay, Vad, Wiesner	1:30
2011. 3. 11. 10:00	2011. 3. 11. 13:00	Valóságos use-case- ek megírása	Csomák	3:00
2011. 3. 13. 20:30	2011. 3. 13. 23:30	Szekvenciadiagramok elkészítése	Sziklay, Wiesner	3:00
2011. 3. 13. 20:00	2011. 3. 13. 21:00	Szkeleton kezelői felületének terve, dialogusok	Jéghe	1:30
2011. 3. 13. 21:30	2011. 3. 13. 23:30	Architektúra, dokumentáció véglegesítés	Vad	2:00

4. Szkeleton beadás

4.1 Fordítási és futtatási útmutató

4.1.1 Fájllista

Fájl neve	Méret (byte-ban)	Keletkezés ideje	Tartalom
_TEST.java	1080	2011. 03. 19.	Olyan függvényeket tartalmaz, mely biztosítja a specifikációknak megfelelő logolást, megjelenítést
ANDGate.java	3036	2011. 03. 19.	andGate osztály
DigitalBoard.java	10878	2011. 03. 19.	DigitalBoard osztály
DigitalObject.java	1333	2011. 03. 19.	DigitalObject absztrakt osztály
Gate.java	1494	2011. 03. 19.	Gate absztrakt osztály
GENERATOR.java	4165	2011. 03. 19.	Generator osztály
iComponent.java	1435	2011. 03. 19.	iComponents interfész
Input.java	439	2011. 03. 19.	Input absztrakt osztály
INVERTER.java	2725	2011. 03. 19.	Inverter osztály
LED.java	1081	2011. 03. 19.	LED osztály
main.java	3464	2011. 03. 19.	A szkeleton elindítását, és a tesztelővel való kommunikációt valósítja meg.
ORGate.java	2797	2011. 03. 19.	orGate osztály
Output.java	368	2011. 03. 19.	Output absztrakt osztály
Status.java	197	2011. 03. 19.	A hálózat állapotát leíró enum-okat tartalmazza.
SWITCH.java	1935	2011. 03. 19.	Switch osztály
Wire.java	2699	2011. 03. 19.	Wire osztály
skeleton_build.bat	489	2011. 03. 20.	Fordítási batch fájl
skeleton_run.bat	45	2011. 03. 20.	Futtatási batch fájl

4.1.2 Fordítás

A fordítás a skeleton_build.bat batch fájl futtatásával kezdődik. Ezen belül be kell állítani a Java elérhetőségét, amely alapesetben megegyezik a HSZK gépeken az elérési úttal (PATH=D:\Program Files\Java\jdk1.6.0_14\bin). Eltérés esetén a tesztelőnek kell beállítani. A fájl létrehoz egy build, ezen belül egy classes mappát, melyet feltölt a programhoz használatos .class fájlokkal, valamint létrehoz egy dist mappát, melybe belekerül a skeleton.jar fájl, amit majd a skeleton_run.bat fájl használ a program futtatásánál.

4.1.3 Futtatás

Futtatáshoz a skeleton_run.bat futtatási batch file indításával kezdődik. Ennek előfeltétele, hogy megtörténjen a 6.1.2-ben leírtak szerint a programfordítása.

A program elindításával egy konzol képernyőt kapunk. Az konzolon kiírásra kerül a menü, melyben kiválaszthatjuk, hogy melyik műveletet akarjuk tesztelni. Ehhez a kiválasztott funkció sorszámát (1-től 12-ig) beírjuk és Entert nyomunk. Ekkor meghívódnak a funkciónak elvégzéséhez szükséges metódusok. A metódusok hívása megegyezik az 5.4 fejezetben ismertetett szekvencia diagramokban leírtakkal.

4.2 Értékelés

Tag neve	Munka százalékban
Csomák Gábor	12%
Jégh Tamás	33%
Sziklay György	16%
Vad Zsolt	19%
Wiesner Péter	20%

4.3 Napló

Kezdet	Időtartam	Résztevők	Leírás
2011.03.17 15:00	20:30	Jégh Tamás	Szkeleton megvalósítása. (Classok vázai, Logolás lehetősége, LoadBoard, ParseFile, Run, Stop, Pause metódusok)
2011.03.18 21:30	3:30	Jégh Tamás	Szkeleton megvalósítása. (StepComponents, Step, Count metódusok)
2011.03.19 7:00	3:30	Jégh Tamás	Szkeleton megvalósítása. Hibajavítás (az output nem felelt meg a specifikációnak) Tesztesetek, kezelői felület - main.class - újraírása
2011.03.20 20:30	2:30	Sziklay György	.bat fájlok, dokumentáció elkészítése
2011.03.20 20:30	1:30	Wiesner Péter	Skeleton tesztelése, módosítások megbeszélése
2011.03.20 21:00	4:00	Csomák Gábor	Skeleton tesztelése, kiegészítése, Step() függvény javítása a feedback megoldására, dokumentáció visszamenő módosítása

5. Prototípus koncepciója

Specifikáció hatása a projektre

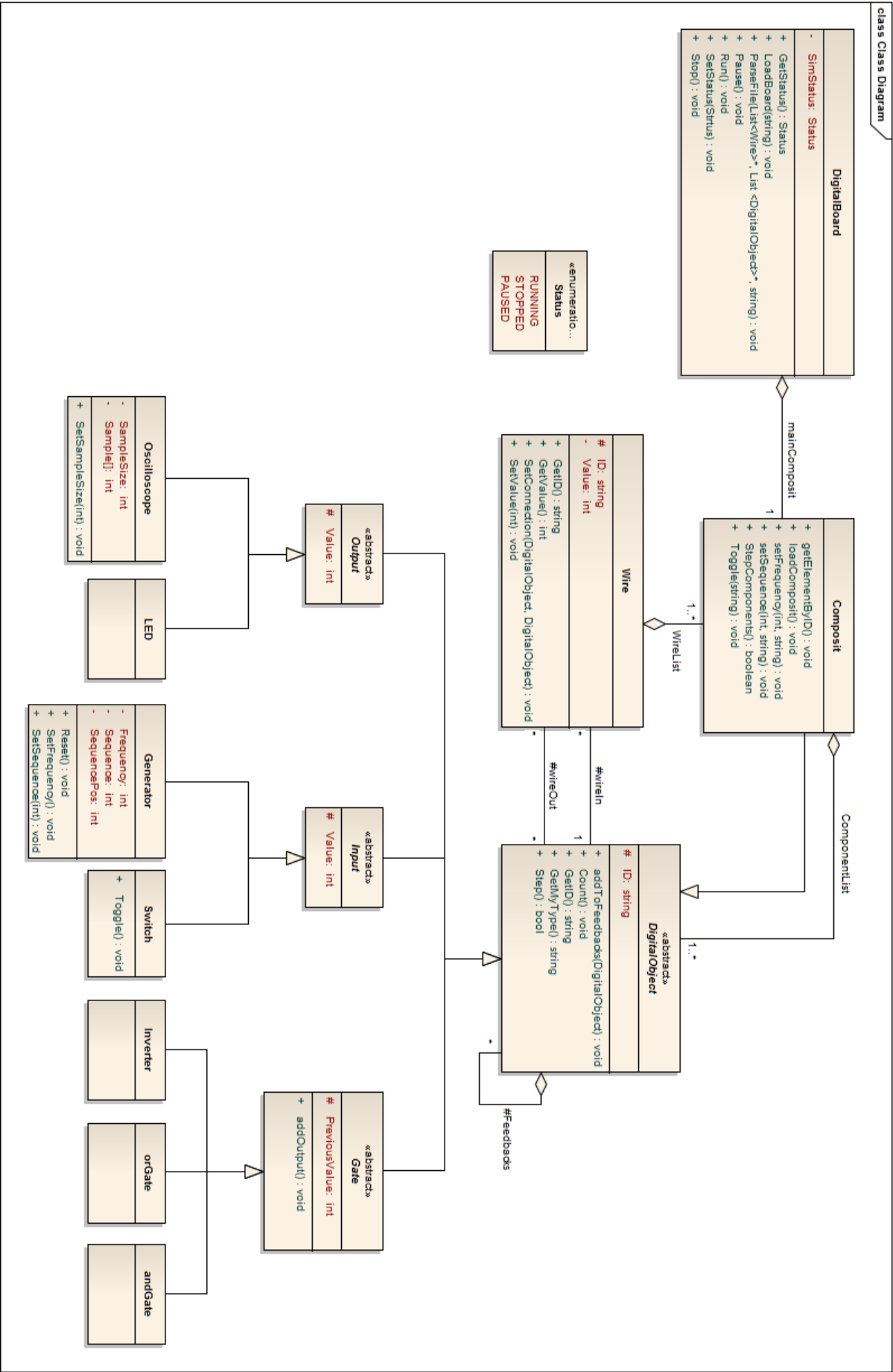
5.1.1 Módosítás leírása

Új építőelem az "Oscilloszkóp", amely egy jelet megjelenítő elem, egy bemenete van. Az adott pillanatig a bemenetére érkezett jeleket jeleníti meg.

Tetszőleges építőelemekből "kompozit" elem képezhető. Kompozit elem tartalmazhat más kompozit elemet is. Egy kompozit elem tetszőleges számú példányban lehet egy hálózatban. Egy hálózatban tetszőleges számú különböző kompozit használható. Kívülről nézve a kompozit elem a kimeneteit egyetlen lépésben állítja elő (mint a VAGY kapu).

5.1.2 Analízis modell változásai

4.3 Statikus struktúra diagramok



5.2 Prototípus interface definíciója

5.2.1 Az interfész általános leírása

A prototípus kezelőfelülete hasonló a szkeleton programban látottakkal. A tesztelő/felhasználó a program helyes működését egy szöveges konzolablakban követheti figyelemmel. A program elindítása után a megfelelő menü kiválasztásával illetve parancs kiadásával lehet a különböző funkciókat, mint például adott kapcsoló értékének átváltása, igénybe venni. A program használatának megkönnyítése végett külön parancssal lekérdezhetők a különböző menüpontok funkciói, a parancsok szintaktikái és következményei.

5.2.2 Bemeneti nyelv

loadBoard filename	
Leírás:	Betölti egy állományt, értelmezi a specifikált nyelven való leírást majd az alapján felépíti az áramkört
Opciók:	filename – egy szöveges állomány, melyben az áramkör leírása található

setOutput mod	
Leírás:	Beállítja a kimenet irányát. "Log"-olási feladatot valósíthatunk meg vele. Automatikusan a wid_test_log.log-ba ír.
Opciók:	mod=0 esetén a képernyőre, mod=1 esetén a kimeneti fájlba, mod=2 esetén mindkettőre ír. Ha a mod értéke 1 vagy 2, akkor automatikusan az stdout-ra kerülő szövegek kiíratódnak a kimeneti fájlba.

run	
Leírás:	Elindítja a szimulációt
Opciók:	

pause	
Leírás:	Megállítja a szimulációt.
Opciók:	

stop	
Leírás:	Alaphelyzetbe állítja a szimulációt.
Opciók:	

stepComponents	
Leírás:	Egyet léptet a szimuláción
Opciók:	

setFrequency <i>generator_id frequency</i>	
Leírás:	Beállítja egy generátor frekvenciáját
Opciók:	generator_id – a beállítani kívánt generátor azonosítója frequency – Ezredmásodpercben annak a gyakorisága, hogy milyen sűrűn váltson a generátor kimenete

setSequence <i>generator_id sequence</i>	
Leírás:	Beállítja egy generátor bitmintáját
Opciók:	generator_id – a beállítani kívánt generátor azonosítója sequence – bitminta, amit kiad a kimenetén egyesével

toggleSwitch <i>switch_id</i>	
Leírás:	Egy kapcsolót fel- vagy lebillent
Opciók:	switch_id – a kapcsoló azonosítója

setSample <i>oscilloscope_id samples</i>	
Leírás:	Oszilloszkóp által tárolandó minták számának beállítása
Opciók:	oscilloscope_id – a beállítani kívánt oszcilloszkóp azonosítója samples – a tárolandó minták száma

exit	
Leírás:	Kilép a programból
Opciók:	

5.2.3 butaHDL[®] ismertetése, alapvető ötletek, technikai elgondolások a megvalósításáról

A butaHDL[®] egy egyszerű hardverleíró nyelv, amely segítségével könnyen szerkeszthető egy digitális áramkör a WID szimulációs programhoz.

A nyelv változókkal dolgozik. A változótipusok a következők:

led	egy outputot egy LED-et definiál
switch	egy kapcsolót hoz létre
generator	egy jelgenerátort definiál
oscilloscope	egy belső memóriával rendelkező kijelzőt definiál
wire	drótot definiál

Egy-egy változó létrehozásának szintaktikája a következő:

```
1. wire <<drót neve>>;
2. switch <<kapcsoló neve>>;
3. generator <<jelgenerátor neve>>(<<frekvencia ms-ban>>);
4. led <<kijelző neve>>;
```

```
5. oscilloscope <<kijelző neve>>(<<tárolt minta hossza>>);
```

Például létrehozunk egy drótot, egy kapcsolót, egy jelgenerátort, amely 1 ezred másodpercenként szolgáltat értéket, egy másik generátort mely egy másodperc alatt ad kimenetet továbbá egy LED-et, majd egy oszcilloszkópot mely egy mintát jegyez meg, azaz LED-ként funkcionál egy-egy 10 mintát tároló kijelzőt.

```
• wire drotom0;
• switch kapcsolo;
• generator 1_ms_generator(1);
• generator 1_s_generator(1000);
• oscilloscope led(1);
• oscilloscope oszcilloszkop(10);
• led led2;
```

A *switch*, *generator* típusú változóknak közvetlenül lehet állítani a kimenetét, ez a set utasítással történhet, a következő módon:

```
8. set <<változó neve>> = <<érték>>;
```

switch esetén, ha ez az érték nem nulla, akkor a kimenetén 1-et fog szolgáltatni, különben nullát.

A *generator* típusú változóknál a megadott érték a generátor mintája lesz, kimenete a belső szekvencia és az mintában való aktuális pozíciója szerint változik

A *wire*, *led*, *oscilloscope* típusoknak nem lehet közvetlenül értéket adni, logikai értékük a hálózattól függ, amire csatlakoznak. Az ilyen értékadás az *assign* utasítással történik

```
4. assign <<elem neve>> = <<logikai kifejezés>>;
```

Az értékadás jobb oldalán tetszőleges logikai kifejezés állhat a megfelelő operátorokkal. A kifejezés egyes operátorai más és más kaput azonosítanak.

Operátor	Neve	kapu
!	not	INVERTER
	or	VAGY kapu
&	and	ÉS kapu
...

A logikai kifejezés operandusainak mindegyike csak olyan változó lehet, melyhez tartozó áramköri elemnek van kimenete. (Igy: *switch*, *generator*, *wire* és *composit*, erről később)

(Azaz megtehető, hogy egy dróthoz hozzárendelünk egy kombinációs hálózat részletét, melyet aztán felhasználunk egy másik drót, vagy LED esetleg oszcilloszkóp értékadásánál.)

Például:

```
• assign drotom0 = kapcsolo0;
• assign drotom1 = (kapcsolol1 & kapcsolol2) + (kapcsolol3+drotom0);
• assign ledem1 = drotom1;
```

Az áramkör alapvetően modulokból áll. Egy áramköri modulhoz tetszőleges számú drót futhat be, illetve szintén tetszőleges számú távozhat. Egy modul további tetszőleges számú modult tartalmazhat. Létezik egy kitüntetett modul a „main”. Ennek nincs bemenete sem kimenete, egyetlen modul sem tartalmazhatja.

A modulokat a nyelvben *composit*-nak hívjuk, és a szintaktikája a következő:

```
1. composit <<modulnév>> (in <<bemenet1, bemenet2, ...>>, out <<kimenet1, kimenet2, ...>>)
```

```

2.
3.   <<modulleírás: változók definiálása, műveletek>>;
4.
5.   endcomposit;

```

FONTOS megjegyzések:

- Alapvető szintaktikai szabályok: Minden utasítást „ ; ” (pontosvessző) zár. (Ez a fájl könnyebb értelmezhetősége miatt van így)
- A kulcsszavak kisbetűsek, a nyelv érzékeny a kisbetű-nagybetű különbségekre!
- Minden változót, modult használata előtt először definiálni kell, nem hivatkozhatunk olyan elemre, melyet korábban még nem hoztunk létre

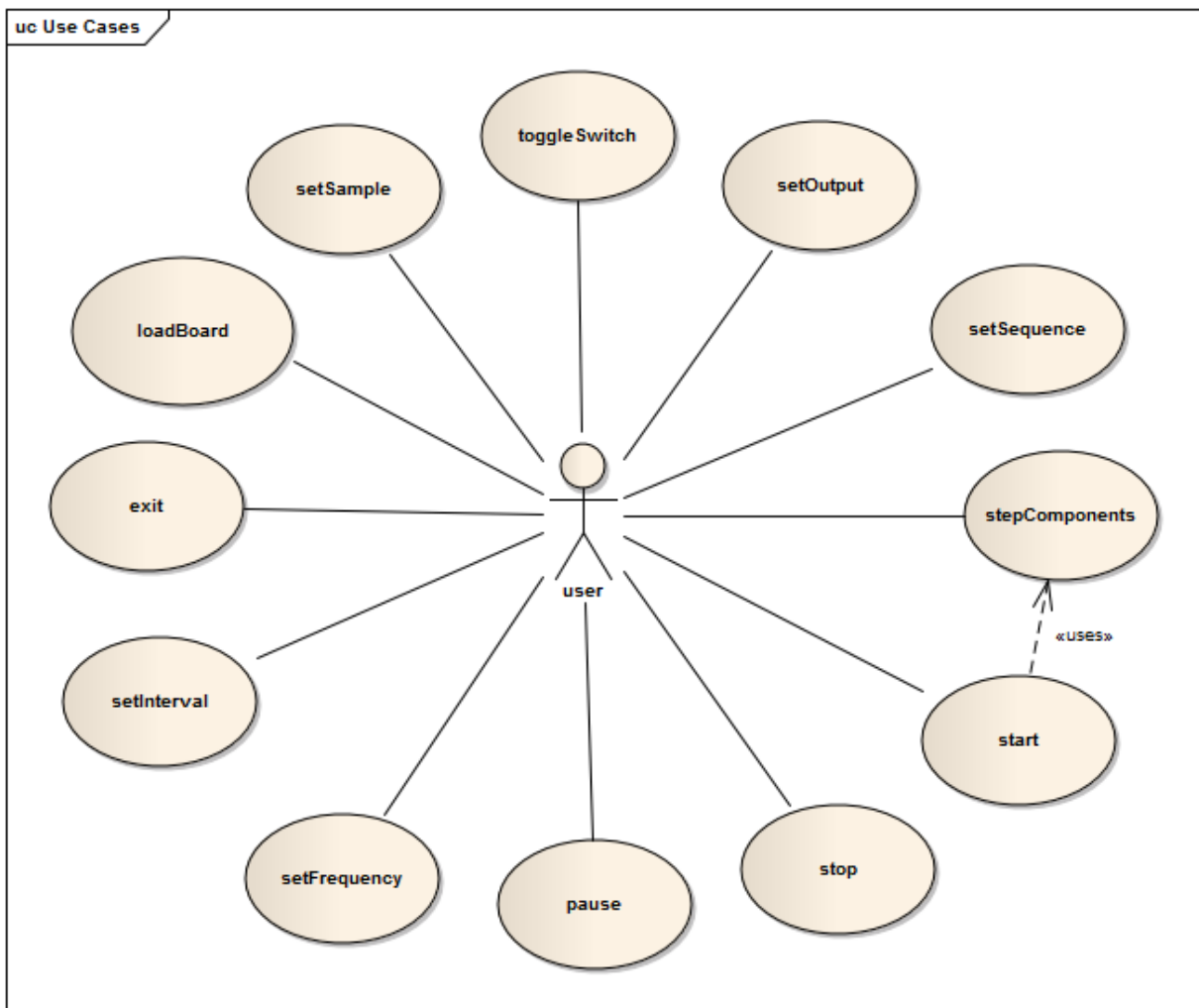
5.2.4 Kimeneti nyelv

A program futásáról, azaz a meghívott függvényekről, és a végbemenő változtatásokról folyamatos visszajelzés olvasható a konzolablakban. Az esetlegesen felmerülő helytelen bevitelből eredő hibák, futás közben keletkezett kivételek ugyanitt lesznek a felhasználó tudtára adva (ezek a 7.3.3-as részben vannak specifikálva). A futás során a program nem használ a standard kimeneten kívül más megjelenítési eszközt, ezért a kimenet bármikor fájlba menthető tesztelés céljából.

A különböző parancsoknak megfelelő kimenetek:

Parancsok	Kimenet
setFrequency <i>gen01 10</i>	gen01's frequency is set to 10
setSequence <i>gen01 100111</i>	gen01's sequence is set to 100111.
loadBoard <i>file1.bhdl</i>	file1's Board is loaded.
stepComponents	Board circuit has stepped.
run	Simulation is running <w01> value is [0 1] <Wire@ID> value is [0 1] ... <led01> value is [0 1]. <led01> value is [0 1]. <osc01> value is [0101110].
pause	Simulation is not running.
stop	Simulation stopped.
setOutput 2	Output mode is set to 2.
setInterval 100	Board's interval is set to 100.
toggleSwitch <i>sw01</i>	sw01's value changed.
setSample <i>osc01 20</i>	osc01's Sample size is set to 20.

5.3 Összes részletes use-case



Use-case neve	<i>loadBoard</i>
Rövid leírás	Ez a use-case tartalmazza adott áramkör betöltését file-ból.
Aktorok	user
Forgatókönyv	A felhasználót megkérdezzük, melyik a betölteni kívánt fájl, majd – amennyiben létezik – betöltjük és felépítjük az áramkört, melynek kezdő állapota 'STOPPED'.

Use-case neve	<i>toggleSwitch</i>
Rövid leírás	Ez a use-case tartalmazza az áramkörben található kapcsolók átállításának lehetőségét.
Aktorok	user
Forgatókönyv	Amennyiben az áramkörben található kapcsoló, megkérdezzük, melyik switchet szeretné átállítani, és milyen értékre. Amennyiben van olyan, beállítjuk a kívánt értékre.

Use-case neve	<i>setSequence</i>
Rövid leírás	Ez a use-case tartalmazza a generátorok szekvenciájának állítását.
Aktorok	user
Forgatókönyv	Amennyiben az áramkörben található generátor, megkérdezzük a felhasználót, melyiket szeretné átállítani. Amennyiben létezik, megkérdezzük a beállítandó szekvenciát, és a generátornak átadjuk.

Use-case neve	<i>stepComponents</i>
Rövid leírás	Ez a use-case tartalmazza az áramkör egy lépését, azaz a generátorok értékváltását, majd a vezetékek értékeinek újraszámítását.
Aktorok	user
Forgatókönyv	A komponens listán sorrendben minden elemet felszólítunk a léptetésre, azaz generátor esetén a szekvencia következő elemét adjuk a kimenetre, belső változókkal nem rendelkező objektumoknál pedig a kimenetet frissítjük, a Step() hívással. Amennyiben egy objektumnak nem üres a visszacsatolás listája, azokra meghívjuk sorrendben a Count() függvényt.

Use-case neve	<i>start</i>
Rövid leírás	Ebben a use-case-ben a felhasználó elindítja az áramkör szimulációját.
Aktorok	user
Forgatókönyv	Elindítjuk a szimulációt, a generátorokat megadott frekvenciával léptetjük, minden lépés után kiírjuk a kimenetek eredményét. Az áramkör állapota 'RUNNING' lesz.

Use-case neve	<i>stop</i>
Rövid leírás	Ebben a use-case-ben a felhasználó leállítja a szimulációt.
Aktorok	user
Forgatókönyv	Amennyiben a szimuláció fut, leállítjuk azt, és a táblát 'reset-eljük', azaz az elemek belső változóit alapállapotba állítjuk. Az áramkör állapota 'STOPPED' lesz.

Use-case neve	<i>pause</i>
Rövid leírás	Ebben a use-case-ben a felhasználó szüneteltetheti a szimulációt.
Aktorok	user
Forgatókönyv	Amennyiben a szimuláció fut, leállítjuk azt. Az áramkör állapota 'PAUSED' lesz.

Use-case neve	<i>setOutput</i>
Rövid leírás	Ebben a use-case-ben a felhasználó megmondhatja, hogy szeretné-e kimenteni egy külső fix fájlba a stdout-ra érkező üzeneteket.
Aktorok	user
Forgatókönyv	Amennyiben a felhasználó úgy dönt, lehetősége van az stdout-ra érkező üzeneteket egy külső fájlba irányítani tesztelés gyanánt.

Use-case neve	<i>setInterval</i>
Rövid leírás	Ebben a use-case-ben a felhasználó beállíthatja, hogy milyen gyorsan történjen a szimuláció.
Aktorok	user
Forgatókönyv	Amennyiben az áramkör állapota 'RUNNING', beállíthatjuk a stepComponent() hívásának gyakoriságát.

Use-case neve	<i>setFrequency</i>
Rövid leírás	Ebben a use-case-ben a felhasználó beállíthatja, hogy milyen gyorsan váltszon a generátorok kimenete.
Aktorok	user
Forgatókönyv	Amennyiben az áramkörben található generátor, megkérdezzük, melyik generátor frekvenciáját szeretné megváltoztatni, és milyen értékre. Amennyiben létezik, megkérdezzük a beállítandó frekvenciát, és a generátornak átadjuk.

Use-case neve	<i>setSample</i>
Rövid leírás	Ez a use-case tartalmazza az áramkörben található oszcilloszkópok által tárolandó minták méretének meghatározását.
Aktorok	user
Forgatókönyv	Amennyiben az áramkörben található oszcilloszkóp, megkérdezzük, melyik oszcilloszkóp beállításait szeretné megváltoztatni, és milyen értékre. Amennyiben van olyan, beállítjuk a kívánt értékre.

Use-case neve	<i>exit</i>
Rövid leírás	Ez a use-case tartalmazza a programból való kilépést
Aktorok	user
Forgatókönyv	Kilépés a programból.

5.4 Tesztelési terv

5.4.1 A tesztelés menete

A prototípus fordítása, futtatása a szkeletonnal megegyező módon történik. A prototípus a parancsokat standard bemenetről olvassa. Minden újsor karakter (\n) az adott parancs végét jelöli. A kimenet a standard kimenetre kerül.

5.4.2 A teszteléshez használható parancsok

Lásd 4.1.2. fejezet.

5.4.3 A kimeneten megjelenő hibaüzenetek, figyelmeztetések

```
x Error: FileNotFound: Nem olvasható a megadott bemeneti fájl.  
x Error: CommandNotFound: Nem létező parancs érkezett.  
x Error: ParameterCountMismatch: Nem megfelelő számú paraméter érkezett.  
x Error: WrongParameter: Nem megfelelő paraméter érkezett.  
x Error: NoBoard: Nincs betöltve pálya a loadBoard segítségével. (Ennek ellenére dolgozni  
akart vele)  
x Error: WrongBoard: Nem megfelelően formázott áramkört próbál betölteni.  
x Error: UnstableCircuit: Instabil áramkör, a szimuláció nem futtatható.  
x Error: ElementHasNoInput: A megjelölt elemnek (elemeknek) nincs bemenete  
x Error: FrequencyIsNull: A meghatározott generátor frekvenciája 0.  
x Error: FrequencyMustBePositive: A frekvenciának pozitív számnak kell lennie  
i Warning: ElementHasNoOutput: Egy kimenettel rendelkező elem nem csatlakozik további  
áramköri elemhez  
...
```

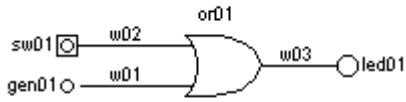
5.4.4 A kimeneten megjelenő egyéb események

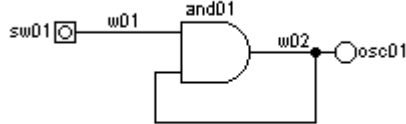
Minden áramkör-betöltés után a képernyőn megjelenik az elemek listája.

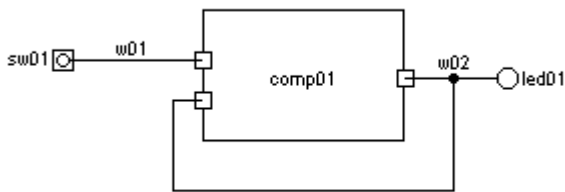
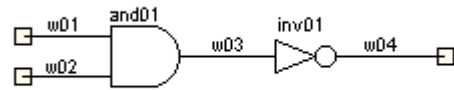
5.4.5 Az áramkör formátuma

Az áramköröket szöveges fájlokban tároljuk, melyből azok egyértelműen felépíthetők. A butaHDL[©] egy egyszerű hardverleíró nyelv mely segítségével könnyen szerkeszthető egy digitális áramkör a szimulációs programhoz. A butaHDL[©] nyelv leírása a 7.1.3 fejezet alatt olvasható.

5.4.6 Tesztesetek

Teszt neve	<i>Egyszerű áramkör</i>
Rövid leírás	Egyszerű áramkör tesztelése, mely egy kapcsolóból, egy jelgenerátorból, egy VAGY kapuból és egy LED-ből áll.
Teszt célja	A hálózatot felépítő elemek tesztelése.
Ábra:	

Teszt neve	<i>Visszacsatolt stabil áramkör</i>
Rövid leírás	Visszacsatolt áramkör tesztelése, meg egy kapcsolóból, egy ÉS kapuból és egy oszcilloszkópból áll.
Teszt célja	Visszacsatolás, illetve további építőelemek tesztelése.
Ábra:	

Teszt neve	<i>Instabil áramkör</i>
Rövid leírás	Instabil áramkör tesztelése, mely áll egy kapcsolóból, egy kompozit elemből és egy LED-ből. A kompozit elem belsejében egy ÉS kapu és egy inverter van.
Teszt célja	Instabil hálózat tesztelése, azaz instabilitás felfedezése, valamint kompozit elem tesztelése.
Ábra:	 <p>A kompozit belülről:</p> 

5.5 Tesztelést támogató segéd- és fordítóprogramok specifikálása

A program otthoni fordításához az *Eclipse IDE*, a teszteléshez a *NetBeans IDE* fordítóját használjuk. A tesztelés fordításának és a program futtatásának megkönnyítésére írt *.bat fájlokat *Notepad* illetve *Notepad++* programokkal hoztuk létre, szerkesztettük.

A tesztelés sikerességének eldöntéséhez, azaz, hogy a kimenet megegyezik az elvárt kimenettel, a *Total Commander* 'összehasonlítás tartalomra' parancsát használjuk. Ennek hatására a két szövegfájl (a kimentett kimenet és a begépelt várt kimenet) különbsége ki lesz emelve. A teszt sikeresnek mondható, ha nincs egyik szövegben sem kiemelés.

5.6 Napló

Kezdés	Vége	Munka megnevezése	Résztevők	Időtartam (óra:perc)
2011.03.24 20:00	2011.03.24 20:45	Osztálydiagramok módosítása	Jégh	00:45
2011.03.25 12:00	2011.03.25 13:00	Értekezlet: Feladatok kiosztása, Osztálydiagram változtatásának átbeszélése, kijavítása	Csomák, Jégh, Sziklay, Vad, Wiesner	1:00
2011.03.25 21:00	2011.03.25 22:30	Dokumentáció	Wiesner	1:30
2011.03.26 10:00	2011.03.26 13:00	Bemeneti nyelv elkészítése	Jégh	3:00
2011.03.26 17:30	2011.03.26 20:00	Use-case-ek megírása	Csomák	2:30
2011.03.26 20:00	2011.03.26 22:30	Tesztelési terv megírása	Csomák	2:30
2011.03.27 12:00	2011.03.27 12:20	Use-case-ek módosítása	Jégh	0:20
2011.03.27 15:00	2011.03.27 17:00	Módosítások dokumentálása	Sziklay	2:00
2011.03.27 22:00	2011.03.27 23:00	Tesztesetek megírása	Sziklay	1:00
2011.03.27 23:00	2011.03.28 3:30	Dokumentáció elkészítése, Use-case-ek kiegészítése, kimeneti nyelv kiegészítése	Vad	4:30
2011.03.28 3:30	2011.03.27 4:30	Dokumentáció véglegesítése	Sziklay, Vad	1:00

6. Részletes tervek

6.1 Osztályok és metódusok tervei.

6.1.1 DigitalBoard

Felelősség

Olyan osztály, amelyben tartalmazza a felhasználói interakcióhoz szükséges attribútumok és metódusok.

Ősosztályok

Nincs.

Interfészek

Nincs

Attribútumok

Composit MainComposit: Azt a Composit-ot valósítja meg, amely az összes hálózati elemet tartalmazza.

Status SimStatus: Háromállapotú változó, amely a szimuláció aktuális állapotát tárolja. Lehetséges értékei: RUNNING, PAUSED, STOPPED.

int MaxNumOfRound: annak az értéke, hogy RUNNING állapotban hányszor fusson le a léptetés.

Metódusok

void SetStatus(Status status): Feladata a belső Status változó értékének beállítása.

DigitalObject GetElementByID(String ElementID): Meghívja a MainComposit GetElementByID() metódusát.

void LoadBoard(String strFilePath): A megfelelő paraméterrel meghívja a ParseFile(String strFilePath) metódust.

void ParseFile(String strFilePath): A megadott útvonalon található fájlt olvassa be és soronként értelmezi az állományt.

void Run(): A szimulációt indítja úgy, hogy elsőnek beállítja a státuszát RUNNING értékre, majd meghívja a StepComponents() metódust.

void Pause(): Lehetőséget ad a szimuláció lépésenkénti végrehajtására. Lehetővé teszi, hogy a felhasználó hívja meg a StepComponents() függvényt. Meghívja a SetStatus(PAUSED) metódust.

void Stop(): A szimulációt megállítja. Az összes DigitalObject és Wire típusú objektum értékét az eredeti, betöltéskori értékre állítja vissza. Meghívja a SetStatus(STOPPED) metódust.

void SetFrequency(int Frequency, String ElementID): Meghívja a MainComposit SetFrequency() metódusát.

void SetSequence(String Sequence, String ElementID): Meghívja a MainComposit SetSequence() metódusát.

void Toggle(String ElementID): Meghívja a MainComposit Toggle() metódusát.

void SetSample(): Meghívja a MainComposit SetSample() metódusát.

void StepComponents(): Meghívja a MainComposit StepComponents() metódusát.

2.5.1 Wire

Felelősség

Az áramkörben található értékek tárolására szolgáló objektum. DigitalObject típusú objektumok között teremt kapcsolatot.

Össztályok

Nincs

Interfészek

Nincs

Attribútumok

String ID: Egyedi karakteres azonosító, mely egyértelműen meghatároz egy Wire objektumot.

String strIDDelimiter: ID generáláshoz szükséges konstans.

int Value: A Wire objektumok által tárolt érték. A Gate objektumok ezek alapján számolják ki kimenetüket.

int WireCounts: Osztályra jellemző statikus tagváltozó, a biztosan egyedi ID generálására.

List <DigitalObject> objectsIn: DigitalObject objektum-referencia, amely a vezeték bemenetéhez kapcsolódik.

List <DigitalObject> objectsOut: DigitalObject objektum-referenciákból álló lista azon objektumokról, melyek a vezeték kimeneteihez kapcsolódnak.

Metódusok

String GetID(): A Wire egyedi azonosítójának lekérdezésére szolgáló metódus.

String GetName(): A Wire objektum nevének lekérdezésére szolgáló metódus.

int GetValue(): A Wire objektum értékének lekérdezésére szolgáló metódus.

void SetValue(int value): A Wire objektum értékének módosítására szolgáló metódus.

void SetConnection(DigitalObject, DigitalObject): Kapcsolatot teremt két DigitalObject között. Beállítja a vezeték inputját, illetve hozzáadja az output tömbjéhez a DigitalObjectet.

2.5.2 DigitalObject

Felelősség

Absztrakt osztály, amelyből az összes nem Wire típusú objektumot származtatjuk. Legnagyobb felelőssége az iComponent interfész megvalósítása, továbbá nyilvántartja a bemeneteit és kimeneteit.

Össztályok

Nincs.

Interfészek

Nincs.

Attribútumok

String ID: Egyedi karakteres azonosító, mely egyértelműen meghatároz egy DigitalObject objektumot.

String strIDDelimiter: ID generáláshoz szükséges konstans.

List<Wire> wireIn: Wire objektum-referenciákból álló lista azon Wire objektumokról, melyek az objektum bemeneteihez kapcsolódnak.

List<Wire> wireOut: Wire objektum-referenciákból álló lista azon Wire objektumokról, melyek az objektum kimeneteihez kapcsolódnak.

List<DigitalObject> Feedbacks: Ha egy Gate egyik bemenete egy visszacsatolás kezdete, akkor tartalmaz egy Feedbacks listát, mely referenciát tárol az összes, az adott visszacsatolásban résztvevő DigitalObject-re.

Metódusok

String GetID(): A DigitalObject egyedi azonosítójának lekérdezésére szolgáló metódus.

void AddToFeedBack(DigitalObject): Hozzáadja a paraméterként kapott DigitalObjectet a Feedbacks tömbjéhez.

void Count(): Ha a DigitalObject kapu vagy kimenet, akkor a bemenő vezetékeiről lekérdezi az értéket, kiszámolja a logikai függvényét. Ha a DigitalObject kapu vagy bemenet, akkor beállítja a kimenő vezetékeinek az értékét a korábbi számítási eredményére vagy a belső értékére.

boolean Step(): Feladata az adott elem értékét kiszámítása, ill. annak eldöntése, hogy a DigitalObject stabil-e. Ezt úgy teszi, hogy kiszámolja az értékét Count()-tal, ha egy visszacsatolás következik a kimenetén, akkor végigmegy a tömbön és mindegyikre kiszámoltatja az adott elem értékét. Ezt megismételi még kétszer úgy, hogy minden alkalommal, amikor végzett a Feedbacks tömbön való számolással kiszámolja az értékét újból és megnézi, hogy a saját korábbi értékével megegyező értéket számolt ki. Ha a harmadik Count() után megegyezik a két érték, akkor stabil a kapu és true-val tér vissza egyébként false-al.

String GetID(): A DigitalObject objektum egyedi azonosítójának lekérdezésére szolgáló metódus.

String GetType(): Visszatér az objektum típusával (osztály neve).

String GetName(): A DigitalObject objektum nevének lekérdezésére szolgáló metódus.

6.1.2 Composit

Felelősség

Legfontosabb feladata a kompozitot felépítő elemek tárolása, és azok irányítása.

Ősosztályok

DigitalObject

Interfészek

Nincs.

Attribútumok

- **ArrayList<List<DigitalObject>> ComponentList:** Ez az attribútum tárolja az összes kaput, kimenetet, bemenetet hierarchikus sorrendben. Ez nem más, mint egy listából szervezett tömb. A tömb indexe azonosítja a hierarchia szintet (0-Források, 1-a forrásokhoz csatlakozó elemek, stb.) az egyes szinteken pedig egy lista van az elemekről.
- **List<Wire> WireList:** Lista a Wire objektumok tárolására.

- **List<PIN> pins_in:** PIN elemekből álló lista, melybe a kívülről érkező drótok futnak.
- **List<PIN> pins_out:** PIN elemekből álló lista, melyből a kompoziton kívüli drótok indulnak.

Metódusok

void GetMyComposits(List<Composit>): A tartalmazott Composit-okat kigyűjti egy verembe

void SetFrequency(int Frequency, String ElementID): A paraméterben megadott azonosítóval rendelkező GENERATOR objektum frekvenciáját módosítja.

void SetFrequency(int Frequency, String ElementID): A paraméterben megadott azonosítóval rendelkező GENERATOR objektum frekvenciáját módosítja.

void SetSequence(String Sequence, String ElementID): Beállítja a paraméterben megadott azonosítóval rendelkező GENERATOR objektum szekvenciáját.

void SetSwitch(int Value, String ElementID): Beállítja a paraméterben megadott azonosítóval rendelkező Switch objektum értékét.

DigitalObject Toggle(String ElementID): A paraméterben megadott azonosítóval rendelkező SWITCH objektum értékét az ellenkezőre állítja azáltal, hogy meghívja az objektum hasonló nevű paraméterét.

void SetSample(): A paraméterben megadott azonosítóval rendelkező OSCILLOSCOPE objektum mintavételezésének nagyságát állítja be.

void AddToFeedbacks(): Hozzáadja a paraméterként kapott DigitalObject-et a Feedbacks tömbjéhez.

DigitalObject GetElementByID(String ElementID): Megkeres egy adott elemet egy Composit ComponentList listájában.

DigitalObject GetElementByName(String ElementName): A kompozit saját elemei közül kikeresi a megadott névvel rendelkező objektumot és visszatér a referenciájával. Ha nem találja, null a visszatérési érték.

Wire GetWireByID(String WireID): Megkeres egy adott Wire objektumot egy Composit ComponentList listájában.

Wire GetWireByName(String WireName): A kompozit saját Wire objektumai közül kikeresi a megadott névvel rendelkező objektumot és visszatér a referenciájával. Ha nem találja, null a visszatérési érték.

Step(): Meghívja a Composit Count() metódusát, illetve ellenőrzi, hogy stabil-e a Composit

Count(): Lekérdezi a bemeneteinek (PIN-ek) értékeit, majd az összes belső elemre meghívja a Step()-et.

void StepComponents(): Meghívja a Composit összes elemére a Step()-et.

void AddToWireList(Wire wire): Hozzáad a WireList-hez egy Wire objektumot.

boolean RemoveFromWireList(Wire wire): Eltávolít a WireList-ből egy Wire objektumot.

void buildHierarchy(): Felépíti a Composit belső hierarchiáját. Első lépésként lekérdez minden forrásnak minősülő elemet (Input osztály leszármazottjai, illetve bejövő PIN-ek) ezután amíg van rendezetlen elem: az előző hierarchia szinten lévő elemeket végignézi és a következő hierarchia szintre pakolja azokat, amik meg nem szerepelnek a rendezett listában.

void getFeedbacks(): Egy hierarchikusan felépített Composit-ban megkeresi a visszacsatolásokat.

List<DigitalObject> getFirstLevelOfComponentList(): A Composit felépítésénél a Compositot alkotó elemeket a ComponentList 0. szintjére gyűjti mindaddig, míg nincs megszervezve a hierarchia.

6.1.3 Input

Felelősség

Absztrakt osztály a beviteli modulok (pl. jelgenerátor, kapcsoló) leszármaztatására.

Ősosztályok

DigitalObject

Interfészek

Nincs.

Attribútumok

int Value: Az adott Input objektum értékét tárolja.

Metódusok

Az őosztály metódusain kívül nincs más szolgáltatása.

6.1.4 Switch

Felelősség

Feladata minden egyes Step() hívásakor a kimenetéhez csatlakozó Wire objektumoknak értéket adni. A felhasználó a Toggle() metódus segítségével képes egy ilyen objektum értékének megváltoztatására.

Ősosztályok

DigitalObject->Input

Interfészek

Nincs.

Attribútumok

Az őosztályok attribútumain kívül nincs más tagváltozója.

Metódusok

void Toggle(): A Value változó értékét állítja nullából egybe, egyből nullába.

6.1.5 Generator

Felelősség

Speciális Input objektum, mely adott frekvencián változtatja a kimenetét. Feladata – adott idő elteltével – a kimenetéhez csatlakozó Wire objektumok értékének be-, ill. átállítása.

Ősosztályok

DigitalObject->Input

Interfészek

Nincs.

Attribútumok

int Frequency: A generátor léptetésének a gyakoriságát tároljuk ebben. Az itt megadott számú Step() hívás után fog csak lépni.

int Sequence: Ebben tároljuk a kimenetre kiküldendő mintát. Értelmezése bináris.

int SequencePos: Az attribútum tartja nyilván az aktuális pozíciót a szekvenciában. A Reset() hívás 0 értékre állítja.

int FrequencyCounter: Egy segédszámláló, mely nyilvántartja, hogy még mennyi Count() maradt hátra.

Metódusok

void Reset(): A SequencePos. értékét állítja alapértelmezettre, azaz a minta elejére.

void SetFrequency(int frequency): a Frequency értékét állítja be, a paraméterben megadott értékre.

void SetSequence(int sequence): a Sequence, azaz a minta értékét állítja be, a paraméterben megadott értékre.

6.1.6 Output

Felelősség

Absztrakt osztály a megjelenítő modulok (pl. LED) leszármaztatására.

Ősosztályok

DigitalObject

Interfészek

Nincs.

Attribútumok

int Value: Ez tárolja a bemeneti Wire objektum értékét.

Metódusok

Az őosztály metódusain kívül nincs más szolgáltatása.

6.1.7 LED

Felelősség

Output absztrakt osztályt valósítja meg. Lekérdezi a hozzá csatolt vezeték (wireIn) értékét, és megjeleníti azt a felhasználó számára.

Ősosztályok

DigitalObject->Output

Interfészek

Nincs.

Attribútumok

Az őosztályok attribútumain kívül nincs más tagváltozója.

Metódusok

Az őosztály metódusain kívül nincs más szolgáltatása.

6.1.8 Oscilloscope

Felelősség

Output absztrakt osztályt valósítja meg. Egy bemenettel rendelkezik, melynek értékét a felhasználó által megadott ideig (lépésszámon keresztül) megjeleníti.

Ősosztályok

Output

Interfészek

Nincs.

Attribútumok

int SampleSize: Meghatározza, hogy az időben később beérkezett jeleket meddig tárolja el az Oscilloscope (azaz a buffer mérete). A tárolás FIFO elv szerint működik.

Queue<Integer> Samples: A mintát tároló FIFO lista

Metódusok

void SetSampleSize(int SampleSize): Beállítja a mintát tároló FIFO lista nagyságát.

6.1.9 Gate

Felelősség

Absztrakt osztály a logikai kapuk (pl. ÉS, VAGY, Inverter) leszármaztatására. A bemeneti vezeték(ek)nek lekérdezi az értékét, és kiszámolja a kimenet új értékét, és be is állítja azt, az leszármazott osztály igazságtáblája szerint.

Ősosztályok

DigitalObject

Interfészek

Nincs.

Attribútumok

- **List<DigitalObject> Feedbacks:** Ha egy Gate egyik bemenete egy visszacsatolás kezdete, akkor tartalmaz egy Feedbacks tömböt, mely referenciát tárol az összes, az adott visszacsatolásban résztvevő DigitalObject-re.
- **int PreviousValue:** A legutolsó ciklus (Step()) eredményét (kimeneti értékét) tárolja, a stabilitásellenőrzés céljából (Count() metódus).

Metódusok

Az őosztály metódusain kívüli szolgáltatások:

void AddOutput(Wire wire): Hozzáad egy Wire objektumot a kimenetéhez (vagyis a wireOut listához).

6.1.10 ANDGate

Felelősség

Gate absztrakt osztályt valósítja meg, bemeneti értékeit lekérdezi (wireIn) a GetValue() metódussal, majd előállítja a kimeneti értéket (wireOut) a logika táblája alapján és beállítja azt, a SetValue(int) segítségével.

Össztályok

DigitalObject->Gate

Interfészek

Nincs.

Attribútumok

int ANDCounts: Osztályra jellemző statikus tagváltozó, a biztosan egyedi ID generálására.

Metódusok

Az osztály metódusain kívül nincs más szolgáltatása.

6.1.11 orGate

Felelősség

Gate absztrakt osztályt valósítja meg, bemeneti értékeit lekérdezi (wireIn) a GetValue() metódussal, majd előállítja a kimeneti értéket (wireOut) a logika táblája alapján és beállítja azt, a SetValue(int) segítségével.

Össztályok

DigitalObject->Gate

Interfészek

Nincs.

Attribútumok

int ORCounts: Osztályra jellemző statikus tagváltozó, a biztosan egyedi ID generálására.

Metódusok

Az osztály metódusain kívül nincs más szolgáltatása.

6.1.12 Inverter

Felelősség

Gate absztrakt osztályt valósítja meg, bemeneti értékeit lekérdezi (wireIn) a GetValue() metódussal, majd előállítja a kimeneti értéket (wireOut) a logika táblája alapján és beállítja azt, a SetValue(int) segítségével. Csak egyetlen bemenete lehet.

Össztályok

DigitalObject->Gate

Interfészek

Nincs.

Attribútumok

int INVERTERCounts: Osztályra jellemző statikus tagváltozó, a biztosan egyedi ID generálására.

Metódusok

Az ősosztály metódusain kívül nincs más szolgáltatása.

6.1.13 bhdIParser

Felelősség

A BHDL fájl beolvasása, értelmezése, áramkör felépítése.

Ősosztályok

Nincs.

Interfészek

Nincs.

Attribútumok

Metódusok

Wire CreateWire(Composit owner, String command): Létrehoz egy Wire objektumot.

LED CreateLed(Composit owner, String command): Létrehoz egy LED objektumot.

Oscilloscope CreateOscilloscope(Composit owner, String command): Létrehoz egy Oscilloscope objektumot.

GENERATOR CreateGenerator(Composit owner, String command): Létrehoz egy GENERATOR objektumot.

SWITCH CreateSwitch(Composit owner, String command): Létrehoz egy SWITCH objektumot.

Wire assign(Composit owner, String command): Összeköt két DigitalObject-et.

String Infix2Postfix(String InfixExpression): A paraméterben megadott kifejezést alakítja át inorder kifejezésből postorder (lengyel módszer) kifejezésbe.

int Precedence(String ch): Ez határozza meg a paraméterben megadott művelet fontosságát.

int NumOfOperand(String ch): Visszaadja, hogy hány operandust vár az adott operátor (Gate objektumok típusa szerint). "&,|" esetén 2, "!" esetén 1.

boolean isOperandChar(String ch): Igazgal tér vissza, ha a paramétere "&|!,()," értékű

boolean isOperator(String ch): Igazgal tér vissza, ha a paramétere "&,|" értékű

boolean isInverter(String ch): Igazgal tér vissza, ha a paramétere "!" értékű

boolean isLeftParental(String ch): Igazgal tér vissza, ha a paramétere "(" értékű

boolean isRightParental (String ch): Igazgal tér vissza, ha a paramétere ")" értékű

Composit CreateComposit(Composit owner, String source, String command): Létrehoz egy Composit objektumot és értelmezi a tartalmát.

boolean SettingElement(Composit owner, String command): Ha a BHDL fájlban kezdőérték van megadva egy elemnek, akkor ez állítja be a létrehozott objektumot.

String matching(String expr, String source): Ellenőrzi, hogy a megadott szövegben található-e reguláris kifejezésnek megfeleltethető részlet, ha igen, akkor visszatér vele.

String remove_CR_LF(String source): eltávolít minden sortörés és kocszi vissza karaktert a forrás szövegből.

String remove_Spaces(String source): eltávolít minden felesleges szóközt a forrás szövegből.

String FindComposite(String source, String compname): Megkeresi és visszaadja a következő Composit-ot a forrás szövegből.

String FindMainComposit(String source): Megkeresi a main(in,out) szignóval ellátott Composit-ot.

Composit CreateMain(String composit): Létrehozza a MainComposit objektumot.

Composit ReadComposit(Composit ThisComposit, String source, String composit_name): Egy Composit tartalmát olvassa be és értelmezi.

String[] getCommands(String bhdLcomposit): Egy Composit BHDL leírásából tömbbe rendezi a benne lévő parancsokat.

void CommandParser(Composit Owner, String source, String[] commands): Értelmez egy BHDL parancsot. A sorokra mintaillesztést végez, és ha talál egy mintát, a parancsnak megfelelően jár el (függvényeket hív).

6.1.14 PIN

Felelősség

Interfészt nyújt a Composit objektumba beérkező illetve kimenő drótok számára.

Ősosztályok

DigitalObject

Interfészek

Nincs.

Attribútumok

Composit ContainerComposit: Composit objektum, mely tartalmazza a PIN objektumot.

Wire wireIn: A PIN-be beérkező Wire objektum.

Wire wireOut: A PIN-ből kimenő Wire objektum.

Metódusok

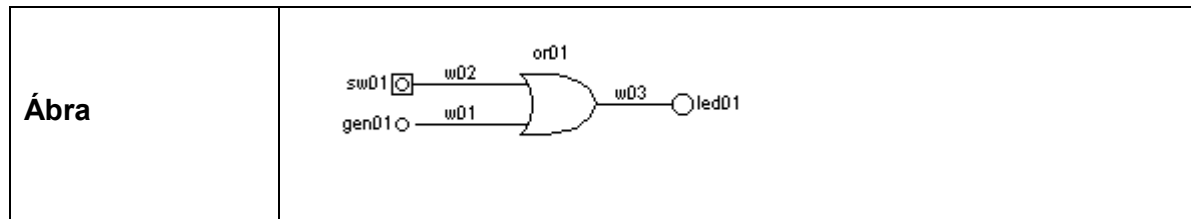
Az ősosztály metódusain kívül nincs más szolgáltatása.

6.2 A tesztek részletes tervei, leírásuk a teszt nyelvén

Egyszerű áramkör

Teszt neve	Egyszerű áramkör
------------	------------------

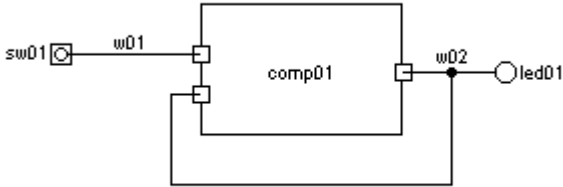
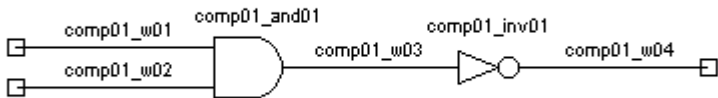
Rövid leírás	Egyszerű áramkör tesztelése, mely egy kapcsolóból, egy jelgenerátorból, egy VAGY kapuból és egy LED-ből áll.
Teszt célja	A hálózatot felépítő elemek tesztelése.
Bemenet	<pre>loadBoard teszt1.bhdl --- toggleSwitch main#SWITCH#sw01 --- setSequence main#GENERATOR#gen01 0110 --- run --- stop</pre>
Kimenet	<pre>create main (Composit). create sw01 (SWITCH). create gen01 (GENERATOR). create led01 (LED). create Wire@eb166b50 (Wire). create Wire@20cdfc9c1 (Wire). create ORGate0 (ORGate). create Wire@338376972 (Wire). teszt1.bhdl is loaded --- sw01's value changed --- gen01's Sequence is set to 0110 --- Simulation is running <Wire@eb166b51> value is 0 <Wire@79616c70> value is 1 <Wire@eb166b51> value is 0 <Wire@79616c70> value is 1 <Wire@20cdfc9c2> value is 1 <led01> value is 1 <Wire@eb166b51> value is 0 <Wire@79616c70> value is 1 <Wire@eb166b51> value is 0 <Wire@79616c70> value is 1 <Wire@20cdfc9c2> value is 1 <led01> value is 1 --- Simulation stopped</pre>



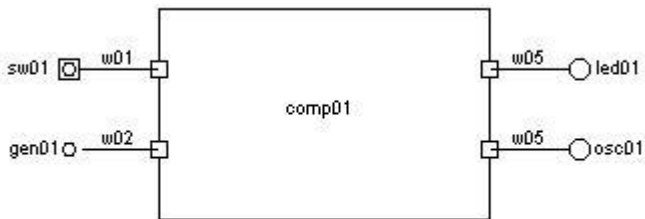
Visszacsatolt stabil áramkör

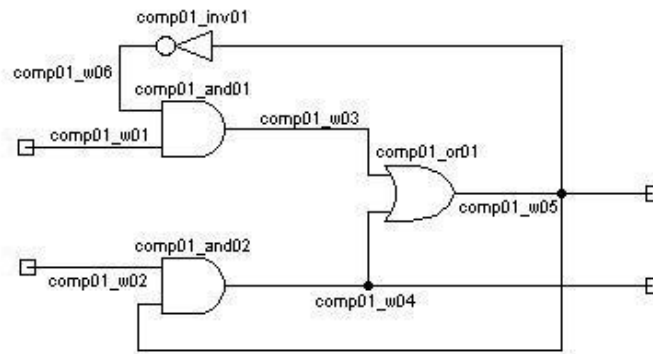
Teszt neve	<i>Visszacsatolt stabil áramkör</i>
Rövid leírás	Visszacsatolt áramkör tesztelése, meg egy kapcsolóból, egy ÉS kapuból és egy oszcilloszkópból áll.
Teszt célja	Visszacsatolás, illetve további építőelemek tesztelése.
Bemenet	<pre> loadBoard teszt1.bhdl --- toggleSwitch main#SWITCH#sw01 --- setSample main#Oscilloscope#osc01 10 --- run --- stop </pre>
Kimenet	<pre> create main (Composit). create sw01 (SWITCH). create osc01 (Oscilloscope). create w02 (Wire). create Wire@55fdc96c3 (Wire). create ANDGate0 (ANDGate). create Wire@8b2fd8f4 (Wire). teszt2.bhdl is loaded --- sw01's value changed --- osc01's Sample size is set to 10 --- Simulation is running </pre>
Ábra	<pre> graph LR sw01[sw01] --- w01[w01] w01 --- and01[and01] and01 --- w02[w02] w02 --- osc01[osc01] w02 --- and01 </pre>

Instabil áramkör

Teszt neve	<i>Instabil áramkör</i>
Rövid leírás	Instabil áramkör tesztelése, mely áll egy kapcsolóból, egy kompozit elemből és egy LED-ből. A kompozit elem belsejében egy ÉS kapu és egy inverter van.
Teszt célja	Instabil hálózat tesztelése, azaz instabilitás felfedezése, valamint kompozit elem tesztelése.
Bemenet	<pre>loadBoard teszt1.bhdl --- toggleSwitch main#SWITCH#sw01 --- run --- stop</pre>
Kimenet	<pre>create main (Composit). create sw01 (SWITCH). create led01 (LED). create w01 (Wire). create comp01 (Composit). create w01 (Wire). create w02 (Wire). create w03 (Wire). create ANDGate1 (ANDGate). create Wire@1f24bbbf5 (Wire). create INVERTER0 (INVERTER). create Wire@24a208926 (Wire). create Wire@9e0bc087 (Wire). teszt3.bhdl is loaded</pre>
Ábra	 <p>A kompozit belülről:</p> 

Bonyolult áramkör

Teszt neve	<i>Bonyolult áramkör</i>
Rövid leírás	A megbeszélés alatti közös feladat letesztelése
Teszt célja	A működés bizonyítása egy bonyolult áramkörön.
Bemenet:	loadBoard teszt4.bhdl
Kimenet:	<pre> create main (Composit). create sw01 (SWITCH). create gen01 (GENERATOR). create led01 (LED). create osc01 (Oscilloscope). create comp01 (Composit). create w01 (Wire). create w02 (Wire). create w03 (Wire). create w04 (Wire). create feedback (Wire). create INVERTER1 (INVERTER). create Wire@4f037c718 (Wire). create ANDGate2 (ANDGate). create Wire@6ddf073d9 (Wire). create ANDGate3 (ANDGate). create Wire@4d546e2510 (Wire). create ORGate1 (ORGate). create Wire@620b66cc11 (Wire). create to_w04 (Wire). create Wire@5a8a0d5d12 (Wire). create Wire@1d73831b13 (Wire). create Wire@aa470b814 (Wire). create Wire@31e4457d15 (Wire). </pre> <p>teszt4.bhdl is loaded</p>
Ábra:	 <p>A kompozit belülről:</p>



6.3 A tesztelést támogató programok tervei

A program otthoni fordításához az *Eclipse IDE*, a teszteléshez a *NetBeans IDE* fordítóját használjuk. A tesztelés fordításának és a program futtatásának megkönnyítésére írt *.bat fájlokat *Notepad* illetve *Notepad++* programokkal hoztuk létre, szerkesztettük.

A tesztelés sikerességének eldöntéséhez, azaz, hogy a kimenet megegyezik az elvárt kimenettel, a *Total Commander* 'összehasonlítás tartalomra' parancsát használjuk. Ennek hatására a két szövegfájl (a kimentett kimenet és a begépelte várt kimenet) különbsége ki lesz emelve. A teszt sikeresnek mondható, ha nincs egyik szövegben sem kiemelés.

6.4 Napló

Kezdés	Vége	Munka megnevezése	Résztevők	Időtartam (óra:perc)
2011.04.01 12:00:00	2011.04.01 13:30:00	8. Megbeszélés a tesztelési tervről, feladatkiosztás	Csomák, Sziklay, Vad, Wiesner	01:30
2011.04.02 17:30:00	2011.04.02 21:30:00	Tesztelési terv, kimenetek, bemenetek	Csomák	04:00
2011.04.03 22:00:00	2011.04.04 2:00:00	Részletes tervek, Dokumentáció	Sziklay	04:00
2011.04.04 1:00:00	2011.04.05 2:00:00	Dokumentáció, Ellenőrzés	Wiesner	01:00

7. Prototípus beadása

7.1 Fordítási és futtatási útmutató

7.1.1 Fájllista

Fájl neve	Keletkezés ideje	Tartalom
ANDGate.java	2011. 04. 22	ÉS kaput megvalósító osztályt tartalmazza.
bhdlParser.java	2011. 04. 22	bhdl kiterjesztésű, hálózat felépítését leíró fájlok beolvasását elvégző osztályt tartalmazza.
Composit.java	2011. 04. 22	Kompozit objektumot megvalósító osztályt.
ExceptionsWithConnection.java	2011. 04. 22	Elemek összeköttetésénél felmerülhető kivételek.
DigitalBoard.java	2011. 04. 22	Áramkört megvalósító osztály.
DigitalObject.java	2011. 04. 22	Hálózati elemeket megvalósító absztrakt osztály.
ExceptionElementInputSize.java	2011. 04. 22	Kevés vezeték van az elem bemenetére kapcsolva kivétel..
ExceptionElementNotConnected.java	2011. 04. 22	Nincs összekötve az elem kivétel.
ExceptionObjectNotFound.java	2011. 04. 22	Objektum nem található kivétel.
ExceptionElementHasNoInputs.java	2011. 04. 22	Az elem bementére nincsen vezeték kapcsolva kivétel.
ExceptionUnstableCircuit.java	2011. 04. 22	Instabil áramkör kivétel.
ExceptionWireHasMultipleInputs.java	2011. 04. 22	Vezeték többszörös bemenettel rendelkezik kivétel.
ExceptionWrongBoard.java	2011. 04. 22	Hibás áramkör kivétel.
ExceptionWrongParameter.java	2011. 04. 22	Hibás paraméter kivétel.
Gate.java	2011. 04. 22	Kapuk absztrakt osztálya.
GENERATOR.java	2011. 04. 22	Generátort megvalósító osztály.
Input.java	2011. 04. 22	Jelet generáló elemek absztrakt osztálya.
INVERTER.java	2011. 04. 22	Invertert megvalósító osztály.
LED.java	2011. 04. 22	Led-et megvalósító osztály.
Logger.java	2011. 04. 22	Eseménynaplózást megvalósító osztály.
Main.java	2011. 04. 22	A prototípus elindítását, és a tesztelővel való kommunikációt valósítja meg.
ORGate.java	2011. 04. 22	VAGY kaput megvalósító osztály.
Oscilloscope.java	2011. 04. 22	Oszilloszkópot megvalósító osztály.
Output.java	2011. 04. 22	Jelmegjelenítő elemek absztrakt osztálya.
PIN.java	2011. 04. 22	A kompozit csatlakozását megvalósító elem osztálya.
Status.java	2011. 04. 22	Hálózat állapotait tartalmazó változó.
SWITCH.java	2011. 04. 22	Kapcsolót megvalósító osztály.

Wire.java	2011. 04. 22	Vezetéket megvalósító osztály.
teszt1.bhdl	2011. 04. 22	Egyszerű áramkör leírása.
teszt2.bhdl	2011. 04. 22	Visszacsatolást tartalmazó áramkör leírása.
teszt3.bhdl	2011. 04. 22	Instabil áramkör leírása.
teszt4.bhdl	2011. 04. 22	Bonyolult áramkör leírása.
proto_build.bat	2011. 04. 22	Fordítási batch fájl.
proto_run.bat	2011. 04. 22	Futtatási batch fájl.

7.1.2 Fordítás

A fordítás a proto_build.bat batch fájl futtatásával kezdődik. Ezen belül be kell állítani a Java elérhetőségét, amely alapesetben megegyezik a HSZK gépeken az elérési úttal (PATH=D:\Program Files\Java\jdk1.6.0_14\bin). Eltérés esetén a tesztelőnek kell beállítani. A fájl létrehoz egy build, ezen belül egy classes mappát, melyet feltölt a programhoz használatos .class fájlokkal, valamint létrehoz egy dist mappát, melybe belekerül a proto.jar fájl, amit majd a proto_run.bat fájl használ a program futtatásánál.

7.1.3 Futtatás

Futtatáshoz a proto_run.bat futtatási batch file indításával kezdődik. Ennek előfeltétele, hogy megtörténjen a 10.1.2-ben leírtak szerint a programfordítása.

A program elindításával egy konzolképernyőt kapunk. A konzolon kiírásra kerül a parancsok listája, a parancsok szintaktikája, illetve a parancsok szöveges leírása.

7.2 Tesztek jegyzőkönyvei

7.2.1 Egyszerű áramkör

Teszt neve	<i>Egyszerű áramkör</i>
Tesztelők neve	Csomák Gábor, Vad Zsolt
Teszt időpontja	2011-04-17 22:50

7.2.2 Visszacsatolt stabil áramkör

Teszt neve	<i>Visszacsatolt stabil áramkör</i>
Tesztelők neve	Csomák Gábor, Vad Zsolt, Wiesner Péter
Teszt időpontja	2011-04-17 22:50

7.2.3 Instabil áramkör

Teszt neve	<i>Instabil áramkör</i>
Tesztelők neve	Csomák Gábor, Wiesner Péter
Teszt időpontja	2011-04-17 22:50

7.2.4 Bonyolult áramkör

Teszt neve	<i>Bonyolult áramkör</i>
Tesztelők neve	Csomák Gábor, Wiesner Péter
Teszt időpontja	2011-04-17 22:50

7.3 Értékelés

Tag neve	Munka százalékban
Csomák Gábor	20%
Jégh Tamás	20%
Sziklay György	20%
Vad Zsolt	20%
Wiesner Péter	20%

7.4 Napló

Kezdés	Munka megnevezése	Résztevők	Időtartam (óra:perc)
2011. 04.10. 10:00	Hierarchiaépítő függvény készítése	Csomák	04:00
2011. 04.16. 12:00	Értekezlet	Csomák, Sziklay, Vad, Wiesner	04:00
2011. 04.17. 10:00	bhdl értelmezés, végrehajtás	Sziklay	01:00
2011. 04.17. 12:00	bhdl értelmezés, végrehajtás (folyt.)	Sziklay	05:30
2011. 04.17. 12:00	FileParse befejezése	Jégh	02:00
2011. 04.17. 10:00	Hierarchiaépítés véglegesítése, tesztelése	Csomák	05:00
2011. 04.17. 20:00	Tesztelés, input nyelv	Wiesner	07:00
2011. 04.17. 22:00	Tesztelés, kód véglegesítése, dokumentáció	Csomák	07:00
2011. 04.18. 16:00	dokumentáció	Vad	02:00
2011. 04. 18. 19:00	programozás	Csomák, Jégh, Sziklay, Vad, Wiesner	10:00
2011. 04. 19. 19:00	programozás	Csomák, Jégh, Sziklay, Vad, Wiesner	10:00

7.5 Változtatások

Az osztályok közé bekerül a **Logger** osztály.

Felelősség

Statikus attribútumai és statikus függvényei segítségével a program naplózhatja a rendszerben történt változásokat, eseményeket.

Ősosztályok

Nincs

Interfészek

Nincs.

Attribútumok

- **log_levels logging_level:** A naplózás szintjét, részletességét tudjuk ezzel az enumerációval szabályozni.
- **int log_mode:** Ezzel a változóval szabályozhatjuk, hogy a standard kimenetre (0) vagy fájlba naplózunk (1). Esetleg mindkettőbe (2).
- **String log_file:** A fájl neve, amibe naplózunk.

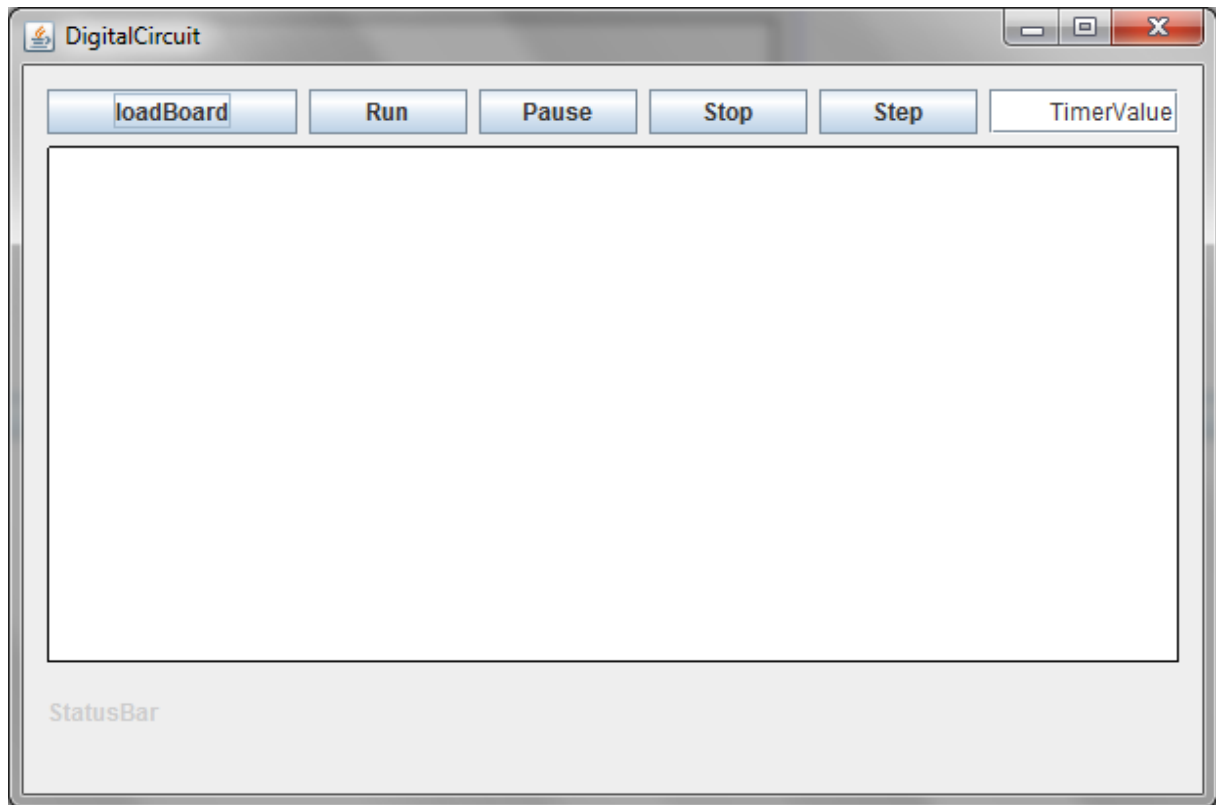
Metódusok

- **static void Log(log_type prio,String InputForLog):** A megadott típusú naplózást elvégezzük. Ha a log_mode 0, akkor standard kimenetre, ha 1, akkor fájlba, ha 2, akkor standard kimenetre és fájlba is naplózunk.
- **static void WriteToFile(String InputForLog):** Ennek a függvénynek a segítségével naplózhatunk fájlba.
- **static void WriteToStandardOutput(String InputForLog):** Ennek a függvénynek a segítségével naplózhatunk a standard kimenetre.

8. Grafikus felület specifikációja

8.1 A grafikus interfész

A program grafikus felületének alapállapotú látványterve a következő:



A felület szerkezete az egyszerűsége törekszik. Felső sorban találhatóak a parancs gombok. alsó sorban a státusz sor és középen a digitális áramkört megjelenítő vászon. Az alsó és felső panel minél kisebbre lett tervezve, hogy ne vonja el a figyelmet és a helyet az áramkör szimulációját mutató paneltől. A fent felsorolt strukturális elrendezésen kívül a megjelenítés 7 részre bontható, ezek a következők:

loadBoard – gomb. Erre való kattintással egy fájlkiválasztó panel jelenik meg, ahol a felhasználó megadhatja, hogy melyik digitális áramkör .bhd1 fájlt szeretné megnyitni, szimulálni. A betöltés után a fő panelon (középen) megjelenik az áramkör képe.

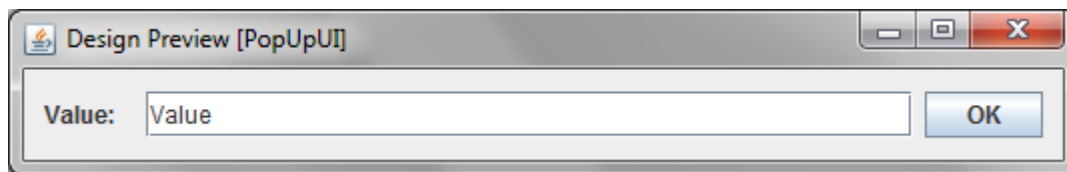
Run – gomb. Erre kattintva indítható el a szimuláció. Az információ folyását figyelemmel lehet követni a fő panelon. Ha az áramkör szimulációját felfüggesztették, ezzel a gombbal lehet azt folytatni.

Pause – gomb. A Pause gomb megnyomása után az áramkör szimulációja fel lesz függesztve. Az áramköri elemek megőrzik pillanatnyi állapotukat.

Stop – gomb. Ennek hatására leállítható az éppen futó szimuláció. Az áramkör alaphelyzetbe kerül.

Step – gomb. Ennek a gombnak a megnyomásával a felhasználó léptetheti egy futási egységgel az áramkört. Tesztelési célokra alkalmas főleg.

Szimulációs tér (fő panel). Ezen a panelon láthatóak a betöltött áramkör elemei. Ezek a mérnöki körökben ismert áramköri jelölésekkel vannak kirajzolva, bemeneteikkel és kimeneteikkel is jelölve (félvezeték vagy gombóc). A vezetékek szögletes vonalakkal vannak reprezentálva, az általa szállított információ a vezeték színe alapján fejthető vissza (fekete - 1, szürke - 0). A logikai kapuk fehér, fekete szegélyes téglalapként vannak ábrázolva. Közepükön '&' jel, ha AND, '>=1', ha OR kapukról van szó. A kapcsoló alakja egy négyzet benne egy körrel. A felhasználó erre kattintva állíthatja a kapcsoló állását. A generátor egy fehér téglalap 'Generator' felirattal, a szekvencia értéke a duplakattintással előtűnő panelban adható meg, melynek grafikus látványterve itt található:



A LED áramköri elem egy üres körként van megjelenítve, ha bemenő vezetékének értéke 0 és telivel, ha annak értéke 1. Az oszcilloszkóp elem egy fehér téglalappal van jelölve, melynek közepén egy logikai órajel egy periódusa található. Az elemen való dupla kattintással előjön egy panel, amiben láthatóak a vezeték korábbi értékei (idő – érték) alakban, táblázatban jelölve. A kompozit elem is egy téglalapként van reprezentálva közepén a kompozit elem nevével.

StatusBar (szöveges mező). Itt kerülnek kiírásra a program és a rendszer állapotával kapcsolatos fontosabb információk. (Teszt01.bhdl betöltve, Szimuláció fut, stb.)

Az összes beállítható értéket, mint például szekvencia, frekvencia, oszcilloszkóp buffer mérete, interval , beállítása felugró panelban adhatjuk meg. A címke jelzi a beállítható érték nevét, a szövegbeviteli mezőben adhatjuk meg az értéket.

TimerValue – mező. A Timer időzítését állíthatjuk be.

8.2 A grafikus rendszer architektúrája

8.2.1 A felület működési elve

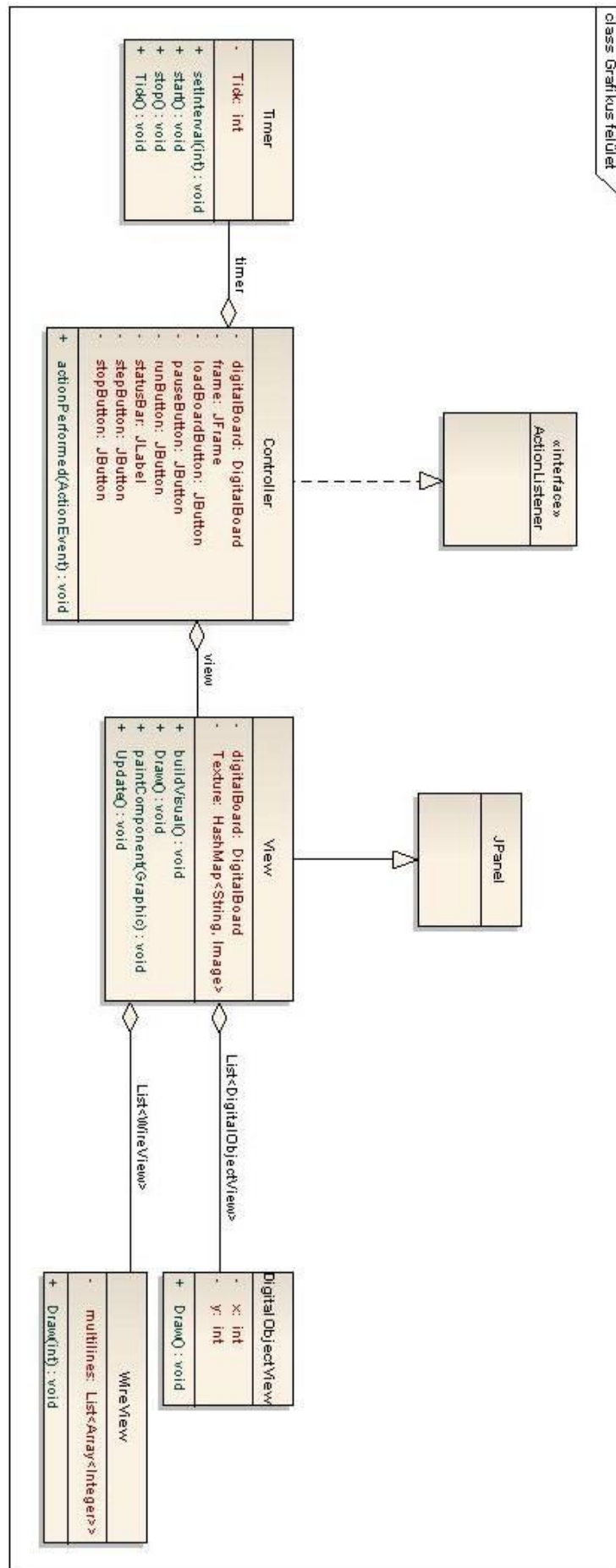
A program grafikus verziója a Model-View-Controller tervezési minta szerint épül fel. Ez a minta alkalmas arra, hogy a program belső adatait elválassza azok konkrét grafikus reprezentációjától, így a kapott program olyan részmodulokból áll, melyek interfészeken keresztül kommunikálnak, és a lehető legkevésbé függenek egymástól - ez az architektúra megfelel a specifikációban korábban deklarált irányelveknek, miszerint a kész programnak modulárisnak, annak részeinek pedig egymástól függetlennek kell lenniük. Megjegyezendő még, hogy a felhasználó által kiadható parancsokat az ActionPerformed , Command vizsgálat tervezési mintával kezeljük le.

Model (modell): Ez a réteg felelős a program belső állapotának, adatainak tárolásáért és kezeléséért. A modell tartalmazza továbbá az alkalmazásban elérhető szolgáltatások implementációját, és a belül megtörtént változásokról folyamatosan értesíti a nézetet.

View (nézet): Magát a modellt jeleníti meg a felhasználói felületen levő elemekkel, melyek lehetőséget adnak a felhasználónak a programmal való interakcióra.

Controller (vezérlő): Az alkalmazás futása során bekövetkező felhasználói műveleteket dolgozza fel, valamint egyszerre kommunikál a modell és nézet modulokkal, változásokra kényszerítve azokat.

8.2.2 A felület osztály-struktúrája



8.3 A grafikus objektumok felsorolása

8.3.1 View

Felelősség

Megjelenítésért felelős. Tárol egy példányt az összes osztály-specifikus megjelenítő segédosztályból, amiknek segítségével kirajzolja, az osztálynak megfelelő képet.

Ősosztályok

JPanel

Interfészek

Nincs.

Attribútumok

- **HashMap<String, Image> Texture:** Hash szerűen tároljuk a képeket.
- **List<DigitalObjectView> digitalObjectViews:** Lista a DigitalObjectView-k részére.
- **List<WireView> wireViews:** A WireView-okat tartalmazó lista.
- **DigitalBoard digitalboard:** Modell tartalmazó változó.

Metódusok

- + **buildVisual():** Az áramkör minden elemének digitalObject-jének kiszámolja az x,y koordinátáját, valamint minden wire-nek a törtvonal reprezentációit. Mindezt a file beolvasás alatt megépített hierarchiaszintezés segítségével algoritmikusan végzi.
- + **Draw():** kirajzolja az áramkör minden elemét. Az objektumokat hierarchia szerint kirajzolja, és az összes vezetéket is behúzza, értékének megfelelően.
- + **Update():** Minden vezetéket és kimenetet újrarajzol. Azért van szükség erre a függvényre, mivel csak a vezetékek és kimenetek változnak az egyes pályabetöltések óta.
- + **paintComponent(Graphics g):** Ha megváltozik a felhasználói felület, akkor meghívódik ez a metódus. A metódus újrarajzolja a vásznat.

Controller

Felelősség

A modell és a megjelenítés összekapcsolásáért, események kezeléséért felelős.

Ősosztályok

Nincs.

Interfészek

ActionListener

Attribútumok

- **DigitalBoard digitalboard:** A modellt tartalmazó változó.
- **View view:** A megjelenítésért felelős.
- **JFrame frame:** Az ablak, amelyben fut az alkalmazásunk.
- **Timer timer:** A futtatás alatti időzítésért felel.
- **JButton loadBoardButton:** Betölti a modellt (hálózatot).
- **JButton runButton:** Futtatja a hálózatot.

- **JButton pauseButton**: Megállítja a hálózat futását.
- **JButton stopButton**: Alaphelyzetbe állítja a hálózatot.
- **JButton stepButton**: Lépteti a hálózatot.
- **JLabel statusBar**: A program lefutása során előforduló események jelzése a felhasználó felé.

Metódusok

- + **void actionPerformed(ActionEvent e)**: A gombok által kiváltott események lekezelésére szolgál.

8.3.2 Timer

Felelősség

Feladata hogy értesítse a Controllert a beállított paraméternek megfelelő időközönként, amennyiben a Controller elindította.

Ősosztályok

Nincsenek

Interfészek

Nincsenek

Attribútumok

- **int tick**: Időzítéshez konstans mely a léptetési mértéket tárolja.

Metódusok

- **void Tick()**: Számláló, jelez a **Controller**-nek amennyiben letelt a kívánt időköz (*tick*).

+ **void setInterval(int interval)**: Beállítja a számláló időközét a paraméterként kapott értékre.

+ **void start()**: Elindítja a számlálót

+ **void stop()**: Leállítja a számlálót.

8.3.3 WireView

Felelősség

Lehetővé tegye és megvalósítsa az egyes vonalak kirajzolását

Össztályok

Nincsenek

Interfészek

Nincsenek

Attribútumok

- **List<Array<Integer>> multilines:** a törött vonal a wire grafikus reprezentációja, a listában a töröttvonal szakaszok végpontjainak x,y koordinátáit tárolja. Mivel egy wire-nek több outputja lehet, több törött vonalat kell tárolnunk.

Metódusok

+ **Draw(int value):** A paraméter értékének megfelelően szürke vagy fekete színűen kirajzolja a törött vonalakat a bemenetétől a kimenetéig

8.3.4 DigitalObjectView

Felelősség

Feladata hogy az egyes DigitalObjectekhez nyilvántartsa a megjelenítéshez szükséges adatokat, valamint kérésre kirajzolja az objektumot.

Ősosztályok

Nincsenek

Interfészek

Nincsenek

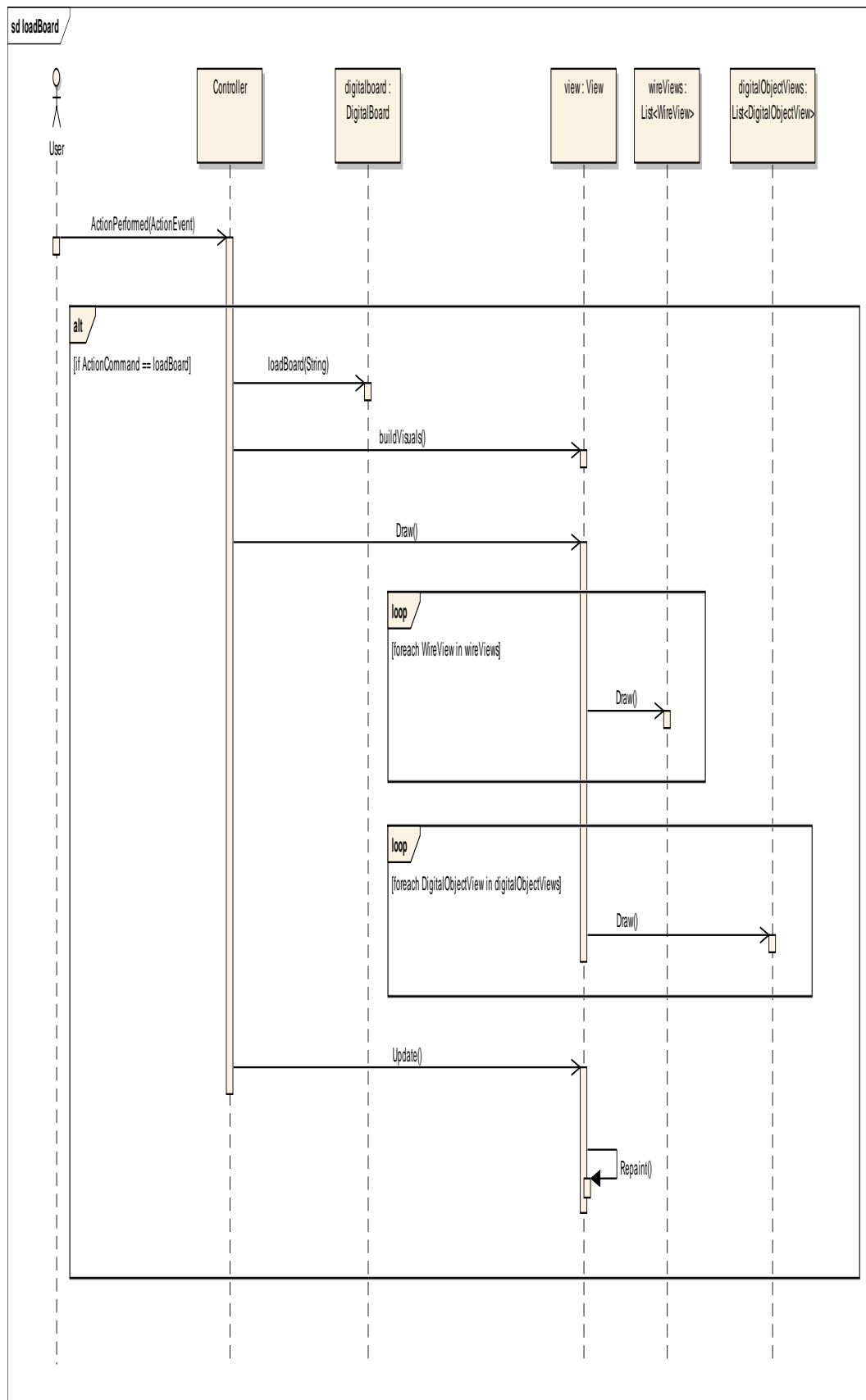
Attribútumok

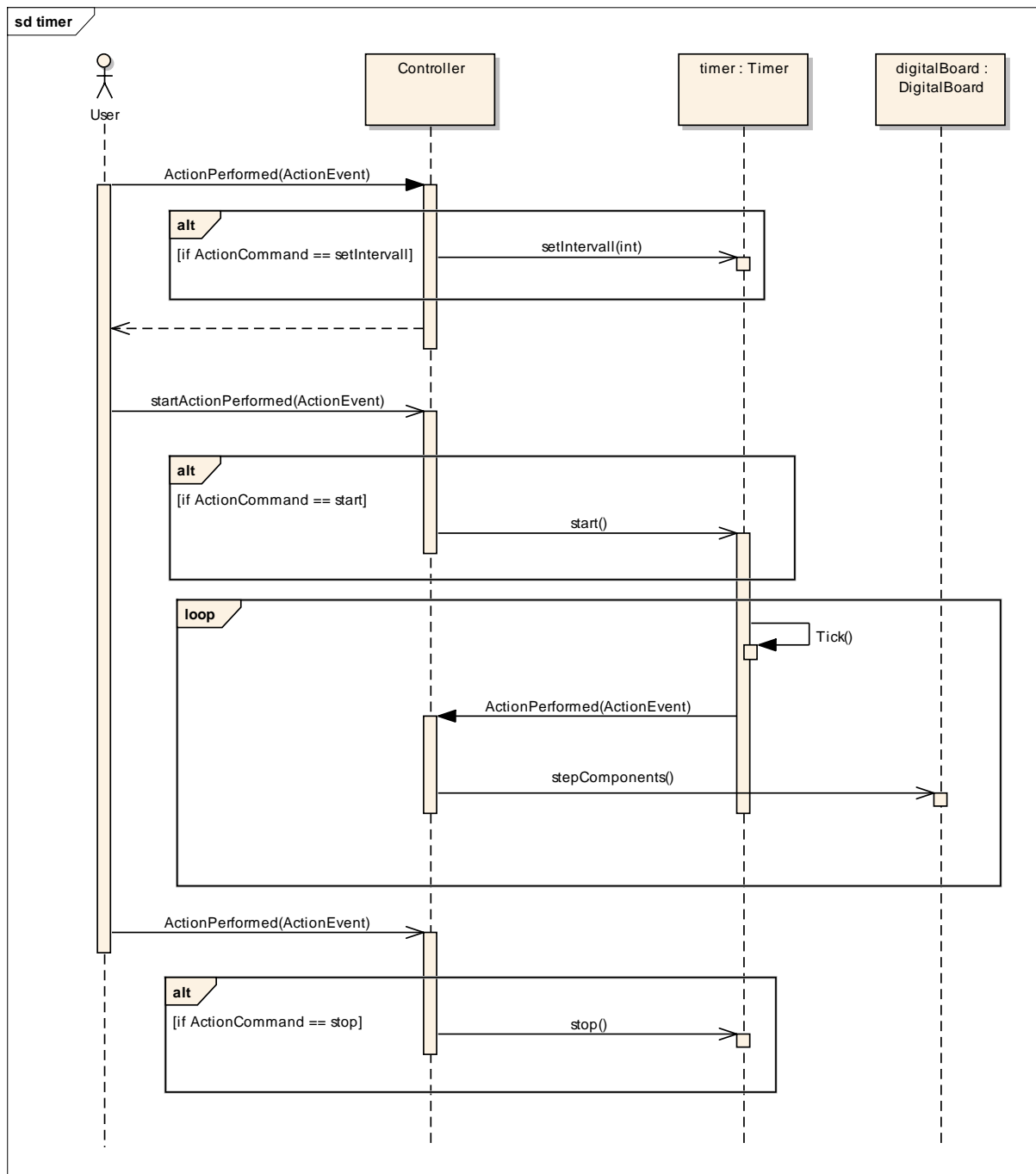
14. - **int x,y**: Az elem koordinátái

Metódusok

15. + **void Draw()**: DigitalObject kirajzolásához segédfüggvény.

8.4 Kapcsolat az alkalmazói rendszerrel





8.5 Napló

Kezdet	Időtartam	Résztevők	Leírás
2011.04.22. 18:00	1 óra	Csomák Jégh	Értekezlet, feladatkiosztás
2010.04.24. 10:30	3 óra	Csomák	Osztályok módosítása, létrehozása, leírása
2010. 04. 24. 20:00	2 óra	Csomák, Jégh, Sziklay, Vad, Wiesner	Grafikus objektumok
2010. 04. 24. 22:00	3 óra	Wiesner	Grafikus interfész
2010. 04. 24. 00:00	1 óra	Sziklay	A felület osztály struktúrája
2010. 04. 25. 01:00	3 óra	Jégh, Sziklay, Vad	Szekvencia diagramok, dokumentáció
2010. 04. 25. 04:00	1 óra	Vad, Wiesner	Dokumentáció véglegesítése

9. Grafikus változat beadása

9.1 Változások

A **View** osztály át lett nevezve **boardView**-ra.

Kikerült a magvalósításból a **WireView** osztály, mivel a vezetékeknek nem szükséges külön megjelenítése, egyszerűen csak ki kell rajzolni őket, ami a **boardView**ben lett megvalósítva.

DigitalObjectView kikerült az osztályok közül.

Új osztályt vettünk fel **viewElem** néven. A **boardView** tartalmaz egy **viewElem**-eket tartalmazó **ViewList** listát.

Felelősség: Segéd view class az elemek megjelenítéséhez. Tarolja az egyes elemek X, Y koordinátáit, továbbá az elem ID-jét, amely a kapcsolatot jelenti az elem és a hozzá tartozó View között

Ősosztályok: Nincs.

Interfész: Nincs.

Attribútumok:

- **public String ID:** Az elem ID-ja, amihez a View osztály tartozik.
- **public int X:** Az elem X koordinátája.
- **public int Y:** Az elem Y koordinátája.
- **public int szint:** Az elem szintje.
- **private ArrayList<viewElem> pins_in**
- **private ArrayList<viewElem> pins_out**
- **private int pinInNo:** Bemeneti pinek száma.
- **private int pinOuNo:** Kimeneti pinek száma.
- **public class ImagePanel extends JPanel:** Az elemhez tartozó képet tartalmazó tároló Component.
- **public ImagePanel ImageContainer**

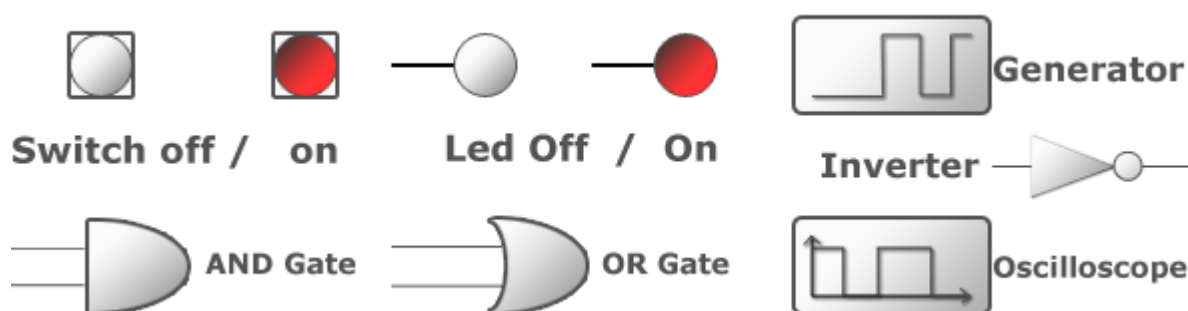
Metódusok:

- **public void addPinIn(viewElem e):** Hozzáadja a paraméterben kapott viewElemet a pins_in tömbhöz.
- **public void addPinOut(viewElem e):** Hozzáadja a paraméterben kapott viewElemet a pins_out tömbhöz.
- **public viewElem getNextPinIn():** Visszaadja a pin_in tömb pinInNo értékének megfelelő helyen lévő elemét.
- **public viewElem getNextPinOut():** Visszaadja a pin_out tömb pinOutNo értékének megfelelő helyen lévő elemét.

9.1.1 A grafikus interfész (bővítés)

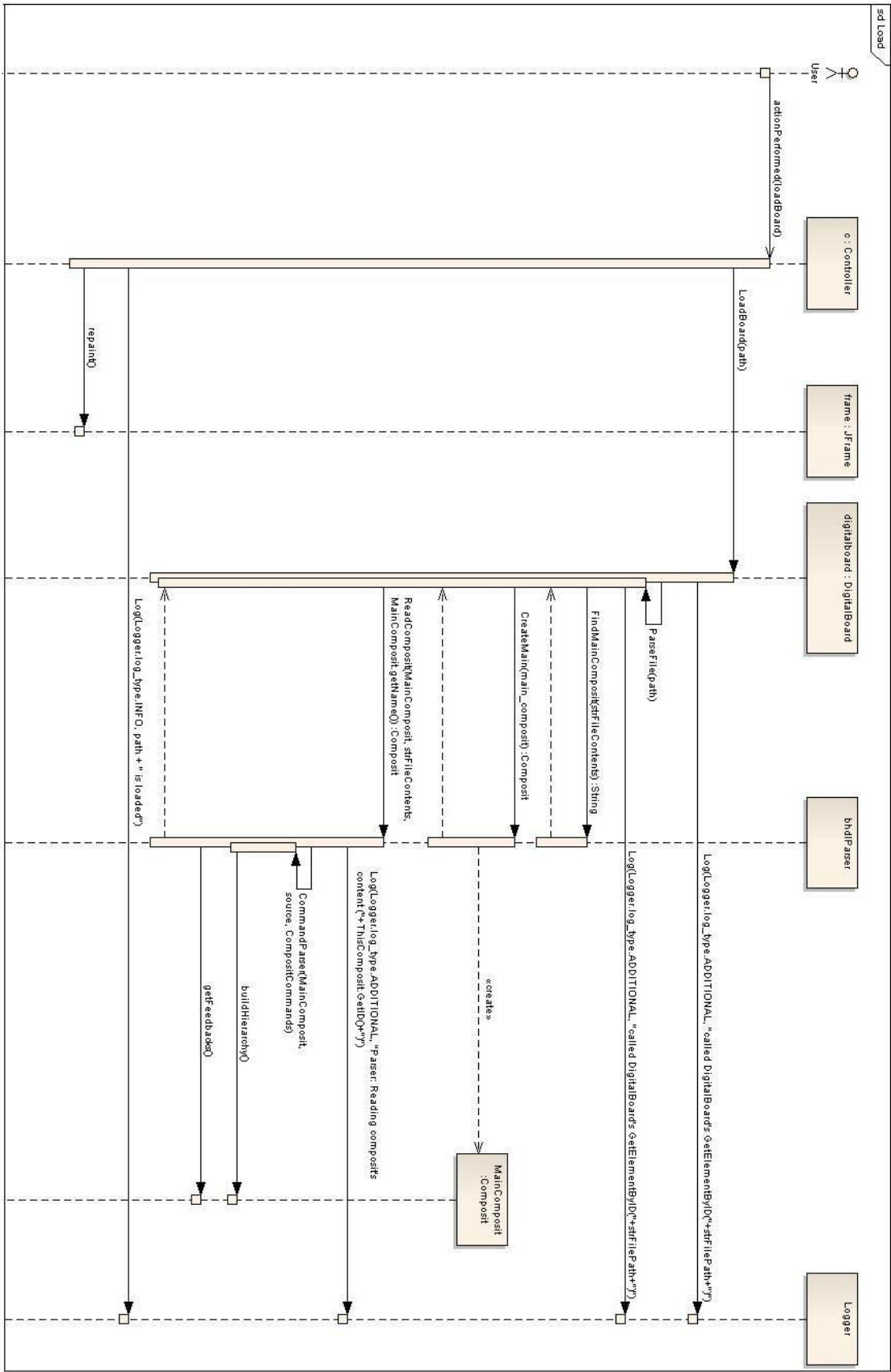
Exit – gomb. Ennek hatására kilép a program.

Szimulációs tér (fő panel). Ezen a fő panelon láthatóak a betöltött áramkör elemei. Ezek a mérnöki körökben ismert áramköri jelölésekkel vannak kirajzolva, bemeneteikkel és kimeneteikkel is jelölve (félvezeték vagy gombóc). A vezetékek szögletes vonalakkal vannak reprezentálva, az általa szállított információ a vezeték színe alapján fejthető vissza (fekete - 1, szürke - 0). A logikai kapuk és áramköri elemek képe az alsó ábrán látható. Az oszcilloszkóp elemen való dupla kattintással előjön egy panel, amiben láthatóak a vezeték korábbi értékei (idő – érték) alakban grafikonon jelölve, a generátor elem tulajdonságait is így lehet beállítani. A kompozit elem egy téglalapként van reprezentálva közepén a kompozit elem nevével. Fontos megjegyezni, hogy az itt megjelenő áramkör teljesen dinamikus épül fel, így a leíró fájlokban semmilyen pozícióra utaló paraméter nincs.

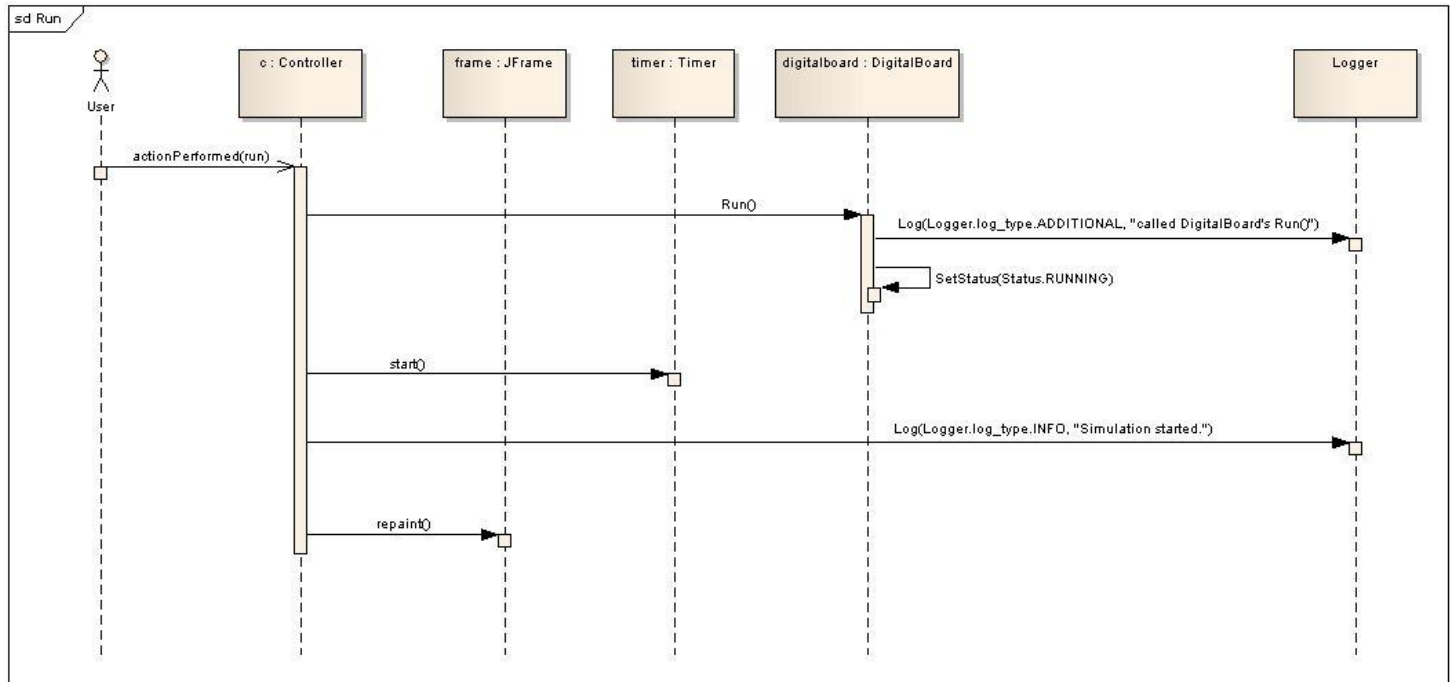


9.1.2 Szekvencia diagramok

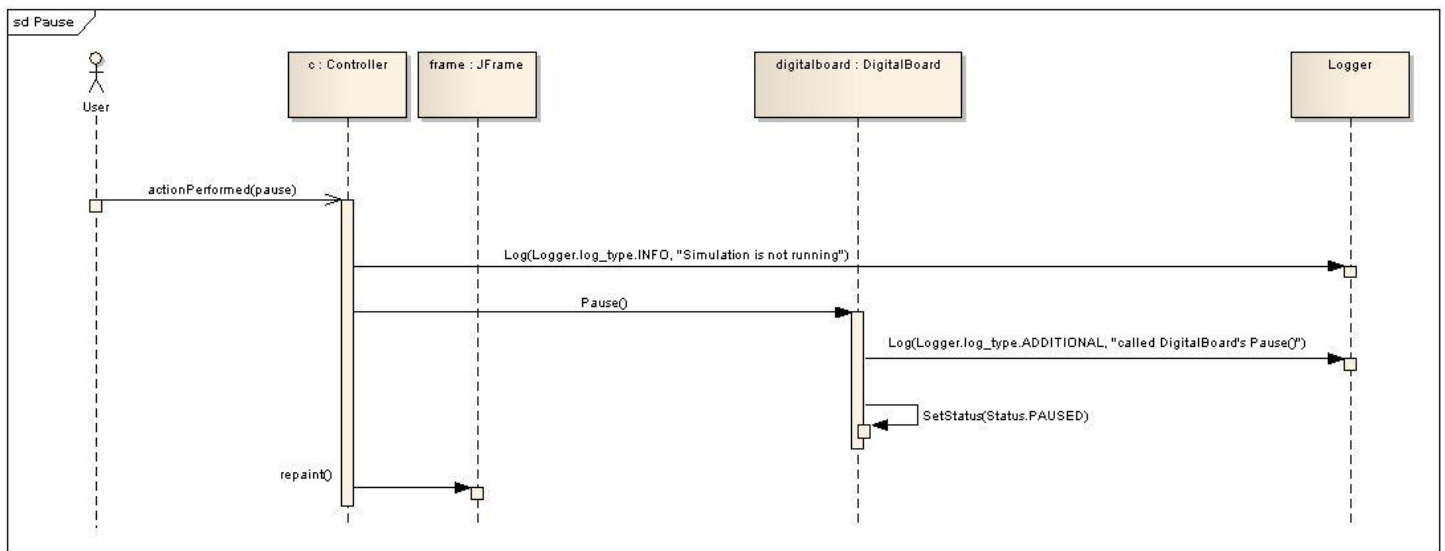
Load



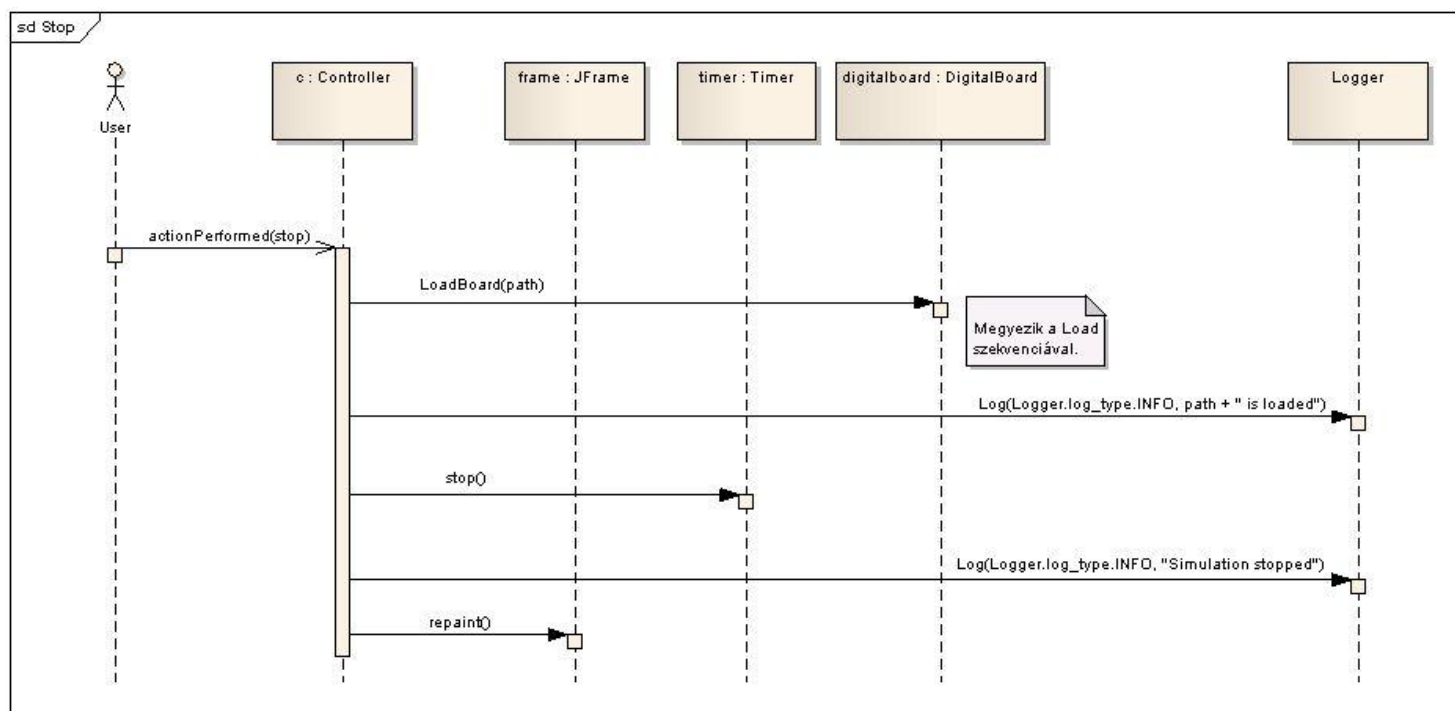
Run



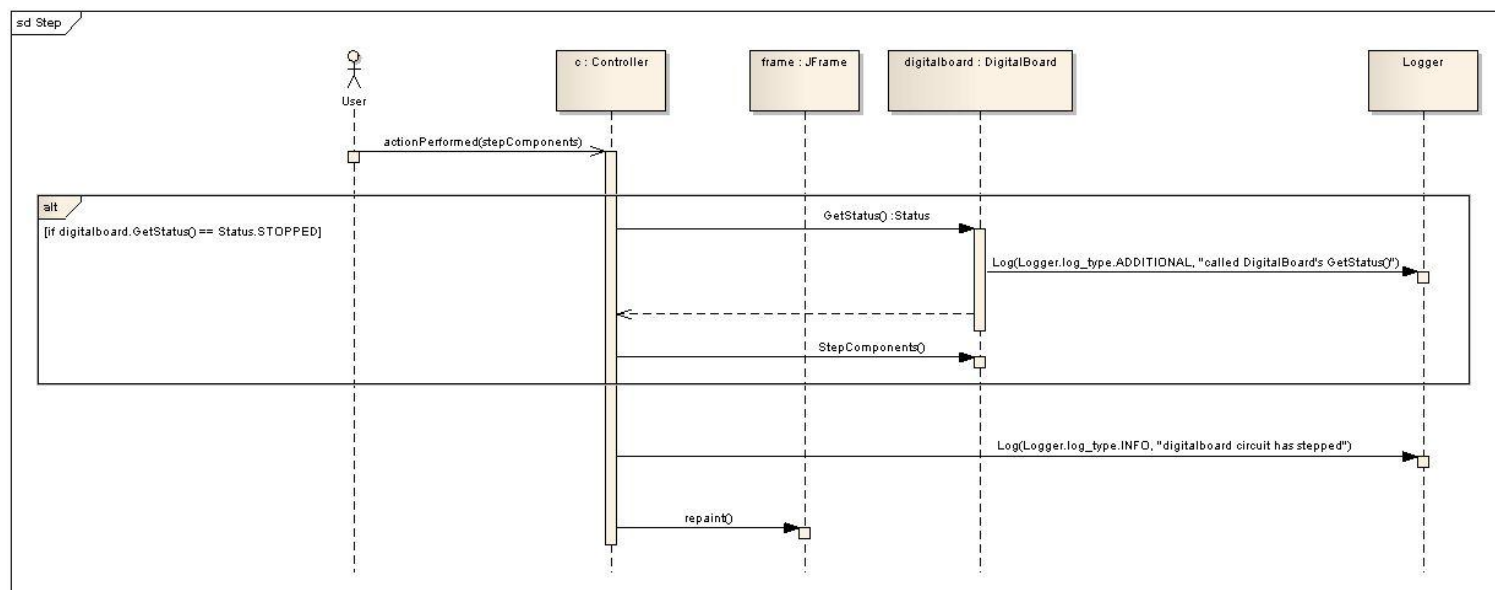
Pause

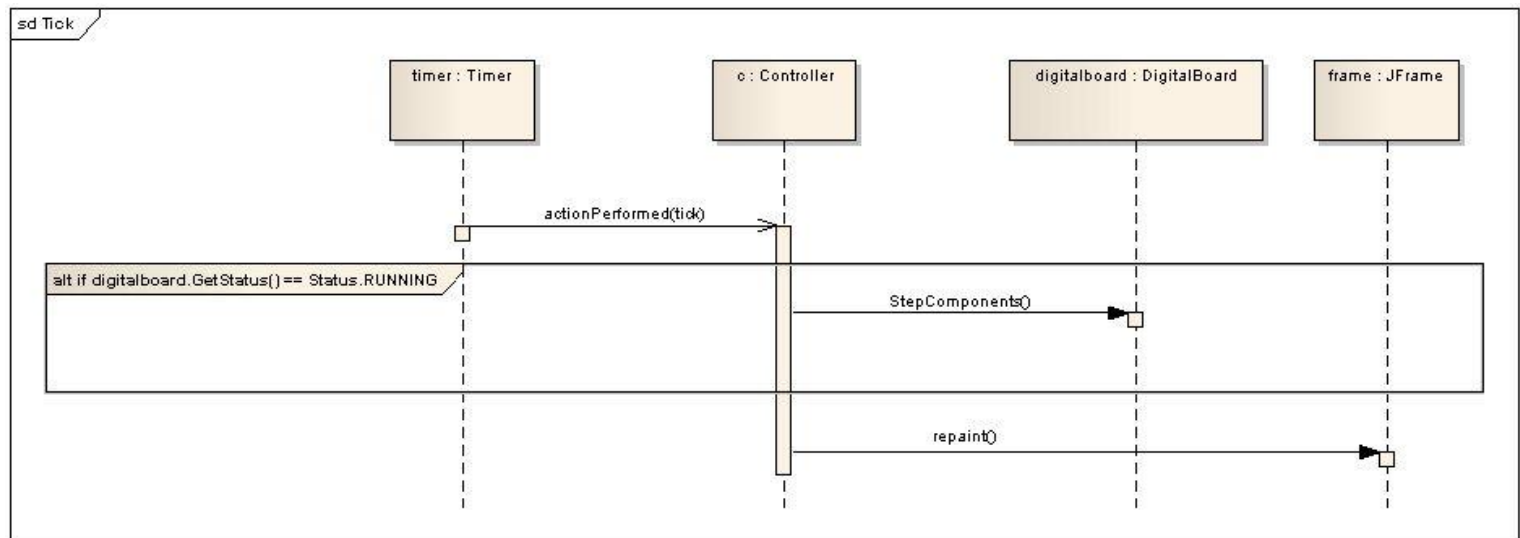
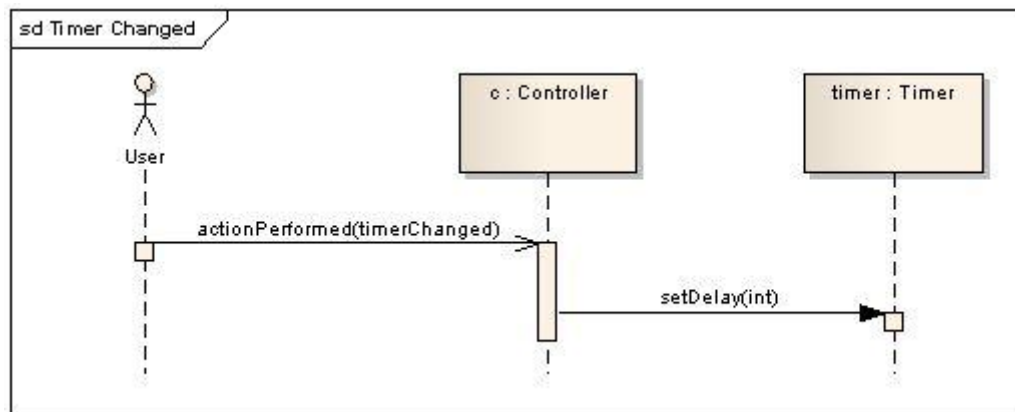


Stop

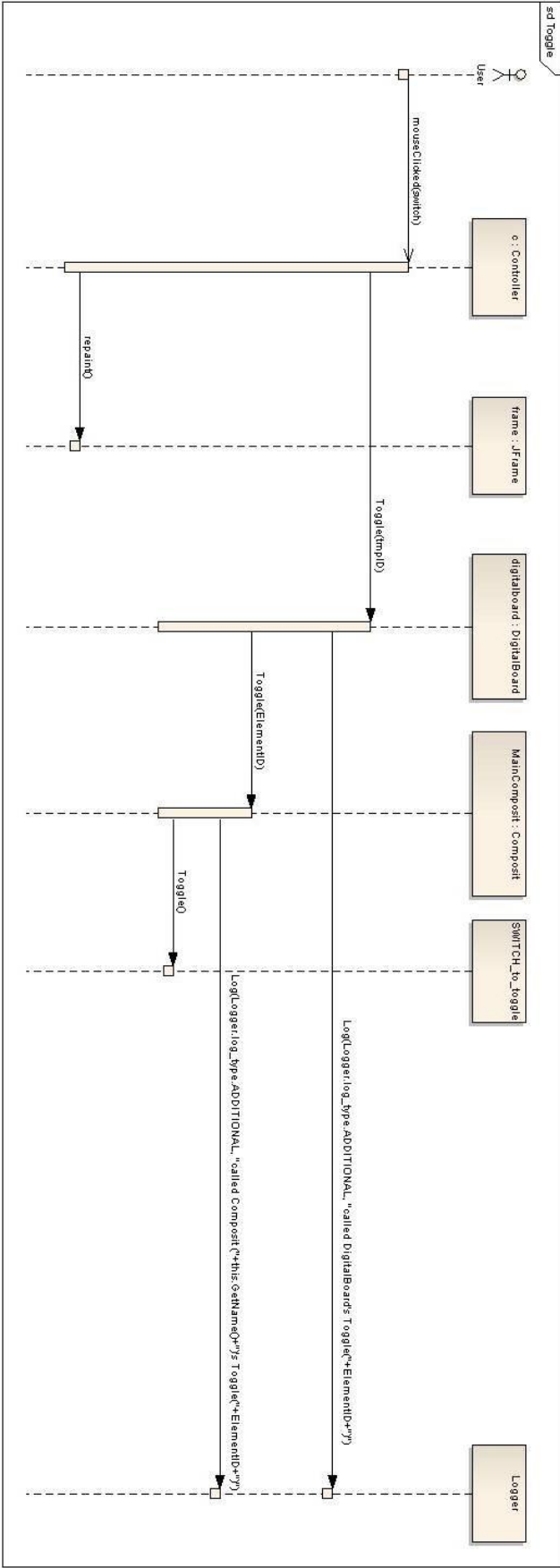


Step

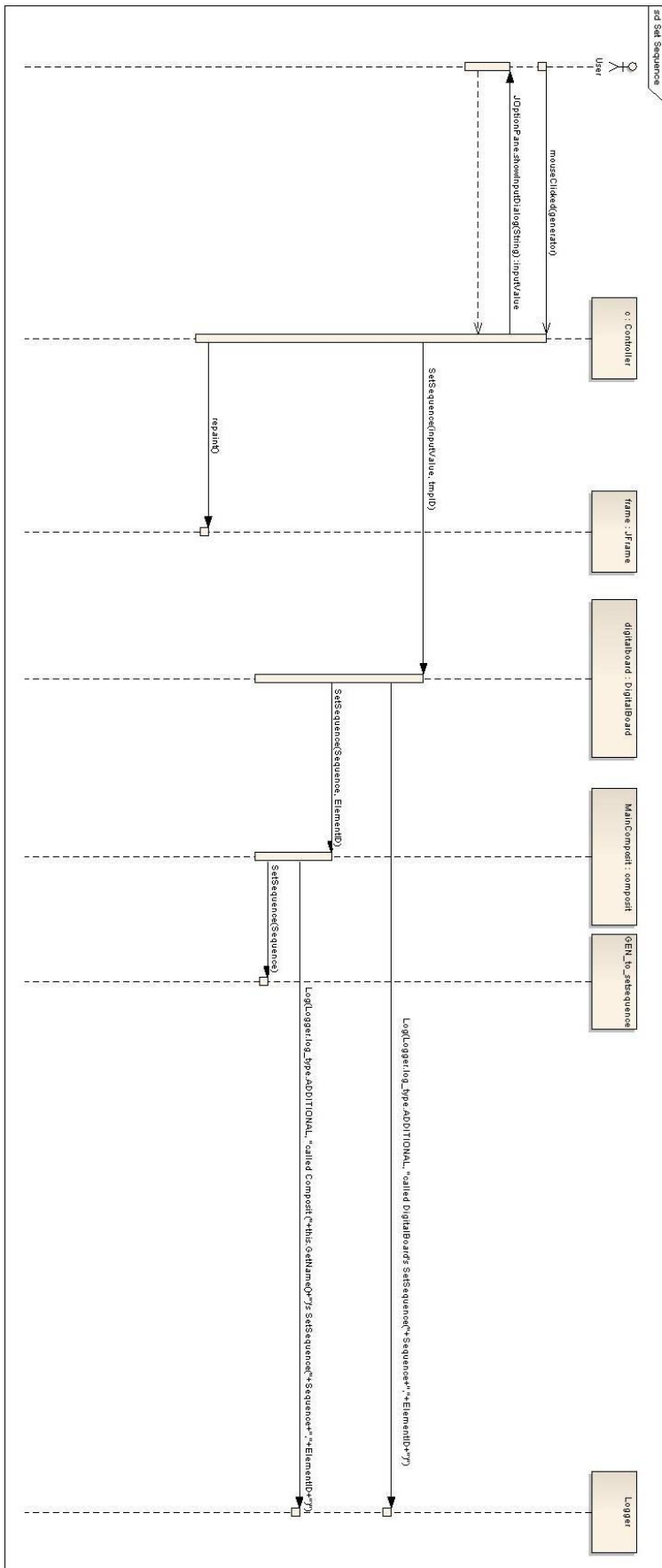


Tick**Timer Changed**

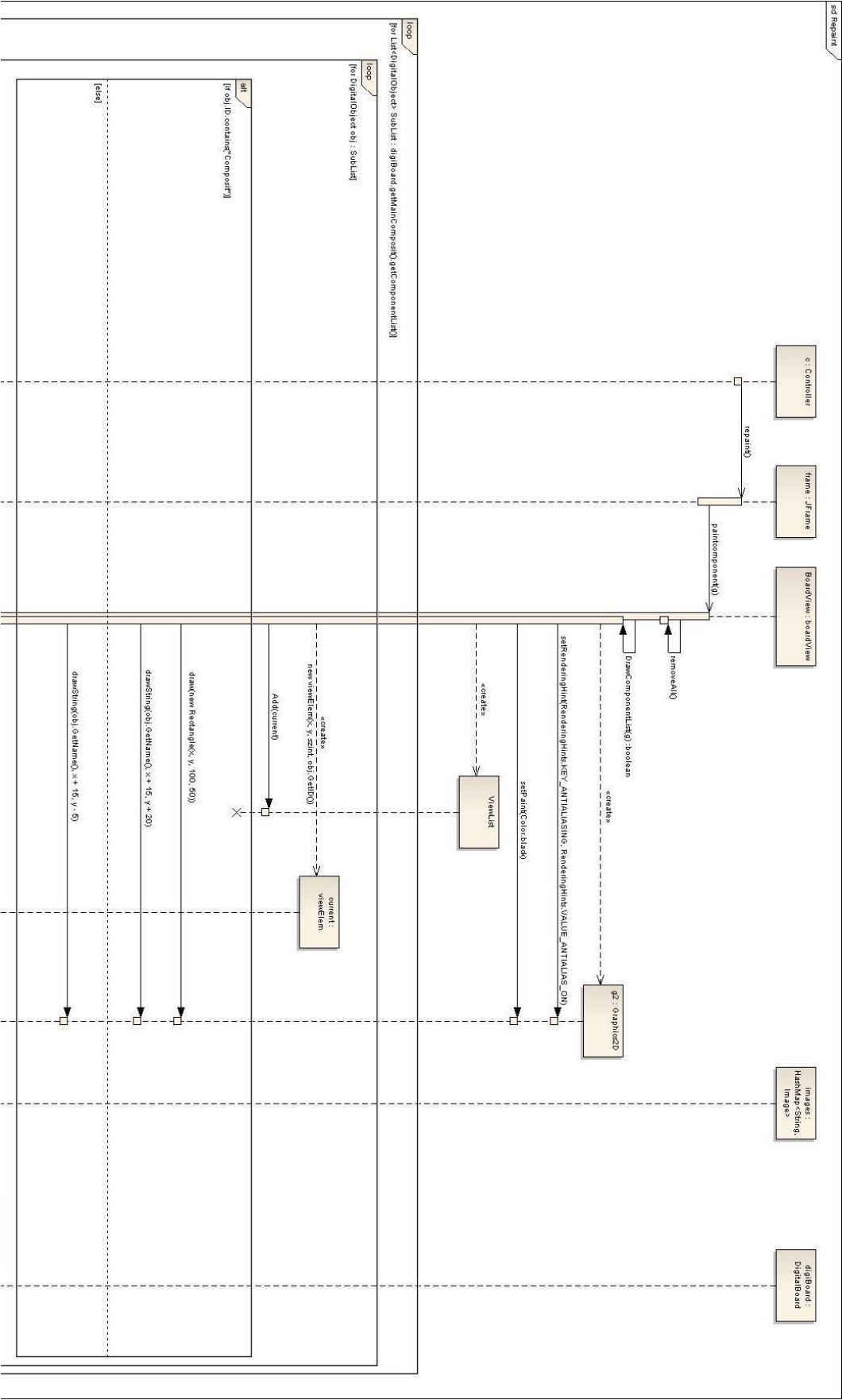
Toggle



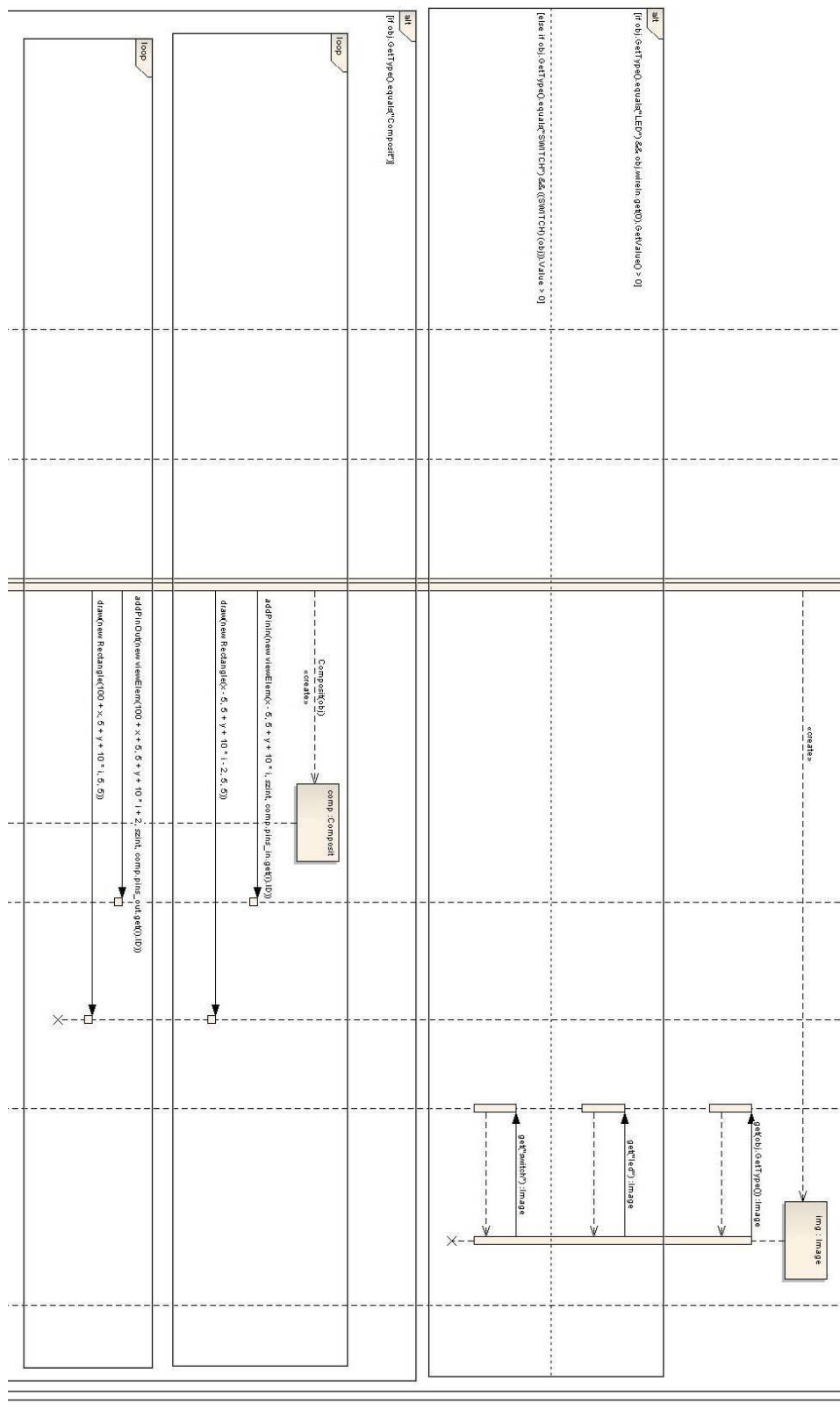
Set Sequence



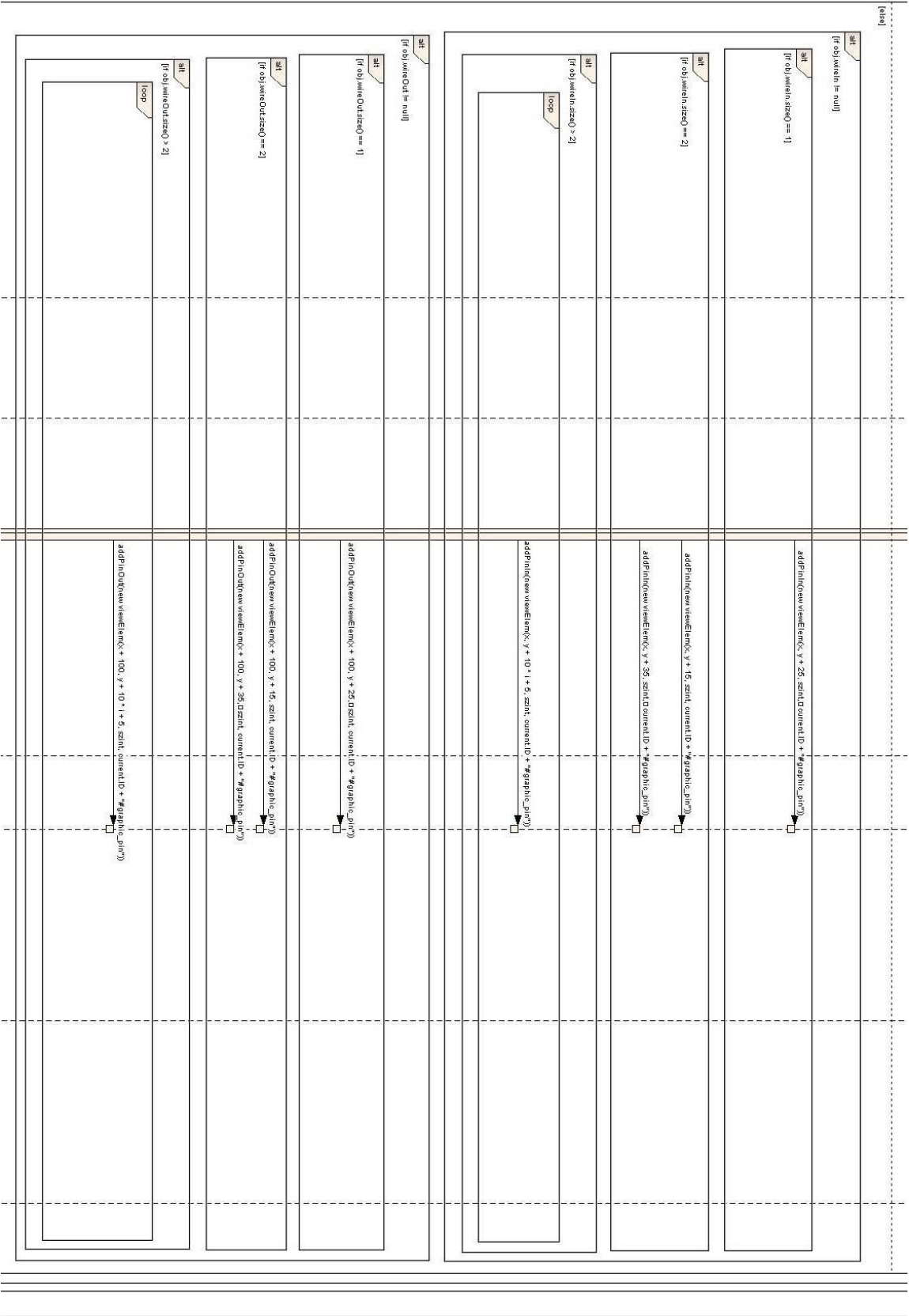
Repaint 1



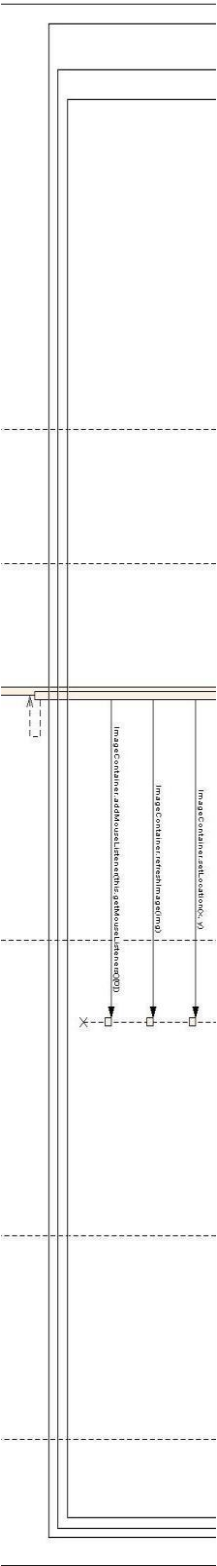
Repaint 2

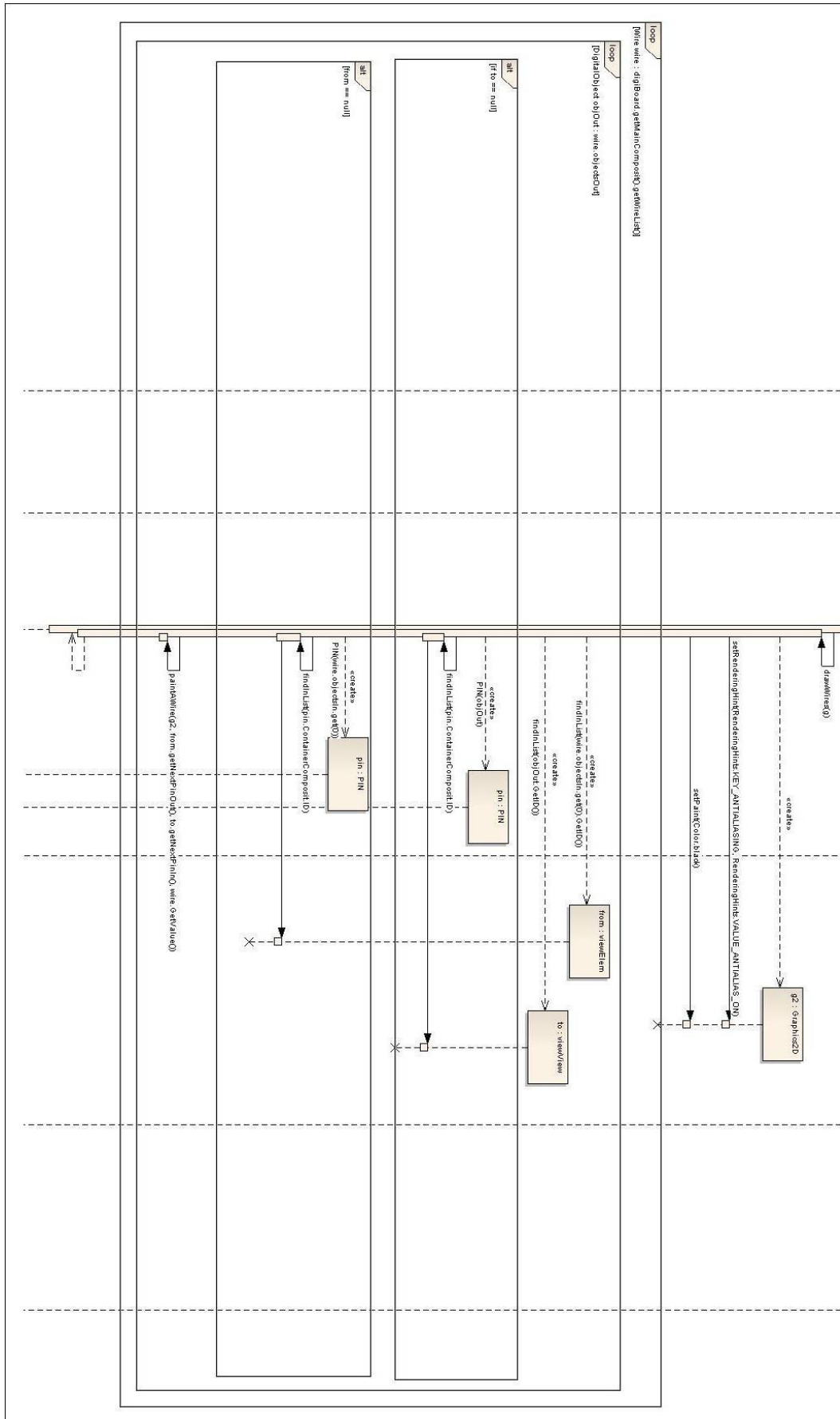


Repaint 3



Repaint 4





9.1.3 Fájllista

Fájl neve	Méret (Byte)	Keletkezés ideje	Tartalom
ANDGate.java	6355	2011.04.26 20:34	ÉS kaput megvalósító osztályt tartalmazza.
bhdlParser.java	35459	2011.04.26 20:35	bhdl kiterjesztésű, hálózat felépítését leíró fájlok beolvasását elvégző osztályt tartalmazza.
boardView.java	12955	2011.05.06 18:44	Osztály mely a DigitalBoard megjelenítésére szolgál.
Composit.java	26517	2011.04.26 20:34	Kompozit objektumot megvalósító osztályt.
Controller.java	27884	2011.04.26 20:34	A program irányításáért felelős.
DigitalBoard.java	8462	2011.04.26 20:34	Áramkört megvalósító osztály.
DigitalObject.java	4404	2011.04.26 20:34	Hálózati elemeket megvalósító absztrakt osztály.
ExceptionElementHasNoInputs.java	281	2011.04.26 20:34	Az elem bementére nincsen vezeték kapcsolva kivétel.
ExceptionElementInputSize.java	279	2011.04.26 20:34	Kevés vezeték van az elem bemenetére kapcsolva kivétel.
ExceptionElementNotConnected.java	281	2011.04.26 20:34	Nincs összekötve az elem kivétel.
ExceptionObjectNotFound.java	298	2011.04.26 20:34	Nem létezik az adott elem kivétel.
ExceptionsWithConnection.java	153	2011.04.26 20:34	Összekötéssel kapcsolatos kivételek.
ExceptionUnstableCircuit.java	256	2011.04.26 20:34	Instabil hálózat kivétel
ExceptionWireHasMultipleInputs.java	271	2011.04.26 20:34	A vezetéknek több bemenete van kivétel.
ExceptionWrongBoard.java	252	2011.04.26 20:34	Hibás betöltendő áramkör kivétel.
ExceptionWrongParameter.java	265	2011.04.26 20:34	Rossz parancs paraméter kivétel
Gate.java	1811	2011.04.26 20:35	Kapuk absztrakt osztálya.
GENERATOR.java	6060	2011.04.26 20:35	Generátort megvalósító osztály.
Input.java	1080	2011.04.26 20:35	Jelet generáló elemek absztrakt osztálya.
INVERTER.java	6042	2011.04.26 20:35	Invertert megvalósító osztály.
LED.java	2859	2011.04.26 20:35	Ledet megvalósító osztály.
Logger.java	3285	2011.04.26 20:35	Naplózásért felelős osztály
Main.java	4193	2011.04.26 20:35	A program elindítását, és a tesztelővel való kommunikációt valósítja meg.
ORGate.java	6480	2011.04.26 20:35	VAGY kaput megvalósító osztály.

Oscilloscope.java	4501	2011.04.26 20:35	Oszilloszkópot megvalósító osztály.
Output.java	837	2011.04.26 20:35	Jelmegjelenítő elemek absztrakt osztálya.
PIN.java	2193	2011.04.26 20:35	A kompozit csatlakozását megvalósító elem osztálya.
Status.java	481	2011.04.26 20:35	Hálózat állapotait tartalmazó változó.
SWITCH.java	3088	2011.04.26 20:35	Kapcsolót megvalósító osztály.
viewElem.java	2159	2011.04.26 20:35	Az elemek megjelenítéséért felelős.
Wire.java	5609	2011.04.26 20:35	Vezetéket megvalósító osztály.
and.png	1679	2011.05.07 15:03	ÉS kapu képe.
generator.png	1459	2011.05.06 18:46	Generátor képe.
inverter.png	1754	2011.05.07 13:39	Inverter képe.
ledoff.png	1093	2011.05.07 14:37	Nulla értékű LED képe.
ledon.png	1122	2011.05.07 14:37	Egy értékű LED képe.
or.png	1903	2011.05.07 13:38	VAGY kapu képe.
oscilloscope.png	1730	2011.05.06 18:46	Oszilloszkóp képe.
swoff.png	1038	2011.05.07 14:36	Nullába állított kapcsoló képe.
swon.png	1055	2011.05.07 14:36	Egybe állított kapcsoló képe.
teszt1.bhdl	114	2011.04.26 20:35	Egyszerű áramkör leírása.
teszt2.bhdl	153	2011.04.26 20:35	Visszacsatolást tartalmazó áramkör leírása.
teszt3.bhdl	227	2011.04.26 20:35	Instabil áramkör leírása.
teszt4.bhdl	395	2011.04.26 20:35	Bonyolult áramkör leírása.
TesztRunner.cmd		2011.04.26 20:34	A fordítást és futtatást végző batch fájl.

9.1.4 Fordítás és telepítés

A fordítás a `grafi_build.bat` batch fájl futtatásával kezdődik. Ezen belül be kell állítani a Java elérhetőségét, amely alapesetben megegyezik a HSZK gépeken az elérési úttal (`PATH=D:\Program Files\Java\jdk1.6.0_14\bin`). Eltérés esetén a tesztelőnek kell beállítani. A fájl létrehoz egy build, ezen belül egy classes mappát, melyet feltölt a programhoz használatos `.class` fájlokkal, valamint létrehoz egy dist mappát, melybe belekerül a `grafikus.jar` fájl, amit majd a `grafi_run.bat` fájl használ a program futtatásánál.

9.1.5 Futtatás

Futtatáshoz a `grafi_build.bat` futtatási batch file indításával kezdődik. Ennek előfeltétele, hogy megtörténjen a 13.2.2-ben leírtak szerint a program fordítása.

9.2 Értékelés

Tag neve	Munka százalékban
Csomák Gábor	18%
Jégh Tamás	25%
Sziklay György	19%
Vad Zsolt	19%
Wiesner Péter	19%

9.3 Napló

Kezdés	Vége	Munka megnevezése	Résztevők	Időtartam (óra:perc)
2011.05.01. 18:00	2011.05.01. 21:00	Grafikus alaposztály megvalósítása	Jégh, Vad	03:00
2011.05.01. 18:00	2011.05.01. 23:00	Komponensek és vezetékek kirajzolása	Csomák	05:00
2011.05.01. 18:00	2011.05.01. 20:00	Kezelőfelület korrigációja, szépítése	Vad	02:00
2011.05.01. 18:00	2011.05.01. 20:00	Grafikus elemek képeinek megvalósítása	Wiesner	02:00
2011.05.01. 18:00	2011.05.01. 21:00	Oszcilloszkóp és generátor popup ablaka	Jégh	03:00
2011.05.01. 18:00	2011.05.01. 19:00	Log mező véglegeítése	Jégh, Vad	01:00
2011.05.06. 15:00	2011.05.06. 16:00	Switch bugfix	Jégh	01:00
2011.05.07. 18:00	2011.05.07. 21:00	Szekvencia diagramok pótlása	Sziklay, Vad	03:00
2011.05.08. 12:00	2011.05.08. 14:00	Végleges dokumentáció	Csomák, Wiesner, Vad	02:00

10. Összegzés és értékelés

10.1 Összegzés

A csapatokra váró feladat eltért a várakozásoktól, azonban nem okozott nagy csalódást. A feladat felidézte elsőéves korunkat, amikor még Digitális Technikából kellett áramköröket szerkesztenünk. Úgy látszik, harmadéves korunkra már érettek lettünk ahhoz, hogy megvalósítsunk egy olyan színvonalú programot, melyben az elsőévesek akár a házi feladatukat megírhatják.

Annak ellenére, hogy a prototípus beadásával csúsztuk, a csapat kitett magáért, és a végeredmény egy olyan szoftver lett, amire minden csapattag büszke lehet. Nagyszerű érzés, hogy együtt milyen eredményeket érhetünk el. Sikertől megvalósítani a dinamikus áramkörépítést, és annak dinamikus kirajzolását.

A dokumentációkban néhol még nem látható, de a félév végére megtanultuk megfogalmazni és leírni elképzeléseinket, színvonalas dokumentációt írni, és ezt a képességünket a jövőbeni munkahelyünkön is magabiztosan fogjuk kamatoztatni.

A csapat tagjainak a modellben gondolkodás nem jelentett különösebb nehézséget, azonban ezek pontos leírása, a vélemények összeegyeztetése annál inkább. A csapatmunka teljesen újszerű volt mindnyájunknak, de mindenki becsületesen végezte a dolgát, és a lehető legtöbbet adta a csapatért. Remek érzés volt, amikor valaki új dolgot tudott mutatni a többieknek, vagy ha valaki elakadt, a többiek gyorsan a segítségére siettek.

Rengeteget tanultunk a félév során, legfőképp alkalmazkodni társainkhoz, és hogy ha jóval előbbre tervezünk, saját dolgunk könnyítjük meg. A csapat minden tagja úgy érzi, hogy a program fejlesztése alatt gyűjtött tapasztalatokat egy sokkal nagyobb feladatban is sikerrel alkalmaznák, és minden problémát megoldanának.

10.2 Értékelés

Mit tanultak a projektből konkrétan és általában?

A projekt során konkrétan megérthettük a Szoftvertechnológiában tanult RUP módszer lépéseit és a Java nyelvet is jobban elsajátítottuk a programozás közben. Projekt alatt a folyamatos tervezés és dokumentálás "rákényszerített" minket a forward engineering típusú fejlesztésre. Megtapasztalhattuk, milyen csapatban dolgozni, kompromisszumokat hozni és együttműködni.

Mi volt a legnehezebb és a legkönnyebb?

A legnehezebb az időnk beosztása, és egymáshoz igazítása volt, sajnos sokszor egymásra kellett várni. Legkönnyebben a grafikus részt valósítottuk meg, ez már amolyan „hab a tortán” jellegű volt, a prototípusban egy hibát találtunk, melyet gyorsan ki tudtunk javítani, és végre látszatja is volt a munkánknak.

Összhangban állt-e az idő és a pontszám az elvégzendő feladatokkal?

Igen, ezzel nem volt probléma

Milyen változtatási javaslatuk van?

Úgy érezzük, hogy az előző évek feladataihoz képest ez egy picit nagyobb hangvételű volt, és 5 ember már nehezen dolgozik össze hierarchia nélkül. Lehet, hogy jobb lett volna, ha van egy kitüntetett szerepű ember, aki csak a munkák emberekhez kiosztását, annak betartását ellenőrzi, így azt is megtanulhatjuk, hogy milyen, ha valaki beosztottjai vagyunk.

Milyen feladatot ajánlanának a projektre?

A csapat szerint nem volt rossz ez a komoly megközelítés, de túlzottan karhoz volt kötve, nem villanykaros barátainknak már nem tudtuk bemutatni. Azonban olyan feladatot, ami mindenki számára ismert, örömmel végeztünk volna, például egy Word, vagy Excel szerű terméket. Továbbá azt sem tartanánk elvetendő ötletnek, hogy a csapatok választhassanak programozási nyelvet, hiszen a modell a nyelvek többségén azonos lenne, ez könnyebbé jelentene.