

# Digitális áramkör szimulátor

54 – *Override*

Konzulens:

Dr. László Zoltán

## Csapattagok:

Kriván Bálint	CBVOEN	balint@krivan.hu
Jákli Gábor	ONZ5G1	j_gab666@hotmail.com
Dévényi Attila	L1YRH0	devenyat@gmail.com
Apagyi Gábor	X8SG3T	apagyi.gabooo@gmail.com
Péter Tamás Pál	N5ZLEG	falconsaglevlist@gmail.com

2011. május 13.

# Tartalomjegyzék

<b>2</b>	<b>Követelmény, projekt, funkcionalitás</b>	<b>9</b>
2.1.	Követelmény definíció . . . . .	9
2.1.1.	A program célja, alapvető feladata . . . . .	9
2.1.2.	A fejlesztőkörnyezet . . . . .	9
2.1.3.	A futtatáshoz szükséges környezet . . . . .	9
2.1.4.	A felhasználói felület . . . . .	9
2.1.5.	Minőségi tényezők . . . . .	9
2.1.6.	A software minősítése . . . . .	9
2.1.7.	A kibocsátás . . . . .	9
2.2.	Projekt terv . . . . .	10
2.2.1.	A fejlesztői csapat . . . . .	10
2.2.2.	Életciklus modell . . . . .	10
2.2.3.	Szervezési struktúra . . . . .	10
2.2.4.	Fejlesztési ütemterv . . . . .	11
2.2.5.	Határidők . . . . .	12
2.3.	Feladatleírás . . . . .	12
2.4.	Szótár . . . . .	14
2.5.	Essential use-case-ek . . . . .	15
2.5.1.	Use-case diagram . . . . .	15
2.5.2.	Use-case leírások . . . . .	16
2.6.	Napló . . . . .	16
<b>4</b>	<b>Analízis modell kidolgozása</b>	<b>19</b>
4.1.	Objektum katalógus . . . . .	19
4.1.1.	<b>Simulation</b> . . . . .	19
4.1.2.	<b>State</b> . . . . .	19
4.1.3.	<b>Circuit</b> . . . . .	19
4.1.4.	<b>Wire</b> . . . . .	19
4.1.5.	<b>Node</b> . . . . .	19
4.1.6.	<b>SequenceGenerator</b> . . . . .	19
4.1.7.	<b>Value</b> . . . . .	19
4.1.8.	<b>AndGate</b> . . . . .	19
4.1.9.	<b>OrGate</b> . . . . .	20
4.1.10.	<b>Inverter</b> . . . . .	20
4.1.11.	<b>Gnd</b> . . . . .	20
4.1.12.	<b>Vcc</b> . . . . .	20
4.1.13.	<b>Led</b> . . . . .	20
4.1.14.	<b>Toggle</b> . . . . .	20
4.1.15.	<b>FlipFlopD</b> . . . . .	20
4.1.16.	<b>FlipFlopJK</b> . . . . .	20
4.1.17.	<b>Mpx</b> . . . . .	20
4.1.18.	<b>SevenSegmentDisplay</b> . . . . .	20
4.2.	Osztályok leírása . . . . .	20
4.2.1.	AbstractComponent . . . . .	20
4.2.2.	AndGate . . . . .	21
4.2.3.	Circuit . . . . .	21
4.2.4.	DisplayComponent . . . . .	22
4.2.5.	FlipFlop . . . . .	22

4.2.6.	FlipFlopD . . . . .	23
4.2.7.	FlipFlopJK . . . . .	23
4.2.8.	Gnd . . . . .	23
4.2.9.	Inverter . . . . .	24
4.2.10.	Led . . . . .	24
4.2.11.	Mpx . . . . .	24
4.2.12.	Node . . . . .	25
4.2.13.	OrGate . . . . .	25
4.2.14.	SequenceGenerator . . . . .	25
4.2.15.	SevenSegmentDisplay . . . . .	26
4.2.16.	Simulation . . . . .	26
4.2.17.	Simulation.State . . . . .	26
4.2.18.	SourceComponent . . . . .	27
4.2.19.	Toggle . . . . .	27
4.2.20.	Value . . . . .	27
4.2.21.	Vcc . . . . .	28
4.2.22.	Wire . . . . .	28
4.3.	Statikus struktúra diagramok . . . . .	29
4.4.	Szekvencia diagramok . . . . .	30
4.5.	State-chartok . . . . .	35
4.6.	Napló . . . . .	35
<b>5</b>	<b>Szkeleton tervezése</b>	<b>37</b>
5.1.	Errata . . . . .	37
5.1.1.	Objektumleírás: <b>Wire</b> . . . . .	37
5.1.2.	Objektumleírás: <b>Node</b> . . . . .	37
5.1.3.	Osztályleírás: <b>AbstractComponent</b> . . . . .	37
5.1.4.	Osztályleírás: <b>Node</b> . . . . .	38
5.1.5.	Osztályleírás: <b>Wire</b> . . . . .	38
5.1.6.	Statikus struktúra diagramok . . . . .	39
5.2.	A skeleton modell valóságos use-case-ei . . . . .	40
5.2.1.	Use-case diagram . . . . .	40
5.2.2.	Use-case leírások . . . . .	40
5.3.	Architektúra . . . . .	47
5.4.	A skeleton kezelői felületének terve, dialógusok . . . . .	47
5.4.1.	Program üzeneteinek formátuma . . . . .	47
5.5.	Szekvencia diagramok a belső működésre . . . . .	49
5.6.	Napló . . . . .	58
<b>6</b>	<b>Skeleton beadás</b>	<b>59</b>
6.1.	Fordítási és futtatási útmutató . . . . .	59
6.1.1.	Fájllista . . . . .	59
6.1.2.	Fordítás . . . . .	60
6.1.3.	Futtatás . . . . .	60
6.2.	Értékelés . . . . .	61
6.3.	Napló . . . . .	61
<b>7</b>	<b>Prototípus koncepciója</b>	<b>62</b>
7.0.	Változtatások . . . . .	62
7.1.	Prototípus interface-definíciója . . . . .	64
7.1.1.	Az interfész általános leírása . . . . .	64
7.1.2.	Bemeneti nyelv . . . . .	64
7.1.3.	Kimeneti nyelv . . . . .	68

7.2. Összes részletes use-case . . . . .	69
7.3. Tesztelési terv . . . . .	70
7.4. Tesztelést támogató segéd- és fordítóprogramok specifikálása . . . . .	71
7.5. Napló . . . . .	72
<b>8 Részletes tervek</b>	<b>73</b>
8.1. Osztályok és metódusok tervei . . . . .	73
8.1.1. Config . . . . .	73
8.1.2. Controller . . . . .	73
8.1.3. Parser . . . . .	73
8.1.4. Proto . . . . .	74
8.1.5. View . . . . .	74
8.1.6. Viewable . . . . .	75
8.1.7. Circuit . . . . .	76
8.1.8. Simulation . . . . .	76
8.1.9. Value . . . . .	77
8.1.10. AbstractComponent . . . . .	77
8.1.11. Composite . . . . .	78
8.1.12. DisplayComponent . . . . .	79
8.1.13. FlipFlop . . . . .	79
8.1.14. SourceComponent . . . . .	80
8.1.15. Wire . . . . .	80
8.1.16. AndGate . . . . .	81
8.1.17. FlipFlopD . . . . .	81
8.1.18. FlipFlopJK . . . . .	81
8.1.19. Gnd . . . . .	82
8.1.20. Inverter . . . . .	82
8.1.21. Led . . . . .	82
8.1.22. Mpx . . . . .	83
8.1.23. Node . . . . .	83
8.1.24. OrGate . . . . .	83
8.1.25. Scope . . . . .	84
8.1.26. SequenceGenerator . . . . .	84
8.1.27. SevenSegmentDisplay . . . . .	85
8.1.28. Toggle . . . . .	85
8.1.29. Vcc . . . . .	86
8.2. A tesztek részletes tervei, leírásuk a teszt nyelvén . . . . .	86
8.2.1. Alap áramkör . . . . .	86
8.2.2. MPX-es áramkör . . . . .	87
8.2.3. Visszacsatolt stabil áramkör . . . . .	89
8.2.4. Visszacsatolt nem stabil áramkör . . . . .	90
8.2.5. Flip-flop-os áramkör . . . . .	91
8.2.6. Kompozitos áramkör . . . . .	92
8.2.7. Kompoziton belüli kompozitos áramkör . . . . .	95
8.3. A tesztelést támogató programok tervei . . . . .	96
8.4. Napló . . . . .	96
<b>10 Prototípus beadása</b>	<b>97</b>
10.1. Fordítási és futtatási útmutató . . . . .	97
10.1.1. Fájllista . . . . .	97
10.1.2. Fordítás . . . . .	98
10.1.3. Futtatás . . . . .	99

10.2. Tesztek jegyzőkönyvei . . . . .	101
10.2.1. 1. tesztet - Kapcsoló, és kapu és LED működésének vizsgálata . . . . .	101
10.2.2. 2. tesztet - Multiplexer és 7 szegmens kijelző vizsgálata . . . . .	101
10.2.3. 3. tesztet - Visszacsatolt vagy kapu vizsgálata - STABIL . . . . .	102
10.2.4. 4. tesztet - Visszacsatolt és kapu és inverter vizsgálata - NEM STABIL . . . . .	102
10.2.5. 5. tesztet - JK Flip-flop és Scope vizsgálata . . . . .	102
10.2.6. 6. tesztet - Kompozitos áramkör vizsgálata . . . . .	102
10.2.7. 7. tesztet - Kompoziton belül kompozit áramkör vizsgálata . . . . .	102
10.3. Értékelés . . . . .	102
10.4. Napló . . . . .	102
<b>11 Grafikus felület specifikációja</b>	<b>104</b>
11.1. A grafikus interfész . . . . .	104
11.2. A grafikus rendszer architektúrája . . . . .	106
11.2.1. A felület működési elve . . . . .	106
11.2.2. A felület osztály-struktúrája . . . . .	107
11.3. A grafikus objektumok felsorolása . . . . .	108
11.3.1. ComponentViewCreator . . . . .	108
11.3.2. Controller . . . . .	108
11.3.3. GuiController . . . . .	109
11.3.4. Parser (vált.) . . . . .	110
11.3.5. AbstractComponent (vált.) . . . . .	111
11.3.6. Composite (vált.) . . . . .	111
11.3.7. Wire (vált.) . . . . .	111
11.3.8. AndGate (vált.) . . . . .	112
11.3.9. FlipFlopD (vált.) . . . . .	112
11.3.10. FlipFlopJK (vált.) . . . . .	112
11.3.11. Gnd (vált.) . . . . .	112
11.3.12. Inverter (vált.) . . . . .	113
11.3.13. Led (vált.) . . . . .	113
11.3.14. Mpx (vált.) . . . . .	113
11.3.15. Node (vált.) . . . . .	113
11.3.16. OrGate (vált.) . . . . .	114
11.3.17. Scope (vált.) . . . . .	114
11.3.18. SequenceGenerator (vált.) . . . . .	114
11.3.19. SevenSegmentDisplay (vált.) . . . . .	114
11.3.20. Toggle (vált.) . . . . .	115
11.3.21. Vcc (vált.) . . . . .	115
11.3.22. CircuitView . . . . .	115
11.3.23. Drawable . . . . .	116
11.3.24. Frame . . . . .	116
11.3.25. FrameView . . . . .	117
11.3.26. ComponentView . . . . .	118
11.3.27. WireView . . . . .	118
11.3.28. AndGateView . . . . .	119
11.3.29. CompositeView . . . . .	119
11.3.30. FlipFlopDView . . . . .	119
11.3.31. FlipFlopJKView . . . . .	120
11.3.32. GndView . . . . .	120
11.3.33. InverterView . . . . .	120
11.3.34. LedView . . . . .	121
11.3.35. MpxView . . . . .	121
11.3.36. NodeView . . . . .	122

11.3.37.OrGateView . . . . .	122
11.3.38.ScopeView . . . . .	122
11.3.39.SequenceGeneratorView . . . . .	123
11.3.40.SevenSegmentDisplayView . . . . .	123
11.3.41.ToggleView . . . . .	124
11.3.42.VccView . . . . .	124
11.4. Kapcsolat az alkalmazói rendszerrel . . . . .	125
11.5. Napló . . . . .	132
<b>13 Grafikus felület specifikációja</b>	<b>134</b>
13.1. Fordítási és futtatási útmutató . . . . .	134
13.1.1. Fájllista . . . . .	134
13.1.2. Fordítás . . . . .	136
13.1.3. Futtatás . . . . .	137
13.2. Értékelés . . . . .	137
13.3. Napló . . . . .	137
<b>14 Összefoglalás</b>	<b>138</b>
14.1. Projekt összegzés . . . . .	138

# Ábrák jegyzéke

2.1. MSN csoport a csapatnak . . . . .	11
2.2. Git történet . . . . .	11
2.3. Ticketek . . . . .	11
2.4. Essential use-case diagram . . . . .	15
4.1. Statikus struktúra nézet . . . . .	29
4.2. Inicializálás . . . . .	30
4.3. Szimuláció . . . . .	31
4.4. Áramkör változásának észlelése . . . . .	32
4.5. Áramkör kiértékelési ciklus . . . . .	32
4.6. Komponens kiértékelése . . . . .	33
4.7. Flipflopok állapotának véglegesítése . . . . .	34
4.8. Jelgenerátorok léptetése . . . . .	34
4.9. Jelforrások módosítása . . . . .	35
4.10. Szimuláció állapotgépe . . . . .	35
5.1. Statikus struktúra nézet . . . . .	39
5.2. A szkeleton modell valóságos use-case-ei . . . . .	40
5.3. Áramkör inicializálása . . . . .	49
5.4. Kapcsoló és Led . . . . .	50
5.5. Kapcsoló, Inverter és Led . . . . .	51
5.6. 2 Kapcsoló, Vagy kapu és Led (1. rész) . . . . .	52
5.7. 2 Kapcsoló, Vagy kapu és Led (2. rész) . . . . .	53
5.8. Inverter visszakötve és Led (1. rész) . . . . .	54
5.9. Inverter visszakötve és Led (2. rész) . . . . .	55
5.10. Kapcsoló, Vagy kapu visszakötve és Led (1. rész) . . . . .	56
5.11. Kapcsoló, Vagy kapu visszakötve és Led (2. rész) . . . . .	57
7.1. Statikus struktúra nézet . . . . .	63
11.1. Főablak . . . . .	104
11.2. Fájl és az Egyéb menü almenüi . . . . .	104
11.3. Szimuláció sebességének beállítása . . . . .	105
11.4. Névjegy . . . . .	105
11.5. Komponens részletei . . . . .	105
11.6. Szekvencia generátor beállítása . . . . .	106
11.7. Statikus struktúra nézet . . . . .	107
11.8. Program indítása . . . . .	125
11.9. Áramkör betöltése . . . . .	126
11.10. Konfigurációs fájl betöltése . . . . .	127
11.11. Konfigurációs fájl mentése . . . . .	127
11.12. Rajzolás indítása . . . . .	128
11.13. Rajzolás . . . . .	128
11.14. Szimuláció léptetése . . . . .	129
11.15. Szekvencia mentése . . . . .	130
11.16. Kapcsolóra kattintás . . . . .	130
11.17. Komponens állapotának kijelzése . . . . .	131
11.18. Start/Stop klikk . . . . .	132

11.19Szimuláció sebességének állítása . . . . .	132
---	-----



## 2. Követelmény, projekt, funkcionalitás

### 2.1. Követelmény definíció

#### 2.1.1. A program célja, alapvető feladata

Az általunk kifejlesztett program célja egy előre megadott digitális áramkör szimulációja és annak megjelenítése grafikus, mindenki számára könnyen kezelhető, átlátható formában. Az alkalmazás az áramköri elemekből felépített digitális hálózat működését szemlélteti úgy, hogy felhasználói interakciók során a rendszer bemenetei átkonfigurálhatóak.

#### 2.1.2. A fejlesztőkörnyezet

A fejlesztéshez a NetBeans 6.9.1 szoftvert választottuk. Az UML diagramok elkészítéséhez a Visual Paradigm for UML nevű alkalmazást használjuk, mely képes az osztály-diagramból Java forráskódot generálni és vice versa. Fejlesztés során szem előtt tartjuk, hogy a program kompatibilis legyen az Oracle által gondozott Java 1.6-os verziójával. Természetesen a cél az, hogy a digitális áramkört modellező program a Hallgatói Számítógép Központban rendszeresített JDK és JRE alatt fordítható és futtatható legyen. A dokumentumokat  $\text{\LaTeX}$ -hel készítjük el a Texmaker nevű alkalmazás segítségével, melyet PDF-be fordítunk le. A unit-tesztekre a JUnit csomagot fogjuk használni.

Mivel a fentebb felsoroltak közül mindegyik alkalmazás fut mind Windows, mind Linux operációs rendszeren, így az egész fejlesztés mindkét platformon megvalósítható.

#### 2.1.3. A futtatáshoz szükséges környezet

Java Runtime Environment 1.6-os verziója, illetve az a számítógép, mely ezt futtatni képes. A modellező alkalmazás használatához billentyűzet, grafikus képernyő és egér szükséges.

#### 2.1.4. A felhasználói felület

A program végső változata grafikus felhasználói felülettel rendelkezik. A programot a felhasználó az egér és a billentyűzet segítségével vezérelheti.

#### 2.1.5. Minőségi tényezők

**Teljesítmény:** A cél az, hogy a digitális rendszermodellező szoftver használható legyen a fentebb meghatározott minimális rendszeren. A grafikus felületnél törekedni fogunk a folyamatos szimuláció megjelenítésére.

**Újrafelhasználhatóság:** A cél az, hogy a grafikus felhasználói felületet a program többi részétől teljesen különválasszuk, így lehetővé téve azt, hogy később a grafikus felület egyszerűen és gyorsan változtatható legyen.

**Rugalmasság:** A rugalmasságot a fejlesztőkörnyezet biztosítja, a modellező szoftvernek ugyanis minden olyan környezetben futtathatónak kell lennie, melyben létezik megfelelő Java futtatókörnyezet.

**Felhasználhatóság:** A használat különösebb tanítást nem igényel, alapfokú számítástechnikai tudással akár a felhasználói kézikönyv elolvasása nélkül is használható.

#### 2.1.6. A software minősítése

A kifejlesztett software akkor megfelelő, ha minél pontosabban megegyezik a fentebb leírtakkal. Ezt ellenőrizni lehet a program futtatásával és kipróbálásával, illetve a forráskód és a modell összevetésével, valamint a funkcionális tesztek futtatásával.

#### 2.1.7. A kibocsátás

A program kibocsátása először a forráskóddal együtt a konzulens felé fog történni.

## 2.2. Projekt terv

### 2.2.1. A fejlesztői csapat

Csapattag neve	feladatköre
Kriván Bálint	csapatvezető, kód, dokumentáció, szervezés
Jákli Gábor	kód, dokumentáció, ticket-koordinátor
Dévényi Attila	kód, dokumentáció, GUI-felelős
Apagyí Gábor	kód, dokumentáció
Péter Tamás Pál	kód, dokumentáció

### 2.2.2. Életciklus modell

A feladat először a program megtervezése, mely a dinamikus és objektum modelleket foglalja magába. Ha ez készen van, elkezdhető a skeleton implementálása. Ez a lépés már teljesen meghatározott, nem merülhet fel semmilyen komplikáció, ha a modellek megfelelőek voltak. A következő feladat a prototípus elkészítése. A programnak ebben az állapotban könnyen tesztelhetőnek kell lennie, hogy a programozási és funkcionális logikai hibák könnyen felismerhetők legyenek. Ha már a prototípus is megfelelő, akkor kezdődhet a grafikus felület megvalósítása. Itt is fontos a tesztelés és a kiértékelés, mert a jó megjelenés sokat számít a modellező használhatóságában. Ha ennek kifejlesztése is sikeres, készen van a program első teljes változata. A kötelező feladat csak eddig tart. Ezt a változatot kell leadni a dokumentációval és a forráskóddal együtt.

### 2.2.3. Szervezési struktúra

A csapat öt emberből áll. A feladat szempontjából a tudásunk nem azonos, mindenki más-más területet érez a magáénak, illetve a feladat eltérő részeinek megoldásához van nagyobb kedvünk. Azt a felépítést választottuk, hogy mindenki az érdeklődésének és tudásának legmegfelelőbb részt kapja az egész feladatból. A feladatok szétosztását találkozókra, illetve az alább meghatározott kommunikációs csatornákon egyeztetjük, ahol az egyéni kívánságok mellett ügyelünk arra, hogy minden feladat kiosztásra kerüljön, valamint a csapattagok az egész feladat megoldásából nagyjából egyenlő mértékben vegyék ki a részüket. A találkozók keretében, mivel a szétosztott feladatok nagy mértékben függenek egymástól, javaslatokat teszünk egymásnak a feladat megoldásának körülményeit és a határidőt illetően.

A forráskódot és minden a fejlesztés során elkészülő dokumentációt, illetve a projekthez tartozó egyéb fájlokat megegyezés alapján egy Git központi tárolóban tároljuk, melyhez a Codaset (<http://codaset.com>) nevű ingyenes szolgáltatását használjuk és erről mindenki egy saját klónt készít.

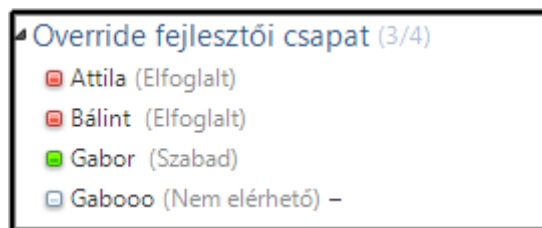
A kiosztott feladatokat a tulajdonosuk elvégzi a megbeszélt határidőig, de ha ez megváltozott funkcionális takar, akkor az adott csapattag köteles a megfelelő teszteseteket megírni, és azok sikeres lefutásáról meggyőződni. Abban az esetben, ha az alkalmazás nem fordul le, vagy valamelyik teszteset nem fut le sikeresen, az adott commit visszaállításra kerülhet annak kijavításáig, melyet a ticket-rendszerben jelezzük a másik felé.

Hogy a fejlesztés minél hatékonyabb és zökkenőmentesebb legyen, a következő eszközöket, technológiákat alkalmazzuk:

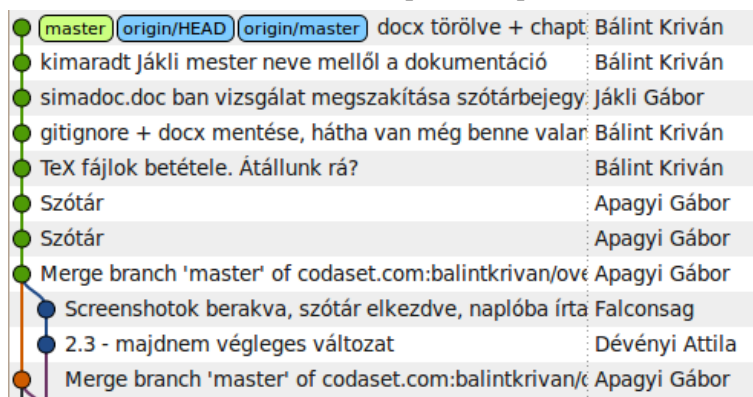
**E-mail** Az egymás számára fontos anyagokat, melyeket a találkozókra előzetesen megbeszéltünk, levélben küldjük el.

**Msn** Felvettük egymást a Microsoft Messenger-be, hogy szükség esetén egymástól is segítséget tudjunk kérni kisebb technikai problémák megoldásában. Természetesen ezek a feladatok lényegét, a projektről hozott döntéseket nem érinthetik, de kivételes helyzetben akár az Interneten is tarthatunk találkozót.

**Git tároló** A feladatok megoldása közben keletkezett anyagokat egy – kizárólag a csapat tagjai által hozzáférhető – helyen tároljuk (lásd fentebb). Így mindig elérhető a fejlesztések legfrissebb változata.



2.1. ábra. MSN csoport a csapatnak



2.2. ábra. Git történet

**Ticket-rendszer** A fejlesztés során felmerülő problémákat, kérdéseket ticket formájában megírjuk egymásnak, amit később a kijelölt felelős személy megold, ha szükséges, akkor együtt konzultálunk a megoldás módjáról, menetéről.

	SUMMARY	STATE	MILESTONE	ASSIGNEE	UPDATED ▼
#7	<b>Szervezési struktúra kialakítása</b> created by falconsag 1 day ago	NEW	2. Követelmény, projek...	gabooo	less than a minute ago by balintkrivan
#6	<b>Szótár elkészítése</b> created by falconsag 1 day ago and has 1 comments	NEW	2. Követelmény, projek...	falconsag	1 minute ago by balintkrivan
#5	<b>Feladatleírás</b> created by falconsag 1 day ago and has 1 comments	NEW	2. Követelmény, projek...	devenyat	1 minute ago by balintkrivan
#4	<b>Essential use-case-ek</b> created by falconsag 1 day ago	NEW	2. Követelmény, projek...	gabooo	1 minute ago by balintkrivan

2.3. ábra. Ticketek

#### 2.2.4. Fejlesztési ütemterv

A program fejlesztésének három fő lépcsőfoka van. Ezek a következők:

**Skeleton:** A cél az, hogy mind a dinamikus, mind az objektum modell jól legyen kitalálva. Ha ezek elkészültek, akkor a fejlesztés szempontjából sikeresen leraktuk az alapokat.

**Prototípus:** Ez már szinte a teljes változat, csak a grafikus felület elemei hiányoznak. Ez a változat tökéletesen megfelelő arra, hogy az objektumok, rutinok, függvények szemantikai helyességét vizsgáljuk.

**Grafikus változat:** A program teljes változata. Tulajdonképpen a prototípus a grafikus felülettel kiegészítve, esetleg kismértékben továbbfejlesztve.

## 2.2.5. Határidők

febr. 11.	Csapat regisztráció
febr. 21.	Követelmény, projekt, funkcionalitás
febr. 28.	Analízis modell kidolgozása 1.
márc. 7.	Analízis modell kidolgozása 2.
márc. 14.	Szkeleton tervezése
márc. 21.	Szkeleton
márc. 28.	Prototípus koncepciója
ápr. 4.	Részletes tervek
ápr. 18.	Prototípus
ápr. 26.	Grafikus felület specifikációja
máj. 9.	Grafikus változat
máj. 13.	Összefoglalás

## 2.3. Feladatleírás

Az általunk készített alkalmazás segítségével a felhasználó egy előre elkészített áramköri listából kiválasztott digitális áramkör szimulációját végezheti el grafikus megjelenítéssel (későbbiekben a felhasználó készíthet saját áramkört a program által elvárt formátumban). A program az alábbi alkatrészeket támogatja áramköri elemként: ÉS kapu, VAGY kapu, Inverter, Kijelző, Jelgenerátor, Kapcsoló, Vcc és Gnd. Ezek mindegyike egy vagy több ki- és/vagy bemenettel rendelkezik.

A komponensek (alkatrészek) részletezése:

1. **Általános komponensek:** A bemenet(ek)re érkező logikai érték(ek) alapján a kimenete(ke)n valamilyen logikai érték(ek)et produkálnak.

- Az ÉS kapu kettő vagy több bemenettel és egy kimenettel rendelkezik. A kimeneten a bemenetre kötött jelek logikai ÉS kapcsolata jelenik meg.
- A VAGY kapu kettő vagy több bemenettel és egy kimenettel rendelkezik. A kimeneten a bemenetre kötött jelek logikai VAGY kapcsolata jelenik meg.
- Az Inverter egyetlen kimenetén az egyetlen bemenetére kötött jel logikai negáltja jelenik meg.
- 4-1-es multiplexer, amely 4 adatbemenettel, 2 kiválasztó-bemenettel és 1 kimenettel rendelkezik. A kiválasztó-bemenetekre adott értéktől függ, hogy melyik adatbemenet értéke jelenik meg az adat-kimeneten.
- D flip-flop, melynek 2 bemenete van és 1 kimenete. A bemenetek közül egyik az adatbemenet a másik az órajel. Az órajel felfutó élére az adatbemeneten lévő érték fog beíródni a flip-flop memóriájába, és ez kerül kiadásra a kimenetén, mindaddig ez marad, amíg új érték nem íródik be. A rendszer indításakor alapértelmezetten a hamis érték van a belső memóriában.
- JK flip-flop, melynek 3 bemenete van: J, K és órajel. Az órajel felfutó élére a J és K állapotától függő érték íródik be a belső memóriájába, ami a kimenetére kerül. Az alábbi táblázat mutatja, hogy az egyes J-K bemenet kombinációira milyen érték íródik be:

J	K	memória
0	0	marad
1	0	1
0	1	0
1	1	negált

Alapállapotban a belső memóriája a hamis értéket tárolja.

2. **Megjelenítők:** Az ide tartozó elemek feladata a logikai értékek vizualizálása.

- A Kijelző komponenssel a felhasználó a bemenetre kötött jelet vizuális formában tudja megjeleníteni.

- 7-szegmenses kijelző komponens, mely 7 bemenettel rendelkezik. Az egyes adatbemenetek megfeleltethetők egy szegmensnek, melyek együtt egy 8-as alakot formálnak. Egy szegmens csak akkor világít, ha a neki megfelelő bemenetre igaz értéket adunk.
3. **Jelforrások:** A harmadik csoport elemei melyeknek nincs bemenete csak kimenete, ez vagy előre definiált (gnd és vcc) vagy a felhasználó által módosítható (kapcsoló, jelgenerátor), ezáltal változtatva az áramkör működését.
- Jelgenerátor segítségével egy bitsorozatot tárolhatunk el, amelyet a szimuláció során az egyetlen kimenetén ciklikusan kiad.
  - A Kapcsolónak egy kimenete van, melynek értéke a kapcsoló állásától függ. „Be” állásban igaz, „Ki” állásban hamis értékű.
  - Gnd („föld”) konstans 0-át (logikai hamis) kiadó jelforrás.
  - Vcc („tápfeszültség”) konstans 1-et (logikai igaz) kiadó jelforrás.

Az összes alkatrésze igaz, hogy nem lehet olyan bemenetük, amelyek szabadok, vagyis sehova sincsenek bekötve, ellenkező esetben a szimulációt nem lehet elindítani és a program figyelmezteti erre a felhasználót (ennek elkerülésére, a programmal szállított szimulálható áramkörök egyik komponensének sincs szabad bemenete).

A felhasználó a fentebb említett alkatrészekből összeállított digitális áramkör szimulációját végezheti. Az alkatrészek és azok egymáshoz kötéseik (összeköttetések) grafikus formában kerül megjelenítésre.

A szimuláció során bármelyik komponens pillanatnyi értékeit a felhasználó lekérdezheti az alkatrészeire való kattintással, ezzel egyidejűleg a szimulációt szünetelteti. Az áramkör vizsgálata közben a Kapcsolók értékeit szabadon változtathatja, melyek hatása valós időben megjelenik. Szimuláció elkezdésekor az összes áramköri elem kimenete hamis értéket vesz fel. Ha a vizsgálandó áramkör bizonyos idő alatt nem áll be stacionárius állapotba változatlan bemenetek mellett, akkor ez jelzésre kerül a felhasználó számára és a szimuláció automatikusan leáll. A szimuláció bármikor megszakítható majd újraindítható, illetve átválthat egy másik digitális áramkör vizsgálatára (az előre elkészített digitális áramkörök közül választva), amennyiben a jelenlegi áramkört nem kívánja tovább használni.

A szimuláció sebessége a felhasználó által konfigurálható, ezáltal a Jelgenerátor kimenetein kiadott jelek váltakozásának sebessége változtatható.

A Kapcsolók, illetve a Jelgenerátorok gyors és egyszerű alap állapotba helyezése érdekében lehet törölni minden addig elvégzett beállítást egyetlen paranccsal, majd előlről kezdeni az egyes elemek konfigurálását. Lehetőség lesz továbbá a szerkeszthető jelforrások beállításainak elmentésére illetve későbbi visszatöltésére is. A konfiguráció sikeres betöltődéséhez teljesülnie kell annak a feltételnek, hogy az abban meghatározott összes elem szerepeljen az áramkörben, amire használni szeretnénk a beállításokat. Amennyiben ez nem áll fent, akkor a nem specifikált jelforrások alapállapotba állnak. Előfordulhat még, hogy a konfigurációban olyan elemek szerepelnek, amelyek az áramkörünkben nem, ekkor hibaüzenet jelenik meg és a betöltés megszakad.

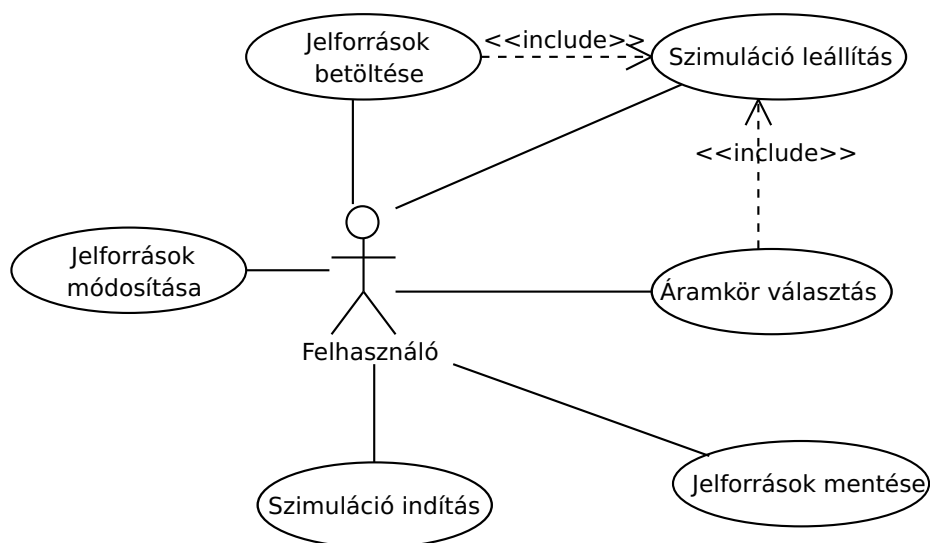
## 2.4. Szótár

<b>Előre elkészített lista</b>	Áramköröket tartalmazó előre elkészített gyűjtemény.
<b>Áramkör</b>	A komponensek egymáshoz kötéséből létrejövő rendszer.
<b>Komponens</b>	Az áramkör alapegysége, mely 3 fajtájú lehet: <i>általános komponens</i> , <i>megjelenítő</i> és <i>jelforrás</i> .
<b>Alkatrész</b>	komponens szinonimája
<b>Általános komponens</b>	Olyan komponens, mely a bemenet(ek)re érkező logikai érték(ek) alapján a kimenete(ke)n valamilyen logikai érték(ek)et produkál.
<b>Bemenet</b>	A komponensek olyan része, melyen keresztül logikai jeleket tudnak fogadni egy másik komponenstől és ezt valamilyen formában felhasználni.
<b>Logikai jel</b>	Az áramkörben levő komponensek által továbbított információ a többi komponens számára, mely az igaz illetve a hamis értéket veheti fel.
<b>Igaz érték</b>	A kétféle logikai jel egyike. (van amikor az '1'-es szimbólum jelöli)
<b>Hamis érték</b>	A kétféle logikai jel egyike. (van amikor a '0'-ás szimbólum jelöli)
<b>Logikai negált</b>	Az igaz érték logikai negáltja hamis, hamis értéké pedig igaz.
<b>Kimenet</b>	A komponens olyan része, melyen keresztül logikai jeleket tud továbbítani más komponenseknek.
<b>Jel továbbítás</b>	A logikai jel egyik komponenstől másik komponensig való áramlása.
<b>Megjelenítő</b>	Olyan komponens, mely a bemenet(ek)re érkező logikai jele(ke)t a felhasználó számára érzékelhető módon szemlélteti.
<b>Jelforrás</b>	Olyan komponens, mely bemenet nélkül továbbítja az áramkörben specifikált vagy a felhasználó által megadott jele(ek)t a kimenetén további komponens(ek) számára.
<b>Gnd</b>	Olyan komponens, mely konstans 0-át ad ki a kimenetén.
<b>Vcc</b>	Olyan komponens, mely konstans 1-et ad ki a kimenetén.
<b>Komponensek egymáshoz kötése</b>	Egy olyan folyamat, amely során 2 komponenst oly módon kapcsolunk össze, hogy az egyik komponens bemenetére a másik komponens kimenetének logikai jelét kapja meg.
<b>Grafikus megjelenítés</b>	Az áramkör felhasználó számára felfogható, érzékelhető megjelenítése.
<b>ÉS kapu</b>	<i>Általános komponens</i> , melynek a bemenetére érkező logikai jelek közt található hamis érték, akkor kimenetén hamis értéket, ellenkező esetben (vagyis minden bemenete logikai igaz) igaz értéket továbbít.
<b>VAGY kapu</b>	<i>Általános komponens</i> , melynek a bemenetére érkező logikai jelek közt található igaz érték, akkor kimenetén igaz értéket, ellenkező esetben (vagyis minden bemenete logikai hamis) hamis értéket továbbít.
<b>Inverter</b>	<i>Általános komponens</i> , mely a bemenetére érkező logikai jel negáltját továbbítja a kimenetén.
<b>Kijelző</b>	Egy darab logikai jelet váró <i>megjelenítő</i> , mely logikai igaz bemenet esetén világít (piros, kék vagy sárga színnel, melyet az áramkör leírója határoz meg), hamis esetén nem.
<b>Áramkör leíró</b>	Egy olyan szöveg, mely a program által elvárt módon leírja a teljes áramkört a komponensek és azok összeköttetéseinek definiálásával.
<b>Jelgenerátor</b>	Olyan <i>jelforrás</i> , mely előre megadott bitsorozatot ad ki ciklikusan a kimenetén.
<b>Bitsorozat</b>	Logikai jelekből létrejövő olyan sorozat, melynél az igaz értéket '1' szimbólum reprezentál, míg a hamis értéket '0'.
<b>Kapcsoló</b>	Olyan <i>jelforrás</i> , mely felhasználói interakció hatására kimenetén igaz, vagy hamis értéket továbbít.

<b>Felhasználói interakció</b>	Olyan esemény, melyet a felhasználó saját maga vált ki valamely tevékenysége során, ezzel potenciálisan befolyásolva az áramkör működését.
<b>Szimuláció</b>	Az a folyamat, mely során minden alkatrész kimenetének logikai jel értékét kiszámoljuk a bemenetére érkező logikai jelekből, vagy ha <i>megjelenítő</i> ről van szó, akkor a felhasználó számára a megjelenítő által meghatározott módon szemléltetjük a bemenetére érkező jeleket. Eközben a felhasználó által megadott időközönként (szimuláció sebessége) léptetjük a jelgenerátort. A szimuláció a felhasználó által indítható és megállítható.
<b>Jelgenerátor léptetése</b>	A jelgenerátorban tárolt bitsorozat következő elemére lépünk és azt adjuk ki a kimenetén, ha a végére értünk, akkor előlről indul.
<b>Szimuláció sebessége</b>	A jelgenerátor egyes állapotai közötti váltás sebessége.
<b>Valós időben</b>	A felhasználó számára lényegében érzékelhetetlen idő alatt.
<b>Stacionárius</b>	A rendszer egy stabil állapota, melyben olyan értékek jelennek meg a komponensek kimenetein, amelyek hatására (visszacsatolás esetén sem) változnak a rendszer komponenseinek kimeneti értékei (Jelgenerátor esetén nincs stacionárius állapot)
<b>Szimuláció megszakítása/leállítása</b>	A rendszer komponenseinek állapota nem változik, azok a megszakítás pillanatában felvett értékeket mutatják a következő indításig.
<b>Állapot</b>	Az áramkör komponenseinek aktuális tulajdonságainak (kimeneti/bemeneti értékek) összessége
<b>Alap állapot</b>	Az a kiindulási állapot, mely az áramkör betöltése után keletkezik. Ilyenkor az <i>általános komponensek</i> kimenetén a hamis érték van.
<b>Jelforrások konfigurációja</b>	A szerkeszthető jelforrások beállításainak (kapcsolók állapota, jelgenerátorok bitsorozata stb.) egy állapota.
<b>Felfutó él</b>	Amikor a logikai érték az adott bemeneten/kimeneten hamisból igazba vált, akkor azt felfutó élnek nevezzük.
<b>Flip-flop belső memóriája</b>	A flip-flopok a szimuláció során megőriznek egy logikai értéket, amit a kimenetükön folyamatosan kiadnak, ezt hívjuk belső memóriának.

## 2.5. Essential use-case-ek

### 2.5.1. Use-case diagram



2.4. ábra. Essential use-case diagram

## 2.5.2. Use-case leírások

Use-case neve	Szimuláció leállítás
Rövid leírás	A futó szimuláció leállítása
Aktorok	Felhasználó
Forgatókönyv	Az éppen futó szimulációt a felhasználó a "Stop" gombbal leállítja.

Use-case neve	Szimuláció indítás
Rövid leírás	A áramkör szimulációjának elindítása
Aktorok	Felhasználó
Forgatókönyv	A felhasználó, az általa korábban kiválasztott áramkör szimulációját a "Start" gombbal elindítja

Use-case neve	Áramkör választás
Rövid leírás	Szimulálni kívánt áramkör kiválasztása
Aktorok	Felhasználó
Forgatókönyv	<ol style="list-style-type: none"> <li>1. A felhasználó a Megnyitás menüpontra kapcsol.</li> <li>2. Kiválaszt egy áramkört a felkínált listából.</li> <li>3. Az áramkör betöltődik a rendszerbe.</li> </ol>

Use-case neve	Jelforrások módosítása
Rövid leírás	Az áramkör jelforrásainak módosítása
Aktorok	Felhasználó
Forgatókönyv	A felhasználó az általa kiválasztott jelforrás értékét (kapcsoló) vagy értékeit (jelgenerátor) módosítja.

Use-case neve	Jelforrások betöltése
Rövid leírás	Az áramkör jelforrásainak betöltése
Aktorok	Felhasználó
Forgatókönyv	<ol style="list-style-type: none"> <li>1. A felhasználó a "Jelforrások betöltése" menüpontra kapcsol.</li> <li>2. Megadja, hogy melyik fájlt olvassa be a rendszer.</li> <li>3. Ha a fájl megfelelő, akkor a betöltés megtörténik, egyéb esetben a felhasználót figyelmeztetjük.</li> </ol>

Use-case neve	Jelforrások mentése
Rövid leírás	Az áramkör jelforrásainak mentése
Aktorok	Felhasználó
Forgatókönyv	<ol style="list-style-type: none"> <li>1. A felhasználó a "Jelforrások mentése" menüpontra kapcsol.</li> <li>2. Megadja, hogy melyik fájlba történjen a mentés.</li> <li>3. A mentés megtörténik, amennyiben ez valami oknál fogva nem sikerült (nincs joga, nincs elég terület stb.), a felhasználót figyelmeztetjük.</li> </ol>

## 2.6. Napló



Kezdet	Időtartam	Résztevők	Leírás
2011.02.15. 21:30	2 óra	<b>Kriván B. Dévényi A. Jákli G.</b>	Értekezlet. Döntések: Megegyeztünk a feladat értelmezését illetően. Milyen kérdéseket teszünk fel a konzulensnek az első konzultáción? Ezeket a Apagyi G. és Péter T. számára továbbítottuk.
2011.02.16. 09:00	2 óra	<b>Kriván B. Dévényi A. Apagyi G. Péter T.</b>	Értekezlet. Döntések: <ul style="list-style-type: none"> <li>• Fejlesztői környezetben megállapodtunk (2.1.2)</li> <li>• Projekt szervezési struktúráját rögzítettük (2.2.3)</li> <li>• A felmerülő algoritmusokról is konzultáltunk.</li> </ul>
2011.02.16. 15:00	1 óra	<b>Jákli G.</b>	Tevékenység: Projekt terv bővítése, formázása, javítása (2.2)
2011.02.17. 16:00	1 óra	<b>Jákli G. Kriván B. Dévényi A.</b>	Tevékenység: a 2.1 és a 2.2 alfejezet közös átdolgozása
2011.02.17. 19:15	45 perc	<b>Jákli G. Kriván B. Dévényi A.</b>	Értekezlet. Döntések: A 2.3 alfejezet főbb gondolatait megfogalmaztuk, és meghatároztuk, hogy mik legyenek a mindenképpen lejegyezendő dolgok.
2011.02.17. 20:00	50 perc	<b>Jákli G.</b>	Tevékenység: A megbeszéltek alapján elkezdte a 2.3 alfejezet megírását.
2011.02.17. 23:00	30 perc	<b>Péter T.</b>	Tevékenység: Szervezési struktúra kiegészítése képernyőképekkel
2011.02.18. 00:00	30 perc	<b>Kriván B. Dévényi A. Péter T</b>	Értekezlet (Msn megbeszélés). Döntések: A 2.3 alfejezet módosításának elhatározása és a szótárba (2.4) bekerülő szavak rögzítése
2011.02.18. 00:30	30 perc	<b>Péter T.</b>	Az előző értekezleten meghatározott szavak felvétele a szótárba, még csak felsorolás szintjén
2011.02.18. 14:00	1 óra	<b>Péter T.</b>	Szótárban lévő szavak magyarázatainak kitöltése
2011.02.18. 15:30	30 perc	<b>Apagyi G.</b>	Szótárban lévő szavak megmagyarázásának folytatása
2011.02.19. 12:00	2 óra	<b>Kriván B.</b>	Tevékenység: helyesírási hibák javítása, szótár (2.4) szerkesztése (sorrendek változtatása, további szavak bevezetése, meglévők magyarázatainak szerkesztése)
2011.02.19. 19:30	30 perc	<b>Kriván B. Jákli G.</b>	Értekezlet (Msn megbeszélés). Döntések: Új essential use-caset kell rajzolni. Szükség van Gnd és Vcc komponensre.
2011.02.19. 20:00	30 perc	<b>Jákli G.</b>	Új essential use-case megrajzolása (2.5.1), use-case leírások megírása (2.5.2)

Kezdet	Időtartam	Résztevők	Leírás
2011.02.19. 20:00	30 perc	<b>Kriván B.</b>	Gnd és Vcc komponens felvétele (2.3), megfelelő részek szerkesztése, use-case konvertálása $\text{\LaTeX}$ -kompatibilis formátumba és felvétele a dokumentációba
2011.02.19. 22:00	30 perc	<b>Dévényi A.</b>	A dokumentáció figyelmes átolvasása, az összes formai és nyelvtani hiba kijavítása
2011.02.20. 18:30	20 perc	<b>Kriván B.</b> <b>Dévényi A.</b> <b>Jákli G.</b>	Értekezlet (Msn megbeszélés) Döntések: Szükség van új komponensekre a jobb használhatóság érdekében: 7-szegmenses kijelző, 4-1-es multiplexer, D és JK flip-flop
2011.02.20. 18:50	30 perc	<b>Jákli G.</b>	Tevékenység: komponensek felvétele a feladat leírásba (2.3), szükséges szavak felvétele a szótárba (2.4)

## 4. Analízis modell kidolgozása

### 4.1. Objektum katalógus

#### 4.1.1. Simulation

A szimulációért felelős objektum. Létrehozza a szimulálni kívánt áramkört. Utasítja az áramkört több kiértékelési ciklus lefuttatásához, amíg az áramkörben van változás. Ha a változás megadott lépés szám limiten belül nem áll meg, tájékoztatja a felhasználót, hogy nincs stacionárius állapot.

#### 4.1.2. State

A szimuláció állapotát megvalósító objektum. 3 állapotot különböztetünk meg:

- a szimuláció fut
- a szimuláció sikeresen lefutott
- a szimuláció során hiba történt

#### 4.1.3. Circuit

Az áramkör objektum. Ez az objektum hozza létre és köti össze egymással az áramköri elemeket. Egy kiértékelési ciklusban minden komponens kiértékel. A flip-flopokat utasítja arra, hogy mentse el a jelenlegi kimenetüket belső állapotként és az órajel bemenetén lévő értéket az éldetektálás érdekében. Megmondja, hogy az áramköri elemek értékei megváltoztak-e. Ezen kívül lépteti a jelgenerátorokat.

#### 4.1.4. Wire

Vezeték, mely az áramköri komponensek ki és bemeneteit köti össze. Egy vezetéket egy darab kimenetet és egy darab bemenetet köt össze. A rajta lévő értéket le lehet tőle kérdezni, illetve be lehet azt állítani.

#### 4.1.5. Node

Csomópont, mely a bemenetén lévő értéket a kimeneteire adja. Segítségével lehet egy vezetéket „szétágaztatni”.

#### 4.1.6. SequenceGenerator

Jelgenerátor, az áramkört felépítő egyik alapelem, kiértékelési kezdeményezés hatására az előre betáplált jelso-rozat éppen aktuális elemét adja ki a kimenetén. Nincs áramköri bemenete. Mikor az áramkör kéri tőle, hogy lépjen, akkor a bitsorozat következő elemét fogja kiadni kimenetén.

#### 4.1.7. Value

A logikai értékeket megvalósító objektum. Jelenleg két érték lehetséges: logikai igaz, logikai hamis.

#### 4.1.8. AndGate

ÉS kapu, az áramkör egyik alapeleme. Bemenetein lévő értékek logikai ÉS kapcsolatát valósítja meg, melynek eredményét a kimenetén kiadja.

#### 4.1.9. OrGate

VAGY kapu, az áramkör egyik alapeleme. Bemenetein lévő értékek logikai VAGY kapcsolatát valósítja meg, melynek eredményét a kimenetén kiadja.

#### 4.1.10. Inverter

Invertáló, az áramkör alapelemei közé tartozik. A bemenetére érkező jel logikai negáltját valósítja meg, amit a kimenetén kiad.

#### 4.1.11. Gnd

Föld, az áramkört felépítő egyik elem, állandó értéke logikai hamis. Bemenete nem létezik.

#### 4.1.12. Vcc

Tápfeszültség, az áramkör egyik alapeleme, mely állandóan a logikai igazt adja ki a kimenetén.

#### 4.1.13. Led

Egy kijelző, az áramkör alapeleme, bemenetén lévő értéket jelzi egy a felhasználó számára érzékelhető módon.

#### 4.1.14. Toggle

Kapcsoló, az áramkör egyik eleme, melynek aktuális értéke állítható, ez jelenik meg a kimenetén. Áramköri bemenete nincs.

#### 4.1.15. FlipFlopD

D flip flopot megvalósító objektum. Csak akkor lép működésbe, mikor az órajelbemenetén a logikai érték hamisról igazra változik, ekkor az értékbemenettől függően változtatja a kimeneti értékét.

#### 4.1.16. FlipFlopJK

JK flip flopot megvalósító objektum. Csak akkor lép működésbe, mikor az órajelbemenetén a logikai érték hamisról igazra változik, ekkor az értékbemenetektől és a belső állapotától függően változtatja a kimeneti értékét.

#### 4.1.17. Mpx

4-1-es multiplexer áramköri építőelemet megvalósító objektum. A választó bemenet függvényében adja ki a kimeneten az egyik, vagy másik értékbemenetére kötött értéket.

#### 4.1.18. SevenSegmentDisplay

Hétszegmenses kijelző objektuma. Minden bemenete egy-egy szegmensért felelős, melyek 8-as alakban helyezkednek el, és a bemenetre kötött érték függvényében világítanak.

### 4.2. Osztályok leírása

#### 4.2.1. AbstractComponent

Absztrakt osztály.

- Felelősség  
Egy komponens absztrakt megvalósítása, ebből származik az összes többi komponens. A közös logikát valósítja meg. A gyakran használt feladatokra ad alapértelmezett implementációt (pl. vezetékek bekötése). Tudja magáról, hogy a legutóbbi két kiértékelés között változtak-e a kimenetei.
- Ősosztályok: (nincs)
- Interfészek: (nincs)
- Attribútumok
  - `protected Wire[] inputs`: Bemeneteire kötött vezetékek.
  - `protected Wire[] outputs`: Kimeneteire kötött vezetékek.
- Metódusok
  - `addTo(Circuit c)`: Meghívja az áramkör `add(AbstractComponent ac)` metódusát.
  - `void evaluate()`: Komponens kimenetein lévő értékek kiszámolása a bemenetek alapján.
  - `boolean isChanged()`: Visszaadja, hogy a legutóbbi két kiértékelés között változtak-e a kimenetek.
  - `void setInput(int inputPin, Wire wire)`: Az adott bemeneti lábára rákötjük a megadott vezetékét.
  - `void setOutput(int outputPin, Wire wire)`: Az adott kimeneti lábára rákötjük a megadott vezetékét.

#### 4.2.2. AndGate

- Felelősség  
ÉS kapu, az áramkör egyik alapeleme. Bemeneteire kötött értékeken a logikai ÉS műveletet hajtva végre, és ennek eredményét adja ki a kimenetén.
- Ősosztályok: `AbstractComponent`
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - (nincs)

#### 4.2.3. Circuit

- Felelősség  
A komponensek létrehozása és összekötése a szimuláció elején, szimuláció által indított kiértékelési ciklus végrehajtása. Rajta keresztül léptethetőek a jelgenerátorok és véglegesíthető a FF-ok állapota.
- Ősosztályok: (nincs)
- Interfészek: (nincs)
- Attribútumok
  - `private Collection<AbstractComponent> components`: Összes áramköri komponens
  - `private Collection<DisplayComponent> displays`: Megjelenítő típusú komponensek (kijelző, 7-segmenses kijelző)

- `private Collection<SourceComponent> sources`: Jelforrás típusú komponensek (kapcsoló, jelgenerátor)
- `private Collection<FlipFlop> flipFlops`: Flipflopok (D és JK flipflopok)
- `private Collection<SequenceGenerator> sequenceGens`: Jelgenerátorok

- **Metódusok**

- `void init()`: Áramkör inicializálása; komponensek létrehozása és összekötése.
- `void add(AbstractComponent component)`: Komponens hozzáadása a `components`-hez. Ezt minden komponensre meg kell hívni.
- `void add(SourceComponent sc)`: Jelforrás hozzáadása a `sources`-hoz és az `add(AbstractComponent component)` meghívása.
- `void add(FlipFlop ff)`: FlipFlop hozzáadása a `flipFlops`-hoz és az `add(AbstractComponent component)` meghívása.
- `void add(SequenceGenerator sg)`: Jelgenerátor hozzáadása a `sequenceGens`-hez és az `add(AbstractComponent component)` meghívása.
- `void add(DisplayComponent dc)`: Megjelenítő hozzáadása a `displays`-hez és az `add(AbstractComponent component)` meghívása.
- `void doEvaluationCycle()`: Egy kiértékelési ciklus lefuttatása. Az áramkörtől ezután lekérdezhető, hogy változott-e a rendszer állapota, azaz valamelyik komponens eltérő kimenetet ad-e, mint az előző ciklusban.
- `boolean isChanged()`: Áramkör változásának lekérdezése. Igazsal tér vissza, ha van olyan komponens, ami azt jelzi magáról, hogy változott a kimenete az előző kiértékeléshez képest.
- `void commitFlipFlops()`: A flipflopok jelenlegi kimenetének elmentése belső állapotnak, és az órajel bemenetén lévő érték eltárolása az éldetektálás érdekében.
- `void stepGenerators()`: Jelgenerátorok léptetése.

#### 4.2.4. DisplayComponent

Absztrakt osztály.

- **Felelősség**  
Megjelenítő típusú komponensek őszosztálya.
- **Őszosztályok**: `AbstractComponent`.
- **Interfészek**: (nincs).
- **Attribútumok**  
- (nincs)
- **Metódusok**  
- `addTo(Circuit c)`: Meghívja az áramkör `add(DisplayComponent dc)` metódusát.

#### 4.2.5. FlipFlop

Absztrakt osztály.

- **Felelősség**  
Flipflopok őszosztálya, itt vannak leírva a flipflopok közös logikája.
- **Őszosztályok**: `AbstractComponent`

- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - `addTo(Circuit c)`: Meghívja az áramkör `add(FlipFlop ff)` metódusát.
  - `void commit()`: Az FF jelenlegi kimenetét és az órajel bemenetét elmenti. Előbbivel frissítjük a belső állapotot, utóbbi pedig az éldetektáláshoz kell. Ezt akkor kell meghívni, amikor az áramkör az adott áramköri bemenetekre stabil állapotba ért.
  - `boolean isActive()`: Számolhat-e az FF? Ezt kell ellenőrizniük a konkrét flipflop implementációknak, hiszen ekkor kellhet a belső állapottól eltérő állapotot kiadni.

#### 4.2.6. FlipFlopD

- Felelősség

D flipflop, mely felfutó órajelnél beírja a belső állapotába az adatbemeneten lévő értéket. Kimenetén a belső állapota jelenik meg.
- Ősosztályok: `AbstractComponent` → `FlipFlop`.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - (nincs)

#### 4.2.7. FlipFlopJK

- Felelősség

JK flipflop, mely felfutó órajelnél a Követelmények résznél leírt módon a J és K bemenetén lévő értéktől és belső állapotától függően változtatja a belső állapotát. Kimenetén a belső állapota jelenik meg.
- Ősosztályok: `AbstractComponent` → `FlipFlop`.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - (nincs)

#### 4.2.8. Gnd

- Felelősség

A "föld" komponens, mely állandóan a hamis értéket adja ki. Nincs bemenete.
- Ősosztályok: `AbstractComponent`.
- Interfészek: (nincs)
- Attribútumok

- (nincs)

- Metódusok

- (nincs)

#### 4.2.9. Inverter

- Felelősség

Inverter alkatrész, mely invertálva adja ki a kimenetén a bemenetén érkező jelet.

- Ősosztályok: AbstractComponent.

- Interfészek: (nincs)

- Attribútumok

- (nincs)

- Metódusok

- (nincs)

#### 4.2.10. Led

- Felelősség

Egy LED-et reprezentál, mely világít, ha bemenetén igaz érték van.

- Ősosztályok: AbstractComponent → DisplayComponent.

- Interfészek: (nincs)

- Attribútumok

- (nincs)

- Metódusok

- `Value getValue()`: Visszaadja a bemenetére kötött értéket.

#### 4.2.11. Mpx

- Felelősség

4-1-es multiplexer, amely 4 adatbemenettel, 2 kiválasztó-bemenettel és 1 kimenettel rendelkezik. A kiválasztó-bemenetekre adott értéktől függ, hogy melyik adatbemenet értéke jelenik meg az adatkimeneten.

- Ősosztályok: AbstractComponent.

- Interfészek: (nincs)

- Attribútumok

- (nincs)

- Metódusok

- (nincs)



## 4.2.12. Node

- Felelősség  
Csomópont, mely a bemenetén lévő értéket a kimeneteire adja. Segítségével lehet egy vezetéket „szét-ágaztatni”.
- Ősosztályok: AbstractComponent.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - (nincs)

## 4.2.13. OrGate

- Felelősség  
VAGY kapu, az áramkör egyik alapeleme. Bemeneteire kötött értékeken a logikai VAGY műveletet hajtva végre, és ennek eredményét adja ki a kimenetén.
- Ősosztályok: AbstractComponent.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - (nincs)

## 4.2.14. SequenceGenerator

- Felelősség  
Jelgenerátort reprezentál, amely a beállított bitsorozatot adja ki.
- Ősosztályok: AbstractComponent → SourceComponent.
- Interfészek: (nincs)
- Attribútumok
  - `private int index`: A bitsorozat egy indexe, ez határozza meg, hogy éppen melyik értéket adja ki a kimenetén.
  - `private Value[] sequence`: Tárolt bitsorozat
- Metódusok
  - `addTo(Circuit c)`: Meghívja az áramkör `add(SequenceGenerator sg)` metódusát.
  - `Value[] getValues()`: Jelgenerátor bitsorozatának lekérdezése
  - `void setValues(Value[] values)`: Jelgenerátor bitsorozatának beállítása
  - `void step()`: A jelgenerátor lép, a bitsorozat következő elemére ugrik. A következő léptetésig ez kerül kiadásra a kimeneteken.

## 4.2.15. SevenSegmentDisplay

- Felelősség  
7-szegmenses kijelzőt reprezentál, melynek 7 bemenete vezérli a megfelelő szegmenseket, ezek világítanak, ha az adott bemenetre logikai igaz van kötve.
- Ősosztályok: `AbstractComponent` → `DisplayComponent`.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - `Value getSegment(int idx)`: Visszaadja az adott szegmenshez tartozó bemenetre kötött értéket.

## 4.2.16. Simulation

- Felelősség  
Az áramkörön kiértékelési ciklusok futtatása az adott áramkör bemenetekre (kapcsolók állapota, jelgenerátorok jelenlegi értéke) nézve addig, amíg az áramkör nem stabilizálódik.
- Ősosztályok: (nincs)
- Interfészek: (nincs)
- Attribútumok
  - `private Circuit circuit`: Szimulált áramkör
- Metódusok
  - `boolean start()`: Szimuláció elindítása a jelenlegi áramköri bemenetekre (kapcsolók állapota, jelgenerátorok jelenlegi értéke). Amennyiben stacionárius állapot jött létre, léptetjük a jelgenerátorokat és elmentjük a flipflopok állapotát. Így újbóli hívásra már a következő időpillanatban érvényes áramköri bemenetekre lehet szimulálni az áramkört. A visszatérési érték a sikerességet jelzi (sikertelen, ha nincs stacionárius állapot).
  - `State getState(State state)`: Szimuláció állapotának lekérdezése.

## 4.2.17. Simulation.State

Enumeráció.

- Felelősség  
Szimuláció állapotait írja le
- Ősosztályok: (nincs)
- Interfészek: (nincs)
- Attribútumok
  - `public static final State READY` A szimuláció kész a futásra. Ilyenkor hívható rajta a `start()` metódus.
  - `public static final State WORKING` Szimuláció éppen dolgozik, egy konkrét jelforrás-kombinációt alkalmazva szimulálja az áramkört.

- `public static final State FAILED` A szimuláció leállt, mert az áramkörnek nincs stationárius állapota. A `start()` metódus újra hívható (ha a bemenetek nem változnak, továbbra is le fog állni).

- Metódusok
  - (nincs)

#### 4.2.18. SourceComponent

Absztrakt osztály.

- Felelősség  
Jelforrás típusú komponensek őssztálya.
- Őssztályok: `AbstractComponent`.
- Interfészek: (nincs).
- Attribútumok
  - (nincs)
- Metódusok
  - `addTo(Circuit c)`: Meghívja az áramkör `add(SourceComponent dc)` metódusát.
  - `abstract Value[] getValues()`: Lekérhetjük a jelforrás értékeit. Ennek megvalósítása a konkrét implementációk feladata.
  - `abstract setValues(Value[] values)`: Beállítjuk a jelforrás értékét. Ennek megvalósítása a konkrét implementációk feladata. (pl. kapcsoló csak 1 elemű tömböt kaphat)

#### 4.2.19. Toggle

- Felelősség  
Kapcsoló jelforrás, melynek két állapota lehet; egyikben logikai igazat, másikban logikai hamist ad ki.
- Őssztályok: `AbstractComponent` → `SourceComponent`.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - `Value[] getValues()`: Lekérjük a kapcsoló értékét (1 elemű tömb)
  - `void setValues(Value[] values)`: Kapcsoló állapotának változtatása, csak 1 elemű tömböt kaphat paraméterül.

#### 4.2.20. Value

Enumeráció.

- Felelősség  
Az áramkörben előfordulható értéket reprezentál.
- Őssztályok: (nincs)
- Interfészek: (nincs)

- Attribútumok
  - `public static final Value FALSE`
  - `public static final Value TRUE`
- Metódusok
  - `Value invert()`: Invertálja az adott értéket. Ennek addig van értelme, amíg 2 féle állapot fordulhat elő a rendszerben.

#### 4.2.21. Vcc

- Felelősség

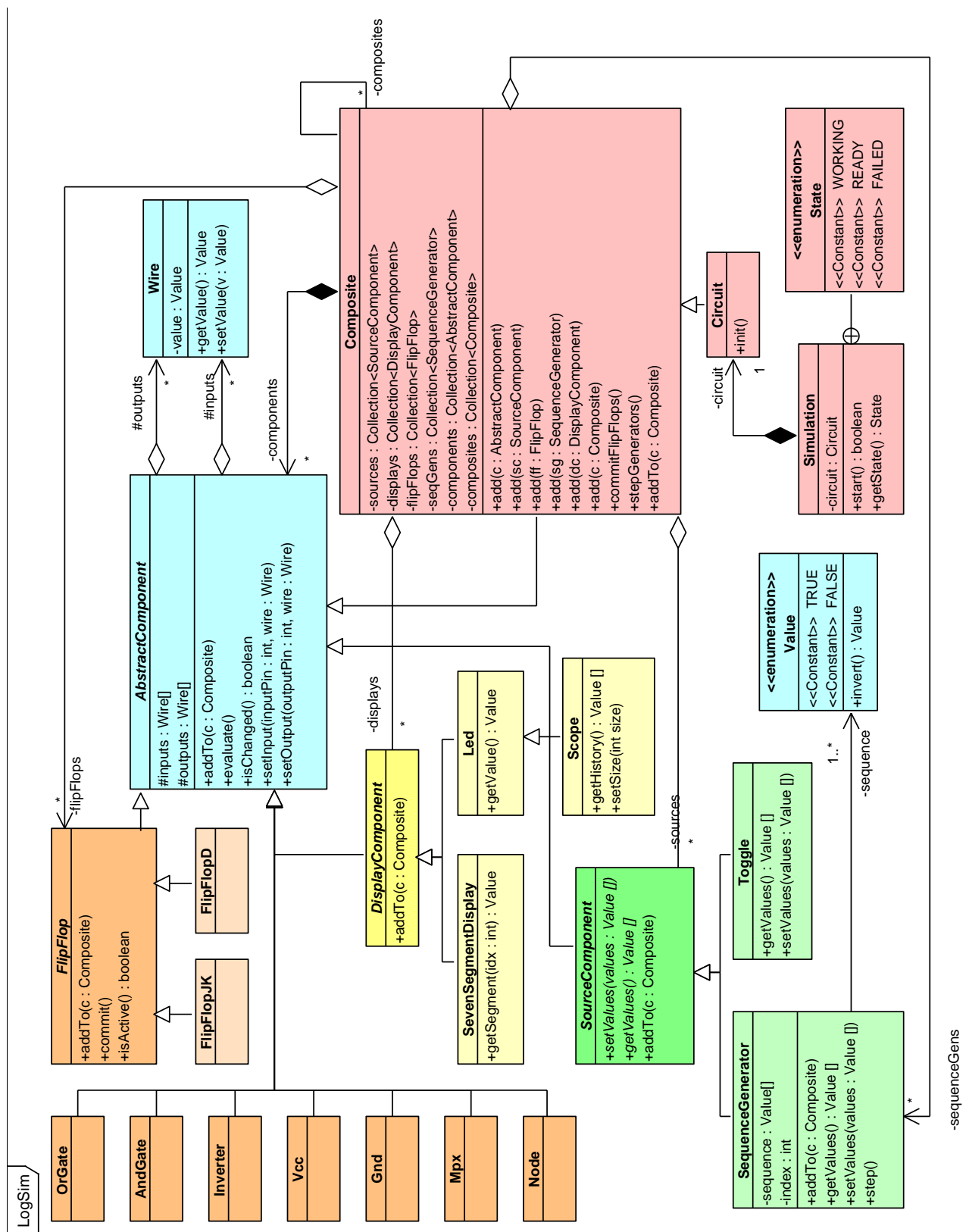
A tápfeszültség komponens, ami konstans igaz értéket ad. Nincs bemenete.
- Ősosztályok: `AbstractComponent`.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - (nincs)

#### 4.2.22. Wire

- Felelősség

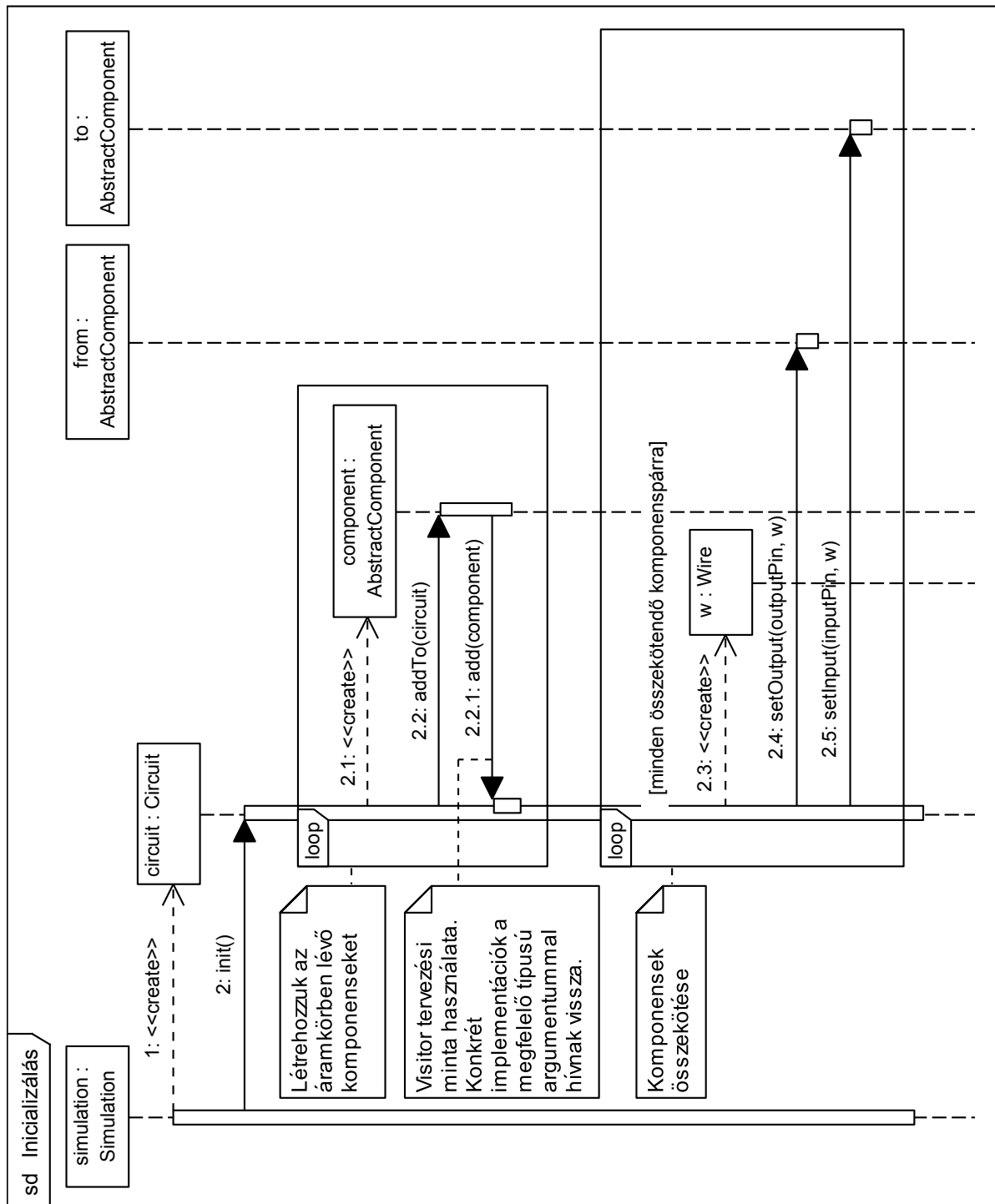
Vezeték, mely az áramköri komponensek ki és bemeneteit köti össze. Egy vezetéket egy darab kimenetet és egy darab bemenetet köt össze. A rajta lévő értéket le lehet tőle kérdezni, illetve be lehet azt állítani.
- Ősosztályok: `AbstractComponent`.
- Interfészek: (nincs)
- Attribútumok
  - `private Value value`: Vezetéken lévő érték
- Metódusok
  - `Value getValue()`: Visszaadja a vezetéken lévő értéket.
  - `void setValue(Value v)`: Beállítja a vezetéken lévő értéket.

## 4.3. Statikus struktúra diagramok

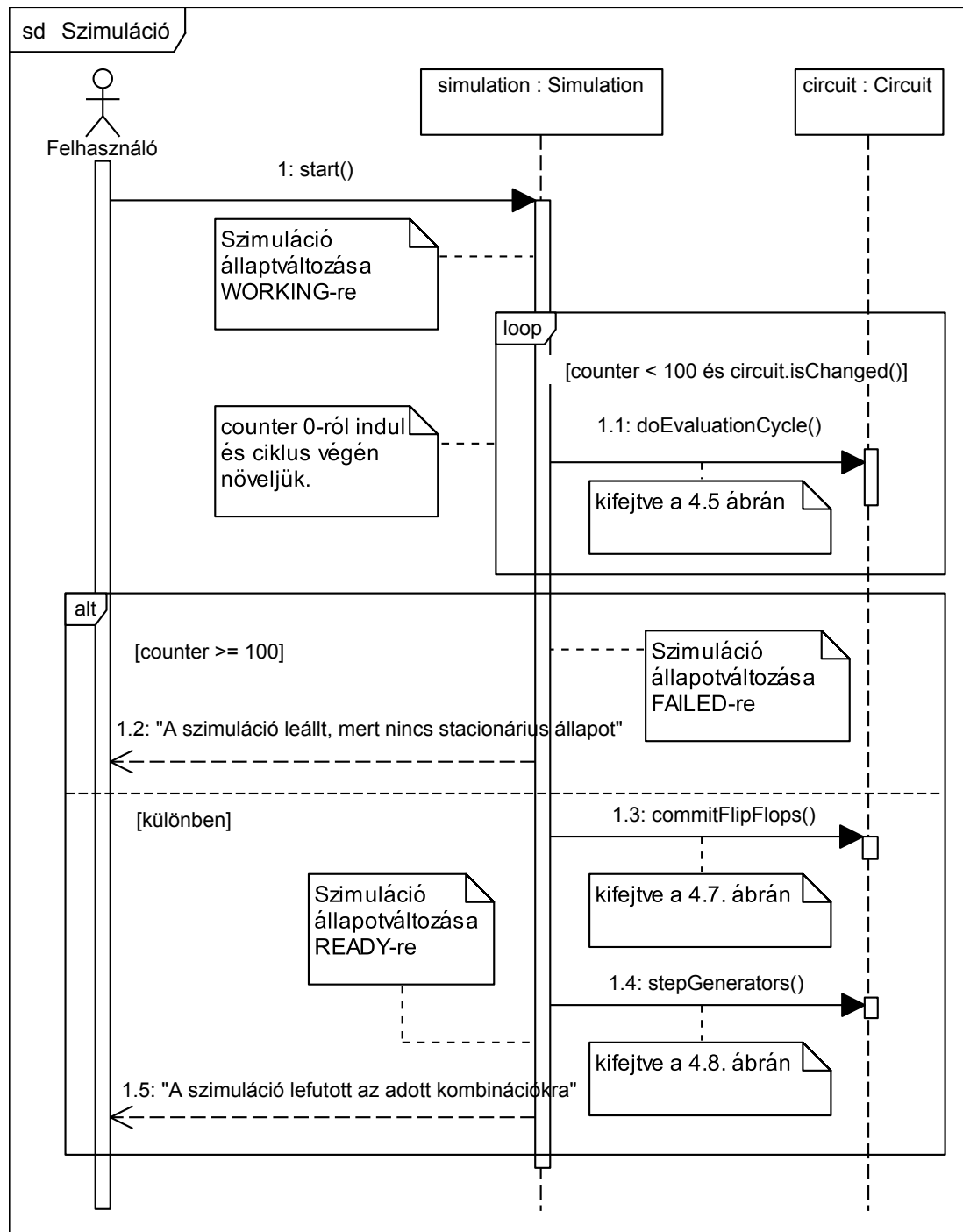


4.1. ábra. Statikus struktúra nézet

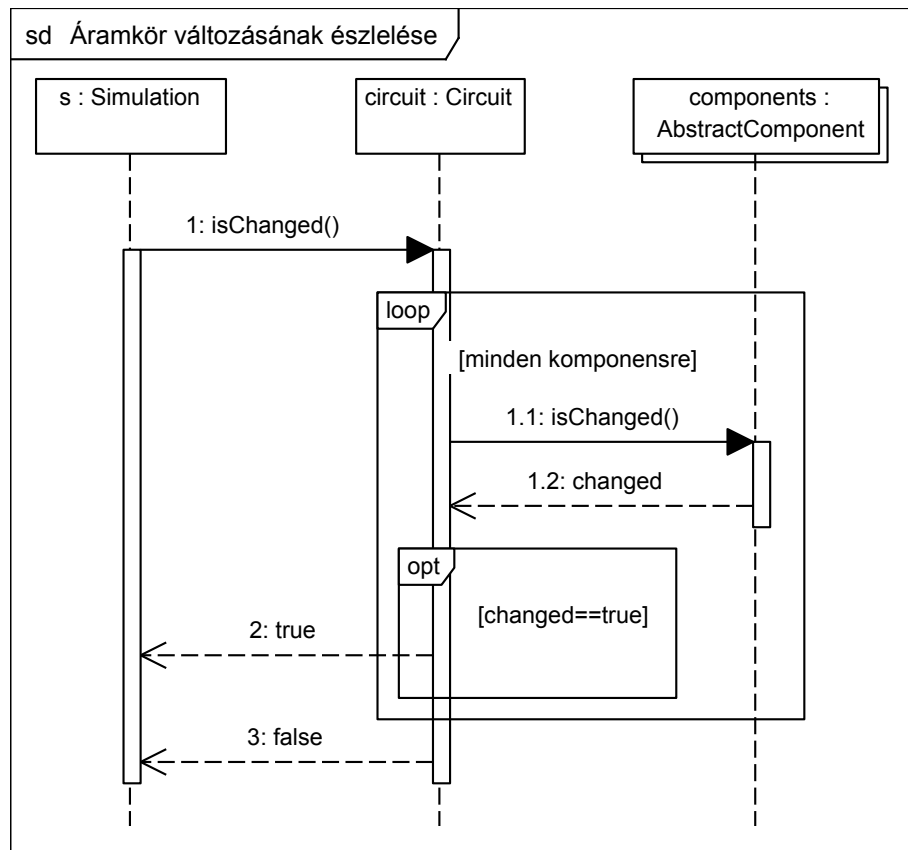
#### 4.4. Szekvencia diagramok



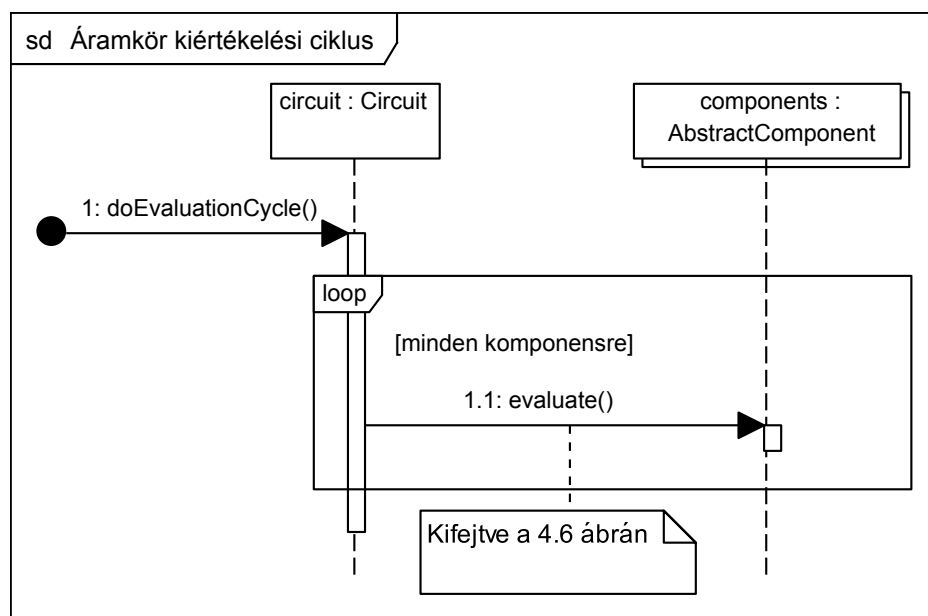
4.2. ábra. Inicializálás



4.3. ábra. Szimuláció

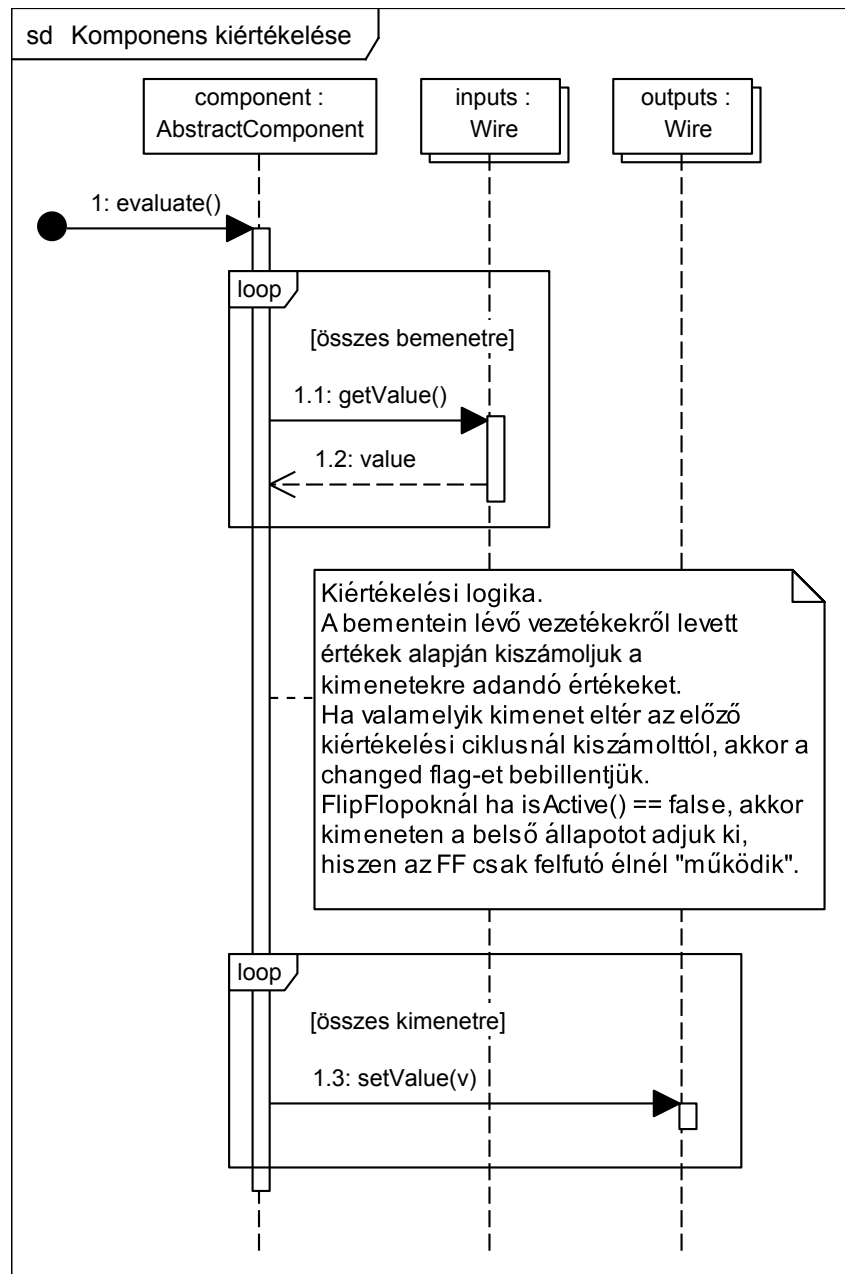


4.4. ábra. Áramkör változásának észlelése

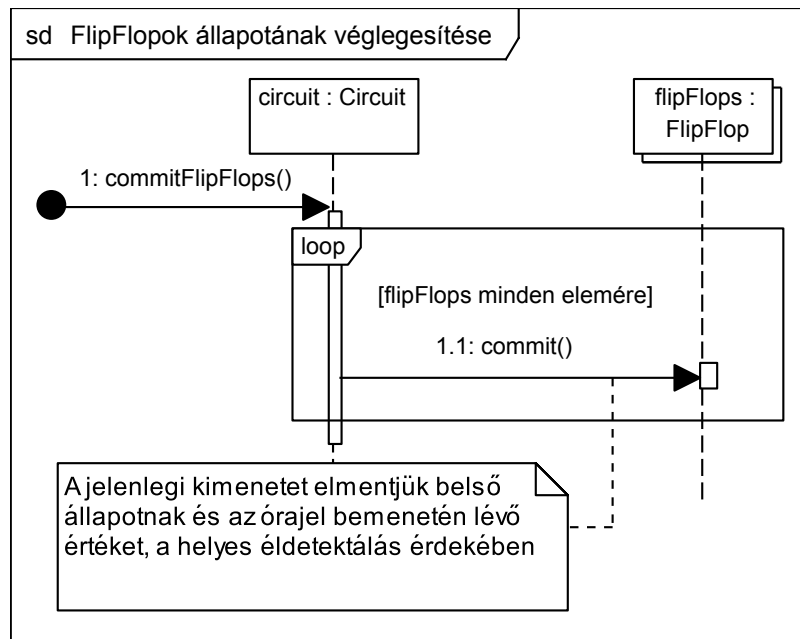


4.5. ábra. Áramkör kiértékelési ciklus

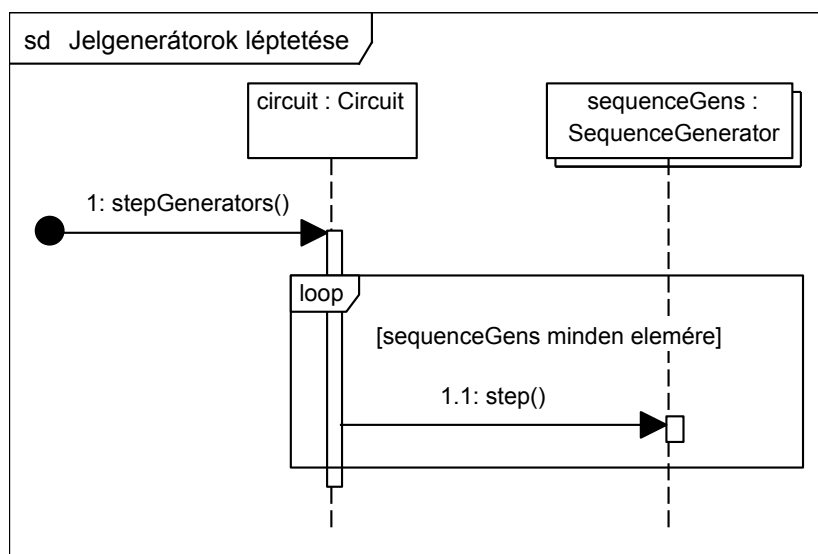




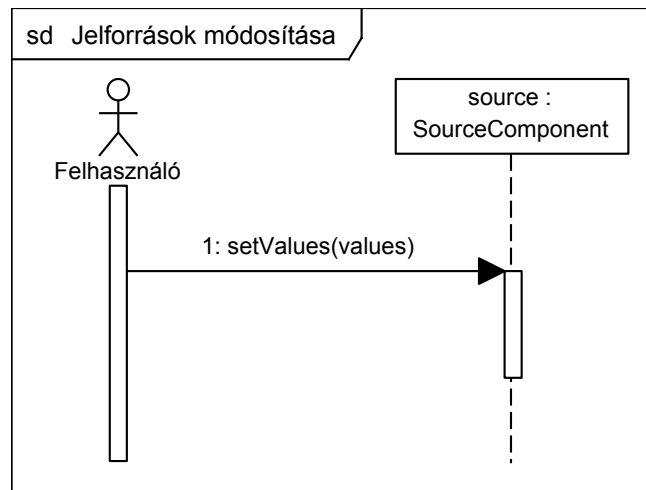
4.6. ábra. Komponent kiértékelése



4.7. ábra. Flipflopok állapotának véglegesítése

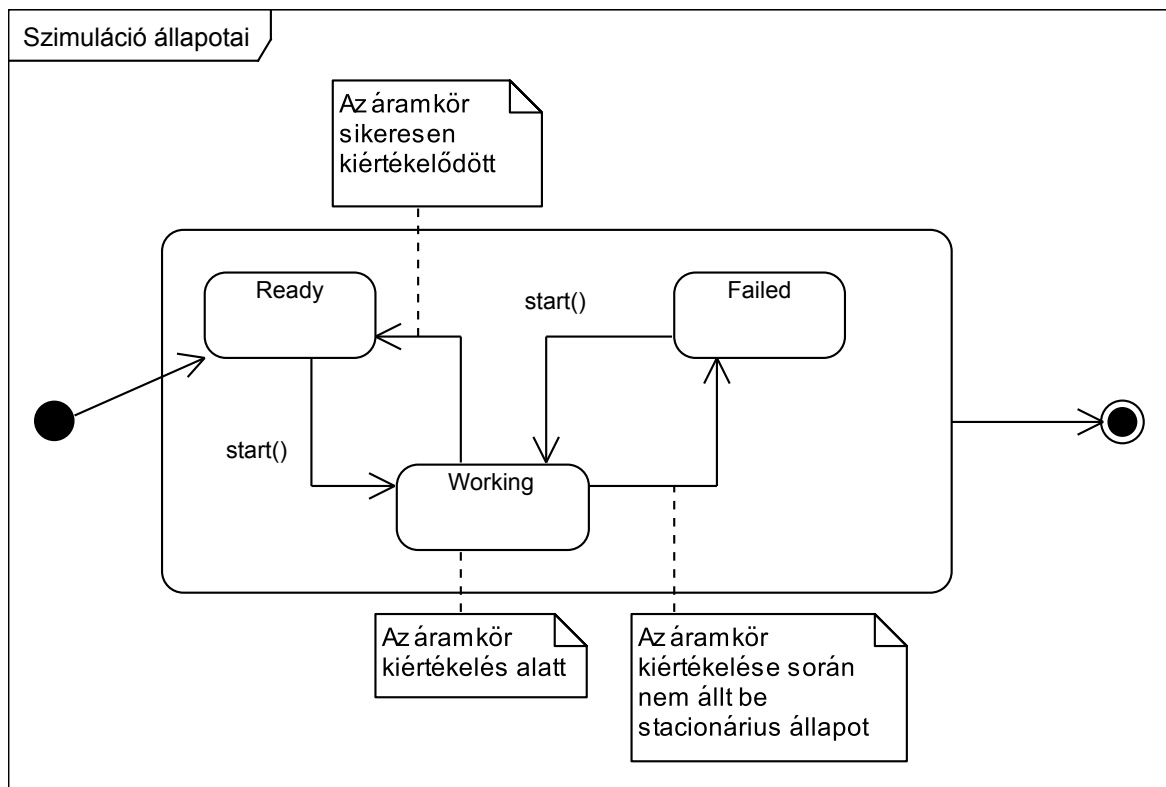


4.8. ábra. Jelgenerátorok léptetése



4.9. ábra. Jelforrások módosítása

#### 4.5. State-chartok



4.10. ábra. Szimuláció állapotgépe

#### 4.6. Napló

Kezdet	Időtartam	Résztevők	Leírás
2011.03.02. 9:00	1,5 óra	<b>Apagyi G. Dévényi A. Jákli G. Kriván B. Péter T.</b>	Konzultáción elhangzottak megtárgyalása, diagramok revíziója
2011.03.02. 18:00	1 óra	<b>Jákli G.</b>	Új diagram: inicializálás (5.3)
2011.03.02. 18:00	1,5 óra	<b>Kriván B.</b>	Új diagram: szimuláció (4.3)
2011.03.03. 15:00	1 óra	<b>Jákli G.</b>	Új diagram: szimuláció állapotgépe (4.10)
2011.03.03. 18:00	1 óra	<b>Dévényi A.</b>	Új diagram: jelgenerátorok léptetése, jelforrások módosítása (4.8)
2011.03.03. 18:00	1 óra	<b>Kriván B.</b>	Új diagram: áramkör kiértékelési ciklus (4.5)
2011.03.03. 18:00	1 óra	<b>Péter T.</b>	Új diagram: komponens kiértékelési ciklus (4.6)
2011.03.03. 19:00	20 perc	<b>Dévényi A.</b>	Jelforrások módosítása diagram frissítése (4.9)
2011.03.03. 20:00	1 óra	<b>Jákli G.</b>	Új diagram: áramkör változásának észlelése (4.4)
2011.03.03. 20:00	1 óra	<b>Dévényi A.</b>	Osztálydiagram újraszerkesztése
2011.03.04. 18:00	30 perc	<b>Apagyi G. Kriván B.</b>	Osztályok leírásának kritikus részeinek átbeszélése
2010.03.04. 18:30	2 óra	<b>Apagyi G.</b>	Objektum katalógus és osztályok leírásának átdolgozása az aktuális modellnek megfelelően
2010.03.04. 20:00	30 perc	<b>Kriván B. Dévényi A.</b>	Értekezlet. Döntés: Komponensek áramkörbe való beregisztrálásának módosítása. (Visitor tervezési minta)
2011.03.04. 21:00	30 perc	<b>Kriván B.</b>	Új diagram: flipflopok állapotának véglegesítése (4.7)
2010.03.05. 10:00	2 óra	<b>Kriván B.</b>	Komponensek áramkörbe való beregisztrálásának módosítása, osztálydiagram átalakítása, szekvenciadiagramok módosítása
2010.03.05. 16:00	2,5 óra	<b>Kriván B.</b>	Objektum katalógus és osztályok leírásának átdolgozása a 2010.03.04. 20:00-i döntés alapján, refaktorálás, apróbb hibák javítása.
2010.03.06. 18:00	1 óra	<b>Dévényi A.</b>	Osztálydiagram végső simításai, rendezgetés. Helyesírási, nyelvtani hibák javítása, apróbb finomítások.

## 5. Szkeleton tervezése

### 5.1. Errata

Az előző fejezetben leírtak egy apró részletben megváltoztak. Az elemeket már nem közvetlenül kötjük össze, hanem *vezetékek* segítségével, melyeket egymással *csomópont*okkal lehet összekötni, ha szükséges. Így javítottuk a láthatósággal kapcsolatosan felmerült problémákat, ehhez fel kellett venni 2 új osztályt (Wire, Node), illetve az AbstractComponent módosítani, ezekhez tartozó objektum és osztályleírások alább olvashatóak, valamint mellékeljük a módosított statikus osztálydiagramot is. (Egy-két egyéb objektumleírás is módosult, de csak azért mert a kiértékelés logikája változott – nem hátulról megyünk, hanem az összes kiértékeli magát, ez nem szükséges a jelen fejezethez, hiszen magától értetődő apróbb átfogalmazásokról van szó)

#### 5.1.1. Objektumleírás: **Wire**

Vezeték, mely az áramköri komponensek ki és bemeneteit köti össze. Egy vezetéket egy darab kimenetet és egy darab bemenetet köt össze. A rajta lévő értéket le lehet tőle kérdezni, illetve be lehet azt állítani.

#### 5.1.2. Objektumleírás: **Node**

Csomópont, mely a bemenetén lévő értéket a kimeneteire adja. Segítségével lehet egy vezetéket „szétágaztatni”.

#### 5.1.3. Osztályleírás: **AbstractComponent**

Absztrakt osztály.

- Felelősség  
Egy komponens absztrakt megvalósítása, ebből származik az összes többi komponens. A közös logikát valósítja meg. A gyakran használt feladatokra ad alapértelmezett implementációt (pl. vezetékek bekötése). Tudja magáról, hogy a legutóbbi két kiértékelés között változtak-e a kimenetei.
- Ősosztályok: (nincs)
- Interfészek: (nincs)
- Attribútumok
  - `protected Wire[] inputs`: Bemeneteire kötött vezetékek.
  - `protected Wire[] outputs`: Kimeneteire kötött vezetékek.
- Metódusok
  - `addTo(Circuit c)`: Meghívja az áramkör `add(AbstractComponent ac)` metódusát.
  - `void evaluate()`: Komponens kimenetein lévő értékek kiszámolása a bemenetek alapján.
  - `boolean isChanged()`: Visszaadja, hogy a legutóbbi két kiértékelés között változtak-e a kimenetek.
  - `void setInput(int inputPin, Wire wire)`: Az adott bemeneti lábára rákötjük a megadott vezetéket.
  - `void setOutput(int outputPin, Wire wire)`: Az adott kimeneti lábára rákötjük a megadott vezetéket.

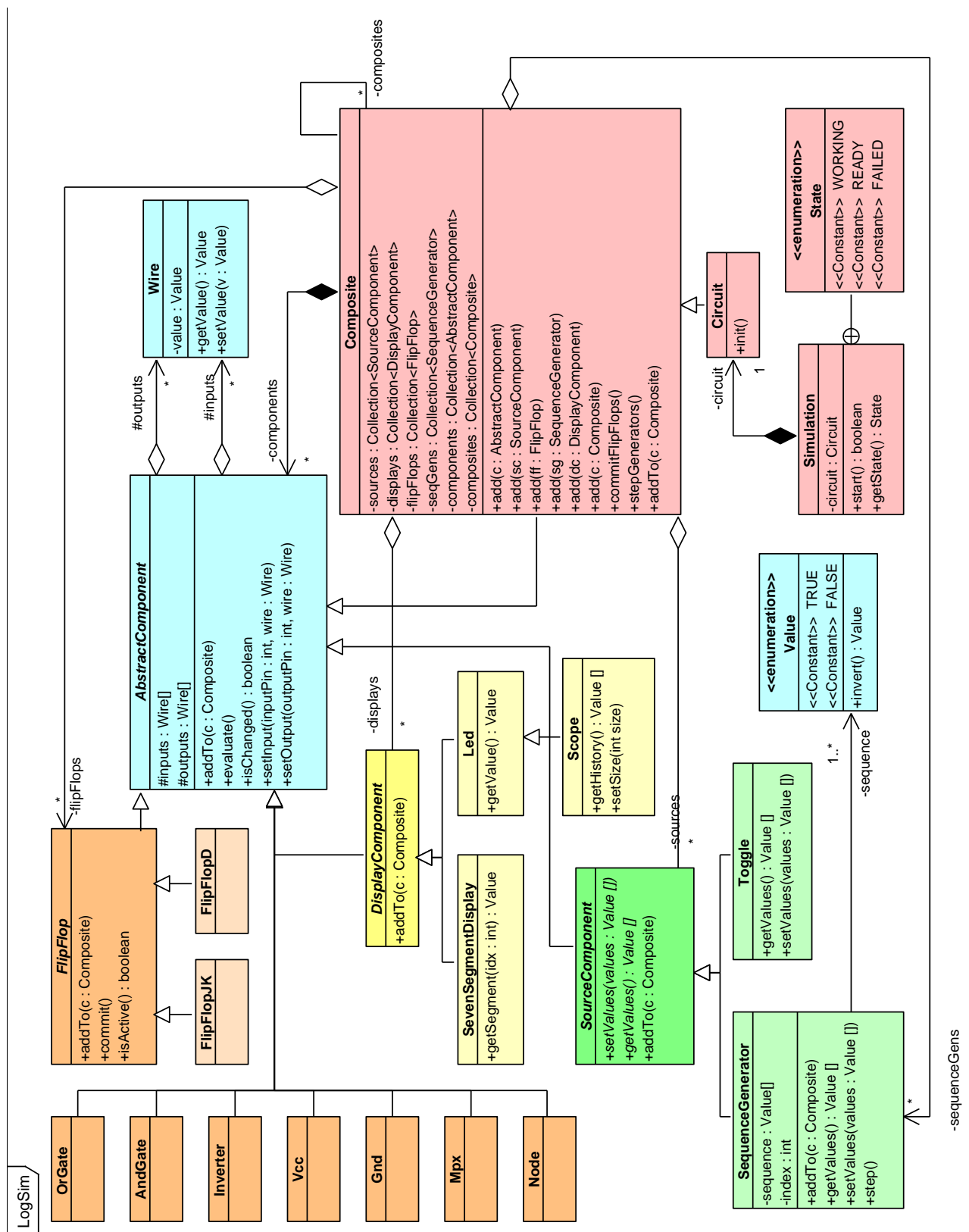
5.1.4. Osztályleírás: **Node**

- Felelősség  
Csomópont, mely a bemenetén lévő értéket a kimeneteire adja. Segítségével lehet egy vezetéket „szét-ágaztatni”.
- Ősosztályok: `AbstractComponent`.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - (nincs)

5.1.5. Osztályleírás: **Wire**

- Felelősség  
Vezeték, mely az áramköri komponensek ki és bemeneteit köti össze. Egy vezetéket egy darab kimenetet és egy darab bemenetet köt össze. A rajta lévő értéket le lehet tőle kérdezni, illetve be lehet azt állítani.
- Ősosztályok: `AbstractComponent`.
- Interfészek: (nincs)
- Attribútumok
  - `private Value value`: Vezetéken lévő érték
- Metódusok
  - `Value getValue()`: Visszaadja a vezetéken lévő értéket.
  - `void setValue(Value v)`: Beállítja a vezetéken lévő értéket.

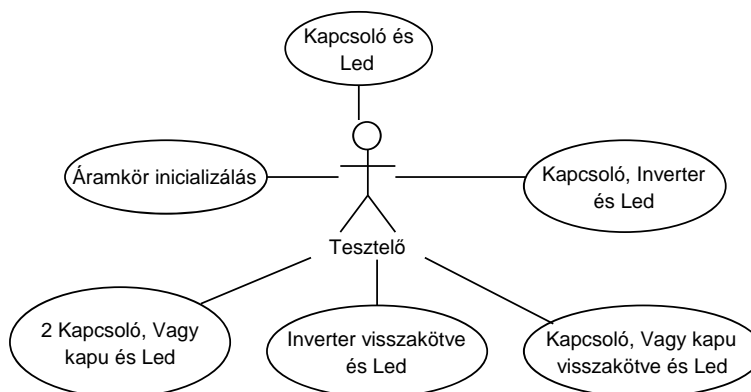
## 5.1.6. Statikus struktúra diagramok



5.1. ábra. Statikus struktúra nézet

## 5.2. A szkeleton modell valóságos use-case-ei

### 5.2.1. Use-case diagram



5.2. ábra. A szkeleton modell valóságos use-case-ei

### 5.2.2. Use-case leírások

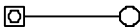
Lenti use-caseknél, ahol valamilyen információ szerint dönteni kell, vagy csak szükségünk van a skeleton jellegéből adódó hiányzó információkra, azt a felhasználótól kérjük be. Ahhoz, hogy a szekvenciadiagramokon lévő szekvenciákat kapjunk, a javasolt értéket kell beírnia a tesztelőnek.


Nem akartuk, hogy az áramkör inicializálása minden use-casenél ott legyen, ezzel elfedve a lényegi részeket, ezért ezt egy külön use-case-ben bemutatjuk, a többinél csak jelezzük, hogy ott is van ilyen lépés. A tesztelő, majd egy adott tesztesetnél választhat, hogy kíváncsi-e az inicializálásra vagy sem.

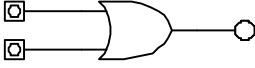
Kapcsolók állítása nincs benne egyik use-caseben sem, azért mert ez ténylegesen majd a szimuláción kívül fog történni, de a kapcsolók bekérik az állapotukat a felhasználótól, így a szimuláció teszteléséhez ez nem is szükséges.

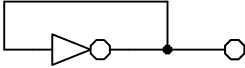
Use-case neve	Áramkör inicializálása
Rövid leírás	Ez a usecase egy áramkör és a hozzá tartozó szimuláció inicializálását mutatja be, hogyan jönnek létre a komponensek és a közöttük lévő összeköttetések. Jelen példa egy Kapcsoló és egy Led összeköttetését prezentálja.
Aktorok	Tesztelő
Forgatókönyv	<ul style="list-style-type: none"> <li>• szimuláció létrehozása</li> <li>• áramkör létrehozása</li> <li>• áramkör inicializálása               <ul style="list-style-type: none"> <li>– kapcsoló létrehozása</li> <li>– vezeték létrehozása</li> <li>– kapcsoló kimenetére vezeték kötése</li> <li>– led létrehozása</li> <li>– led bemenetére vezeték kötése</li> <li>– kapcsoló áramkörbe regisztrálása</li> <li>– led áramkörbe regisztrálása</li> </ul> </li> <li>• áramkör beregisztrálása a szimulációba</li> </ul>

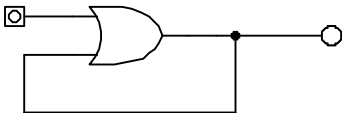


Use-case neve	Kapcsoló és Led
Rövid leírás	<p>Ez a usecase egy olyan áramkör tesztelését mutatja be, amely egy kapcsolóból és rá kötött ledből áll.</p> <p style="text-align: center;"><b>Kapcsoló és Led</b></p> <p style="text-align: center;">  </p>
Aktorok	Tesztelő
Forgatókönyv	<ul style="list-style-type: none"> <li>• Áramkör és komponensek létrehozása</li> <li>• szimuláció indítása             <ul style="list-style-type: none"> <li>– hálózat kiértékelés indítása                 <ul style="list-style-type: none"> <li>* kapcsoló kiértékelése                     <ul style="list-style-type: none"> <li>· kapcsoló állapotának lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kapcsoló értékének kiadása a vezetékekre</li> </ul> </li> <li>* led kiértékelése                     <ul style="list-style-type: none"> <li>· led bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> </ul> </li> </ul> </li> <li>– áramkör változásának vizsgálata                 <ul style="list-style-type: none"> <li>* kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>* led változásának vizsgálata (<i>idáig nem kéne eljutni, ha fent jól válaszolt a tesztelő</i>)</li> </ul> </li> <li>– áramkör változott, ezért új ciklus</li> <li>– hálózat kiértékelés indítása                 <ul style="list-style-type: none"> <li>* <i>ugyanazon lépések történnek mint az előző kiértékelésnél, és ugyanazon válaszokat javasolt adni.</i></li> </ul> </li> <li>– áramkör változásának vizsgálata                 <ul style="list-style-type: none"> <li>* kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* led változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> </ul> </li> <li>– áramkör nem változott, stabil állapot</li> <li>– FF-okat véglegesítjük (nem történik semmi, mert nincs FF)</li> <li>– jelgenerátorokat léptetjük (nem történik semmi, mert nincs jelgenerátor)</li> <li>– szimuláció vége</li> </ul> </li> </ul>

Use-case neve	Kapcsoló, Inverter és Led
Rövid leírás	<p>Ez a usecase egy olyan áramkör tesztelését mutatja be, amely egy kapcsolóból egy rá kötött inverterből és egy arra kötött ledből áll.</p> <p style="text-align: center;"><b>Kapcsoló, Inverter és Led</b></p> 
Aktorok	Tesztelő
Forgatókönyv	<ul style="list-style-type: none"> <li>• Áramkör és komponensek létrehozása</li> <li>• szimuláció indítása <ul style="list-style-type: none"> <li>– hálózat kiértékelés indítása <ul style="list-style-type: none"> <li>* kapcsoló kiértékelése <ul style="list-style-type: none"> <li>· kapcsoló állapotának lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kapcsoló értékének kiadása a vezetékre</li> </ul> </li> <li>* inverter kiértékelése <ul style="list-style-type: none"> <li>· bemenet lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kimenet kiadása a vezetékre (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> </ul> </li> <li>* led kiértékelése <ul style="list-style-type: none"> <li>· led bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> </ul> </li> </ul> </li> <li>– áramkör változásának vizsgálata <ul style="list-style-type: none"> <li>* kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>* inverter változásának vizsgálata (<i>idáig nem kéne eljutni, ha fent jól válaszolt a tesztelő</i>)</li> <li>* led változásának vizsgálata (<i>idáig nem kéne eljutni, ha fent jól válaszolt a tesztelő</i>)</li> </ul> </li> <li>– áramkör változott, ezért új ciklus</li> <li>– hálózat kiértékelés indítása <ul style="list-style-type: none"> <li>* ugyanazon lépések történnek mint az előző kiértékelésnél, és ugyanazon válaszokat javasolt adni.</li> </ul> </li> <li>– áramkör változásának vizsgálata <ul style="list-style-type: none"> <li>* kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* inverter változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* led változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> </ul> </li> <li>– áramkör nem változott, stabil állapot</li> <li>– FF-okat véglegesítjük (nem történik semmi, mert nincs FF)</li> <li>– jelgenerátorokat léptetjük (nem történik semmi, mert nincs jelgenerátor)</li> <li>– szimuláció vége</li> </ul> </li> </ul>

Use-case neve	2 Kapcsoló, Vagy kapu és Led
Rövid leírás	<p>Ez a usecase egy olyan áramkör tesztelését mutatja be, amely egy vagy kapura kötött két kapcsolóból és a vagy kapu kimenetére kötött ledből áll.</p> <p style="text-align: center;"><b>2 Kapcsoló, Vagy kapu és Led</b></p> 
Aktorok	Tesztelő
Forgatókönyv	<ul style="list-style-type: none"> <li>• Áramkör és komponensek létrehozása</li> <li>• Szimuláció indítása</li> <li>– hálózat kiértékelés indítása <ul style="list-style-type: none"> <li>* 1. kapcsoló kiértékelése <ul style="list-style-type: none"> <li>· kapcsoló állapotának lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>· kapcsoló értékének kiadása a vezetékre</li> </ul> </li> <li>* 2. kapcsoló kiértékelése <ul style="list-style-type: none"> <li>· kapcsoló állapotának lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kapcsoló értékének kiadása a vezetékre</li> </ul> </li> <li>* VAGY kapu kiértékelése <ul style="list-style-type: none"> <li>· kapu egyik bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>· kapu másik bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kapu értékének kiadása a vezetékre (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> </ul> </li> <li>* led kiértékelése <ul style="list-style-type: none"> <li>· led bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> </ul> </li> </ul> </li> <li>– áramkör változásának vizsgálata <ul style="list-style-type: none"> <li>* 1. kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* 2. kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>* kapu változásának vizsgálata (<i>idáig nem kéne eljutni, ha fent jót választott a tesztelő</i>)</li> <li>* led változásának vizsgálata (<i>idáig nem kéne eljutni, ha fent jót választott a tesztelő</i>)</li> </ul> </li> <li>– áramkör változott, ezért új ciklus</li> <li>– hálózat kiértékelés indítása <ul style="list-style-type: none"> <li>* <i>ugyanazon lépések történnek mint az előző kiértékelésnél, és ugyanazon válaszokat javasolt adni.</i></li> </ul> </li> <li>– áramkör változásának vizsgálata <ul style="list-style-type: none"> <li>* 1. kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* 2. kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* kapu változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* led változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> </ul> </li> <li>– áramkör nem változott, stabil állapot</li> <li>– FF-okat véglegesítjük (nem történik semmi, mert nincs FF)</li> <li>– jelgenerátorokat léptetjük (nem történik semmi, mert nincs jelgenerátor)</li> <li>– stacionárius állapot, szimuláció vége</li> </ul>

Use-case neve	Inverter visszakötte és Led
Rövid leírás	<p>Ez a usecase egy olyan áramkör tesztelését mutatja be, amely egy inverterből és egy ledből áll. Az inverter kimenete a ledbe illetve saját bemenetére van kötve. Oszcillálni fog, ezért a szimuláció rövid időn belül leáll.</p> <p style="text-align: center;"><b>Inverter visszakötte és Led</b></p> 
Aktorok	Tesztelő
Forgatókönyv	<ul style="list-style-type: none"> <li>• Áramkör és komponensek létrehozása</li> <li>• szimuláció indítása <ul style="list-style-type: none"> <li>– hálózat kiértékelés indítása <ul style="list-style-type: none"> <li>* inverter kiértékelése <ul style="list-style-type: none"> <li>· inverter bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>· inv. értékének kiadása a vezetékekre (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> </ul> </li> <li>* node kiértékelése <ul style="list-style-type: none"> <li>· node bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· node értékének kiadása a vezetékekre (<b>megkérdezi a tesztelőt</b>, javasolt: mindkettőre 1)</li> </ul> </li> <li>* led kiértékelése <ul style="list-style-type: none"> <li>· led bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> </ul> </li> </ul> </li> <li>– áramkör változásának vizsgálata <ul style="list-style-type: none"> <li>* inverter változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>* node változásának vizsgálata (<i>idáig nem kéne eljutni, ha fent jót választott a tesztelő</i>)</li> <li>* led változásának vizsgálata (<i>idáig nem kéne eljutni, ha fent jót választott a tesztelő</i>)</li> </ul> </li> <li>– áramkör változott, ezért új ciklus</li> <li>– hálózat kiértékelés indítása <ul style="list-style-type: none"> <li>* inverter kiértékelése <ul style="list-style-type: none"> <li>· inverter bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· inv. értékének kiadása a vezetékekre (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> </ul> </li> <li>* node kiértékelése <ul style="list-style-type: none"> <li>· node bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>· node értékének kiadása a vezetékekre (<b>megkérdezi a tesztelőt</b>, javasolt: mindkettőre 0)</li> </ul> </li> <li>* led kiértékelése <ul style="list-style-type: none"> <li>· led bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> </ul> </li> </ul> </li> <li>– áramkör változásának vizsgálata <ul style="list-style-type: none"> <li>* inverter változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>* node változásának vizsgálata (<i>idáig nem kéne eljutni, ha fent jót választott a tesztelő</i>)</li> <li>* led változásának vizsgálata (<i>idáig nem kéne eljutni, ha fent jót választott a tesztelő</i>)</li> </ul> </li> <li>– az áramkör ismét változott, nincs stacionárius állapota, szim. vége</li> </ul> </li> </ul>

Use-case neve	Kapcsoló, Vagy kapu visszakötve és Led
Rövid leírás	<p>Ez a usecase egy olyan áramkör tesztelését mutatja be, amely egy kapcsolóból, egy VAGY kapuból, melynek egyik bemenetére a kapcsoló, másik bemenetére a saját kimenete van kötve és egy ledből, melyre szintén a VAGY kapu kimenetét kötöttük. Ez egy olyan visszakötéses hálózat, mely stabil állapotban van.</p> <p><b>Kapcsoló, Vagy kapu visszakötve és Led</b></p>  <p>The diagram shows a logic circuit. On the left, there is a switch symbol (a square with a circle inside). A line connects the switch to the top input of an OR gate (a D-shaped symbol). The output of the OR gate is connected to a small black dot, which then splits into two paths: one path goes down and left, then up to the bottom input of the OR gate, forming a feedback loop; the other path goes right to a small circle representing an LED.</p>
Aktorok	Tesztelő

Forgatókönyv	<ul style="list-style-type: none"> <li>• Áramkör és komponensek létrehozása</li> <li>• szimuláció indítása</li> <li>– hálózat kiértékelés indítása <ul style="list-style-type: none"> <li>* kapcsoló kiértékelése <ul style="list-style-type: none"> <li>· kapcsoló állapotának lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kapcsoló értékének kiadása a vezetékre</li> </ul> </li> <li>* kapu kiértékelése <ul style="list-style-type: none"> <li>· kapu egyik bemenetének (amelyiken a kapcsoló van) lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kapu másik bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>· kapu értékének kiadása a vezetékre (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> </ul> </li> <li>* node kiértékelése <ul style="list-style-type: none"> <li>· node bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· node értékének kiadása a vezetékekre (<b>megkérdezi a tesztelőt</b>, javasolt: mindkettőre 1)</li> </ul> </li> <li>* led kiértékelése <ul style="list-style-type: none"> <li>· led bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> </ul> </li> </ul> </li> <li>– áramkör változásának vizsgálata <ul style="list-style-type: none"> <li>* kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>* kapu változásának vizsgálata (<i>idáig nem kéne eljutni. . .</i>)</li> <li>* node változásának vizsgálata (<i>idáig nem kéne eljutni. . .</i>)</li> <li>* led változásának vizsgálata (<i>idáig nem kéne eljutni. . .</i>)</li> </ul> </li> <li>– áramkör változott, ezért új ciklus</li> <li>– hálózat kiértékelés indítása <ul style="list-style-type: none"> <li>* kapcsoló kiértékelése <ul style="list-style-type: none"> <li>· kapcsoló állapotának lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kapcsoló értékének kiadása a vezetékre</li> </ul> </li> <li>* kapu kiértékelése <ul style="list-style-type: none"> <li>· kapu egyik bemenetének (amelyiken a kapcsoló van) lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kapu másik bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· kapu értékének kiadása a vezetékre (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> </ul> </li> <li>* node kiértékelése <ul style="list-style-type: none"> <li>· node bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> <li>· node értékének kiadása a vezetékekre (<b>megkérdezi a tesztelőt</b>, javasolt: mindkettőre 1)</li> </ul> </li> <li>* led kiértékelése <ul style="list-style-type: none"> <li>· led bemenetének lekérdezése (<b>megkérdezi a tesztelőt</b>, javasolt: 1)</li> </ul> </li> </ul> </li> <li>– áramkör változásának vizsgálata <ul style="list-style-type: none"> <li>* kapcsoló változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* kapu változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* node változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> <li>* led változásának vizsgálata (<b>megkérdezi a tesztelőt</b>, javasolt: 0)</li> </ul> </li> <li>– áramkör nem változott, stabil állapot</li> <li>– FF-okat véglegesítjük (nem történik semmi, mert nincs FF)</li> <li>– jelgenerátorokat léptetjük (nem történik semmi, mert nincs jelgenerátor)</li> <li>– stacionárius állapot, szimuláció vége</li> </ul>
--------------	--

### 5.3. Architektúra

#### 5.4. A szkeleton kezelői felületének terve, dialógusok

Az általunk elkészített szkeleton egy program váz melynek felülete egy egyszerű konzolos megjelenítési felület, amely alkalmas arra, hogy a use case-k által leírt teszteseteket bemutassuk. Az egyes tesztesetek a neki megfelelő use case sorszámaival van elnevezve, így program indítás után egy szám bevitelét követően a kiválasztott teszteset lefut.

A teszteset futása közben kiír minden objektumot amin metódust hív, illetve kiírja a metódus nevét a paraméterekkel együtt, majd a visszatérési értéket. Ez azért lehetséges, mert a szkeleton már tartalmazza az elkészítendő szoftver összes fontos osztályát és metódusát, azonban az üzleti logikát még nem. Így könnyen eldönthető, hogy a use case-nek megfelelően viselkedik a program és továbbiakban képes lesz-e megfelelően működni.

A tesztelési folyamat során döntési helyzet léphet fel. Ilyenkor a program felteszi a kérdést, majd a kapott válasz alapján folytatja a további futást. Ezzel csökkentjük a tesztesetek számát, anélkül, hogy bizonyos esetek kimaradnának a tesztelés alól.

Futás közben megjegyzés formájában a program tájékoztat bizonyos fontosabb lépésekről (például inicializálás), vagy a megértést segítő egyéb dolgokról.

Az elvárás, hogy a szkeleton a szekvenciadiagramok által leírt működést mutassa. A program egyszerű és könnyen összehasonlítható formában írja ki a működését, amelyet könnyen összevethetjük a szekvencia diagrammokkal.

##### 5.4.1. Program üzeneteinek formátuma

A program a következő eseményeket jelzi ki:

- Konstruktor hívás.  
Formátum: `CREATE osztálynév objektumnév`  
Példa: `CREATE Circuit circuit`
- Tagfüggvény hívás.  
Formátum: `CALL objektumnév.metódus (paraméterek)`  
Példa: `CALL wire.setValue(Value.TRUE)`
- Konstruktor/tagfüggvény visszatér  
Formátum: `RETURN` vagy `RETURN objektumnév/érték`  
Ha van visszatérési érték az kétféle lehet: referencia valamelyik objektumra (ekkor az objektumnevet írjuk ki), vagy egy konkrét érték (ilyenkor az értéket, pl: `false`).
- Kérdés a felhasználónak.  
Formátum: `QUESTION objektumnév üzenet? [opciók]`  
Példa: `QUESTION wire.vezetéken lévő érték? [0/1]`  
Objektumnév annak az objektumnak a neve, aki kérdezi a felhasználót. Az opciókban lévő elemek / jellel vannak elválasztva. Addig nem megy tovább a program, amíg nem ezek közül kap választ.
- Megjegyzés kijelzése.  
Formátum: `# üzenet`

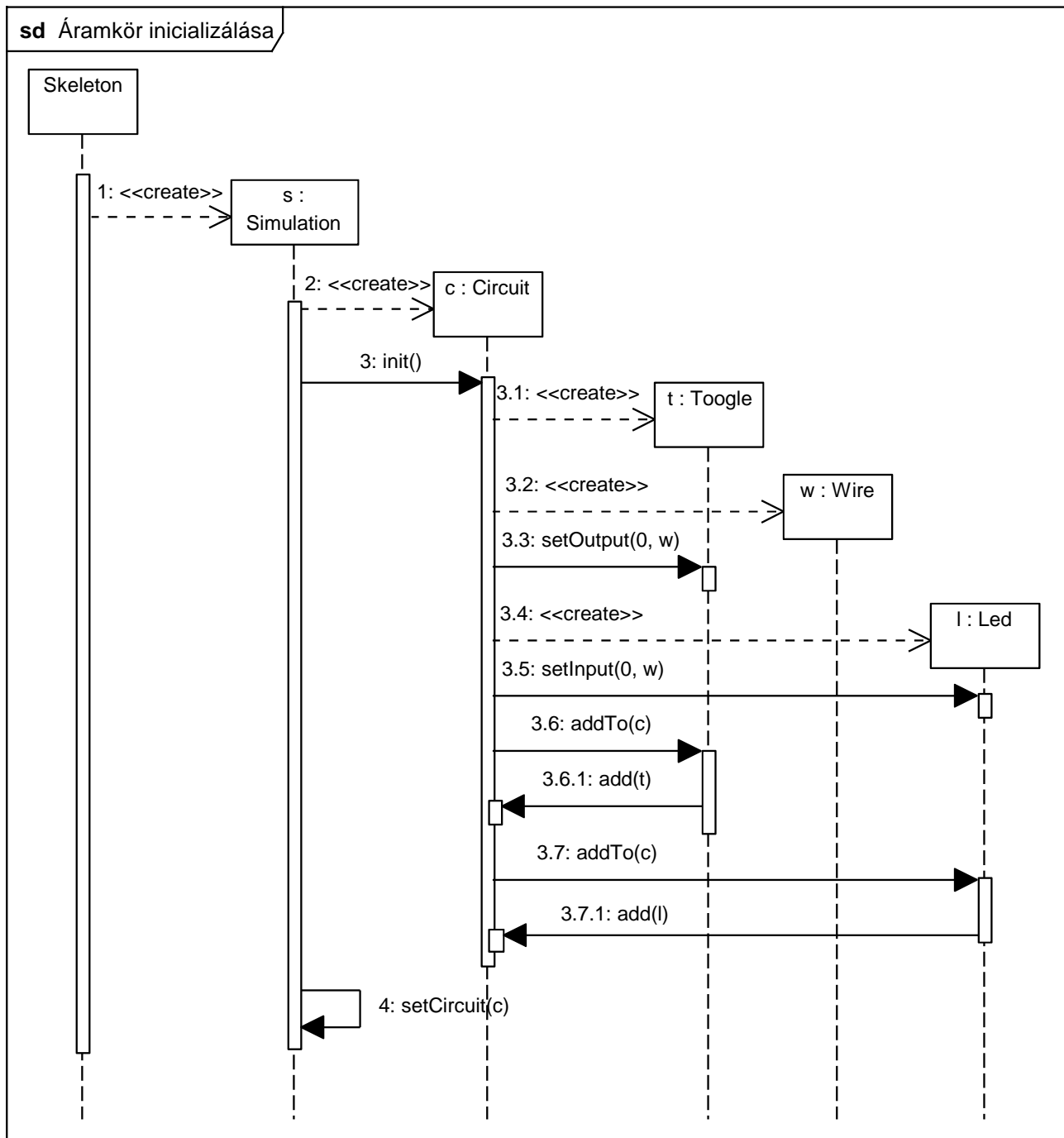
Formátum bemutatásához egy összetettebb példa:

```
CALL simulation.start()
CALL circuit.doEvaluationCycle()
CALL toggle.evaluate()
  QUESTION toggle állapot? [0/1] 1
  CALL wire.setValue(Value.TRUE)
  RETURN
```

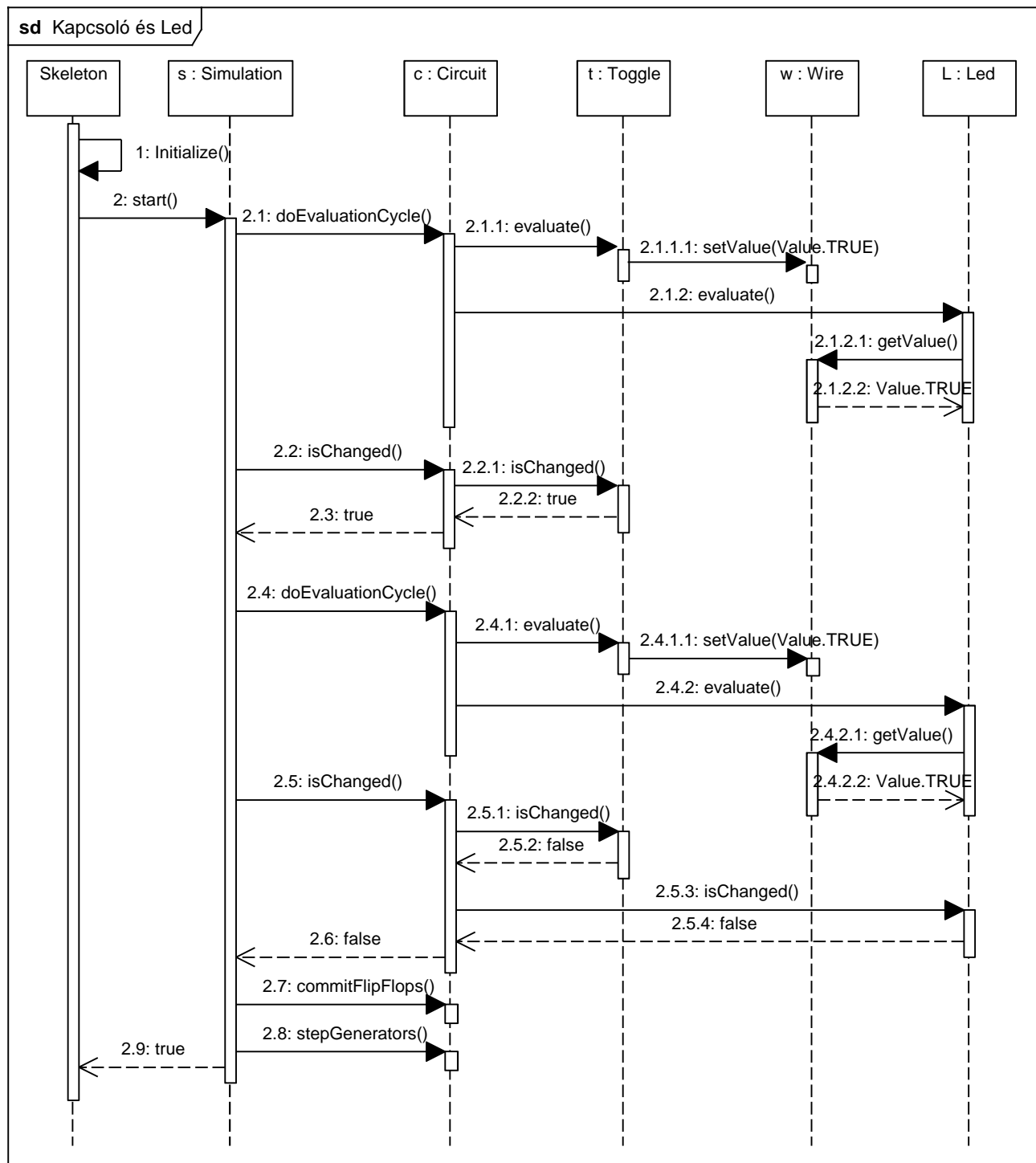
```
    RETURN
    CALL led.evaluate()
    CALL wire.getValue()
    QUESTION wire vezetéken lévő érték? [0/1] 1
    RETURN Value.TRUE
    RETURN
RETURN
CALL circuit.isChanged()
    CALL toggle.isChanged()
    QUESTION toggle változott? [0/1] 1
    RETURN true
RETURN true
CALL circuit.doEvaluationCycle()
    CALL toggle.evaluate()
    QUESTION toggle állapot? [0/1] 1
    CALL wire.setValue(Value.TRUE)
    RETURN
    RETURN
    CALL led.evaluate()
    CALL wire.getValue()
    QUESTION wire vezetéken lévő érték? [0/1] 1
    RETURN Value.TRUE
    RETURN
RETURN
CALL circuit.isChanged()
    CALL toggle.isChanged()
    QUESTION toggle változott? [0/1] 0
    RETURN false
    CALL led.isChanged()
    QUESTION led változott? [0/1] 0
    RETURN false
RETURN false
CALL circuit.commitFlipFlops()
RETURN
CALL circuit.stepGenerators()
RETURN
RETURN true
```



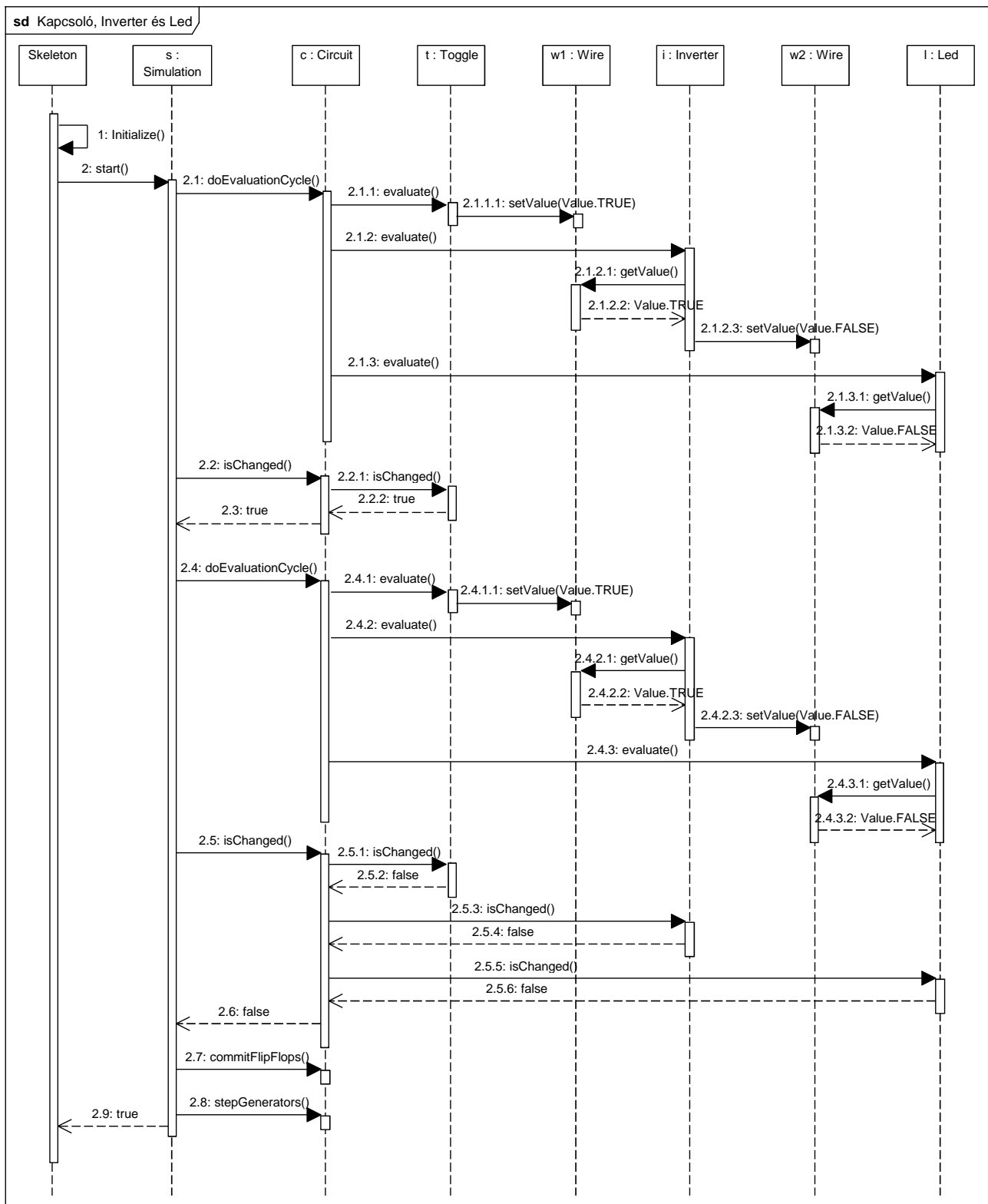
## 5.5. Szekvencia diagramok a belső működésre



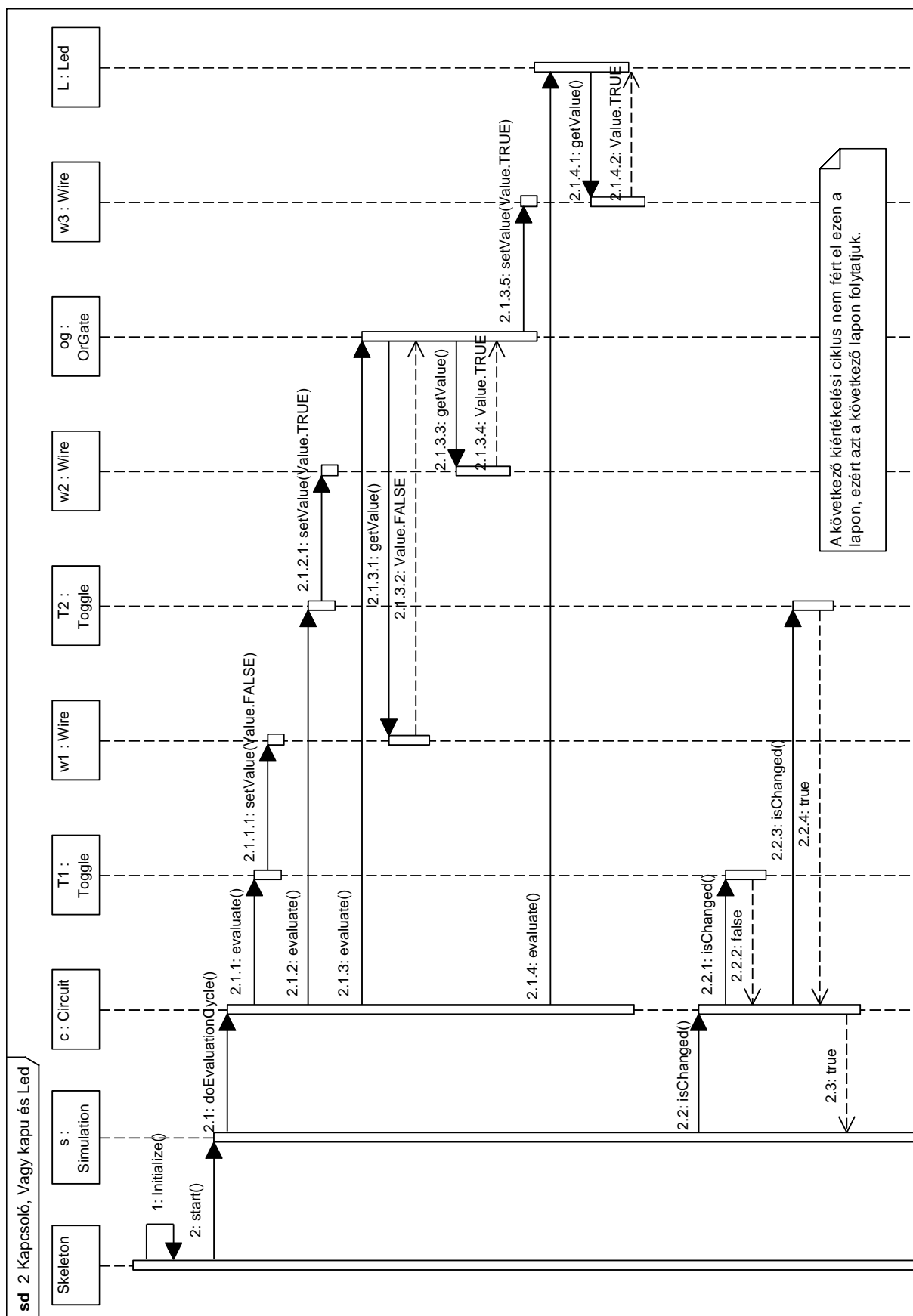
5.3. ábra. Áramkör inicializálása



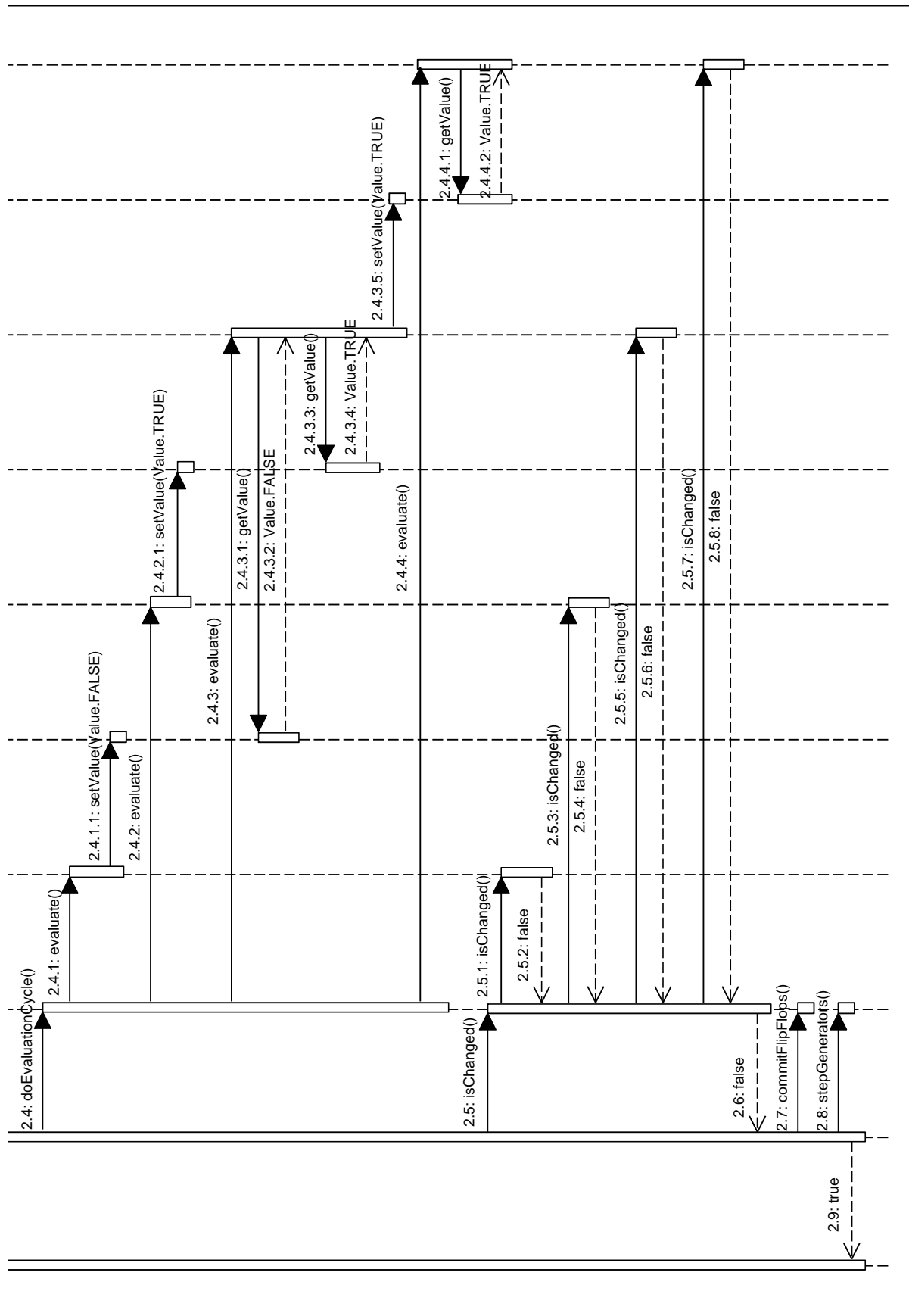
5.4. ábra. Kapcsoló és Led



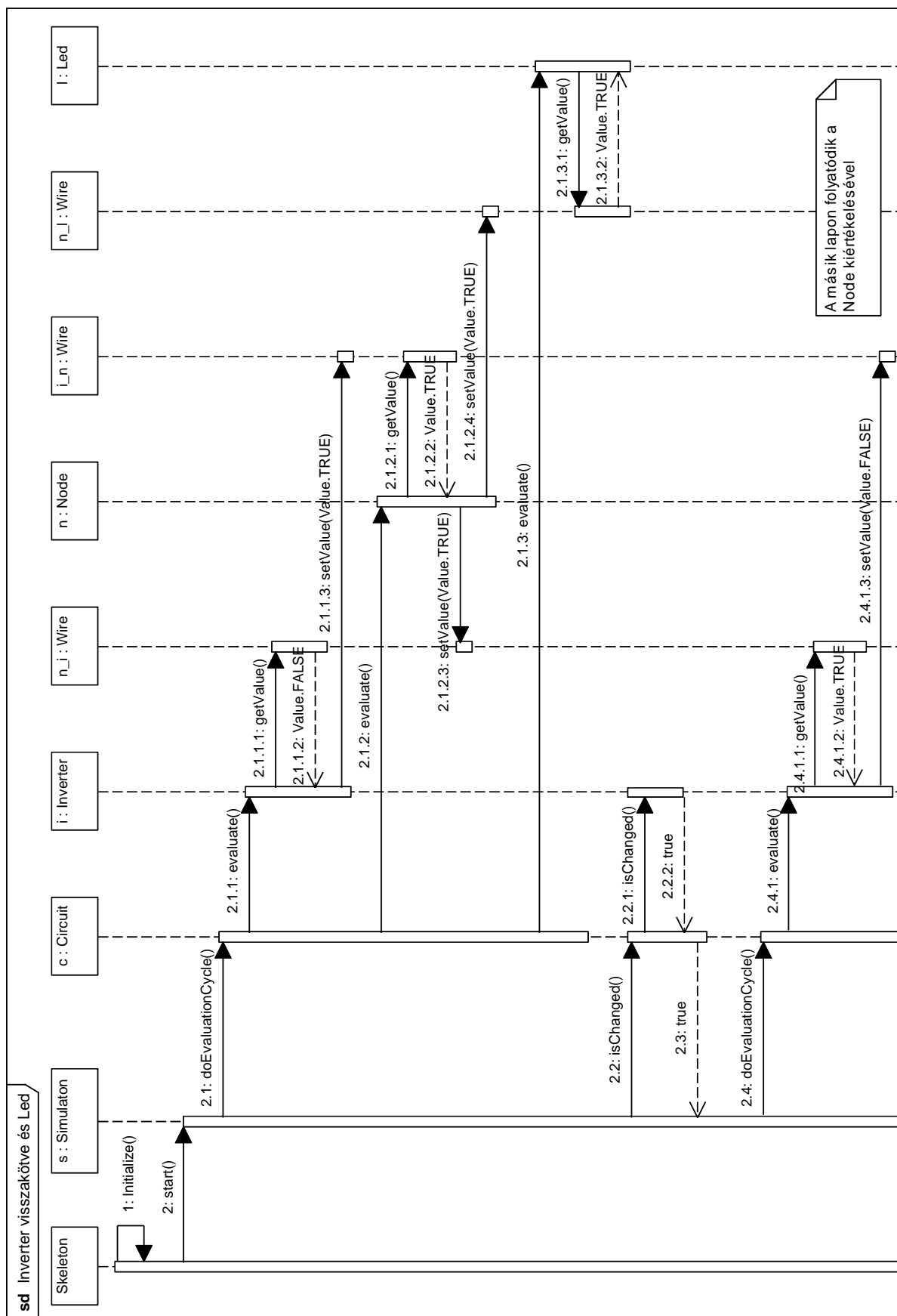
5.5. ábra. Kapcsoló, Inverter és Led



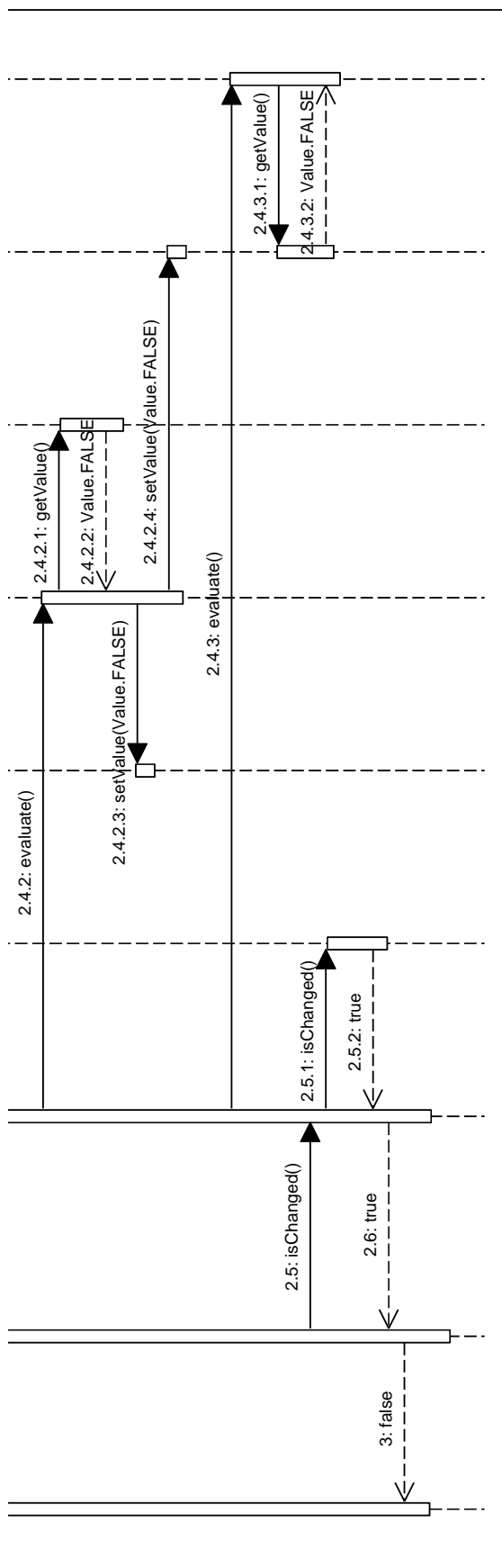
5.6. ábra. 2 Kapcsoló, Vagy kapu és Led (1. rész)



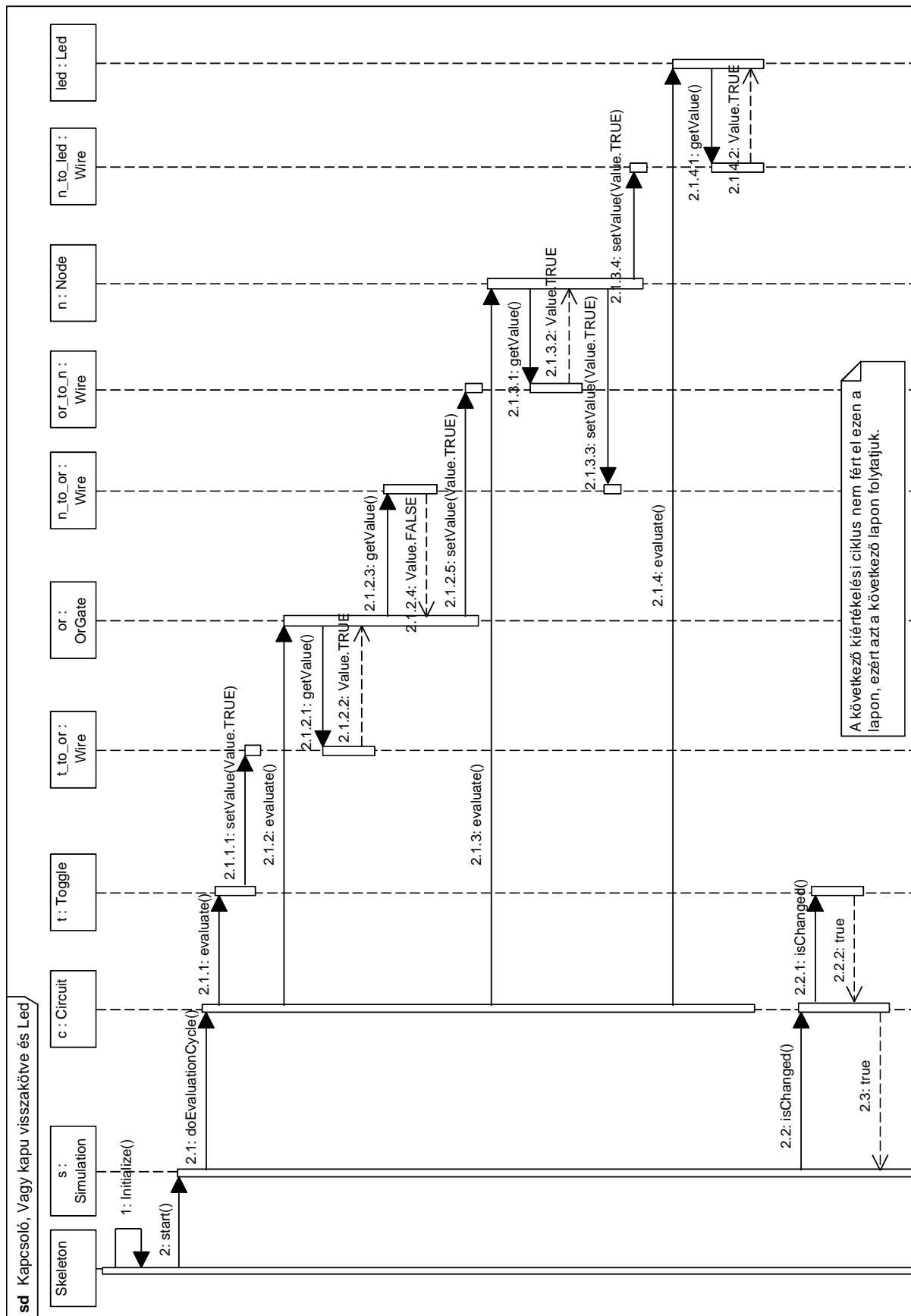
5.7. ábra. 2 Kapcsoló, Vagy kapu és Led (2. rész)



5.8. ábra. Inverter visszakötte és Led (1. rész)

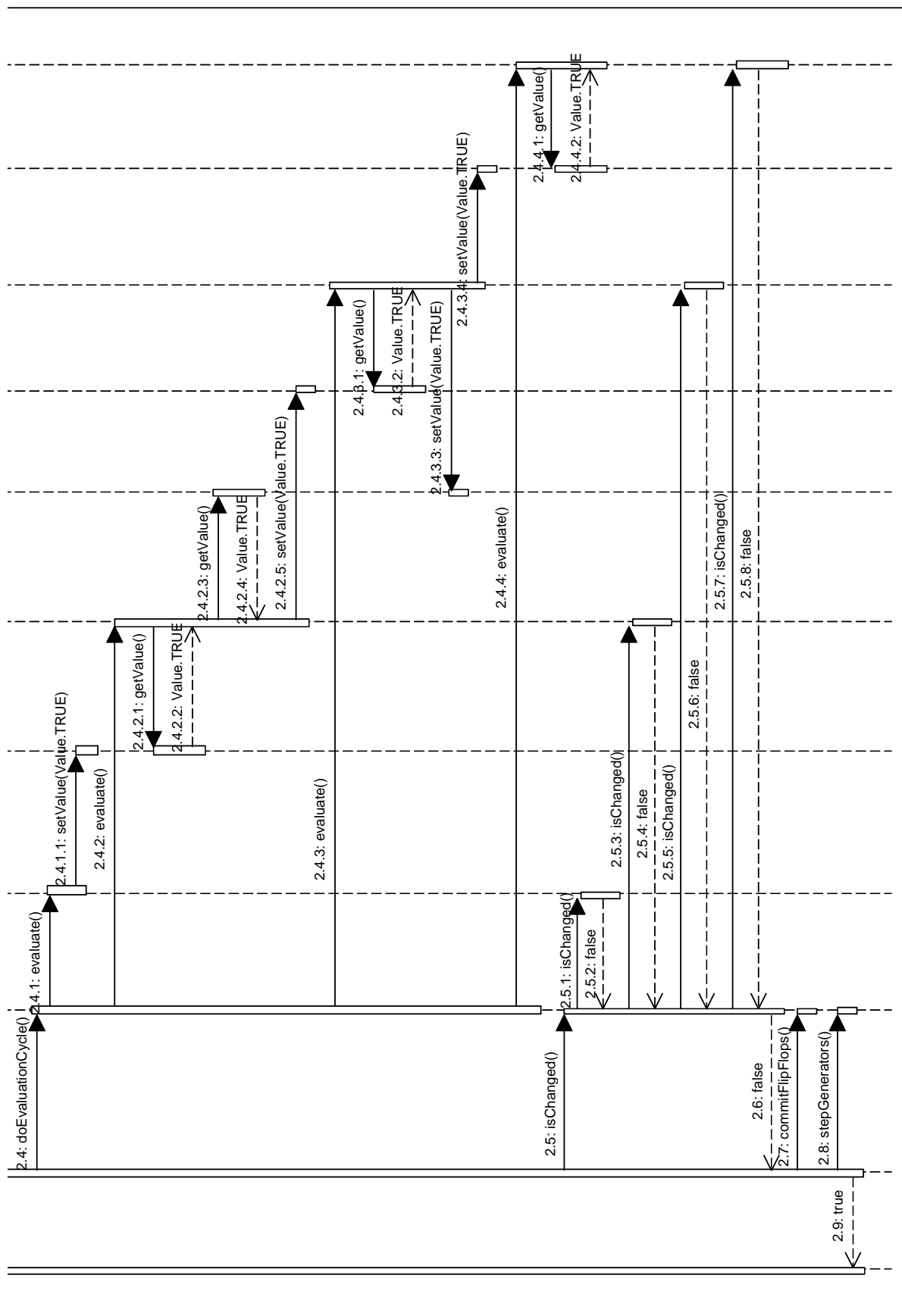


5.9. ábra. Inverter visszakötve és Led (2. rész)



5.10. ábra. Kapcsoló, Vagy kapu visszakötve és Led (1. rész)





5.11. ábra. Kapcsoló, Vagy kapu visszakötve és Led (2. rész)

## 5.6. Napló

Kezdet	Időtartam	Résztvevők	Leírás
2010.03.11. 14:00	1,5 óra	<b>Kriván B.</b>	Javasolt módosítások elvégzése az előző fejezetben, rövid errate készítése jelen fejezet elé.
2010.03.12. 00:00	2 óra	<b>Péter T.</b>	Use-casek leírása szöveges formátumban
2010.03.12. 09:30	30 perc	<b>Kriván B.</b>	Use-case diagram megrajzolása
2010.03.12. 10:00	2 óra	<b>Kriván B.</b>	Use-casek leírásának L <sup>A</sup> T <sub>E</sub> X formátumra való alakítása, apróbb finomítások
2010.03.13. 16:00	2 óra	<b>Apagyi G.</b>	Első 3 (5.3, 5.4, 5.5) szekvenciadiagram megrajzolása
2010.03.13. 17:00	1 óra	<b>Kriván B.</b> <b>Dévényi A.</b> <b>Péter T.</b>	Értekezlet Döntés: kimeneti és bemeneti értékeket egyaránt bekérjük a felhasználótól, ciklusok száma: 2 (elégnek kell lennie ezeknél az áramköröknél), nagy diagramokat félbevágjuk
2010.03.13. 18:00	2 óra	<b>Kriván B.</b>	5.3, 5.4, és 5.5 diagramok és az utolsó (5.10 és 5.11) szekvenciadiagram formázása, tömörítése, leírások frissítése
2010.03.13. 18:00	3 óra	<b>Dévényi A.</b>	5.6 és 5.7 diagram formázása, tömörítése és a 5.8 és 5.9 szekvenciadiagram megrajzolása leírások frissítése
2010.03.13. 18:00	1 óra	<b>Jákli G.</b>	5.4-es fejezet megírása

## 6. Szkeleton beadás

### 6.1. Fordítási és futtatási útmutató

#### 6.1.1. Fájllista

Fájl neve	Méret	Keletkezés ideje	Tartalom
compile.bat	178 byte	2011.03.19. 11:11	Fordításra használt batch fájl
doc.bat	276 byte	2011.03.19. 11:11	Dokumentáció generálására készített batch fájl
run.bat	106 byte	2011.03.19. 11:11	Futtatáshoz használt batch fájl
src/logsim/Skeleton.java	6246 byte	2011.03.13. 14:34	Menüt tartalmazza. A felhasználó választása alapján a megfelelő tesztesetet elindítja
src/logsim/log/Loggable.java	362 byte	2011.03.13. 14:53	Loggolást segítő interfész
src/logsim/log/LoggableInt.java	604 byte	2011.03.13. 15:49	Wrapper osztály az int típus köré, mely lehetővé teszi az intek loggolását
src/logsim/log/Logger.java	5062 byte	2011.03.13. 14:38	Loggolást megvalósító osztály
src/logsim/model/Circuit.java	4405 byte	2011.03.13. 14:36	Az áramkört megvalósító osztály
src/logsim/model/Simulation.java	1684 byte	2011.03.13. 14:36	A szimulációt megvalósító osztály
src/logsim/model/Value.java	721 byte	2011.03.13. 14:36	Az áramkörben előforduló értékeket tartalmazó osztály
src/logsim/model/component/AbstractComponent.java	3379 byte	2011.03.13. 14:46	Az alkatrészek absztrakt ősosztálya
src/logsim/model/component/DisplayComponent.java	767 byte	2011.03.13. 14:46	Megjelenítő típusú alkatrészek absztrakt ősosztálya
src/logsim/model/component/SourceComponent.java	1119 byte	2011.03.13. 14:46	Forrás típusú alkatrészek absztrakt ősosztálya
src/logsim/model/component/Wire.java	1295 byte	2011.03.13. 14:46	Vezetéket megvalósító osztály
src/logsim/model/component/impl/Inverter.java	946 byte	2011.03.13. 14:46	Az inverter alkatrészt megvalósító osztály
src/logsim/model/component/impl/Led.java	833 byte	2011.03.13. 14:46	A led megjelenítőt megvalósító osztály
src/logsim/model/component/impl/Node.java	1110 byte	2011.03.13. 14:46	Csomópont alkatrészt megvalósító osztály
src/logsim/model/component/impl/OrGate.java	1109 byte	2011.03.13. 14:46	VAGY kaput megvalósító osztály
src/logsim/model/component/impl/Toggle.java	1686 byte	2011.03.13. 14:46	A kapcsolót megvalósító osztály
src/logsim/model/skeleton/Simulation1.java	1712 byte	2011.03.19. 13:56	Az első tesztesetet tartalmazó osztály
src/logsim/model/skeleton/Simulation2.java	2147 byte	2011.03.19. 14:16	A második tesztesetet tartalmazó osztály
src/logsim/model/skeleton/Simulation3.java	2622 byte	2011.03.19. 14:17	A harmadik tesztesetet tartalmazó osztály
src/logsim/model/skeleton/Simulation4.java	2326 byte	2011.03.19. 14:17	A negyedik tesztesetet tartalmazó osztály

Fájl neve	Méret	Keletkezés ideje	Tartalom
src/logsim/model/skeleton/Simulation5.java	2924 byte	2011.03.19. 14:18	Az ötödik tesztesetet tartalmazó osztály

### 6.1.2. Fordítás

A hibamentes és minél inkább gördülékeny fordítás érdekében létrehoztunk egy `compile.bat` nevezetű batch fájlt, mely a projekt főkönyvtárában található. Projekt főkönyvtára az, amelyik a batch fájlokat és a "src" nevezetű mappát tartalmazza, melyben a program forráskódja található. Szükség esetén kézzel kell módosítani a batch fájlt

```
set C="C:\Program Files\Java\jdk1.6.0_24\bin\"
```

sorát, attól függően, hogy a gépen éppen melyik Java JDK verzió található és az hová van telepítve!

A `compile.bat` fájl az alábbi parancsokat hajtja végre:

```
@echo off
set C="C:\Program Files\Java\jdk1.6.0_24\bin\"
mkdir build
cd src
%C%\javac -d ../build logsim\Skeleton.java
cd ..
if not errorlevel 1 echo Forditas sikeres
pause
```

Ha hibamentes volt a fordítás, a "Fordítás sikeres" kimenettel értesíti a felhasználót.

A fordítás sikeressége után, lehetőség van a dokumentáció legenerálására is. Ehhez felhasználható a főkönyvtárban található `doc.bat` batch fájl. Szükség esetén kézzel kell módosítani a batch file

```
set C="C:\Program Files\Java\jdk1.6.0_24\bin\"
```

sorát, attól függően, hogy a gépen éppen melyik Java JDK verzió található és az hová van telepítve!

A batch fájl az alábbi parancsokat hajtja végre:

```
@echo off
set C="C:\Program Files\Java\jdk1.6.0_24\bin\"
cd src
%C%\javadoc logsim logsim.log logsim.model logsim.model.component ^
logsim.model.component.impl logsim.model.skeleton -d ../documents
cd ..
if not errorlevel 1 echo Dokumentum generalas sikeres volt.
pause
```

Ha a dokumentum generálás sikeres volt, akkor a documents nevezetű mappában megtalálhatóak a kívánt dokumentumok.

### 6.1.3. Futtatás

A futtatás megkönnyítése érdekében elkészítettük a `run.bat` batch fájlt. Szükség esetén kézzel kell módosítani a batch file

```
set C="C:\Program Files\Java\jdk1.6.0_24\bin\"
```

sorát, attól függően, hogy a gépen éppen melyik Java JDK verzió található és az hová van telepítve!

A `run.bat` fájl az alábbi parancsokat hajtja végre:

2011. május 13.

```
@echo off
set C="C:\Program_Files\Java\jdk1.6.0_24\bin\"
cd build
%C%\java logsim.Skeleton
cd..
PAUSE
```

A "build" könyvtárból elindítja az előzőleg lefordított programot.

## 6.2. Értékelés

Csapatunk már a múlt félévben kialakult, így nem voltunk ismeretlenek egymás számára, ismertük egymás képességeit. Az elején hamar egyezsége jutottunk, hogy miként fog történni a kommunikáció a csapaton belül és milyen módon tároljuk az elkészült anyagokat, úgy, hogy mindig mindenki a legfrissebb fájlokhoz férhessen hozzá.

A közös munka elején mégis gondot okozott számunkra a csapatban dolgozás. Ezt azonban hamar felismertük és közös megbeszéléseket tartva pontosan definiáltuk mindenkinek az aktuális munkáját és a továbbiakban figyelemmel kísértük a másik munkájának a haladását is. Előfordult, hogy a munka haladtával merültek fel kérdések, mely más csapattagok által készített megoldásokat is érintette. Ilyenkor gyors megbeszélés és döntés után a csapat probléma nélkül tudta folytatni a közös munkát. Minden döntésünkről és annak okairól leírás készül, melyet elolvasva azok a csapattagok is követni tudták a fejleményeket, akik esetleg nem tudtak résztvenni egy közös gyűlésen.

A munkaórák eloszlását közös megegyezés alapján korrigáltuk és ebből számítottuk a végső százalékokat, melyek az alábbiak szerint alakultak:

Tag	Munka százalékban	Aláírás
Apagyí G.	10 %	
Dévényi A.	25 %	
Jákli G.	25 %	
Kriván B.	30 %	
Péter T.	10 %	

## 6.3. Napló

Kezdet	Időtartam	Résztvevők	Leírás
2011.03.16. 10:00	2,5 óra	<b>Kriván B.</b>	Szkeletonhoz szükséges osztályok kialakítása, vázának elkészítése
2011.03.16. 12:00	1 óra	<b>Apagyí G.</b>	Komponensek implementációjának elkészítése
2011.03.16. 12:00	1,5 óra	<b>Jákli G.</b>	Osztályok kommentezése és az AbstractComponent átdolgozása
2011.03.16. 19:30	1 óra	<b>Jákli G.</b>	Dokumentum írása
2011.03.18. 14:00	1,5 óra	<b>Péter T.</b>	Teszt áramkörök és skeleton kommentezése
2011.03.19. 12:00	2 óra	<b>Kriván B.</b>	Végső simítások; forrás átnézése, kommentek javítása, igazítása. Skeleton által generált ki-menet összevetése a tervben megadott szek-venciákkal, dokumentáció véglegesítése

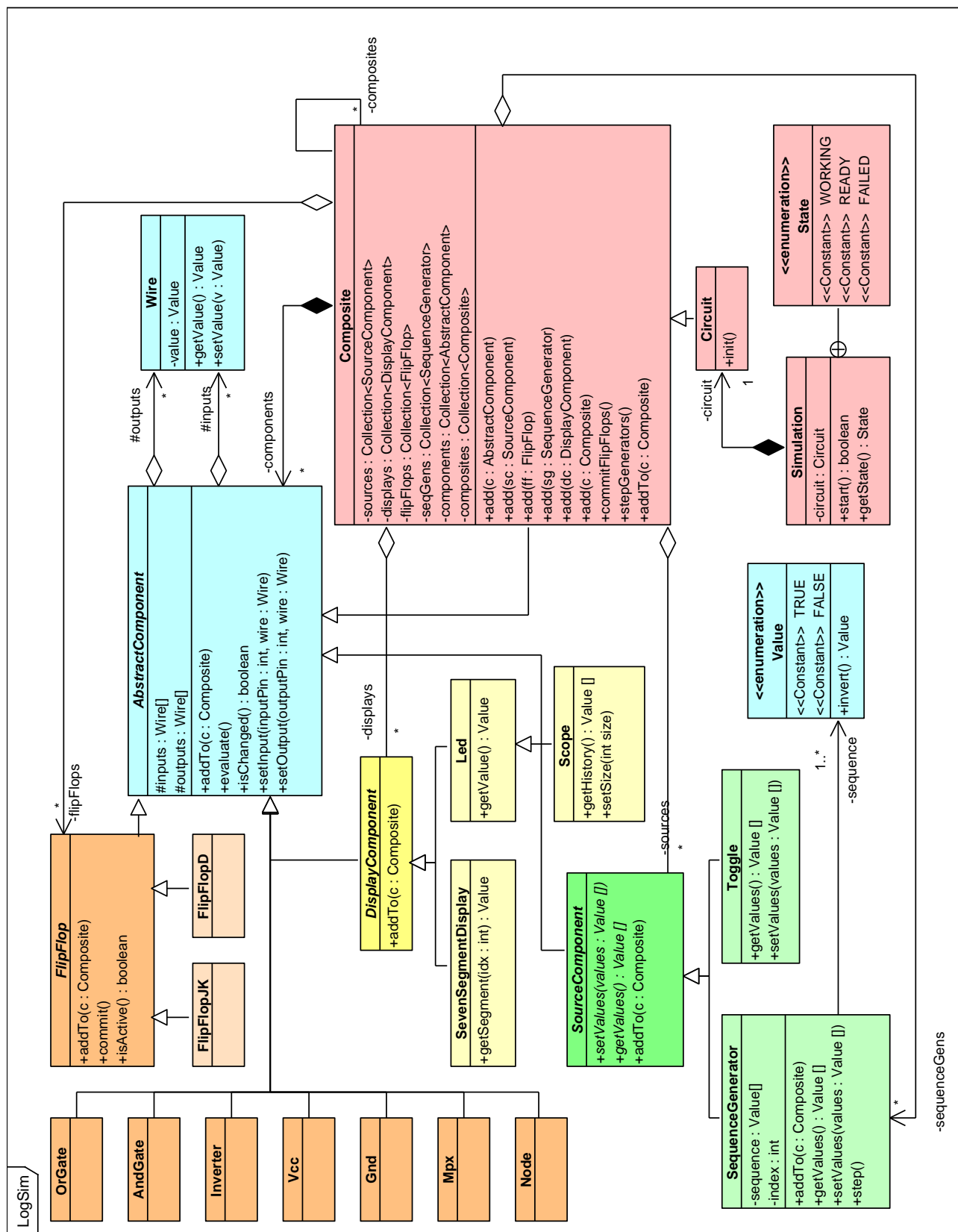
## 7. Prototípus koncepciója

### 7.0. Változtatások

A specifikáció módosulása miatt a következő változtatásokat kellett tenni a modellben. Bevezettük a Composite osztályt, mely egy kompozit áramköri elemet ír le. Mivel az áramkör is lényegében egy kompozit elem, ezért a Circuit osztály a Composite-ből öröklődik. Ha egy kompozit elemnek nincsen stacionárius állapota, akkor azt jelzi kifelé (az őt magába foglaló kompozitnak - amennyiben az áramkörrel van szó, akkor a szimulációt értesíti) és ezáltal az egész áramkörnek nem lesz stacionárius állapota. Az áramkör annyiból speciális kompozit, hogy nincsen be- és kimenete.

A másik új elem, mely bevezetésre került az oszcilloszkóp. Ez egy olyan speciális LED, mely kijelzi az aktuális értékét, de lekérhető tőle az addig eltárolt értékek is. A memóriája véges, ha megtelik, az új érték a legrégebbi érték helyére íródik be.

A megváltozott osztálydiagram a következő oldalon tekinthető meg.



7.1. ábra. Statikus struktúra nézet

## 7.1. Prototípus interface-definíciója

### 7.1.1. Az interfész általános leírása

A prototípus szabványos ki- és bemeneten kommunikál a felhasználóval. Az elkészített prototípus program egy saját parancsrendszert használ. A parancs kiadása után a program végrehajtja azt és kiírja az eredményt a kimenetre. Az automatikus tesztelés elősegítése érdekében lehetőség van arra, hogy a parancsokat egy előre elkészített fájlból olvassa és a kimenetet fájlba mentse. A program az áramkört fájlból olvassa. A tesztelés elősegítése érdekében elkészítünk néhány áramkört, azonban a felhasználó a megadott áramkört leíró fájl specifikációja alapján saját áramkört is készíthet, majd tesztelhet.

### 7.1.2. Bemeneti nyelv

#### 7.1.2.1. Felhasználói parancsok

A parancsokat a standard bemenetről, illetve fájlból olvassa be a program. Minden parancsot egy sorvége karakter zár le.

Megjegyzés: minden parancs ad visszajelzést a felhasználónak a végrehajtott eseményről, ennek formátuma a 7.1.3 alfejezetben olvasható.

*loadCircuit* <file>

- Leírás: A megadott áramkört betölti a szimulációs program. A szintaktikát lásd 7.1.2.2 fejezetet.

*loadSettings* <file>

- Leírás: A jelenlegi áramkörhöz a megadott konfigurációs fájl betöltése. A szintaktikát lásd 7.1.2.3 fejezetet.

*saveSettings* <file>

- Leírás: A pillanatnyilag használt konfiguráció fájlba mentése.

*switch* <név>

- Leírás: A megnevezett kapcsoló átállítása.

*setSeqGen* <név> <bit1>[<bit2>...]

- Leírás: A megnevezett szekvenciagenerátor az értékparaméterek szerint beállítódik.
- Megjegyzés: A szekvencia legalább 1 bitből áll.

*check* <név> | -all

- Leírás: A megadott áramköri elem bemeneteinek és kimeneteinek kilistázása.
- Opció: a *check -all* parancs kilistázza az összes áramköri elem bemenetét és kimenetét.

*step*

- Leírás: A parancs hatására lefut egy szimulációs ciklus, melynek két eredménye lehet:
  - véges lépésen belül stabilizálódik a rendszer, ekkor a kapcsoló(k), szekvenciagenerátor(ok) és ki-jelző(k) értéke(i) kiíródnak.
  - nem stabilizálódik az áramkör; hibaüzenet



## 7.1.2.2. Az áramkör leíró fájl nyelvtana

Az áramkör leíró fájlban adjuk meg a szimulálandó hálózatot. Egyszerű szövegfájl, melyben az értelmezendő parancsok soronként tagolódnak. A program feltételezi, hogy ez egy, a Követelmények c. fejezetben megfogalmazott hálózatot ír le, azaz pl. nincsen lebegő ki- és/vagy bemenet.

A fájl egy sora az alábbi formátumú:

```
<név> = <komponens> ( <paraméter1> [, <paraméter2>, ...] )
```

Megjegyzés: a szóközök nem bírnak jelentéssel, ezek csak az olvashatóságot növelik.

Minden sor egy komponensre ír le, az egyenlőség baloldalán található a komponens neve, mellyel a továbbiak során az adott komponensre, illetve annak valamelyik kimenetére hivatkozhatunk. A jobb oldalán pedig az adott komponens típusát határozzuk meg, illetve a paramétereket, melyek főképpen bemenetek:

- 0: konstans 0.
- 1: konstans 1.
- <név>[i]: egy adott komponens i. kimenete. Ha nem írjuk oda, hogy [i], akkor az alapértelmezett az, hogy a komponens 1. kimenetét kötjük oda. A név csak az angol ábc betűit és a számokat tartalmazhatja, illetve csak betűvel kezdődhet.
- Egyéb paraméter típus, ami a komponens létrehozásához kell (ilyen pl. a csomópontnak a kimeneti száma, illetve az oszcilloszkóp által eltárolható értékek száma), mely általában egy egész szám.

Ettől jelentősen eltér a kompozit definiálása (ennek felhasználása az áramkörben viszont ilyen alakú), ezt lásd később.

A lehetséges komponens típusok (és azok bemeneteinek) az alábbiak:

- OR(in1, in2 [, in3, ...])
  - Leírás: VAGY kapu létrehozása.
  - Paraméterek:
    - \* N bemenetű kapu esetén ide N db bemenet kerül. A bemenetek számát nem kell megadni, azt a feldolgozó automatikusan észleli a megkapott paraméterek számából. Minimum 2 bemenetet meg kell adni.
  - Példa: or = OR(kapcs1, kapcs2, kapcs3) - három komponens rákapcsolása a kapura, mely így egy három bemenetes vagy kapu lesz.
- AND(in1, in2 [, in3, ...])
  - Leírás: ÉS kapu létrehozása.
  - Paraméterek:
    - \* N bemenetű kapu esetén ide N db bemenet kerül. A bemenetek számát nem kell megadni, azt a feldolgozó automatikusan észleli a megkapott paraméterek számából. Minimum 2 bemenetet meg kell adni.
  - Példa: and = AND(kapcs1, kapcs2) - két komponens rákapcsolása a kapura, mely így egy két bemenetes vagy kapu lesz.
- INV(in)
  - Leírás: inverter létrehozása.
  - Paraméterek:
    - \* in: az inverter bemenete.
  - Példa: inv = INV(kapcs1) - egy kapcsoló rákapcsolása az inverterre.

- `MPX(in1, in2, in3, in4, s1, s2)`
  - Leírás: 4:1 multiplexer létrehozása.
  - Paraméterek:
    - \* `in1..in4`: a 4 adatbemenet.
    - \* `s1, s2`: a 2 kiválasztó bemenet
  - Példa: `mpx = MPX(kapcs1, kapcs2, kapcs3, kapcs4, or1, and1)`
- `FFJK(clk, J, K)`
  - Leírás: J-K flip-flop létrehozása
  - Paraméterek:
    - \* `clk`: órajel bemenet
    - \* `J`: a J bemenet
    - \* `K`: a K bemenet
  - Példa: `jk = FFJK(seqgen1, and1, and2)`
- `FFD(clk, D)`
  - Leírás: D flip-flop létrehozása.
  - Paraméterek:
    - \* `clk`: órajel bemenet
    - \* `D`: a D bemenet
  - Példa: `ffd = FFD(seqgen1, or1)`
- `LED(in)`
  - Leírás: LED létrehozása.
  - Paraméterek:
    - \* `in`: led bemenete
  - Példa: `led = LED(kapcs1)`
- `7SEG(seg1, seg2, seg3, seg4, seg5, seg6, seg7)`
  - Leírás: 7 szegmenses kijelző létrehozása.
  - Paraméterek:
    - \* `seg1..seg7`: szegmensek bemenetei
  - Példa: `display = 7SEG(seg1, seg2, seg3, seg4, seg5, seg6, seg7)`
- `TOGGLE()`
  - Leírás: kapcsoló létrehozása.
  - Példa: `t = TOGGLE()`
- `SEQGEN(seq)`
  - Leírás: szekvencia generátor létrehozása.
  - Paraméterek:
    - \* `seq`: meg kell adni egy sorozatot, melyet a generátor egymás után kiad.
  - Példa: `seqgen = SEQGEN(011000110)`
- `NODE(in, n)`

- Leírás: csomópont létrehozása.
- Paraméterek:
  - \* in: a csomópont bemenete
  - \* n: csomópont kimeneteinek a száma
- Példa: `node = NODE(kapcs1, 3)` - három kimenetű csomópontot hoz létre, melynek bemenete a `kapcs1`.
- `SCOPE(in, size)`
  - Leírás: oszcilloszkóp létrehozása.
  - Paraméterek:
    - \* in: az oszcilloszkóp bemenete
    - \* size: az oszcilloszkóp által tárolható értékek száma
  - Példa: `scope = SCOPE(kapcs1, 3)` - 3 értéket eltároló oszcilloszkóp, melynek bemenetére a `kapcs1` nevű komponens van kötve.

#### Kompozit definiálása:

A kompozitok definiálása lényegesen eltér az előzőektől, hiszen ott le kell írni a kompozit belsejét, majd erre a hálózatban hivatkozni lehet, ahogy a többi komponensre. Példa a kompozitra:

```
composite MY_COMPOSITE( x, y, z ) {

and = AND(x, y, z)
or = OR(and, y)

} (or)
```

A fenti példa egy olyan kompozitot ír le, melynek 3 bemenete van: `x`, `y` és `z`. Ezeket a bemeneteket egy 3-bemenetű ÉS és egy 2-bemenetű VAGY kapuban használjuk fel. A kompozitnak egy kimenete van, ezen pedig a VAGY kapu kimenete fog látszódni. Példa egy ilyen kompozit használatára:

```
toggle = TOGGLE()
comp = MY_COMPOSITE(toggle, 1, 1)
led = LED(comp)
```

#### Példa egy áramkör leíró fájlra:

Egy olyan minta hálózatot hozunk létre melyben található két kapcsoló egy és kapura kötve és az és kapu kimenete egy inverteren keresztül egy ledre kapcsolódik.

```
composite MY_COMPOSITE( x, y, z ) {

    and = AND(x, y, z)
    or = OR(and, y)

} (or)

t1 = TOGGLE()
t2 = TOGGLE()
comp = MY_COMPOSITE(t1, t2, 1)
led = LED(comp)
```

### 7.1.2.3. A konfigurációs fájl nyelvtana

A konfigurációs fájl minden sorában egy jelforrás beállítása szerepel, mely a következő egységekből áll:

- az elem neve
- egyenlőségjel
- az elem értéke (szekvencia generátor esetében a bitszekvenciát egybe kell megadni, lásd példa)

példa:

```
toggle1 = 0  
seqGen1 = 01101
```

### 7.1.3. Kimeneti nyelv

A program történései, visszajelzése a standard kimeneten jelennek meg, illetve ezek fájlba is kiíródhatnak. A program minden parancs után visszajelzést ad a felhasználónak a végrehajtott eseményről. A fentebb definiált parancsokra a következő jelzéseket kapja a felhasználó:

*loadCircuit <file>*

Lehetséges kimenetek

- `load successful`
  - Leírás: a betöltés sikeres, amennyiben az áramkört tartalmazó fájl szintaktikája megfelel a 7.1.2.2 alfejezetben leírtaknak.
- `load failed`
  - Leírás: a betöltés sikertelen, amennyiben az áramkört tartalmazó fájl szintaktikája nem felel meg a 7.1.2.2 alfejezetben leírtaknak.

*loadSettings <file>*

Lehetséges kimenetek

- `load successful`
  - Leírás: az értékek betöltése sikeres, amennyiben a konfigurációs fájlban szereplő áramköri elemek megfeleltethetők az aktuális áramkörben szereplő elemekkel, illetve a megadott értékek helyesek.
  - Megjegyzés: azon elemek, melyek beállítására nem volt információ a konfigurációs fájlban automatikusan nullázódnak.
- `load failed`
  - Leírás: az értékek betöltése sikeres, amennyiben a konfigurációs fájlban szereplő áramköri elem nem feleltethető meg az aktuális áramkörben szereplő elemek egyikével sem, illetve ha valamelyik érték helytelen.

*saveSettings <file>*

Lehetséges kimenetek

- `save successful`
  - Leírás: a konfigurációs értékek sikeresen fájlba mentődtek.

*switch <név>*

Lehetséges kimenetek

- `<név>: <érték>`

- Leírás: az [elem] megadja a módosított kapcsoló nevét, míg az [érték] megmutatja, hogy milyen értékre változott az aktuális kapcsoló kimenete.

*setSeqGen* <név> <bit1>[<bit2><bit3>...]

Lehetséges kimenetek

- <név>: <bit1>[<bit2><bit3>...]

- Leírás: az [elem] megadja a módosított generátor nevét, míg az [érték1, érték2, ...] megmutatja, hogy milyen értékekre változott az aktuális generátor kimenete.

*check* <név> | -all

Lehetséges kimenetek

- <név>:  
in: <in1> [, <in2>, ...]  
out: <out1> [, <out2>, ...]

- Leírás: kiírja a megadott elem ki és bemeneteit a fenti formátumban.
- Megjegyzés: a *check -all* parancsra az összes elemet kilistázza a megadott formában új sor karakterrel elválasztva

*step*

Lehetséges kimenetek

- simulation successful  
<elem1>: <érték>  
<elem2>: <érték>  
...

- Leírás: a szimuláció sikeres, amennyiben véges lépésen belül stabilizálódni tud az áramkör. Ekkor a kapcsoló(k), szekvenciagenerátor(ok) és a megjelenítő(k) értéke(i) kiíródnak.

- simulation failed

- Leírás: a szimuláció sikertelen, amennyiben véges lépésen belül nem tud stabilizálódni az áramkör.

## 7.2. Összes részletes use-case

Use-case neve	Áramkör betöltése
Rövid leírás	Az áramkört leíró fájl betöltése
Aktorok	Felhasználó
Forgatókönyv	A loadCircuit parancsot használva betöltheti az áramkört leíró fájlt, amely a program követelményeinek megfelel

Use-case neve	Konfiguráció betöltése
Rövid leírás	Egy áramkör konfigurációjának betöltése
Aktorok	Felhasználó
Forgatókönyv	A loadSettings paranccsal betölt egy egyedi a konfigurációt az áramkörhöz, amely például tartalmazhatja a szekvencia generátorok által kiadott bitsorozatot vagy a kapcsolók állását.

Use-case neve	Konfiguráció mentése
Rövid leírás	Áramkör konfigurációjának mentése

Aktorok	Felhasználó
Forgatókönyv	A saveSettings parancs kiadásával menti az aktuális áramkör konfigurációját.

<b>Use-case neve</b>	<b>Kapcsoló kapcsolása</b>
Rövid leírás	Kapcsoló állásnak módosítás
Aktorok	Felhasználó
Forgatókönyv	Az adott áramkörben a neve alapján azonosított kapcsoló állásának módosítása a switch parancs használatával.

<b>Use-case neve</b>	<b>Szekvenciagenerátor módosítás</b>
Rövid leírás	Szekvenciagenerátor bitsorozatának megadása
Aktorok	Felhasználó
Forgatókönyv	Az adott áramkörben a neve alapján azonosított szekvenciagenerátor által kiadott bitsorozat megadása a setSeqGen paranccsal.

<b>Use-case neve</b>	<b>Elem vizsgálata</b>
Rövid leírás	Egy, az áramkörben lévő alkatrész vizsgálata
Aktorok	Felhasználó
Forgatókönyv	Az adott áramkörben a check parancs használatával a megadott nevű alkatrész be- és kimeneteinek lekérdezése.

<b>Use-case neve</b>	<b>Áramkör szimulálása</b>
Rövid leírás	A betöltött áramkör szimulálása
Aktorok	Felhasználó
Forgatókönyv	A step parancs kiadásával szimulálja a betöltött áramkört.

<b>Use-case neve</b>	<b>Teszt eredményének ellenőrzése</b>
Rövid leírás	A program által generált kimenetet összehasonlítja a referencia kimenettel
Aktorok	Felhasználó
Forgatókönyv	A teszt lefutását követően egy script összehasonlítja a kapott eredményeket a várt eredményekkel.

### 7.3. Tesztelési terv

A tesztelés lehetőséget nyújt a program funkcióinak és menetének széleskörű vizsgálatára. Tesztelés során lehetőség nyílik a különböző tesztesetek kipróbálására. A teszt bemenetét bemeneti állományokból kapja, és a teszt eredményét kimeneti fájlban, és konzolon jeleníti meg. Ezáltal lehet összevetni a kiválasztott teszt várt, és tényleges eredményét.

<b>Teszt-eset neve</b>	<b>Alap áramkör</b>
Rövid leírás	Alap áramkör, mely kapcsolókból, és egyszerű nem visszacsatolt ÉS kapukból áll, kimeneti értékeket leddek jelzik
Teszt célja	Ez a teszteset leteszteli a kapcsoló, az ÉS kapu, és a led működését.

<b>Teszt-eset neve</b>	<b>MPX-es áramkör</b>
------------------------	-----------------------

Rövid leírás	Olyan áramkör, mely kapcsolókból és MPX-ből áll, a kimeneti értékeket 7 szegmenses kijelző jelzi.
Teszt célja	Ez a teszteset leteszteli a MPX és a 7szegmenses kijelző működését.

<b>Teszt-eset neve</b>	<b>Visszacsatolt stabil áramkör</b>
Rövid leírás	Olyan áramkör, melyben egy visszacsatolás történik, de az áramkör stabil marad, tehát egy VAGY kapu visszakötvé, kimenet értékét egy led jelzi.
Teszt célja	Ez a teszteset első sorban a visszacsatolás helyes működését teszteli le, de mivel visszacsatolás is történik és egyúttal a kimeneti értéket is megjelenítjük leden, ezért csomópont is kell; így másodsorban a csomópont elem helyes működését is teszteli.

<b>Teszt-eset neve</b>	<b>Visszacsatolt nem stabil áramkör</b>
Rövid leírás	Olyan áramkör, melyben egy visszacsatolás történik, de az áramkör nem lesz stabil, mert egy visszacsatolt invertert tartalmaz.
Teszt célja	Ez a teszteset azt teszteli, hogy a szimuláció ilyen esetben leáll és ezt jelzi a felhasználónak.

<b>Teszt-eset neve</b>	<b>Flip-flop-os áramkör</b>
Rövid leírás	Olyan áramkör melyben szerepel egy flipflop, egy jelgenerátor, és egy oszcilloszkóp, melyre a flipflop kimenetét kötjük
Teszt célja	E teszteset során letesztelhetjük a jelgenerátor helyes működését és megvizsgálhatjuk, hogy a flip-flop helyesen lép-e felfutó élre, illetve helyesen működik-e, valamint az oszcilloszkóp helyesen tárolja-e az értékeket.

<b>Teszt-eset neve</b>	<b>Kompozitos áramkör</b>
Rövid leírás	Egy olyan áramkör, melyben szerepel egy kompozit elem, mely egy egyszerű áramköri hálózat tartalmaz.
Teszt célja	Ez a teszteset a kompozit elem működését teszteli le, annak helyességét ellenőrizhetjük.

<b>Teszt-eset neve</b>	<b>Kompoziton belüli kompozitos áramkör</b>
Rövid leírás	Előző áramkörhöz hasonló áramkör, a különbség az, hogy a kompoziton belüli áramköri hálózat tartalmaz egy újabb kompozitot a többi elemen kívül.
Teszt célja	Ez az áramkör leteszteli, hogyan működik a program olyan esetben, mikor a kompozit további kompozitot tartalmaz, illetve a kompoziton belüli kompozit jól működik-e más elemekkel összekötve.

#### 7.4. Tesztelést támogató segéd- és fordítóprogramok specifikálása

A program által generált kimeneti fájl és az elvárt eredményeket tartalmazó fájlok összehasonlítására a DiffUtils-ban (<http://www.gnu.org/software/diffutils/>) található cmp.exe-t fogjuk használni.

## 7.5. Napló

Kezdet	Időtartam	Résztevők	Leírás
2011.03.22. 12:00	2,5 óra	<b>Apagyi G.</b>	Prototípus áramkör leíró nyelvének definiálása
2011.03.22. 14:00	1,5 óra	<b>Kriván B. Jákli G. Dévényi A.</b>	Értekezlet: Specifikáció módosítása miatt szükséges szerű változtatások megbeszélése
2011.03.22. 20:00	1 óra	<b>Jákli Gábor</b>	Összes részletes use-case
2011.03.22. 22:00	1 óra	<b>Dévényi A.</b>	Felhasználói parancsok
2011.03.23. 14:00	1 óra	<b>Dévényi A.</b>	Konfigurációs fájl nyelvtana, kimeneti nyelv
2011.03.23. 15:00	45 perc	<b>Jákli G.</b>	Új use case, 7.1.1 és 7.4
2011.03.26. 16:00	1,5 óra	<b>Péter T.</b>	Tesztelési terv és tesztesetek
2011.03.28. 11:30	2 óra	<b>Kriván B.</b>	Oszilloszkóp és kompozit elem felvétele a megfelelő fejezetekbe, illetve az osztálydiagram javítása.
2011.03.28. 12:00	30 perc	<b>Dévényi A.</b>	7.0-ás fejezet megírása.
2011.03.28. 12:30	45 perc	<b>Jákli G.</b>	A dokumentum átnézése, formázás javítása és helyesírás ellenőrzés.



## 8. Részletes tervek

### 8.1. Osztályok és metódusok tervei

#### 8.1.1. Config

- Felelősség  
Konfigurációs fájlok kezelése, azok írása az áramkör alapján, illetve azok betöltése az áramkörbe.
- Ősosztályok Object → Config.
- Interfészek (nincs)
- Attribútumok
  - `Circuit circuit` Áramkör, aminek mentjük a dolgait
  - `Pattern sourceComponentPattern` Regex kifejezés az illesztéshez (beolvasásnál)
- Metódusok
  - + `Config(Circuit circuit)`: Példány létrehozása az áramkörhöz.
  - + `int load(File file)`: Betölti egy fájlból a kapcsolók illetve jelgenerátorok konfigurációját
  - + `int save(File file)`: Elmenti a kapcsolók illetve jelgenerátorok aktuális állapotát egy fájlba

#### 8.1.2. Controller

Interfész.

- Felelősség  
Kontroller interfész.
- Ősosztályok Controller.
- Interfészek (nincs)
- Metódusok
  - + `void run(BufferedReader input)`: Vezérlés elindítása adott bemenetről.

#### 8.1.3. Parser

- Felelősség  
Áramkör értelmező objektum, feladata, hogy a paraméterként átadott, illetve fájlban elhelyezett komponenseket értelmezze, a kapcsolatokat feltérképezze, elvégezze az összeköttetéseket, és ezáltal felépítse az áramkört.
- Ősosztályok Object → Parser.
- Interfészek (nincs)
- Attribútumok
  - `Circuit circuit` A leíróból létrehozott áramkör.
  - `Pattern componentPattern` Regex minta egy komponens-sor feldolgozásához
  - `Pattern compositeEndPattern` Regex minta egy kompozit véghez

- `Map composites` **Komponensek** listája név szerint.
- `Pattern compositeStartPattern` **Regex** minta egy kompozit kezdetéhez
- `Map parameters` **Kompozitokban** lévő komponensek paraméter listája.

- **Metódusok**

- + `Parser()`:
  - `void addComponentsToComposite(Composite composite, List lines, String[] inputs, String[] outputs)`: **Komponens** hozzáadása a kompozithoz
  - `void parse(BufferedReader br)`: **Bementről** feldolgozás
- + `Circuit parse(File file)`: **Létrehoz** egy áramkört a megadott fájlból
  - `AbstractComponent parseComponentFromLine(Matcher matcher, Composite composite)`: **Egy komponens-sor** feldolgozása a fájlban

#### 8.1.4. Proto

- **Felelősség**

Prototípus vezérlő osztálya.

- **Ősosztályok** `Object` → `Proto`.

- **Interfészek** `Controller`.

- **Attribútumok**

- `Circuit c` **Áramkör**
- `Config config` **Konfiguráció** menedzselése
- `Simulation s` **Szimuláció**
- `Viewable view` **Megjelenítő**

- **Metódusok**

- + `Proto(String[] args)`:
  - `void eval(String s)`: **Parancs** értelmezése
- + `static void main(String[] args)`: **Program** belépési pontja.
- + `final void run(BufferedReader input)`: **Felhasználó** parancsait olvassa

#### 8.1.5. View

- **Felelősség**

Egy konkrét kimeneti implementáció, mely `OutputStreamWriter`-be ír ki, így a konzolos megjelenítés és fájlba írás megoldott.

- **Ősosztályok** `Object` → `View`.

- **Interfészek** `Viewable`.

- **Attribútumok**

- `Controller controller` **Kontroller**
- `PrintWriter out` **Kimeneti adatfolyam**, ide írunk.

- **Metódusok**

- + `View(Controller c, OutputStreamWriter out)`: **Létehozzuk** a `View`t egy `kontroller`rel és a `kimenettel`, ide fog menni a `kimenet`.

```

+ void newline(): Új sor a kimeneten
+ void writeDetails(AbstractComponent ac): Kiírunk egy komponenst (be és ki-
menetek)
+ void writeLedValue(Led led): Kiírja a led értékét
+ void writeLoadFailed(): Kiírjuk, hogy a betöltés sikertelen
+ void writeLoadSuccessful(): Kiírjuk, hogy a betöltés sikeres
+ void writeSaveFailed(): Kiírjuk, hogy a config fájl mentése sikertelen
+ void writeSaveSuccessful(): Kiírjuk, hogy a config fájl mentés sikeres
+ void writeScopeDetails(Scope ac): Kiírunk egy scope-ot
+ void writeScopeValues(Scope scope): Kiírja a scope által tárolt értékeket
+ void writeSequenceGenerator(SequenceGenerator sg): Szekvenciagenerátor
szekvenciájának kiírása
+ void writeSequenceGeneratorDetails(SequenceGenerator sg): Kiírunk egy
jelgenerátort
+ void writeSequenceGeneratorSequence(SequenceGenerator sg): Kiírja a
jelgenerátor szekvenciáját
+ void writeSequenceGeneratorValue(SequenceGenerator sg): Kiírja a jel-
generátor éppen kiadott értékét
+ void writeSevenSegmentDisplayValues(SevenSegmentDisplay seg): Ki-
írja a 7-szegmentes kijelző szegmenseit.
+ void writeSimulationFailed(): Kiírjuk, hogy a szimuláció sikertelen
+ void writeSimulationSuccessful(): Kiírjuk, hogy a szimuláció sikeres
+ void writeToggleValue(Toggle sc): Kiírja a kapcsoló állapotát

```

#### 8.1.6. Viewable

##### Interfész.

- Felelősség  
A kimenet interfésze.
- Ősosztályok Viewable.
- Interfészek (nincs)
- Metódusok

```

+ void newline(): Új sor a kimeneten
+ void writeDetails(AbstractComponent ac): Kiírunk egy komponenst (be és ki-
menetek)
+ void writeLedValue(Led led): Kiírja a led értékét
+ void writeLoadFailed(): Kiírjuk, hogy a betöltés sikertelen
+ void writeLoadSuccessful(): Kiírjuk, hogy betöltés sikeres
+ void writeSaveFailed(): Kiírjuk, hogy a config fájl sikertelen
+ void writeSaveSuccessful(): Kiírjuk, hogy a config fájl mentés sikeres
+ void writeScopeDetails(Scope scope): Kiírunk egy scope-ot
+ void writeScopeValues(Scope scope): Kiírja a scope által tárolt értékeket

```

```

+ void writeSequenceGenerator(SequenceGenerator sg): Szekvenciagenerátor
szekvenciájának kiírása
+ void writeSequenceGeneratorDetails(SequenceGenerator sg): Kiírnk egy
jelgenerátort
+ void writeSequenceGeneratorSequence(SequenceGenerator sg): Kiírja a
jelgenerátor szekvenciáját
+ void writeSequenceGeneratorValue(SequenceGenerator sg): Kiírja a jel-
generátor éppen kiadott értékét
+ void writeSevenSegmentDisplayValues(SevenSegmentDisplay seg): Ki-
írja a 7-szegmentes kijelző szegmenseit.
+ void writeSimulationFailed(): Kiírjuk, hogy a szimuláció sikertelen
+ void writeSimulationSuccessful(): Kiírjuk, hogy a szimuláció sikeres
+ void writeToggleValue(Toggle toggle): Kiírja a kapcsoló állapotát

```

#### 8.1.7. Circuit

- Felelősség  
Áramkört reprezentáló osztály, igazából egy kompozit. Felelőssége megegyzik a kompozitéval.
- Ősosztályok `Object` → `AbstractComponent` → `Composite` → `Circuit`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
 

```
+ Circuit():
```

#### 8.1.8. Simulation

- Felelősség  
Egy szimulációt reprezentáló objektum. Utasítja az áramkört, hogy értékelje ki magát. Ha az áramkör azt jelzi magáról, hogy nincs stacionárius állapota akkor jelezzük a felhasználónak.
- Ősosztályok `Object` → `Simulation`.
- Interfészek (nincs)
- Attribútumok
 

```
# Circuit circuit Szimulált áramkör
```
- Metódusok
 

```
+ Simulation():
+ void setCircuit(Circuit circuit): Szimulált áramkör beállítása
+ boolean start(): Egy adott bemeneti kombinációkra kiértékeli a hálózatot.
```

## 8.1.9. Value

- Felelősség  
Az áramkörben előfordulható értéket reprezentál.
- Ősosztályok `Object` → `Enum` → `Value`.
- Interfészek (nincs)
- Attribútumok
  - + `final Value FALSE`
  - + `final Value TRUE`
- Metódusok
  - `Value()`:
  - + `Value invert()`: Érték invertálása
  - + `static Value valueOf(String name)`:
  - + `static Value[] values()`:

## 8.1.10. AbstractComponent

Absztrakt osztály.

- Felelősség  
Egy komponens absztrakt megvalósítása, ebből származik az összes többi komponens. A közös logikát valósítja meg. A gyakran használt dolgokra ad alapértelmezett implementációt (kimenetekre és bemenetekre kötés, kiértékelés stb.)
- Ősosztályok `Object` → `AbstractComponent`.
- Interfészek (nincs)
- Attribútumok
  - `boolean changed` Változott-e a komponens kimenete
  - # `Wire[] inputs` Bemenetekre kötött vezetékek
  - # `String name` Komponens neve
  - # `Wire[] outputs` Kimenetekre kötött vezetékek
- Metódusok
  - # `AbstractComponent(String name, int inputCount, int outputCount)`:  
Konstruktor
  - + `void addTo(Composite composite)`: Komponens hozzáadása az áramkörhöz
  - + `abstract AbstractComponent copy(String newName)`: Lemásoljuk a komponenst.
  - + `void evaluate()`: Komponens kimeneti lábain lévő vezetékeken lévő értékek újraszámolása a bemenetek alapján.
  - # `Value getInput(int inputPin)`: Lekérjük egy adott bemenetre kötött értéket
  - + `int getInputsCount()`: Bemeneti lábak száma
  - + `Wire getInputWire(int inputPin)`: Lekérünk egy bemeneti lábon lévő vezeték
  - + `String getName()`: Komponens nevének lekérése.
  - + `int getOutputsCount()`: Kimeneti lábak száma

```

+ Wire getOutputWire(int outputPin): Lekérünk egy kimeneti lábon lévő vezetéket
+ boolean isChanged(): Visszaadja, hogy a komponensünk kimeneti értéke változott-e a kiértékelés során
# abstract void onEvaluation(): Ebben a metódusban kell implementálni az alkatrész logikáját, vagyis az adott bemenet(ek) függvényében mit kell kiadnia a kimenet(ek)re.
+ void setInput(int inputPin, Wire wire): Beállítunk egy bemenetet
+ void setOutput(int outputPin, Wire wire): Beállítunk egy kimenetet
+ void writeTo(Viewable view): Komponens kiírása a viewra.
+ void writeValueTo(Viewable view): Kiírja az értékét a viewra (csak kijelző és forrásra!)

```

### 8.1.11. Composite

- Felelősség  
Kompozit elem leírása, kiértékelésnél a tartalmazott komponenseket kiértékeli, lépteti a jelgenerátorokat stb. Ha nem áll be stacionárius állapotba a kiértékelésnél, akkor ezt jelzi kifelé.
- Ősosztályok `Object` → `AbstractComponent` → `Composite`.
- Interfészek (nincs)
- Attribútumok
  - `Map components` **Komponensek listája**
  - `List composites` **Kompozit típusú komponensek listája**
  - `final int cycleLimit` **Max. ciklusok száma**
  - `List displays` **Megjelenítő típusú komponensek listája (pl. led)**
  - `List flipFlops` **Flipflopok listája**
  - `List generators` **Jelgenerátorok listája**
  - `Node[] inputNodes` **Bemeneti csomópontok**
  - `Pattern inputPattern` **Regex minta egy komponens bemeneteinek a feldolgozásához**
  - `Node[] outputNodes` **Kimeneti csomópontok**
  - `List scopes` **Oszcillátor típusú komponensek listája**
  - `List sources` **Jelforrás típusú komponensek listája (pl. kapcsoló)**
  - `String type` **Kompozit típusa**
- Metódusok
  - + `Composite(String type, String name, int inputCount, int outputCount):` **Adott típusú és nevű komponens létrehozása a megfelelő lábszámmal.**
  - + `void add(AbstractComponent c):` **Általános típusú komponens hozzáadása**
  - + `void add(Composite c):` **Kompozit típusú komponens hozzáadása**
  - + `void add(DisplayComponent dc):` **Kijelző típusú komponens hozzáadása**
  - + `void add(FlipFlop ff):` **Flipflop komponens hozzáadása**
  - + `void add(Scope scope):` **Oszcillátor típusú komponens hozzáadása**
  - + `void add(SequenceGenerator sg):` **Jelgenerátor komponens hozzáadása**
  - + `void add(SourceComponent sc):` **Jelforrás típusú komponens hozzáadása**
  - + `void addTo(Composite composite):` **Kompozit hozzáadása kompozithoz.**

- `void commitFlipFlops()`: A flipflopok jelenlegi kimenetét elmentjük belső állapotnak, és az órajel bemenetén lévő értéket pedig eltároljuk az éldetektálás érdekében.
- `void commitScopes()`: Oszilloszkópok véglegesítése
- + `void connectComponents(Map connections, String[] inputs, String[] outputs)`: Komponensek összekötése
- + `Composite copy(String variableName)`: Kompozit lemásolása (példányosításnál használjuk.)
- + `AbstractComponent getComponentByName(String name)`: Komponens lekérése a neve alapján (delegálja a kérést, ha kell).
- + `Collection getComponents()`: Összes tartalmazott komponens listája
- + `List getDisplayComponents()`: Megjelenítők listája
- + `List getSourceComponents()`: Jelforrások listája
- + `List getStepGenerators()`: Jelgenerátorok listája
- # `void onEvaluation()`: Kiértékelési ciklus
- + `void setInput(int inputPin, Wire wire)`: Bemenet beállítása
- + `void setOutput(int outputPin, Wire wire)`: Kimenet beállítása
- `void stepGenerators()`: Jelgenerátorok léptetése

#### 8.1.12. DisplayComponent

Absztrakt osztály.

- Felelősség  
Megjelenítő típusú komponens reprezentál. Tőle származnak a megjelenítők (pl. led).
- Ősosztályok `Object` → `AbstractComponent` → `DisplayComponent`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - # `DisplayComponent(String name, int inputCount)`: Konstruktor. Nem lesz kimenete.
  - + `void addTo(Composite composite)`: Hozzáadás kompozithoz

#### 8.1.13. FlipFlop

Absztrakt osztály.

- Felelősség  
Flipflopok őszosztálya, minden flipflop 1. bemenete az órajel!
- Ősosztályok `Object` → `AbstractComponent` → `FlipFlop`.
- Interfészek (nincs)
- Attribútumok
  - # `Value clk` Előző érvényes órajel, ettől és a kiértékelés pillanatában lévő órajel értékétől függően észlelhetjük, hogy felfutó él van-e vagy sem.

```
# final int CLK Fixen az 1. bemenet az órajel
# Value q Belső memóriája, ami a kimenetén megjelenik, órajel felfutó élénél változhat az állapota.
```

- Metódusok

```
+ FlipFlop(String name, int inputCount):
+ void addTo(Composite composite): Hozzáadás kompozithoz
+ void commit(): Véglegesítés
+ boolean isActive(): Számolhat-e az FF? Ezt hívja meg az FF-ek onEvaluation() metódusa, mielőtt bármit is csinálnának.
# abstract void onCommit(): Kimenetre értékadás a logika elvégzése után.
# void onEvaluation(): Nem csinálunk semmit, majd csak commit()-nál.
```

#### 8.1.14. SourceComponent

Absztrakt osztály.

- Felelősség  
Jelforrás típusú komponenszt reprezentál. Tőle származnak a jelforrások (pl. toggle).
- Ősosztályok `Object` → `AbstractComponent` → `SourceComponent`.
- Interfészek (nincs)
- Attribútumok  
– (nincs)
- Metódusok

```
# SourceComponent(String name): Konstruktor. Nincs bemenete és egy kimenete van
+ void addTo(Composite composite): Hozzáadás kompozithoz.
+ abstract Value[] getValues(): Lekérhetjük a jelforrás értékeit, hogy el tudjuk menteni.
+ abstract void reset(): Jelforrás nullázása
+ abstract void setValues(Value[] values): Beállítjuk a jelforrás értékét. Kapcsoló esetén csak 1 elemű tömb adható paraméterként!
```

#### 8.1.15. Wire

- Felelősség  
Vezeték osztály. Két komponens-lábat köt össze. A rajta lévő érték lekérdezhető és beállítható.
- Ősosztályok `Object` → `Wire`.
- Interfészek (nincs)
- Attribútumok  
– `Value value` Vezetéken lévő érték
- Metódusok
 

```
+ Wire():
+ Value getValue(): Vezeték értékének lekérése
+ void setValue(Value value): Vezeték értékének beállítása
```



## 8.1.16. AndGate

- Felelősség  
ÉS kapu, az áramkör egyik alapeleme. Bemeneteire kötött komponensek kiértékelését kezdeményezi, s a kapott értékek logikai ÉS kapcsolatát valósítja meg, amit a kimenetén kiad.
- Ősosztályok `Object` → `AbstractComponent` → `AndGate`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
 

```
+ AndGate(int pinsCount, String name):
+ AndGate copy(String newName):
# void onEvaluation():
```

## 8.1.17. FlipFlopD

- Felelősség  
D flipflop, mely felfutó órajelnél beírja a belső memóriába az adatbemeneten (D) lévő értéket.
- Ősosztályok `Object` → `AbstractComponent` → `FlipFlop` → `FlipFlopD`.
- Interfészek (nincs)
- Attribútumok
  - `final int D` bemenet lábának a száma.
- Metódusok
 

```
+ FlipFlopD(String name):
+ FlipFlopD copy(String newName):
# void onCommit(): Flipflop logika véglegesítésnél
```

## 8.1.18. FlipFlopJK

- Felelősség  
JK flipflop, mely a belső memóriáját a Követelmények résznél leírt módon a J és K bemenetektől függően változtatja.
- Ősosztályok `Object` → `AbstractComponent` → `FlipFlop` → `FlipFlopJK`.
- Interfészek (nincs)
- Attribútumok
  - `final int J` J bemenet lábának a száma
  - `final int K` K bemenet lábának a száma
- Metódusok
 

```
+ FlipFlopJK(String name):
+ FlipFlopJK copy(String newName):
# void onCommit(): Flipflop logika véglegesítésnél
```

## 8.1.19. Gnd

- Felelősség  
A "föld" komponens, mely állandóan a hamis értéket adja ki. Nincs bemenete.
- Ősosztályok  $\text{Object} \rightarrow \text{AbstractComponent} \rightarrow \text{Gnd}$ .
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `Gnd(String name):`
  - + `AbstractComponent copy(String newName):`
  - # `void onEvaluation():`

## 8.1.20. Inverter

- Felelősség  
Inverter alkatrész, mely invertálva adja ki a kimenetén a bemenetén érkező jelet.
- Ősosztályok  $\text{Object} \rightarrow \text{AbstractComponent} \rightarrow \text{Inverter}$ .
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `Inverter(String name):` Konstruktor. 1 bemenet és 1 kimenet
  - + `AbstractComponent copy(String name):`
  - # `void onEvaluation():`

## 8.1.21. Led

- Felelősség  
Egy LED-et reprezentál, mely világít, ha bemenetén igaz érték van.
- Ősosztályok  $\text{Object} \rightarrow \text{AbstractComponent} \rightarrow \text{DisplayComponent} \rightarrow \text{Led}$ .
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `Led(String name):` Konstruktor. 1 bemenetű megjelenítő
  - + `Led copy(String name):`
  - + `Value getValue():` Visszaadja a led értékét
  - # `void onEvaluation():`
  - + `void writeValueTo(Viewable view):`

## 8.1.22. Mpx

- Felelősség  
4-1-es multiplexer, melynek a bemeneti lábak sorrendje a következő: D0, D1, D2, D3, S0, S1. Ahol Dx az adatbemenetek, Sy a kiválasztóbemenetek. Kimenetén a kiválasztóbemenetektől függően valamelyik adatbemenet kerül kiadásra.
- Ősosztályok Object → AbstractComponent → Mpx.
- Interfészek (nincs)
- Attribútumok
  - final int DATA0
  - final int DATA1
  - final int DATA2
  - final int DATA3
  - final int SEL0
  - final int SEL1
- Metódusok
  - + Mpx(String name):
  - + Mpx copy(String newName):
  - # void onEvaluation():

## 8.1.23. Node

- Felelősség  
Csomópont elem. Az egyetlen bemenetére kötött értéket kiadja az összes kimeneti lábán.
- Ősosztályok Object → AbstractComponent → Node.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + Node(int outputPinsCount, String name): **Konstruktor. 1 bemenete van**
  - + AbstractComponent copy(String name):
  - # void onEvaluation():

## 8.1.24. OrGate

- Felelősség  
VAGY kapu, az áramkör egyik alapeleme. Bemenetein lévő értékek logikai VAGY kapcsolatát valósítja meg, amit a kimenetén kiad.
- Ősosztályok Object → AbstractComponent → OrGate.
- Interfészek (nincs)
- Attribútumok
  - (nincs)

- Metódusok

```
+ OrGate(int inputPinsCount, String name): Konstruktor. 1 kimenete van
+ AbstractComponent copy(String name):
# void onEvaluation():
```

## 8.1.25. Scope

- Felelősség

Egy oszcilloszkópot reprezentál. Eltárolt értékek egy sorba kerülnek bele, mely fix méretű.

- Ősosztályok `Object` → `AbstractComponent` → `DisplayComponent` → `Led` → `Scope`.

- Interfészek (nincs)

- Attribútumok

- `Queue memory` Eltárolt értékek sora.
- `int size` Eltárolható értékek száma.

- Metódusok

```
+ Scope(int size, String name): Konstruktor. 1 bemenetű megjelenítő
+ void addTo(Composite composite): Hozzáadás kompozithoz.
+ void commit(): Eltároljuk az értéket a memóriában
+ Scope copy(String name):
+ Value[] getValues(): Visszaadja az eddig eltárolt értékeket
# void onEvaluation():
+ void writeTo(Viewable view): Komponens kiírása a viewra.
+ void writeValueTo(Viewable view): Érték kiírása a kimenetre.
```

## 8.1.26. SequenceGenerator

- Felelősség

Jelgenerátort reprezentál, amely a beállított bitsorozatot adja ki. Alapértelmezetten (amíg a felhasználó nem állítja be, vagy tölt be másikat) a 0,1-es szekvenciát tárolja.

- Ősosztályok `Object` → `AbstractComponent` → `SourceComponent` → `SequenceGenerator`.

- Interfészek (nincs)

- Attribútumok

- `int index` Bitsorozat egy indexe, ez határozza meg, hogy éppen melyik értéket adja ki.
- `Value[] sequence` Tárolt bitsorozat

- Metódusok

```
+ SequenceGenerator(String name): Konstruktor, ami alapállapotban a 0,1-es szek-
venciát állítja be.
+ void addTo(Composite composite): Hozzáadás kompozithoz.
+ SequenceGenerator copy(String newName):
+ Value[] getValues(): Jelgenerátor bitsorozatának lekérdezése
# void onEvaluation():
```

```

+ void reset(): Jelgenerátor alaphelyzetbe állítása (01-es szekvencia)
+ void setValues(Value[] values): Jelgenerátor bitsorozatának beállítása
+ void step(): A jelgenerátor lép, a bitsorozat következő elemére ugrik. A következő lépé-
tésig ez kerül kiadásra a kimeneteken.
+ void writeTo(Viewable view): Komponens kiírása a viewra.
+ void writeValueTo(Viewable view): Érték kiírása a megjelenítőre

```

#### 8.1.27. SevenSegmentDisplay

- Felelősség  
7-szegmenses kijelzőt reprezentál, melynek 7 bemenete vezérli a megfelelő szegmenseket, ezek világítanak, ha az adott bemenetre logikai igaz van kötve.
- Ősosztályok `Object` → `AbstractComponent` → `DisplayComponent` → `SevenSegmentDisplay`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
 

```

+ SevenSegmentDisplay(String name):
+ AbstractComponent copy(String newName):
+ Value getSegment(int segment): Egy szegmens értékének lekérdezése
# void onEvaluation():
+ void writeValueTo(Viewable view):

```

#### 8.1.28. Toggle

- Felelősség  
Kapcsoló jelforrás, melyet a felhasználó szimuláció közben kapcsolgathat.
- Ősosztályok `Object` → `AbstractComponent` → `SourceComponent` → `Toggle`.
- Interfészek (nincs)
- Attribútumok
  - `Value v` Kapcsoló állapota
- Metódusok
 

```

+ Toggle(String name): Konstruktor
+ AbstractComponent copy(String name):
+ Value[] getValues(): Lekérjük a kapcsoló értékét (1 elemű tömb)
# void onEvaluation():
+ void reset(): Kapcsoló alaphelyzetbe állítása.
+ void setValues(Value[] newValues): Kapcsoló állapotának változtatása, csak 1
elemű tömböt kaphat paraméterül.
+ void writeValueTo(Viewable view): Érték kiírása a kimenetre.

```

## 8.1.29. Vcc

- Felelősség  
A tápfeszültség komponens, ami konstans igaz értéket ad. Nincs bemenete.
- Ősosztályok `Object`  $\rightarrow$  `AbstractComponent`  $\rightarrow$  `Vcc`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `Vcc(String name):`
  - + `AbstractComponent copy(String newName):`
  - # `void onEvaluation():`

**8.2. A tesztek részletes tervei, leírásuk a teszt nyelvén**

## 8.2.1. Alap áramkör

- Leírás  
Olyan áramkör, melyben 2 kapcsolóval állíthatjuk egy ÉS kapu bemeneteit, melyet egy LED jelenít meg.
- Ellenőrzött funkcionalitás, várható hibahelyek  
Ellenőrizzük a kapcsoló helyes váltását, az ÉS kapu kimenetének helyes kiszámítását és a LED működését
- Áramkör létrehozása

```
kapcs1=TOGGLE()  
kapcs2=TOGGLE()  
es=AND(kapcs1, kapcs2)  
led=LED(es)
```

- Bemenet és kimenet

<i>Bemenet</i>	<i>Kimenet</i>
loadCircuit test1.ovr step switch kapcs1 step check -all switch kapcs2 step	simulation successful kapcs1: 0 kapcs2: 0 led: 0  kapcs1: 1  simulation successful kapcs1: 1 kapcs2: 0 led: 0  led: in: 0 out: kapcs1: in: out: 1 kapcs2: in: out: 0 es: in: 1, 0 out: 0  kapcs2: 1  simulation successful kapcs1: 1 kapcs2: 1 led: 1

### 8.2.2. MPX-es áramkör

- Leírás

Olyan áramkört hozunk létre, melyben egy 7 szegmenses kijelzőt hajtunk meg kapcsolókkal és egy MPX-xel. A 7szegmenses kijelző [2]-[7] bemeneteire kapcsolókat kötünk, a [1] bemenetét egy MPX adja, mely 4 kapcsolóból választja ki az egyiket, tehát egy 4/1-es MPX.

- Ellenőrzött funkcionalitás, várható hibahelyek

Ellenőrizzük a MPX helyes működését, és a 7 szegmenses kijelzőt. Hiba a MPX kiválasztása során történhet, hogy rossz jelet juttat a kimenetére.

- Áramkör létrehozása

```
inmpx1=TOGGLE()
inmpx2=TOGGLE()
inmpx3=TOGGLE()
inmpx4=TOGGLE()
selmpx1=TOGGLE()
```

```
selmpx2=TOGGLE()  
mux=MPX(inmpx1,inmpx2,inmpx3,inmpx4,selmpx1,selmpx2)  
seg=TOGGLE()  
display=7SEG(mux,seg,0,0,0,0,0)
```

- Bemenet és kimenet



<i>Bemenet</i>	<i>Kimenet</i>
loadCircuit test2.ovr	load successful
switch inmpx1	
switch inmpx3	inmpx1: 1
step	
switch selmpx2	inmpx3: 1
switch seg2	
step	simulation successful
switch selmpx2	inmpx1: 1
switch selmpx1	inmpx2: 0
step	inmpx3: 1
	inmpx4: 0
	selmpx1: 0
	selmpx2: 0
	seg: 0
	display: 1, 0, 0, 0, 0, 0, 0
	selmpx2: 1
	seg: 1
	simulation successful
	inmpx1: 1
	inmpx2: 0
	inmpx3: 1
	inmpx4: 0
	selmpx1: 0
	selmpx2: 1
	seg: 1
	display: 1, 1, 0, 0, 0, 0, 0
	selmpx2: 0
	selmpx1: 1
	simulation successful
	inmpx1: 1
	inmpx2: 0
	inmpx3: 1
	inmpx4: 0
	selmpx1: 1
	selmpx2: 0
	seg: 1
	display: 0, 1, 0, 0, 0, 0, 0

### 8.2.3. Visszacsatolt stabil áramkör

- Leírás

Egy olyan áramkört hozunk létre, melyben egy VAGY kapu szerepel, aminek egyik bemenete egy kapcsoló, kimenetét pedig visszakötjük a második bemenetére, illetve egy csomóponton keresztül egy LED-

re is eljuttatjuk.

- Ellenőrzött funkcionalitás, várható hibahelyek

Ellenőrizzük, hogy az áramkör helyesen stabilnak érzékeli-e a kapcsolást, illetve a VAGY kapu helyes működését is ellenőrizzük. Hibát a visszakötés okozhat.

- Áramkör létrehozása

```
kapcs=TOGGLE()
vagy=OR(kapcs,node[2])
node=NODE(vagy,2)
led=LED(node[1])
```

- Bemenet és kimenet

Bemenet	Kimenet
loadCircuit test3.ovr step switch kapcs step	load successful  simulation successful kapcs: 0 led: 0  kapcs: 1  simulation successful kapcs: 1 led: 1

#### 8.2.4. Visszacsatolt nem stabil áramkör

- Leírás

Egy olyan áramkört hozunk létre, melyben egy ÉS kapu szerepel, aminek egyik bemenete egy kapcsoló, kimenetét pedig visszakötjük egy inverteren keresztül a második bemenetére, illetve egy csomóponton keresztül egy LED-re is eljuttatjuk.

- Ellenőrzött funkcionalitás, várható hibahelyek

Ellenőrizzük, hogy az áramkör helyesen instabilnak érzékeli-e a kapcsolást. Hibás működést ez okozhat, tehát ha az áramkör ezt rosszul állapítja meg, és nem jelzi.

- Áramkör létrehozása

```
kapcs=TOGGLE()
inv=INV(node[2])
es=AND(kapcs,inv)
node=NODE(es,2)
led=LED(node[1])
```

- Bemenet és kimenet

<i>Bemenet</i>	<i>Kimenet</i>
loadCircuit test4.ovr switch kapcs step	load successful  kapcs: 1  simulation failed

#### 8.2.5. Flip-flop-os áramkör

- **Leírás**  
Egy olyan áramkört hozunk létre, melyben egy JK flipflop szerepel, J és K bemenetére kapcsolókat kötünk, órajelét egy jelgenerátorból kapja, és a kimenetét egy oszcilloszkóp kapja meg.
- **Ellenőrzött funkcionalitás, várható hibahelyek**  
Ellenőrizzük a jelgenerátort, hogy megfelelő jelet adja-e ciklikusan, ellenőrizzük a JK flipflop működését, illetve, hogy megfelelően lép-e az órajelre, továbbá ellenőrizzük, hogy az oszcilloszkóp helyesen működik-e. Hiba lehetséges a jelgenerátor működésében, a JK flipflop működésében illetve számolásában, és az oszcilloszkóp működésében.
- **Áramkör létrehozása**

```
j=TOGGLE()
k=TOGGLE()
seqgen=SEQGEN()
jk=FFJK(seqgen, j, k)
scope=SCOPE(jk, 3)
```

- **Bemenet és kimenet**

<i>Bemenet</i>	<i>Kimenet</i>
loadCircuit test5.ovr	load successful
switch k	
step	k: 1
step	
switch j	simulation successful
step	j: 0
step	k: 1
switch j	seqgen: 0
switch k	scope: 0
step	
step	simulation successful
	j: 0
	k: 1
	seqgen: 1
	scope: 00
	j: 1
	simulation successful
	j: 1
	k: 1
	seqgen: 0
	scope: 000
	simulation successful
	j: 1
	k: 1
	seqgen: 1
	scope: 001
	j: 0
	k: 0
	simulation successful
	j: 0
	k: 0
	seqgen: 0
	scope: 011
	simulation successful
	j: 0
	k: 0
	seqgen: 1
	scope: 111

#### 8.2.6. Kompozitos áramkör

- Leírás

Egy olyan áramkört valósítunk meg, melyben egy kompozit szerepel. Ez a kompozit egy 2 bites balról  
2011. május 13.

tölthető shiftregisztert valósít meg. A kompozitnak két bemenete van egy kapcsoló ami a balról bejövő értéket adja, és egy jelgenerátor, amely az órajelet. Belül 2 db D flipflop található összekötve. Az első flipflop kimenetét kiadja a kompozit kimenetén is, és a második flipflop bemenetére is ráadja, ezért NODE is kell. A kompozit kimenete a 2 bit és a carry.

- Ellenőrzött funkcionalitás, várható hibahelyek  
Kompozit helyes működését ellenőrizzük.
- Áramkör létrehozása

```
input=TOGGLE()  
seqgen=SEQGEN()  
composite SHR(clk, in){  
    in2 = NODE(in, 1)  
    d1 = FFD(clk, in)  
    node1 = NODE(d1, 2)  
    d2 = FFD(clk, node1[1])  
} (in2, node1[2], d2)  
myshr = SHR(seqgen, input)  
led1=LED(myshr[1])  
led2=LED(myshr[2])  
ledcarry=LED(myshr[3])
```

- Bemenet és kimenet

<i>Bemenet</i>	<i>Kimenet</i>
loadCircuit test6.ovr switch input step step switch input step step step step	load successful  input: 1  simulation successful input: 1 seqgen: 0 led1: 1 led2: 0 ledcarry: 0  simulation successful input: 1 seqgen: 1 led1: 1 led2: 0 ledcarry: 0  input: 0  simulation successful input: 0 seqgen: 0 led1: 0 led2: 1 ledcarry: 0  simulation successful input: 0 seqgen: 1 led1: 0 led2: 1 ledcarry: 0  simulation successful input: 0 seqgen: 0 led1: 0 led2: 0 ledcarry: 1  simulation successful input: 0 seqgen: 1 led1: 0 led2: 0 ledcarry: 1

## 8.2.7. Kompoziton belüli kompozitos áramkör

- Leírás

Egy olyan áramkört hozunk létre melyben egy kompozit szerepel, ami egy másik kompozitot foglal magába. A belső kompozit egyetlen invertert tartalmaz. A külső kompozit tartalmaz még egy VAGY kaput, melynek egyik bemenetére a belső kompozit kimenetét, másik bemenetére pedig a külső kompozit bemenetére érkező jelet kötjük. A külső kompozit bemenetére egy kapcsolót, kimenetére egy LED-et kötünk.

- Ellenőrzött funkcionalitás, várható hibahelyek

Leteszteljük, hogy működik-e a kompozit elem, ha belül bonyolultabb áramköri hálózat szerepel, jelen esetben egy kompozit, illetve egy VAGY kapu.

- Áramkör létrehozása

```
tog = TOGGLE()
composite innerComp(in){
  inv = INV(in)
} (inv)
composite Main(in){
  inC = innerComp(in)
  or = OR(in, inC)
} (or)
m = Main(tog)
led = LED(m)
```

- Bemenet és kimenet

<i>Bemenet</i>	<i>Kimenet</i>
loadCircuit test7.ovr	load successful
step	simulation successful
step	tog: 0
switch tog	led: 1
step	simulation successful
step	tog: 0
	led: 1
	tog: 1
	simulation successful
	tog: 1
	led: 1
	simulation successful
	tog: 1
	led: 1

### 8.3. A tesztelést támogató programok tervei

Az ellenőrizendő tesztadatokat a prototípus a kijelzőre vagy az argumentumban megadott fájlba írja a kimenetet. Ezt tudjuk összehasonlítani az előre legyártott referencia kimenettel, ami a helyes kimenetet tartalmazza. A két fájl összehasonlításához a DiffUtils cmp.exe programot használjuk.

Az ellenőrzés megkönnyítése érdekében a prototípus mellé szállítunk egy batch fájlt, amivel az összes tesztet lefut, és a generált kimenetet összehasonlítja az elvárt kimenetekkel (ezeket is szállítjuk a prototípus mellé).

A batch fájl kimenete futtatása után, minden teszt esetén az alábbi lehet:

- "Teszt sikeres!" ha a generált tesztfájl megegyezik a referencia fájlal
- Egyéb esetben pedig cmp.exe által generált hibaüzenet jelenik meg, mely megmutatja a két fájl közti eltéréseket

### 8.4. Napló

Kezdet	Időtartam	Résztvevők	Leírás
2011.04.01. 15:00	2,5 óra	<b>Péter T.</b>	Tesztesetek megtervezése, leírása, felépítésük megadása a bemeneti nyelvnek megfelelően
2011.04.02. 10:00	3 óra	<b>Apagyi G.</b>	Tesztesetek felhasználói interakciójának, illetve várt kimeneteinek megtervezése.
2011.04.04. 10:00	4 óra	<b>Kriván B.</b>	Komponensek implementálása, osztályok leírásának megcsinálása
2011.04.04. 11:00	3 óra	<b>Jákli G.</b>	Konzultáció Kriván B-vel.
2011.04.04. 10:00	4 óra	<b>Dévényi A.</b>	Tesztesetek ellenőrzése, szépítése, FF logika áttervezése.



## 10. Prototípus beadása

### 10.1. Fordítási és futtatási útmutató

#### 10.1.1. Fájllista

Fájl neve	Méret	Keletkezés ideje	Tartalom
compile.bat	178 byte	2011.04.18. 11:27	Fordításra használt batch fájl
doc.bat	276 byte	2011.04.18. 11:27	Dokumentáció generálására készített batch fájl
run.bat	106 byte	2011.04.18. 11:27	Futtatáshoz használt batch fájl
runtests.bat	106 byte	2011.04.18. 11:27	Az összes tesztet lefuttatásához használt batch file
verifytests.bat	106 byte	2011.04.18. 11:27	A kiemenetek ellenőrzésére szolgáló batch file
src/logsim/Config.java	4195 byte	2011.04.05. 11:54	A kapcsolók és szekvenciagenerátorok ki-mentéséért és betöltéséért felelős
src/logsim/Controller.java	390 byte	2011.04.05. 00:52	A vezérlés interfészét tartalmazza
src/logsim/Parser.java	10347 byte	2011.04.17. 21:58	Az áramkörleíró fájl feldolgozását végzi
src/logsim/Proto.java	5402 byte	2011.04.18. 11:27	A szimuláció működéséért felelős; felhasználói utasítások értelmezése
src/logsim/View.java	6712 byte	2011.04.16. 16:10	Konkrét kimeneti implementáció; a konzolos megjelenítésért és fájlba írásért felelős
src/logsim/Viewable.java	2475 byte	2011.04.16. 16:08	A kimenet interfésze
src/logsim/model/Circuit.java	297 byte	2011.04.05. 00:52	Áramkört reprezentáló osztály
src/logsim/model/Simulation.java	857 byte	2011.04.05. 00:52	Egy szimulációt reprezentáló osztály
src/logsim/model/Value.java	714 byte	2011.04.05. 00:52	Az áramkörben előforduló értékeket tartalmazó osztály
src/logsim/model/component/AbstractComponent.java	4588 byte	2011.04.16. 16:19	Az alkatrészek absztrakt őssztálya
src/logsim/model/component/Composite.java	14385 byte	2011.04.17. 21:58	A kompozit elem leírása
src/logsim/model/component/DisplayComponent.java	671 byte	2011.04.05. 00:52	Megjelenítő típusú alkatrészek absztrakt őssztálya
src/logsim/model/component/FlipFlop.java	1688 byte	2011.04.16. 16:21	Flipflop típusú alkatrészek absztrakt őssztálya
src/logsim/model/component/SourceComponent.java	1099 byte	2011.04.05. 11:18	Forrás típusú alkatrészek absztrakt őssztálya
src/logsim/model/component/Wire.java	600 byte	2011.04.04. 12:39	Vezetéket megvalósító osztály
src/logsim/model/component/impl/AndGate.java	915 byte	2011.04.04. 12:39	Az ÉS kapu alkatrészt megvalósító osztály
src/logsim/model/component/impl/FlipFlopD.java	870 byte	2011.04.05. 00:52	A D flipflop alkatrészt megvalósító osztály
src/logsim/model/component/impl/FlipFlopJK.java	1453 byte	2011.04.05. 00:52	A JK flipflop alkatrészt megvalósító osztály
src/logsim/model/component/impl/Gnd.java	565 byte	2011.04.04. 12:39	A permanens logikai nullát megvalósító osztály

Fájl neve	Méret	Keletkezés ideje	Tartalom
src/logsim/model/component/impl/Inverter.java	638 byte	2011.04.05. 00:52	Az inverter alkatrészt megvalósító osztály
src/logsim/model/component/impl/Led.java	846 byte	2011.04.05. 00:52	A led megjelenítőt megvalósító osztály
src/logsim/model/component/impl/Mpx.java	1222 byte	2011.04.04. 12:39	A multiplexer alkatrészt megvalósító osztály
src/logsim/model/component/impl/Node.java	1012 byte	2011.04.05. 00:52	Csomópont alkatrészt megvalósító osztály
src/logsim/model/component/impl/OrGate.java	983 byte	2011.04.05. 00:52	a VAGY kapu alkatrészt megvalósító osztály
src/logsim/model/component/impl/Scope.java	1839 byte	2011.04.05. 00:52	Oscilloszkópot megvalósító osztály
src/logsim/model/component/impl/SequenceGenerator.java	2562 byte	2011.04.04. 12:39	A szekvenciagenerátor alkatrészt megvalósító osztály
src/logsim/model/component/impl/SevenSegmentDisplay.java	1115 byte	2011.04.04. 12:39	A 7 szegmenses kijelző alkatrészt megvalósító osztály
src/logsim/model/component/impl/Toggle.java	1585 byte	2011.04.05. 11:24	A kapcsolót megvalósító osztály
src/logsim/model/component/impl/Vcc.java	537 byte	2011.04.04. 12:39	A permanens logikai egyet megvalósító osztály
diff/cmp.exe	57344 byte	2011.04.04. 12:39	A diffUtils összehasonlító exe-je
diff/libiconv2.dll	898048 byte	2011.04.04. 12:39	a cmp.exe használja
diff/libintl3.dll	92672 byte	2011.04.04. 12:39	a cmp.exe használja
tesztek/input1.txt	89 byte	2011.04.17. 21:58	Az 1. teszteset bemeneti parancsai
tesztek/input2.txt	131 byte	2011.04.17. 21:58	A 2. teszteset bemeneti parancsai
tesztek/input3.txt	49 byte	2011.04.17. 21:58	A 3. teszteset bemeneti parancsai
tesztek/input4.txt	43 byte	2011.04.17. 21:58	A 4. teszteset bemeneti parancsai
tesztek/input5.txt	99 byte	2011.04.17. 21:58	Az 5. teszteset bemeneti parancsai
tesztek/input6.txt	85 byte	2011.04.17. 21:58	A 6. teszteset bemeneti parancsai
tesztek/input7.txt	57 byte	2011.04.17. 21:58	A 7. teszteset bemeneti parancsai
tesztek/ref_output1.txt	309 byte	2011.04.18. 11:27	Az 1. teszteset elvárt kimenete
tesztek/ref_output2.txt	490 byte	2011.04.18. 11:27	A 2. teszteset elvárt kimenete
tesztek/ref_output3.txt	117 byte	2011.04.18. 11:27	A 3. teszteset elvárt kimenete
tesztek/ref_output4.txt	52 byte	2011.04.18. 11:27	A 4. teszteset elvárt kimenete
tesztek/ref_output5.txt	408 byte	2011.04.18. 11:27	Az 5. teszteset elvárt kimenete
tesztek/ref_output6.txt	505 byte	2011.04.18. 11:27	A 6. teszteset elvárt kimenete
tesztek/ref_output7.txt	193 byte	2011.04.18. 11:27	A 7. teszteset elvárt kimenete
tesztek/test1.txt	78 byte	2011.04.17. 21:58	Az 1. teszteset áramköre
tesztek/test2.txt	221 byte	2011.04.17. 21:58	A 2. teszteset áramköre
tesztek/test3.txt	83 byte	2011.04.17. 21:58	A 3. teszteset áramköre
tesztek/test4.txt	96 byte	2011.04.17. 21:58	A 4. teszteset áramköre
tesztek/test5.txt	89 byte	2011.04.17. 21:58	Az 5. teszteset áramköre
tesztek/test6.txt	263 byte	2011.04.17. 21:58	A 6. teszteset áramköre
tesztek/test7.txt	161 byte	2011.04.17. 21:58	A 7. teszteset áramköre

### 10.1.2. Fordítás

A hibamentes és minél inkább gördülékeny fordítás érdekében létrehoztunk egy `compile.bat` nevezetű batch fájlt, mely a projekt főkönyvtárban található. Projekt főkönyvtára az, amelyik a batch fájlokat és a "src" 2011. május 13.

nevezetű mappát tartalmazza, melyben a program forráskódja található. Szükség esetén kézzel kell módosítani a batch fájlt

```
set C="C:\Program Files\Java\jdk1.6.0_23\bin\"
```

sorát, attól függően, hogy a gépen éppen melyik Java JDK verzió található és az hová van telepítve!

A compile.bat fájl az alábbi parancsokat hajtja végre:

```
@echo off
set C="C:\Program_Files\Java\jdk1.6.0_23\bin\"
mkdir build
cd src
%C%\javac -d ../build logsim\Proto.java
cd..
if not errorlevel 1 echo Forditas sikeres
pause
```

Ha hibamentes volt a fordítás, a "Fordítás sikeres" kimenettel értesíti a felhasználót.

A fordítás sikeressége után, lehetőség van a dokumentáció legenerálására is. Ehhez felhasználható a főkönyvtárban található doc.bat batch fájl. Szükség esetén kézzel kell módosítani a batch file

```
set C="C:\Program Files\Java\jdk1.6.0_23\bin\"
```

sorát, attól függően, hogy a gépen éppen melyik Java JDK verzió található és az hová van telepítve!

A batch fájl az alábbi parancsokat hajtja végre:

```
@echo off
set C="C:\Program_Files\Java\jdk1.6.0_23\bin\"
cd src
%C%\javadoc logsim logsim logsim.model logsim.model.component ^
logsim.model.component.impl -d ../documents
cd..
if not errorlevel 1 echo Dokumentum generalas sikeres volt.
pause
```

Ha a dokumentum generálás sikeres volt, akkor a documents nevezetű mappában megtalálhatóak a kívánt dokumentumok.

### 10.1.3. Futtatás

A futtatás és a beépített tesztesetek ellenőrzésének megkönnyítése érdekében elkészítettük a run.bat, runtests.bat és a verifytests.bat batch fájlt. Szükség esetén kézzel kell módosítani a run.bat batch fájlt

```
set C="C:\Program Files\Java\jdk1.6.0_23\bin\"
```

sorát, attól függően, hogy a gépen éppen melyik Java JDK verzió található és az hová van telepítve!

A runtests.bat fájl lefuttatja a 7 beépített teszt esetet. A runtests.bat fájl az alábbi parancsokat hajtja végre:

```
@echo off
set C="C:\Program_Files\Java\jdk1.6.0_23\bin\"
xcopy tesztek build
cd build
%C%\java logsim.Proto input1.txt output1.txt
%C%\java logsim.Proto input2.txt output2.txt
%C%\java logsim.Proto input3.txt output3.txt
%C%\java logsim.Proto input4.txt output4.txt
%C%\java logsim.Proto input5.txt output5.txt
```

```
%C%\java logsim.Proto input6.txt output6.txt
%C%\java logsim.Proto input7.txt output7.txt
cd..
echo Tesztek lefutottak
PAUSE
```

A "build" könyvtárba outputX.txt néven lesznek a tesztek kimenetei, ahol X a teszteset számát jelüli.

A verifytests.bat fájl összehasonlítja a 7 beépített teszteset kimenetét a referencia kimenetekkel, majd egyenként kiírja, hogy a tesztesetek megegyeznek-e a referenciával. A verifytests.bat fájl az alábbi parancsokat hajtja végre:

```
@echo off
diff\cmp.exe -s build\output1.txt build\ref_output1.txt
IF %ERRORLEVEL%==0 ECHO Teszt1 (input1.txt) kimenete helyes
IF NOT %ERRORLEVEL%==0 ECHO Teszt1 (input1.txt) kimeneteben HIBA van!

diff\cmp.exe -s build\output2.txt build\ref_output2.txt
IF %ERRORLEVEL%==0 ECHO Teszt2 (input2.txt) kimenete helyes
IF NOT %ERRORLEVEL%==0 ECHO Teszt2 (input2.txt) kimeneteben HIBA van!

diff\cmp.exe -s build\output3.txt build\ref_output3.txt
IF %ERRORLEVEL%==0 ECHO Teszt3 (input3.txt) kimenete helyes
IF NOT %ERRORLEVEL%==0 ECHO Teszt1 (input3.txt) kimeneteben HIBA van!

diff\cmp.exe -s build\output4.txt build\ref_output4.txt
IF %ERRORLEVEL%==0 ECHO Teszt4 (input4.txt) kimenete helyes
IF NOT %ERRORLEVEL%==0 ECHO Teszt4 (input4.txt) kimeneteben HIBA van!

diff\cmp.exe -s build\output5.txt build\ref_output5.txt
IF %ERRORLEVEL%==0 ECHO Teszt5 (input5.txt) kimenete helyes
IF NOT %ERRORLEVEL%==0 ECHO Teszt5 (input5.txt) kimeneteben HIBA van!

diff\cmp.exe -s build\output6.txt build\ref_output6.txt
IF %ERRORLEVEL%==0 ECHO Teszt6 (input6.txt) kimenete helyes
IF NOT %ERRORLEVEL%==0 ECHO Teszt6 (input6.txt) kimeneteben HIBA van!

diff\cmp.exe -s build\output7.txt build\ref_output7.txt
IF %ERRORLEVEL%==0 ECHO Teszt7 (input7.txt) kimenete helyes
IF NOT %ERRORLEVEL%==0 ECHO Teszt7 (input7.txt) kimeneteben HIBA van!

pause
```

A run.bat fájl az alábbi parancsokat hajtja végre:

```
@echo off
set C="C:\Program_Files\Java\jdk1.6.0_23\bin\"
cd build
%C%\java logsim.Proto
cd..
PAUSE
```

A "build" könyvtárból elindítja az előzőleg lefordított programot.

**10.2. Tesztek jegyzőkönyvei**

## 10.2.1. 1. teszteset - Kapcsoló, és kapu és LED működésének vizsgálata

<b>Tesztelő neve</b>	<b>Apagyi Gábor</b>
<b>Teszt időpontja</b>	2011.04.16

## 10.2.2. 2. teszteset - Multiplexer és 7 szegmenses kijelző vizsgálata

<b>Tesztelő neve</b>	<b>Apagyi Gábor</b>
<b>Teszt időpontja</b>	2011.04.16. 11 óra 14 perc
<b>Teszt eredménye</b>	Futás idejű hiba
<b>Lehetséges hibák</b>	Mivel az eddigi tesztek sikeresen lefutottak valószínűleg a bemeneti fájlokkal lehet gond, illetve esetleg a multiplexer, vagy a 7 szegmenses kijelző implementációjával.
<b>Változtatások</b>	Az előző részben definiált felhasználói bemenetben az egyik kapcsolóra való hivatkozáskor rossz nevet írtunk a tesztesetbe: seg2, a helyes az áramkör létrehozásakor megadott seg név. A hibát javítva a teszt sikeresen lefutott.

<b>Tesztelő neve</b>	<b>Apagyi Gábor</b>
<b>Teszt időpontja</b>	2011.04.16. 11 óra 22 perc
<b>Teszt eredménye</b>	Kimenet nem megfelelő
<b>Lehetséges hibák</b>	Mivel a 7 szegmenses kijelző kimenetén az 1-esek száma megfelelő, valószínű, hogy a multiplexer belső logikájával lesz a probléma, azon belül is a kiválasztó jel és kimenet hozzárendeléssel
<b>Változtatások</b>	<p>Valóban a multiplexer implementációja volt hibás, a belső kiválasztó logikában a sorrend megcserélődött, ezt át kellett írni:</p> <pre>private static final int DATA0 = 1; private static final int DATA1 = 2; private static final int DATA2 = 3; private static final int DATA3 = 4; private static final int SEL0 = 5; private static final int SEL1 = 6;</pre> <p>Illetve a számolásnál a selected változó értékét 0-ról indítottuk, azonban a megfelelő tömb indexelésnél ez 1-ről indul.</p> <pre>int selected = 1; if (getInput(SEL0) == Value.TRUE) {     selected += 1; } if (getInput(SEL1) == Value.TRUE) {     selected += 2; }</pre> <p>A hibát javítva a teszt sikeresen és jó eredménnyel lefutott.</p>

<b>Tesztelő neve</b>	<b>Apagyi Gábor</b>
----------------------	---------------------

Teszt időpontja	2011.04.16
-----------------	------------

10.2.3. 3. tesztet - Visszacsatolt vagy kapu vizsgálata - STABIL

Tesztelő neve	Apagyi Gábor
Teszt időpontja	2011.04.16

10.2.4. 4. tesztet - Visszacsatolt és kapu és inverter vizsgálata - NEM STABIL

Tesztelő neve	Apagyi Gábor
Teszt időpontja	2011.04.16

10.2.5. 5. tesztet - JK Flip-flop és Scope vizsgálata

Tesztelő neve	Jákli Gábor
Teszt időpontja	2011.04.18

10.2.6. 6. tesztet - Kompozitos áramkör vizsgálata

Tesztelő neve	Apagyi Gábor
Teszt időpontja	2011.04.16

10.2.7. 7. tesztet - Kompoziton belül kompozit áramkör vizsgálata

Tesztelő neve	Apagyi Gábor
Teszt időpontja	2011.04.16

### 10.3. Értékelés

Tag	Munka százalékban	Aláírás
Apagyi G.	15 %	
Dévényi A.	22 %	
Jákli G.	22 %	
Kriván B.	30 %	
Péter T.	11 %	

### 10.4. Napló

Kezdet	Időtartam	Résztvevők	Leírás
2011.04.09. 10:00	6 óra	Kriván B.	Implementáció
2011.04.13. 14:00	2 óra	Kriván B. Dévényi A. Jákli G.	Összes áramköri alkatrész implementálása
2011.04.15. 16:00	1,5 óra	Jákli G.	Parancsértelmező implementálása
2011.04.16. 10:00	1,5 óra	Dévényi A.	Konfiguráció kimentésének, betöltésének implementálása

<b>Kezdet</b>	<b>Időtartam</b>	<b>Résztevők</b>	<b>Leírás</b>
2011.04.16. 10:00	2,5 óra	<b>Apagyi G.</b>	Tesztesetek futtatása, kiértékelése javítása
2011.04.17. 17:00	1,5 óra	<b>Péter T.</b>	Tesztek eredményének ellenőrzése
2011.04.18. 10:00	2 óra	<b>Dévényi A.</b>	Fájllista, kommentezés
2011.04.18. 10:00	2 óra	<b>Jákli G.</b>	Batch fájlok, diff utils, hibajavítás a paraméterkezelésben

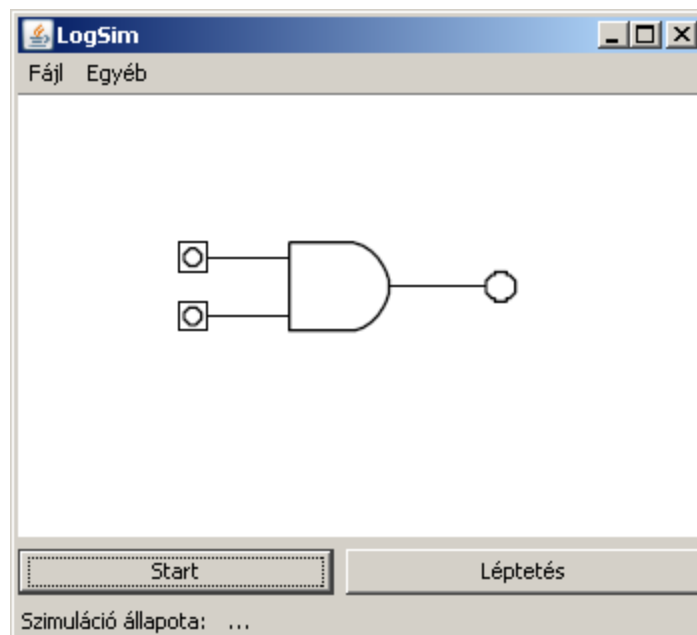
## 11. Grafikus felület specifikációja

### 11.1. A grafikus interfész

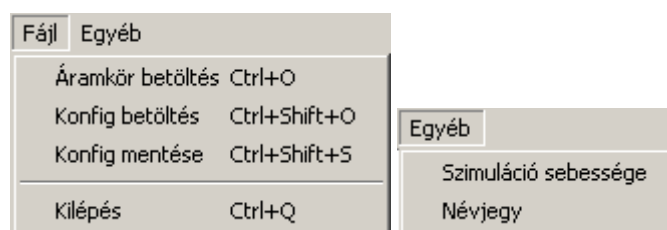
A 11.1. ábra mutatja a főablakot, a benne lévő áramkör csak illusztráció. A két menü almenüi a 11.2. ábrán látszódnak. A Fájll menü almenüi beszédesek, a felső három menüpontra megnyílik egy fájlválasztó ablak, ahol megadható egy fájl, majd az adott akció lefut. A Kilépés menüpont segítségével kiléphetünk az alkalmazásból. Az Egyéb menü Névjegyet menüpontjára kapcsolva pedig megnyílik a 11.4. ábrán látható ablak.

A Start gombra kattintva a szimuláció magától lép, az éppen beállított időközönként. Ennek módosítása az Egyéb/Szimuláció sebessége menüpontra kattintva lehetséges, ahol msec-ben megadhatjuk a két léptetés közt eltelt időt.

Lehetőség van bármikor megnézni egy komponens részleteit - ha rákattintunk egy külön ablakban láthatjuk a komponens bemeneteit és kimeneteit. Szekvenciagenerátor szekvenciájának módosítása is hasonlóképpen történik.



11.1. ábra. Főablak

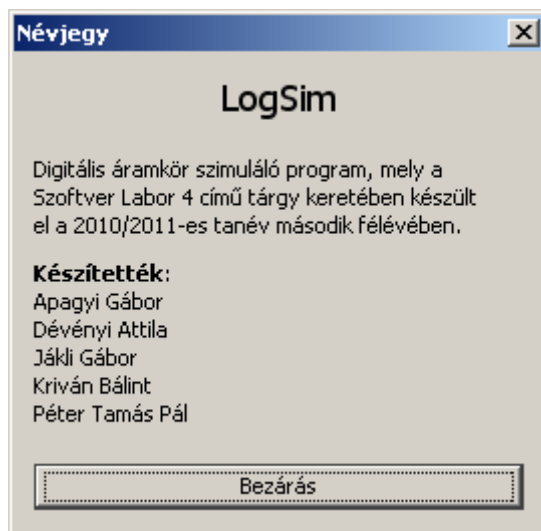


11.2. ábra. Fájl és az Egyéb menü almenüi

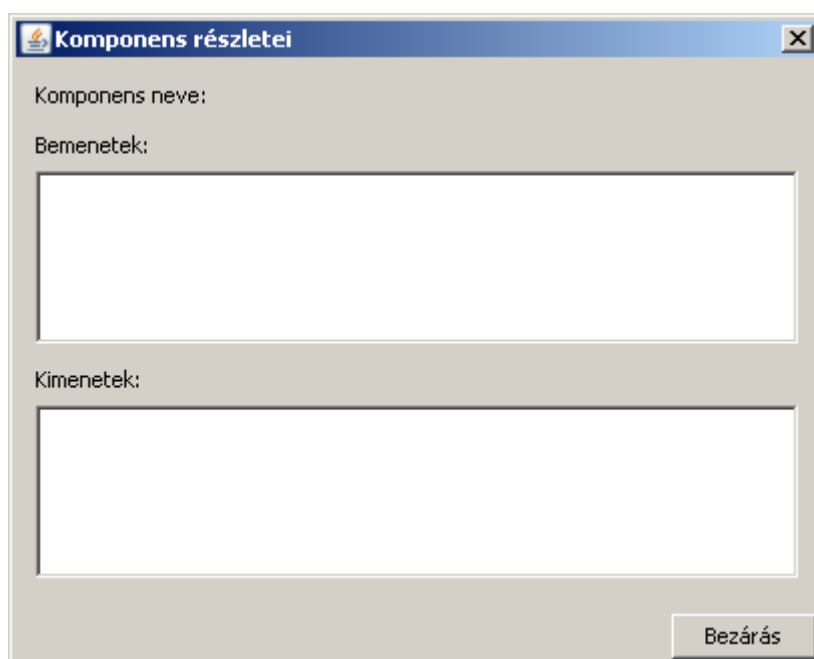




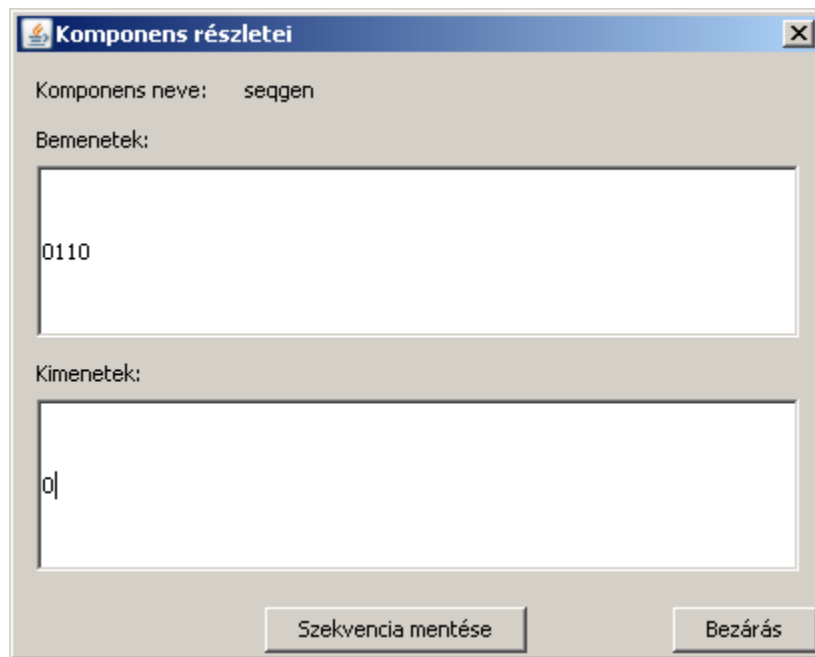
11.3. ábra. Szimuláció sebességének beállítása



11.4. ábra. Névjegy



11.5. ábra. Komponens részletei



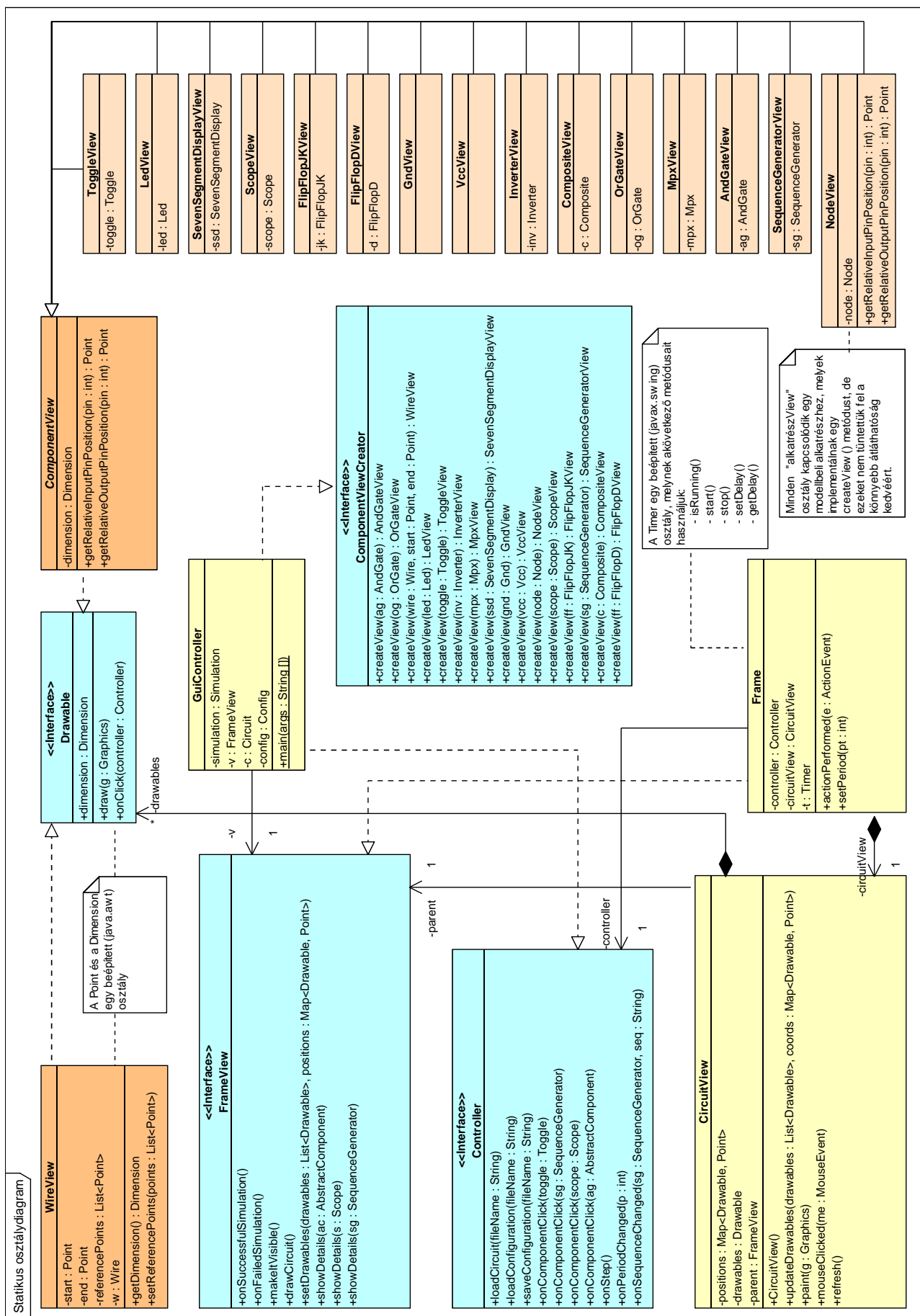
11.6. ábra. Szekvencia generátor beállítása

## 11.2. A grafikus rendszer architektúrája

### 11.2.1. A felület működési elve

Az általunk elkészített grafikus felület "pull" típusú, vagyis a grafikus rendszer kérdezi le a modell objektumoktól az aktuális állapotukat. Azokhoz a modellobjektumokhoz, melyeket megjelenítünk, elkészítettünk egy-egy wrapper osztályt, mely a megjelenítésért és a megjelenítéshez szükséges információk tárolásáért felel. Az áramkört egy JPanel-ra rajzoljuk, mely biztosítja számunkra, hogy az elhelyezhető legyen bármilyen ablakon. Áramkör újrarajzoláskor, az eltárolt objektumok egyenként rajzolják ki magukat az előzőleg megadott koordináták alapján. Bármilyen felhasználói interakciónál, melynél változhat az áramkör állapota, az egész áramkört újrarajzoljuk, biztosítva ezzel, hogy a kirajzolt áramkör mindig az aktuális állapotban legyen megjelenítve.

## 11.2.2. A felület osztály-struktúrája



### 11.3. A grafikus objektumok felsorolása

#### 11.3.1. ComponentViewCreator

Interfész.

- Felelősség  
Az egyes alkatrészekhez létrehozza a "megjeleníthető" wrapper objektumokat.
- Ősosztályok ComponentViewCreator.
- Interfészek (nincs)
- Metódusok
  - + AndGateView createView(AndGate ag): Megjeleníthető ÉS kapu létrehozása
  - + CompositeView createView(Composite c): Megjeleníthető Kompozit létrehozása
  - + FlipFlopDView createView(FlipFlopD ff): Megjeleníthető D flip-flop létrehozása
  - + FlipFlopJKView createView(FlipFlopJK ff): Megjeleníthető JK flip-flop létrehozása
  - + GndView createView(Gnd gnd): Megjeleníthető GND komponens létrehozása
  - + InverterView createView(Inverter inv): Megjeleníthető Inverter komponens létrehozása
  - + LedView createView(Led led): Megjeleníthető LED komponens létrehozása
  - + MpxView createView(Mpx mpx): Megjeleníthető Multiplexer komponens létrehozása
  - + NodeView createView(Node node): Megjeleníthető Node komponens létrehozása
  - + OrGateView createView(OrGate og): Megjeleníthető VAGY kapu létrehozása
  - + ScopeView createView(Scope scope): Megjeleníthető Scope komponens létrehozása
  - + SequenceGeneratorView createView(SequenceGenerator sg): Megjeleníthető jelgenerátor létrehozása
  - + SevenSegmentDisplayView createView(SevenSegmentDisplay ssd): Megjeleníthető Hétszegmentes komponens létrehozása
  - + ToggleView createView(Toggle toggle): Megjeleníthető Kapcsoló komponens létrehozása
  - + VccView createView(Vcc vcc): Megjeleníthető VCC komponens létrehozása
  - + WireView createView(Wire wire, Point start, Point end): Megjeleníthető vezeték létrehozása

#### 11.3.2. Controller

Interfész.

- Felelősség  
A program ezeket a szolgáltatásokat nyújtja a grafikus felület felé
- Ősosztályok Controller.
- Interfészek (nincs)
- Metódusok

```

+ void loadCircuit(String fileName): Áramkör betöltése
+ void loadConfiguration(String fileName): Áramkör konfigurációs fájl betöl-
tése
+ void onComponentClick(AbstractComponent ag): Általános komponens infor-
máció megjelenítés (név, bemenet, kimenet)
+ void onComponentClick(Scope scope): Scope megjelenítés (eddig eltárolt értékek)
+ void onComponentClick(SequenceGenerator sg): Jelgenerátor megjelenítése és
konfigurálása
+ void onComponentClick(Toggle toggle): Kapcsoló változtatása
+ void onPeriodChanged(int p): Szimuláció sebességének megváltoztatása
+ void onSequenceChanged(SequenceGenerator sg, String seq): Új szek-
vencia mentése
+ void onStep(): Áramkör léptetése
+ void saveConfiguration(String fileName): Konfigurációs fájl mentése

```

### 11.3.3. GuiController

- Felelősség  
Az alkalmazás vezérlője
- Ősosztályok Object → GuiController.
- Interfészek ComponentViewCreator, Controller.
- Attribútumok
  - Circuit c: vezérelt áramkör
  - Config config: vezérelt áramkörhöz tartozó konfiguráció
  - Simulation simulation: szimuláció
  - FrameView v: alkalmazás főablaka
- Metódusok
  - + GuiController(): Konstruktor
  - + AndGateView createView(AndGate ag): Megjeleníthető ÉS kapu létrehozása
  - + CompositeView createView(Composite c): Megjeleníthető Kompozit létrehozása
  - + FlipFlopDView createView(FlipFlopD ff): Megjeleníthető D flip-flop létreho-  
zása
  - + FlipFlopJKView createView(FlipFlopJK ff): Megjeleníthető JK flip-flop létre-  
hozása
  - + GndView createView(Gnd gnd): Megjeleníthető GND komponens létrehozása
  - + InverterView createView(Inverter inv): Megjeleníthető Inverter komponens lét-  
rehozása
  - + LedView createView(Led led): Megjeleníthető LED komponens létrehozása
  - + MpxView createView(Mpx mpx): Megjeleníthető Multiplexer komponens létrehozása
  - + NodeView createView(Node node): Megjeleníthető Node komponens létrehozása
  - + OrGateView createView(OrGate og): Megjeleníthető VAGY kapu létrehozása
  - + ScopeView createView(Scope scope): Megjeleníthető Scope komponens létreho-  
zása

```

+ SequenceGeneratorView createView(SequenceGenerator sg): Megjeleníthető
jelgenerátor létrehozása
+ SevenSegmentDisplayView createView(SevenSegmentDisplay ssd): Meg-
jeleníthető Hétszegmenses komponens létrehozása
+ ToggleView createView(Toggle toggle): Megjeleníthető Kapcsoló komponens lét-
rehozása
+ VccView createView(Vcc vcc): Megjeleníthető VCC komponens létrehozása
+ WireView createView(Wire wire, Point start, Point end): Megjeleníthető
vezeték létrehozása
+ void loadCircuit(String fileName): Áramkör betöltése
+ void loadConfiguration(String fileName): Áramkör konfigurációs fájl betöl-
tése
+ static void main(String[] args): Program belépési pontja
+ void onComponentClick(AbstractComponent ag): Általános komponens infor-
máció megjelenítés (név, bemenet, kimenet)
+ void onComponentClick(Scope scope): Scope megjelenítés (eddig eltárolt értékek)
+ void onComponentClick(SequenceGenerator sg): Jelgenerátor megjelenítése és
konfigurálása
+ void onComponentClick(Toggle toggle): Kapcsoló változtatása
+ void onPeriodChanged(int p): Szimuláció sebességének megváltoztatása
+ void onSequenceChanged(SequenceGenerator sg, String seq): Szekven-
ciagenerátor értékének változtatása
+ void onStep(): Áramkör léptetése
- void run(): főablakot kirajzoljuk
+ void saveConfiguration(String fileName): Konfigurációs fájl mentése

```

#### 11.3.4. Parser (vált.)

- Felelősség  
Áramkör értelmező objektum, feladata, hogy a paraméterként átadott, illetve fájlban elhelyezett kompo-  
nenseket értelmezze, a kapcsolatokat feltérképezze, elvégezze az összeköttetéseket, és ezáltal felépítse  
az áramkört.
- Ősosztályok Object → Parser.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + Point getPosition(AbstractComponent ac): Komponens pozíciójának a lekér-  
dezése
  - AbstractComponent parseComponent(String variableName,  
String componentName, String argumentsStr, Composite composite):
  - AbstractComponent parseComponentFromLine(Matcher matcher, Composite  
composite): Egy komponens-sor feldolgozása a fájlban
  - AbstractComponent parseTopLevelComponentFromLine(Matcher matcher,  
Circuit circuit): Egy olyan komponens-sor feldolgozása a fájlban, ami a legfelső szinten  
szerepel, azaz a kompozit amiben szerepel az az áramkör. Itt a pozíció információt is feldolgozzuk!

## 11.3.5. AbstractComponent (vált.)

Absztrakt osztály.

- Felelősség  
Egy komponens absztrakt megvalósítása, ebből származik az összes többi komponens. A közös logikát valósítja meg. A gyakran használt dolgokra ad alapértelmezett implementációt (kimenetekre és bemenetekre kötés, kiértékelés stb.)
- Ősosztályok `Object` → `AbstractComponent`.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + `ComponentView createView(ComponentViewCreator cvc)`: Lekérjük a komponens ábrázoló viewt, de a tényleges rajzolást nem mi végezzük, hanem a `ComponentViewCreator`, kihasználva a `Visitor` tervezési mintát.

## 11.3.6. Composite (vált.)

- Felelősség  
Kompozit elem leírása, kiértékelésnél a tartalmazott komponenseket kiértékeli, lépteti a jelgenerátorokat stb. Ha nem áll be stacionárius állapotba a kiértékelésnél, akkor ezt jelzi kifelé.
- Ősosztályok `Object` → `AbstractComponent` → `Composite`.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + `ComponentView createView(ComponentViewCreator cvc)`:

## 11.3.7. Wire (vált.)

- Felelősség  
Vezeték osztály. Két komponens-lábat köt össze. A rajta lévő érték lekérdezhető és beállítható.
- Ősosztályok `Object` → `Wire`.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + `WireView createView(ComponentViewCreator cvc, Point start, Point end)`: Lekérjük a vezetéket ábrázoló viewt, de a tényleges rajzolást nem mi végezzük, hanem a `ComponentViewCreator`, kihasználva a `Visitor` tervezési mintát.

## 11.3.8. AndGate (vált.)

- Felelősség  
ÉS kapu, az áramkör egyik alapeleme. Bemeneteire kötött komponensek kiértékelését kezdeményezi, s a kapott értékek logikai ÉS kapcsolatát valósítja meg, amit a kimenetén kiad.
- Ősosztályok `Object` → `AbstractComponent` → `AndGate`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.9. FlipFlopD (vált.)

- Felelősség  
D flipflop, mely felfutó órajelnél beírja a belső memóriába az adatbemeneten (D) lévő értéket.
- Ősosztályok `Object` → `AbstractComponent` → `FlipFlop` → `FlipFlopD`.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.10. FlipFlopJK (vált.)

- Felelősség  
JK flipflop, mely a belső memóriáját a Követelmények résznél leírt módon a J és K bemenetektől függően változtatja.
- Ősosztályok `Object` → `AbstractComponent` → `FlipFlop` → `FlipFlopJK`.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.11. Gnd (vált.)

- Felelősség  
A "föld" komponens, mely állandóan a hamis értéket adja ki. Nincs bemenete.
- Ősosztályok `Object` → `AbstractComponent` → `Gnd`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`



## 11.3.12. Inverter (vált.)

- Felelősség  
Inverter alkatrész, mely invertálva adja ki a kimenetén a bemenetén érkező jelet.
- Ősosztályok `Object` → `AbstractComponent` → `Inverter`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.13. Led (vált.)

- Felelősség  
Egy LED-et reprezentál, mely világít, ha bemenetén igaz érték van.
- Ősosztályok `Object` → `AbstractComponent` → `DisplayComponent` → `Led`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.14. Mpx (vált.)

- Felelősség  
4-1-es multiplexer, melynek a bemeneti lábak sorrendje a következő: D0, D1, D2, D3, S0, S1. Ahol Dx az adatbemenetek, Sy a kiválasztóbemenetek. Kimenetén a kiválasztóbemenetektől függően valamelyik adatbemenet kerül kiadásra.
- Ősosztályok `Object` → `AbstractComponent` → `Mpx`.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.15. Node (vált.)

- Felelősség  
Csomópont elem. Az egyetlen bemenetére kötött értéket kiadja az összes kimeneti lábán.
- Ősosztályok `Object` → `AbstractComponent` → `Node`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.16. OrGate (vált.)

- Felelősség  
VAGY kapu, az áramkör egyik alapeleme. Bemenetein lévő értékek logikai VAGY kapcsolatát valósítja meg, amit a kimenetén kiad.
- Ősosztályok `Object` → `AbstractComponent` → `OrGate`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.17. Scope (vált.)

- Felelősség  
Egy oszcilloszkópot reprezentál. Eltárolt értékek egy sorba kerülnek bele, mely fix méretű.
- Ősosztályok `Object` → `AbstractComponent` → `DisplayComponent` → `Led` → `Scope`.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.18. SequenceGenerator (vált.)

- Felelősség  
Jelgenerátort reprezentál, amely a beállított bitsorozatot adja ki. Alapértelmezetten (amíg a felhasználó nem állítja be, vagy tölt be másikat) a 0,1-es szekvenciát tárolja.
- Ősosztályok `Object` → `AbstractComponent` → `SourceComponent` → `SequenceGenerator`.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.19. SevenSegmentDisplay (vált.)

- Felelősség  
7-szegmenses kijelzőt reprezentál, melynek 7 bemenete vezérli a megfelelő szegmenseket, ezek világítanak, ha az adott bemenetre logikai igaz van kötve.
- Ősosztályok `Object` → `AbstractComponent` → `DisplayComponent` → `SevenSegmentDisplay`.
- Interfészek (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - + `ComponentView` `createView(ComponentViewCreator cvc):`

## 11.3.20. Toggle (vált.)

- Felelősség  
Kapcsoló jelforrás, melyet a felhasználó szimuláció közben kapcsolgathat.
- Ősosztályok `Object` → `AbstractComponent` → `SourceComponent` → `Toggle`.
- Interfészek (nincs)
- Attribútumok
- Metódusok  
+ `ComponentView createView(ComponentViewCreator cvc):`

## 11.3.21. Vcc (vált.)

- Felelősség  
A tápfeszültség komponens, ami konstans igaz értéket ad. Nincs bemenete.
- Ősosztályok `Object` → `AbstractComponent` → `Vcc`.
- Interfészek (nincs)
- Attribútumok  
– (nincs)
- Metódusok  
+ `ComponentView createView(ComponentViewCreator cvc):`

## 11.3.22. CircuitView

- Felelősség  
Áramkört kirajzoló panel.
- Ősosztályok `JPanel` → `CircuitView`.
- Interfészek `MouseListener`.
- Attribútumok
  - `List drawables`: kirajzolandók listája
  - `FrameView parent`: főablak
  - `Map positions`: kirajzolandók pozíciója
- Metódusok
  - + `CircuitView()`: Áramkört kirajzoló panel
  - + `void mouseClicked(MouseEvent me)`: Egérekattintás kezelő
  - + `void paint(Graphics g)`: Áramkör kirajzolása
  - + `void refresh()`: Áramkör újrakirajzolása.
  - + `void setParent(FrameView parent)`: Szülő beállítása
  - + `void updateDrawables(List drawables, Map coords)`: Kirajzolandó objektumok és koordinátáik beállítása

## 11.3.23. Drawable

Interfész.

- Felelősség  
Áramköri panelre rajzolható objektum.
- Ősosztályok Drawable.
- Interfészek (nincs)
- Metódusok
  - + void draw(Graphics g): Kirajzoló logika
  - + Dimension getDimension(): Lekérhetjük az objektumtól a méretét, ha beszélhetünk ilyenről.
  - + void onClick(Controller controller): Komponensre kapcsolás logikája (visszahívhat a vezérlőre)

## 11.3.24. Frame

- Felelősség  
Alkalmazás főablaka. Ő tartalmazza a CircuitView-t és a menüsört valamint a gombokat.
- Ősosztályok JFrame → Frame.
- Interfészek ActionListener, FrameView.
- Attribútumok
  - CircuitView circuitView
  - Controller controller: vezérlő
  - Timer t: időzítő
- Metódusok
  - + Frame(Controller controller): Kontruktor
  - void aboutCloseBtnActionPerformed(ActionEvent evt): Névjegy ablak bezárása
  - void aboutMIAActionPerformed(ActionEvent evt): Névjegy menüpont eseményvezérlője
  - + void actionPerformed(ActionEvent e): Timer tick eventje
  - void closeDetailedBTNActionPerformed(ActionEvent evt): Komponens részletei ablak bezárása
  - + void drawCircuit(): Áramkör kirajzolása
  - void exitMIAActionPerformed(ActionEvent evt): Kilépés menüpont eseményvezérlője
  - + Controller getController(): Lekérdezhető a vezérlő
  - void loadCircuitMIAActionPerformed(ActionEvent evt): Áramkör betöltése menüpont eseményvezérlője
  - void loadConfigMIAActionPerformed(ActionEvent evt): Konfig fájl betöltése menüpont eseményvezérlője
  - + void makeItVisible(): Megjelenítés
  - + void onFailedSimulation(): Áramkör szimulációja nem sikerült

- + void onSuccessSimulation(): Áramkör szimulációja sikeres
- void saveConfigMIActionPerformed(ActionEvent evt): Konfig fájl mentése menüpont eseményvezérlője
- void saveSeqBTNActionPerformed(ActionEvent evt): Új szekvencia elmentése
- + void setDrawables(List drawables, Map positions): Megjelenítendő objektumok és koordinátáik átadása a megjelenítőnek
- + void setPeriod(int pt): Szimuláció sebességének beállítása
- + void showDetails(AbstractComponent ac): Általános komponens részleteinek megjelenítése
- + void showDetails(Scope s): Scope részleteinek megjelenítése
- + void showDetails(SequenceGenerator sg): Szekvenciagenerátor részleteinek megjelenítése
- void simSpeedSaveBtnActionPerformed(ActionEvent evt): Új sebesség mentése
- void simulationDelayActionPerformed(ActionEvent evt): Szimuláció sebességének beállítására szolgáló ablak megjelenítése
- void StartStopActionPerformed(ActionEvent evt): Szimuláció start/stop
- void stepBtnActionPerformed(ActionEvent evt): Léptetés gomb eseményvezérlője

### 11.3.25. FrameView

#### Interfész.

- Felelősség  
Főablak interfésze
- Ősosztályok FrameView.
- Interfészek (nincs)
- Metódusok
  - + void drawCircuit(): Kirajzoljuk az áramkört.
  - + Controller getController(): Lekérdezzük a vezérlőt
  - + void makeItVisible(): Itt kell megadni, hogy a főablak, hogy tehető láthatóvá.
  - + void onFailedSimulation(): Itt adható meg, hogy mi történjen, ha nem stabil az áramkör
  - + void onSuccessSimulation(): Itt adható meg, hogy mi történjen, ha sikeres egy szimulációs lépés
  - + void setDrawables(List drawables, Map positions): Beállítjuk a kirajzolandó objektumokat és azok pozícióját.
  - + void setPeriod(int pt): Szimuláció sebességének beállítása
  - + void showDetails(AbstractComponent ac): Általános komponens részleteinek megjelenítése
  - + void showDetails(Scope s): Scope részleteinek megjelenítése
  - + void showDetails(SequenceGenerator sg): Szekvenciagenerátor részleteinek megjelenítése

## 11.3.26. ComponentView

Absztrakt osztály.

- Felelősség  
A kirajzoló wrapper objektumok őssosztálya. Itt írhatjuk le a közös kirajzoló logikákat (pl. kábelek pinjei).
- Őssosztályok `Object` → `ComponentView`.
- Interfészek `Drawable`.
- Attribútumok
  - `Dimension dimension` Szélesség-magasság
- Metódusok
  - + `ComponentView(int w, int h)`: Konstruktor a méretek megadásával.
  - + `void draw(Graphics g)`: Kirajzolási logika
  - + `Dimension getDimension()`: Lekérhetjük az objektumtól a méretét.
  - # `abstract int getInputPinsCount()`: Bemeneti pinek száma.
  - # `abstract int getOutputPinsCount()`: Kimeneti pinek száma
  - + `Point getRelativeInputPinPosition(int pin)`: Visszaadja a bemeneti pin relatív pozícióját.
  - + `Point getRelativeOutputPinPosition(int pin)`: Visszaadja a kimeneti pin relatív pozícióját.
  - # `abstract void onDraw(Graphics g)`: Komponens kirajzolásának egyedi logikája.

## 11.3.27. WireView

- Felelősség  
Egy vezetékek megjelenítéséért felelős, amit törött vonallal jelenítünk meg.
- Őssosztályok `Object` → `WireView`.
- Interfészek `Drawable`.
- Attribútumok
  - `Point end` Vezeték vége
  - `List referencePoints` Vezeték referenciapontjai, ahol a vezetékek "törnek".
  - `Point start` Vezeték kezdete
  - `Wire w` Vezeték, aminek a megjelenítéséért felel.
- Metódusok
  - + `WireView(Wire w, Point start, Point end)`: Konstruktor
  - + `void draw(Graphics g)`: Kirajzolási logika
  - + `Dimension getDimension()`:
  - + `void setReferencePoints(List referencePoints)`: Vezeték referenciapontjainak a beállítása

## 11.3.28. AndGateView

- Felelősség  
ÉS kaput kirajzoló osztály
- Ősosztályok `Object` → `ComponentView` → `AndGateView`.
- Interfészek (nincs)
- Attribútumok
  - `AndGate ag` Becsomagolt ÉS kapu
- Metódusok
  - + `AndGateView(AndGate ag):` Konstruktor
  - # `int getInputPinsCount():` Bemeneti pinek száma
  - # `int getOutputPinsCount():` Kimeneti pinek száma
  - + `void onClick(Controller controller):` ÉS kapura kattintás
  - # `void onDraw(Graphics g):` Kirajzolási logika

## 11.3.29. CompositeView

- Felelősség  
Kompozitot kirajzoló osztály
- Ősosztályok `Object` → `ComponentView` → `CompositeView`.
- Interfészek (nincs)
- Attribútumok
  - `Composite c` Becsomagolt kompozit
- Metódusok
  - + `CompositeView(Composite c):` Konstruktor
  - # `int getInputPinsCount():` Bemeneti pinek száma
  - # `int getOutputPinsCount():` Kimeneti pinek száma
  - + `void onClick(Controller controller):` Komponensre kapcsolás
  - # `void onDraw(Graphics g):` Kirajzolási logika

## 11.3.30. FlipFlopDView

- Felelősség  
D flip-flopot kirajzoló osztály
- Ősosztályok `Object` → `ComponentView` → `FlipFlopDView`.
- Interfészek (nincs)
- Attribútumok
  - `FlipFlopD d` Becsomagolt D flip-flop
- Metódusok
  - + `FlipFlopDView(FlipFlopD d):` Konstruktor
  - # `int getInputPinsCount():` Bemeneti pinek száma

```
# int getOutputPinsCount(): Kimeneti pinek száma
+ void onClick(Controller controller): D flip-flopra kattintás
# void onDraw(Graphics g): Kirajzolási logika
```

### 11.3.31. FlipFlopJKView

- Felelősség  
JK flip-flopot kirajzoló osztály
- Ősosztályok Object → ComponentView → FlipFlopJKView.
- Interfészek (nincs)
- Attribútumok
  - FlipFlopJK jk Becsomagolt JK flip-flop
- Metódusok
 

```
+ FlipFlopJKView(FlipFlopJK jk): Konstruktor
# int getInputPinsCount(): Bemeneti pinek száma
# int getOutputPinsCount(): Kimeneti pinek száma
+ void onClick(Controller controller): JK flip-flopra kattintás
# void onDraw(Graphics g): Kirajzolási logika
```

### 11.3.32. GndView

- Felelősség  
GND-t kirajzoló osztály
- Ősosztályok Object → ComponentView → GndView.
- Interfészek (nincs)
- Attribútumok
- Metódusok
 

```
+ GndView(): Konstruktor
# int getInputPinsCount(): Bemeneti pinek száma
# int getOutputPinsCount(): Kimeneti pinek száma
+ void onClick(Controller controller): Komponensre kapcsolás
# void onDraw(Graphics g): Kirajzolási logika
```

### 11.3.33. InverterView

- Felelősség  
Invertert kirajzoló osztály
- Ősosztályok Object → ComponentView → InverterView.
- Interfészek (nincs)
- Attribútumok
  - Inverter inv Becsomagolt inverter



- **Metódusok**

```
+ InverterView(Inverter inv): Konstruktor
# int getInputPinsCount(): Bemeneti pinek száma
# int getOutputPinsCount(): Kimeneti pinek száma
+ void onClick(Controller controller): Komponensre kapcsolás
# void onDraw(Graphics g): Kirajzolási logika
```

## 11.3.34. LedView

- **Felelősség**

LED-et kirajzoló osztály.

- **Ősosztályok** Object → ComponentView → LedView.

- **Interfészek** (nincs)

- **Attribútumok**

– Led led Becsomagolt Led.

- **Metódusok**

```
+ LedView(Led led): Konstruktor
# int getInputPinsCount(): Bemeneti pinek száma
# int getOutputPinsCount(): Kimeneti pinek száma
+ void onClick(Controller controller): Komponensre kapcsolás
# void onDraw(Graphics g): Kirajzolási logika
```

## 11.3.35. MpxView

- **Felelősség**

Multiplexert kirajzoló osztály

- **Ősosztályok** Object → ComponentView → MpxView.

- **Interfészek** (nincs)

- **Attribútumok**

– Mpx mpx Becsomagolt multiplexer.

- **Metódusok**

```
+ MpxView(Mpx mpx): Konstruktor
# int getInputPinsCount(): Bemeneti pinek száma
# int getOutputPinsCount(): Kimeneti pinek száma
+ void onClick(Controller controller): Komponensre kapcsolás
# void onDraw(Graphics g): Kirajzolási logika
```

## 11.3.36. NodeView

- Felelősség  
Node-ot kirajzoló osztály.
- Ősosztályok `Object` → `ComponentView` → `NodeView`.
- Interfészek (nincs)
- Attribútumok
  - `Node node` Becsomagolt csomópont
- Metódusok
  - + `NodeView(Node node):` Konstruktor
  - + `void draw(Graphics g):` Kirajzolási logika
  - # `int getInputPinsCount():` Bemeneti pinek száma
  - # `int getOutputPinsCount():` Kimeneti pinek száma
  - + `Point getRelativeInputPinPosition(int pin):` Megadott bemeneti pin relatív pozícióját adja vissza
  - + `Point getRelativeOutputPinPosition(int pin):` Megadott kimeneti pin relatív pozícióját adja vissza
  - + `void onClick(Controller controller):` Komponensre kapcsolás
  - # `void onDraw(Graphics g):` Kirajzolási logika

## 11.3.37. OrGateView

- Felelősség  
VAGY kaput kirajzoló osztály
- Ősosztályok `Object` → `ComponentView` → `OrGateView`.
- Interfészek (nincs)
- Attribútumok
  - `OrGate og` Becsomagolt VAGY kapu
- Metódusok
  - + `OrGateView(OrGate og):` Konstruktor
  - # `int getInputPinsCount():` Bemeneti pinek száma
  - # `int getOutputPinsCount():` Kimeneti pinek száma
  - + `void onClick(Controller controller):` Komponensre kapcsolás
  - # `void onDraw(Graphics g):` Kirajzolási logika

## 11.3.38. ScopeView

- Felelősség  
Scope-ot kirajzoló osztály.
- Ősosztályok `Object` → `ComponentView` → `ScopeView`.
- Interfészek (nincs)
- Attribútumok

- Scope scope **Becsomagolt** oszcilloszkóp

- Metódusok

- + ScopeView(Scope scope): **Konstruktor**
  - # int getInputPinsCount(): **Bemeneti pinek száma**
  - # int getOutputPinsCount(): **Kimeneti pinek száma**
  - + void onClick(Controller controller): **Komponensre** **kapcsolás**
  - # void onDraw(Graphics g): **Kirajzolási logika**

### 11.3.39. SequenceGeneratorView

- Felelősség

- Jelgenerátort kirajzoló osztály

- Ősosztályok Object → ComponentView → SequenceGeneratorView.

- Interfészek (nincs)

- Attribútumok

- SequenceGenerator sg **Becsomagolt** szekvenciagenerátor

- Metódusok

- + SequenceGeneratorView(SequenceGenerator sg): **Konstruktor**
  - # int getInputPinsCount(): **Bemeneti pinek száma**
  - # int getOutputPinsCount(): **Kimeneti pinek száma**
  - + void onClick(Controller controller): **Komponensre** **kapcsolás**
  - # void onDraw(Graphics g): **Kirajzolási logika**

### 11.3.40. SevenSegmentDisplayView

- Felelősség

- Hétszegmenses kijelzőt kirajzoló osztály.

- Ősosztályok Object → ComponentView → SevenSegmentDisplayView.

- Interfészek (nincs)

- Attribútumok

- SevenSegmentDisplay ssd **Becsomagolt** 7-szegmenses kijelző

- Metódusok

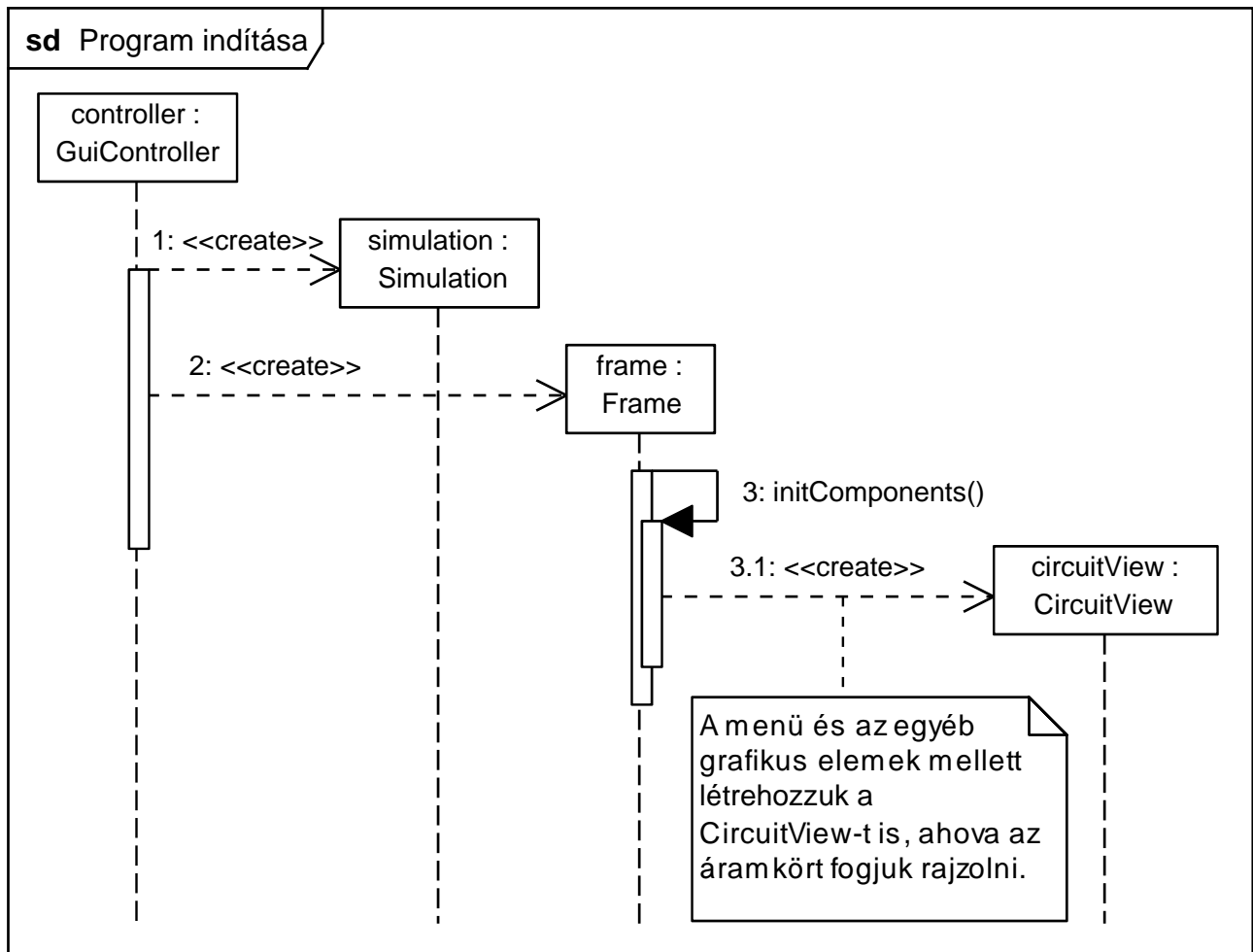
- + SevenSegmentDisplayView(SevenSegmentDisplay ssd): **Konstruktor**
  - # int getInputPinsCount(): **Bemeneti pinek száma**
  - # int getOutputPinsCount(): **Kimeneti pinek száma**
  - + void onClick(Controller controller): **Komponensre** **kapcsolás**
  - # void onDraw(Graphics g): **Kirajzolási logika**

## 11.3.41. ToggleView

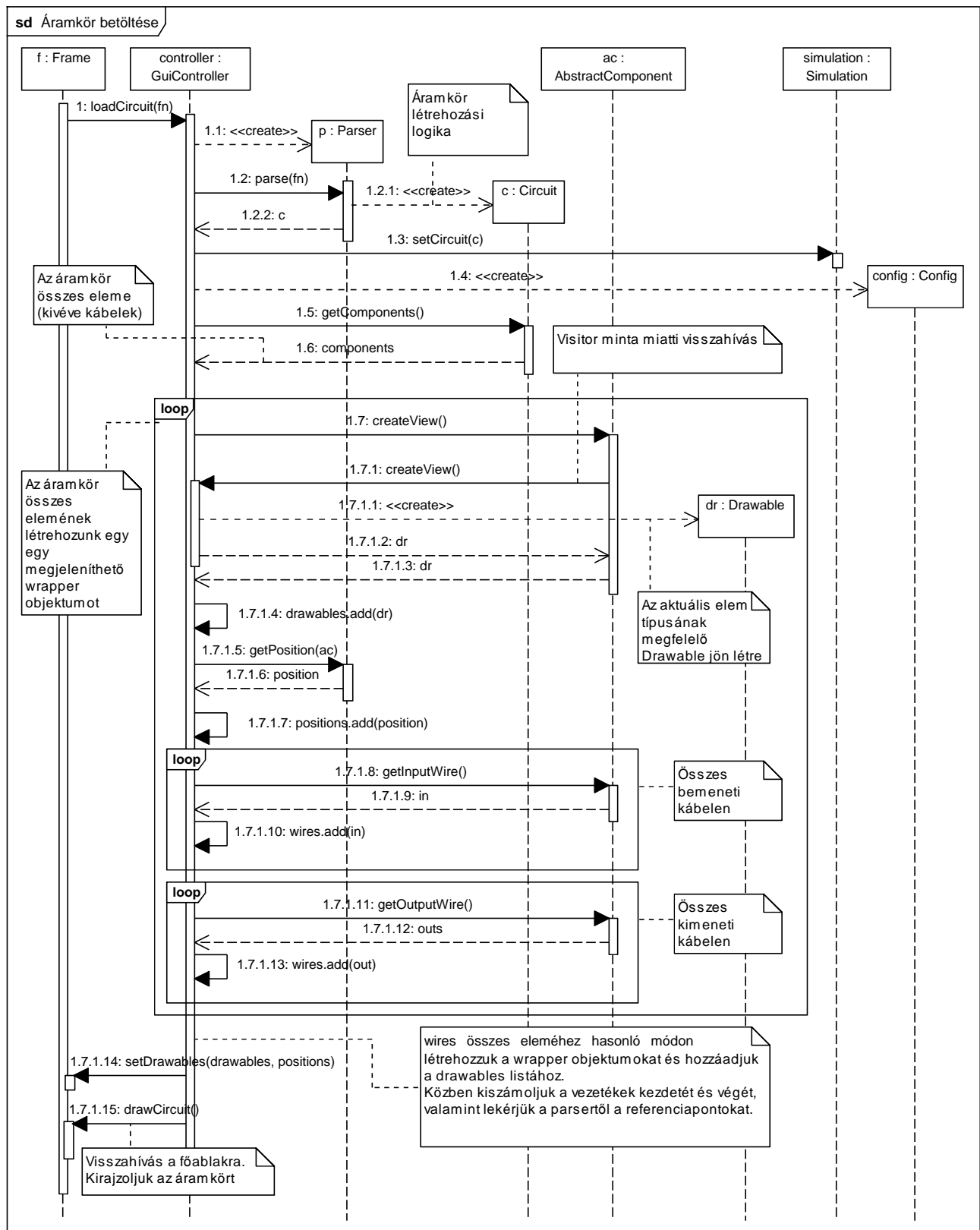
- Felelősség  
Kapcsolót kirajzoló osztály
- Ősosztályok `Object` → `ComponentView` → `ToggleView`.
- Interfészek (nincs)
- Attribútumok
  - `Toggle toggle` Becsomagolt kapcsoló
- Metódusok
  - + `ToggleView(Toggle toggle):` Konstruktor
  - # `int getInputPinsCount():` Bemeneti pinek száma
  - # `int getOutputPinsCount():` Kimeneti pinek száma
  - + `void onClick(Controller controller):` Komponensre kapcsolás
  - # `void onDraw(Graphics g):` Kirajzolási logika

## 11.3.42. VccView

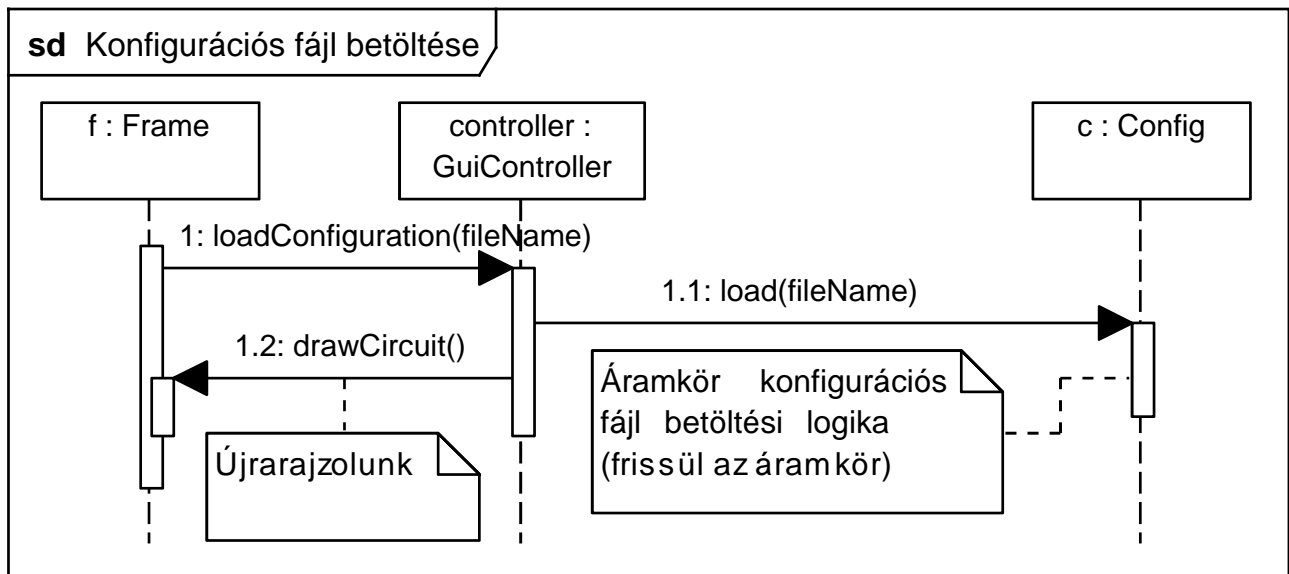
- Felelősség  
VCC-t kirajzoló osztály
- Ősosztályok `Object` → `ComponentView` → `VccView`.
- Interfészek (nincs)
- Attribútumok
- Metódusok
  - + `VccView():` Konstruktor
  - # `int getInputPinsCount():` Bemneti pinek száma
  - # `int getOutputPinsCount():` Kimeneti pinek száma
  - + `void onClick(Controller controller):` Komponensre kapcsolás
  - # `void onDraw(Graphics g):` Kirajzolási logika

**11.4. Kapcsolat az alkalmazói rendszerrel**

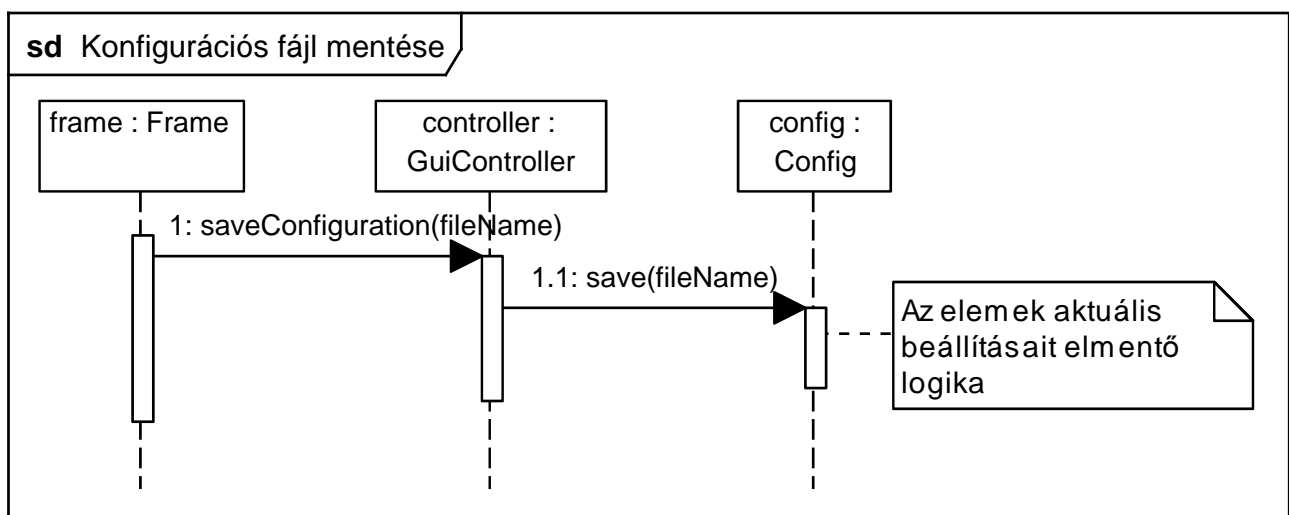
11.8. ábra. Program indítása



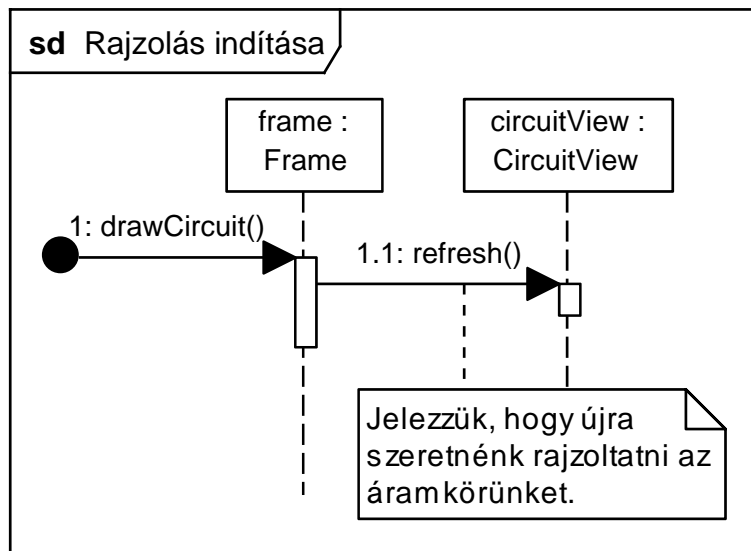
11.9. ábra. Áramkör betöltése



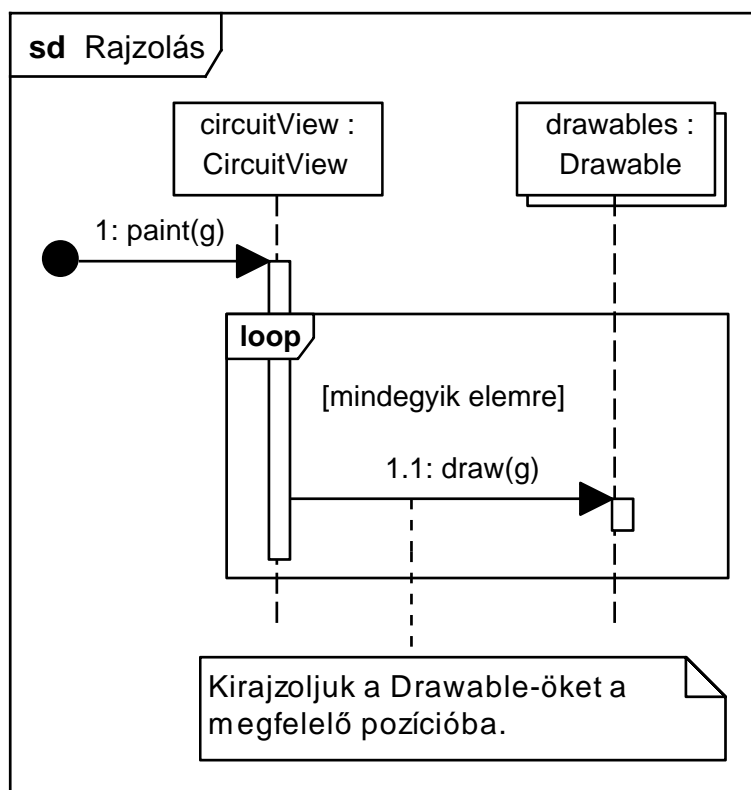
11.10. ábra. Konfigurációs fájl betöltése



11.11. ábra. Konfigurációs fájl mentése

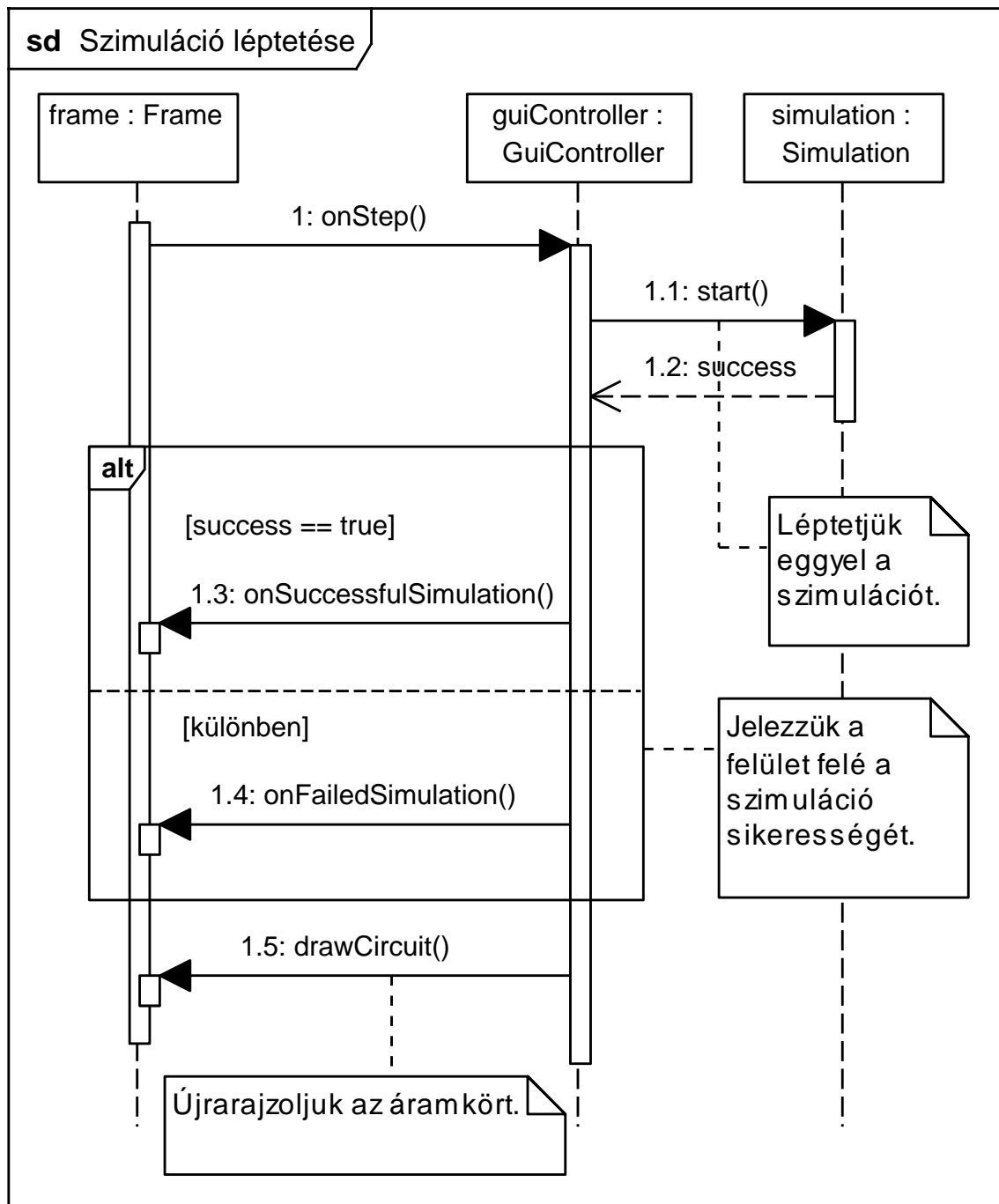


11.12. ábra. Rajzolás indítása

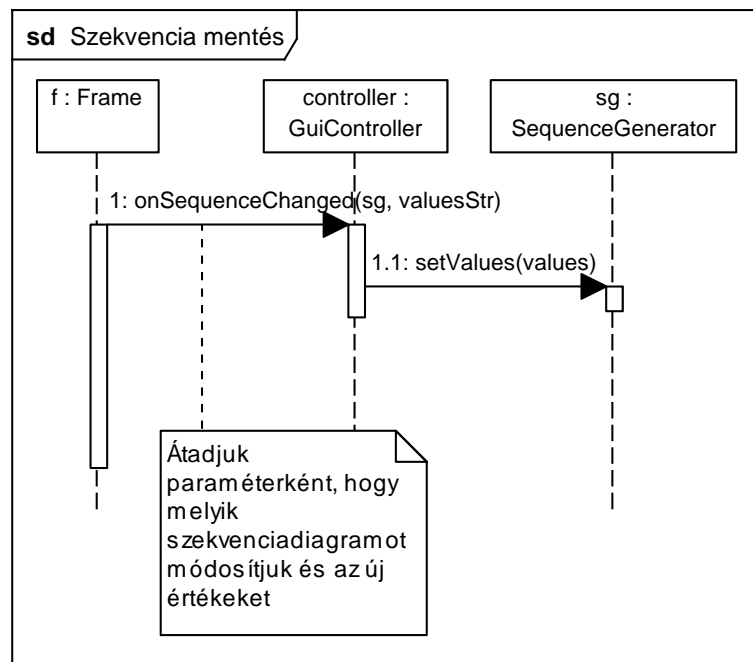


11.13. ábra. Rajzolás

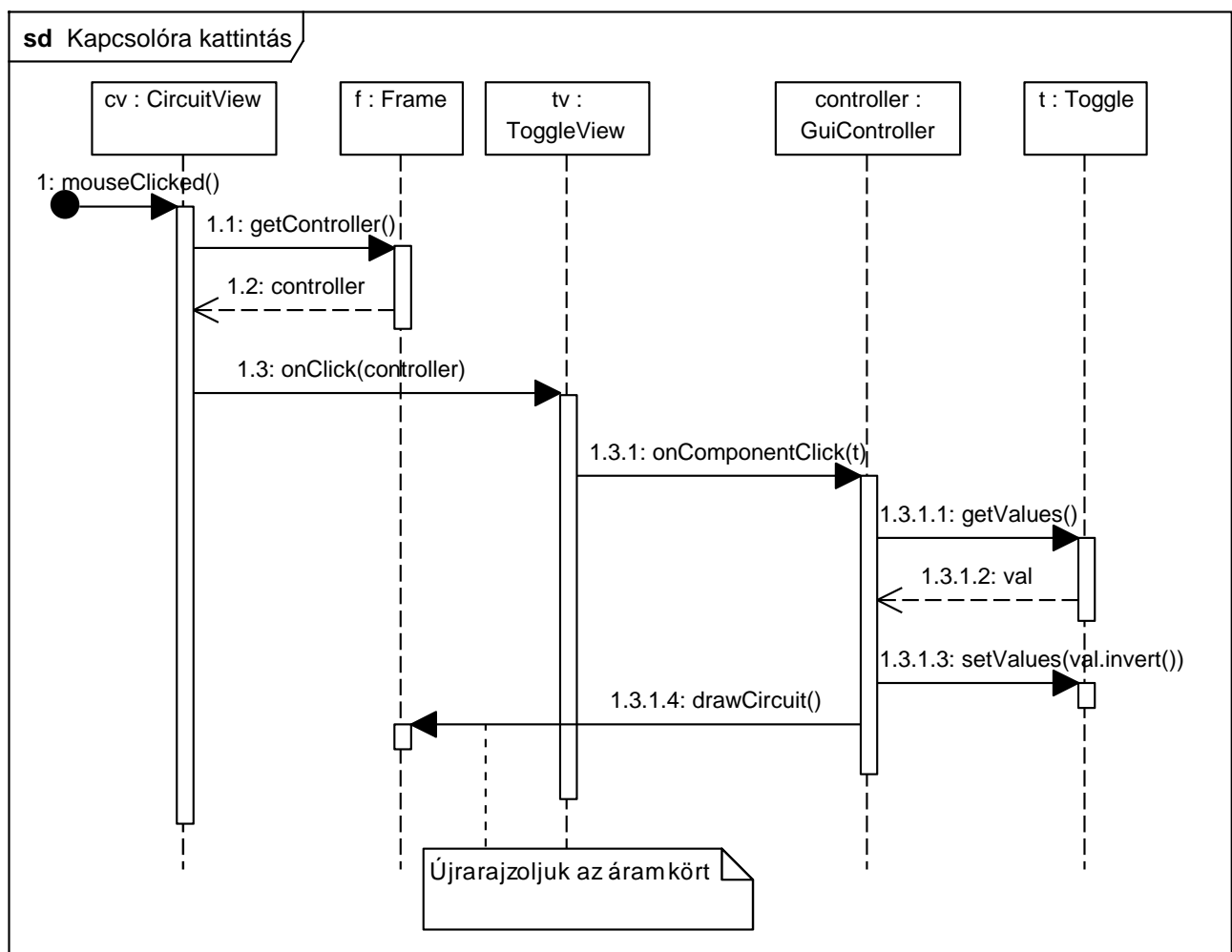




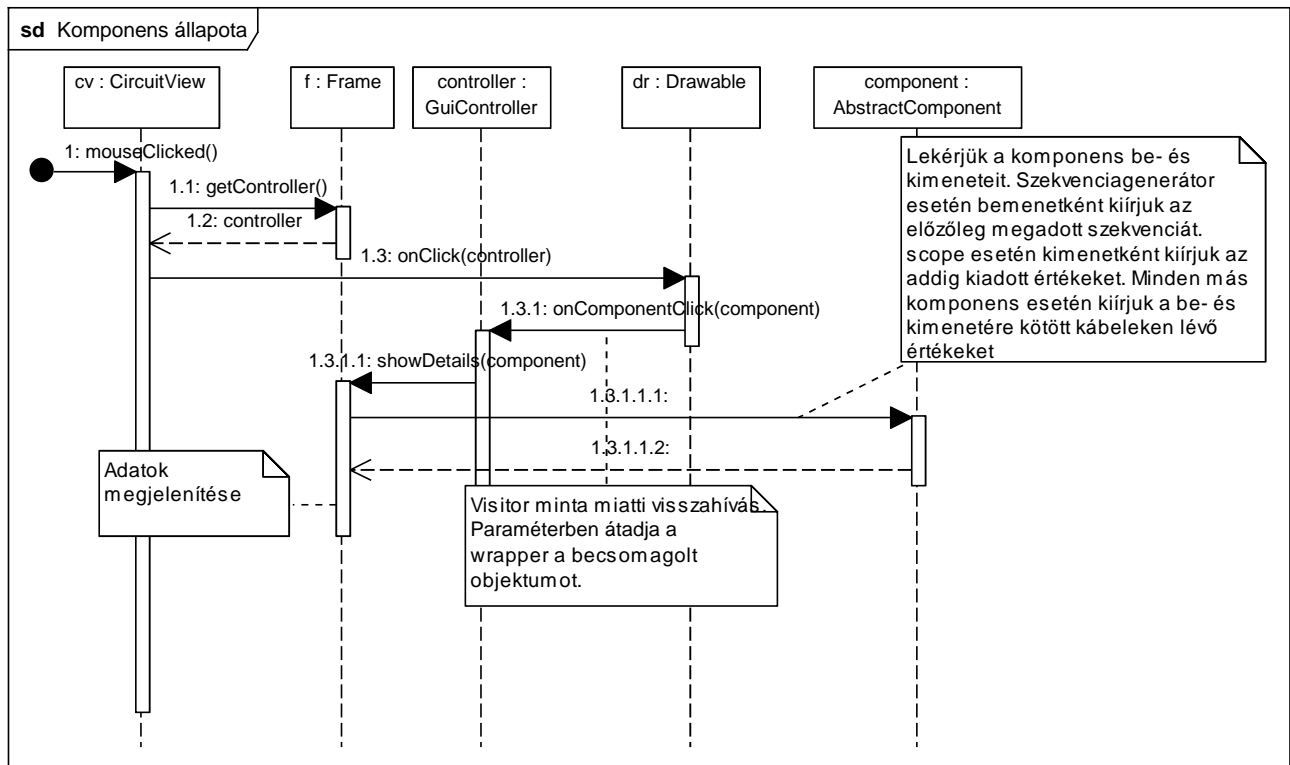
11.14. ábra. Szimuláció léptetése



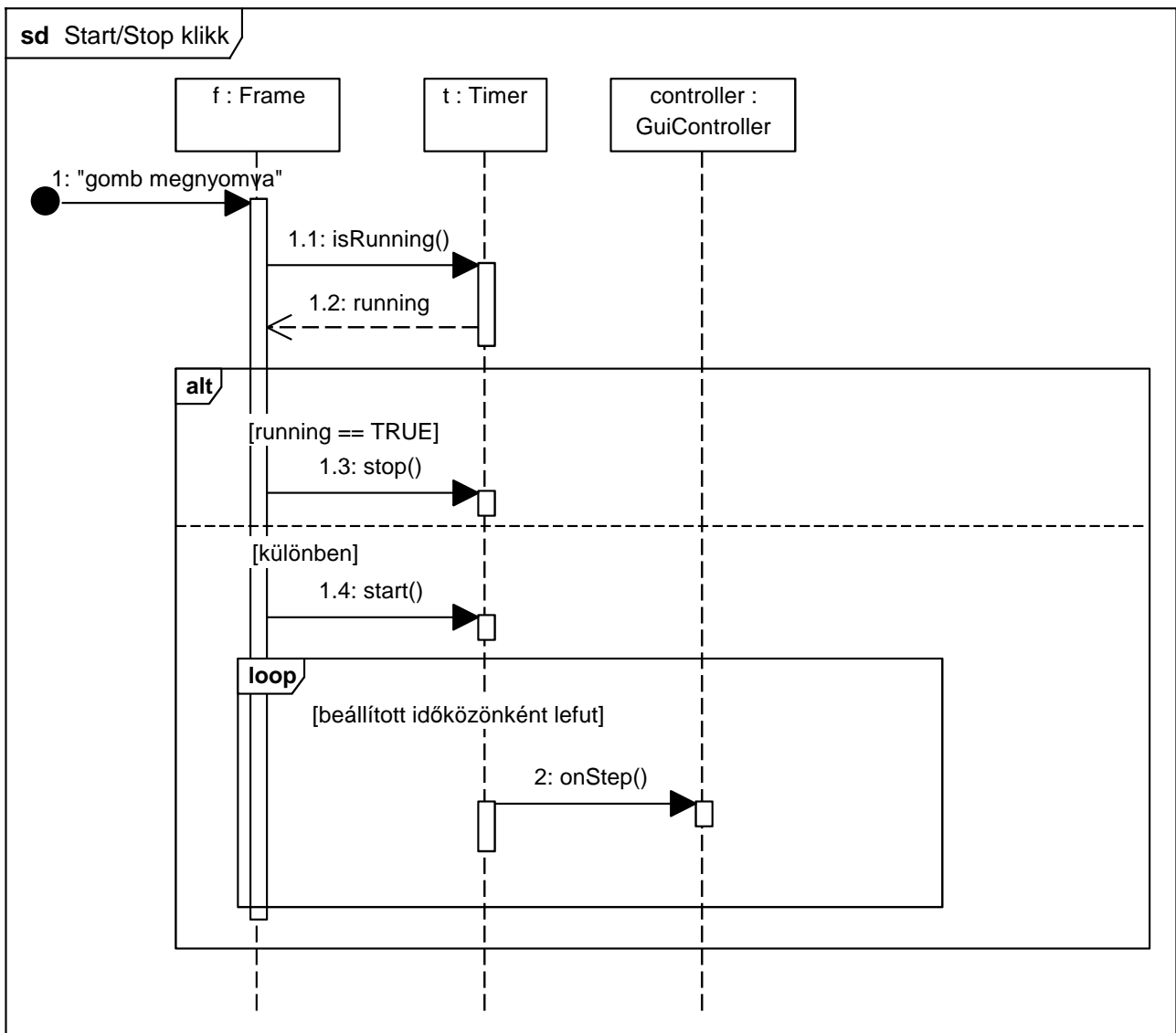
11.15. ábra. Szekvencia mentése



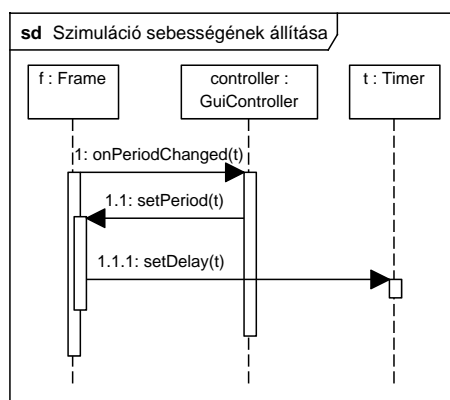
11.16. ábra. Kapcsolóra kattintás



11.17. ábra. Komponens állapotának kijelzése



11.18. ábra. Start/Stop klikk



11.19. ábra. Szimuláció sebességének állítása

## 11.5. Napló

<b>Kezdet</b>	<b>Időtartam</b>	<b>Résztevők</b>	<b>Leírás</b>
2011.04.20. 14:00	3 óra	<b>Dévényi A. Jákli G. Kriván B.</b>	Értekezlet. Megbeszéltük a grafikus felületet, osztályokat és a szekvenciákat. Döntés: szétszöttük a diagramokat
2011.04.23. 11:30	45 perc	<b>Kriván B.</b>	A grafikus interfész c. fejezet elkészítése.
2011.04.23. 14:00	3 óra	<b>Jákli G.</b>	Szekvenciadiagramok készítése. (11.6, 11.7, 11.8, 11.9, 11.13)
2011.04.23. 16:00	2 óra	<b>Kriván B.</b>	Szekvenciadiagramok készítése. (11.10, 11.11, 11.12, 11.15)
2011.04.24. 11:00	1,5 óra	<b>Jákli G.</b>	Szekvenciadiagramok készítése. (11.14, 11.16, 11.17)
2011.04.24. 12:00	2,5 óra	<b>Dévényi A.</b>	Osztálydiagram készítése, grafikus objektumok osztályleírása.
2011.04.24. 19:00	2 óra	<b>Kriván B.</b>	Dokumentáció véglegesítése, összerakása.

## 13. Grafikus felület specifikációja

### 13.1. Fordítási és futtatási útmutató

#### 13.1.1. Fájllista

Fájl neve	Méret	Keletkezés ideje	Tartalom
src/logsim/ ComponentViewCreator.java	4506 byte	2011.05.01. 14:33	Minden áramköri elemhez létrehoz egy megjeleníthető elemet
src/logsim/Config.java	4195 byte	2011.05.01. 14:33	A kapcsolók és szekvenciagenerátorok ki-mentéséért és betöltéséért felelős
src/logsim/Controller.java	1600 byte	2011.05.01. 14:33	A vezérlés interfészét tartalmazza
src/logsim/GuiController.java	12086 byte	2011.05.01. 14:33	A szimuláció működéséért felelős; felhasználói utasítások értelmezése
src/logsim/Parser.java	13730 byte	2011.05.01. 14:33	Az áramkörleíró fájl feldolgozását végzi
src/logsim/model/Circuit.java	1096 byte	2011.05.01. 14:33	Áramkört reprezentáló osztály
src/logsim/model/ Simulation.java	857 byte	2011.05.01. 14:33	Egy szimulációt reprezentáló osztály
src/logsim/model/Value.java	714 byte	2011.05.01. 14:33	Az áramkörben előforduló értékeket tartalmazó osztály
src/logsim/model/component/ AbstractComponent.java	4612 byte	2011.05.01. 14:33	Az alkatrészek absztrakt őssztálya
src/logsim/model/component/ Composite.java	15419 byte	2011.05.09. 10:35	A kompozit elem leírása
src/logsim/model/component/ DisplayComponent.java	644 byte	2011.05.01. 14:33	Megjelenítő típusú alkatrészek absztrakt őssztálya
src/logsim/model/component/ FlipFlop.java	1799 byte	2011.05.09. 10:35	Flipflop típusú alkatrészek absztrakt őssztálya
src/logsim/model/component/ Pin.java	612 byte	2011.05.01. 14:33	A kimeneteket és bemeneteket tároló osztály
src/logsim/model/component/ SourceComponent.java	1074 byte	2011.05.01. 14:33	Forrás típusú alkatrészek absztrakt őssztálya
src/logsim/model/component/ Wire.java	1109 byte	2011.05.01. 14:33	Vezetéket megvalósító osztály
src/logsim/model/component/ impl/AndGate.java	1124 byte	2011.05.01. 14:33	Az ÉS kapu alkatrészt megvalósító osztály
src/logsim/model/component/ impl/FlipFlopD.java	1178 byte	2011.05.09. 10:35	A D flipflop alkatrészt megvalósító osztály
src/logsim/model/component/ impl/FlipFlopJK.java	1800 byte	2011.05.09. 10:35	A JK flipflop alkatrészt megvalósító osztály
src/logsim/model/component/ impl/Gnd.java	774 byte	2011.05.01. 14:33	A permanens logikai nullát megvalósító osztály
src/logsim/model/component/ impl/Inverter.java	847 byte	2011.05.01. 14:33	Az inverter alkatrészt megvalósító osztály
src/logsim/model/component/ impl/Led.java	924 byte	2011.05.01. 14:33	A led megjelenítőt megvalósító osztály
src/logsim/model/component/ impl/Mpx.java	1431 byte	2011.05.01. 14:33	A multiplexer alkatrészt megvalósító osztály
src/logsim/model/component/ impl/Node.java	1221 byte	2011.05.01. 14:33	Csomópont alkatrészt megvalósító osztály

Fájl neve	Méret	Keletkezés ideje	Tartalom
src/logsim/model/component/impl/OrGate.java	1194 byte	2011.05.01. 14:33	a VAGY kapu alkatrészt megvalósító osztály
src/logsim/model/component/impl/Scope.java	1663 byte	2011.05.01. 14:33	Oszilloszkópot megvalósító osztály
src/logsim/model/component/impl/SequenceGenerator.java	2532 byte	2011.05.01. 14:33	A szekvenciagenerátor alkatrészt megvalósító osztály
src/logsim/model/component/impl/SevenSegmentDisplay.java	1176 byte	2011.05.01. 14:33	A 7 szegmenses kijelző alkatrészt megvalósító osztály
src/logsim/model/component/impl/Toggle.java	1719 byte	2011.05.01. 14:33	A kapcsolót megvalósító osztály
src/logsim/model/component/impl/Vcc.java	746 byte	2011.05.01. 14:33	A permanens logikai egyet megvalósító osztály
src/logsim/view/CircuitView.java	3551 byte	2011.05.01. 14:33	Áramkört kirajzoló panel.
src/logsim/view/DrawableView.java	555 byte	2011.05.01. 14:33	Áramköri panelre rajzolható objektum.
src/logsim/view/Frame.form	23241 byte	2011.05.01. 14:33	A főablak elemeinek elrendezését tartalmazó fájl.
src/logsim/view/Frame.java	28642 byte	2011.05.09. 11:34	A főablak elemeinek interakcióinak feldolgozásáért felelős osztály.
src/logsim/view/FrameView.java	1914 byte	2011.05.01. 14:33	Főablak interfésze.
src/logsim/view/component/ComponentView.java	2620 byte	2011.05.01. 14:33	Az elemek megjelenítéséért felelős osztályok absztrakt ősosztálya.
src/logsim/view/component/WireView.java	2050 byte	2011.05.01. 14:33	A kábelek kirajzolását végző osztály.
src/logsim/view/component/impl/AndGateView.java	1446 byte	2011.05.01. 14:33	Az ÉS kapu alkatrészt megjelenítő osztály
src/logsim/view/component/impl/CompositeView.java	1758 byte	2011.05.09. 10:48	A kompozit alkatrészt megjelenítő osztály
src/logsim/view/component/impl/FlipFlopDView.java	1257 byte	2011.05.01. 14:33	A D flipflop alkatrészt megjelenítő osztály
src/logsim/view/component/impl/FlipFlopJKView.java	1272 byte	2011.05.01. 14:33	A JK flipflop alkatrészt megjelenítő osztály
src/logsim/view/component/impl/GndView.java	1128 byte	2011.05.01. 14:33	A permanens logikai nullát megjelenítő osztály
src/logsim/view/component/impl/InverterView.java	1425 byte	2011.05.01. 14:33	Az inverter alkatrészt megjelenítő osztály
src/logsim/view/component/impl/LedView.java	1400 byte	2011.05.01. 14:33	A led megjelenítőt megjelenítő osztály
src/logsim/view/component/impl/MpxView.java	1187 byte	2011.05.01. 14:33	A multiplexer alkatrészt megjelenítő osztály
src/logsim/view/component/impl/NodeView.java	2013 byte	2011.05.09. 10:35	Csomópont alkatrészt megjelenítő osztály
src/logsim/view/component/impl/OrGateView.java	1284 byte	2011.05.01. 14:33	a VAGY kapu alkatrészt megjelenítő osztály
src/logsim/view/component/impl/ScopeView.java	1479 byte	2011.05.01. 14:33	Oszilloszkópot megjelenítő osztály
src/logsim/view/component/impl/SequenceGeneratorView.java	1603 byte	2011.05.01. 14:33	A szekvenciagenerátor alkatrészt megjelenítő osztály

Fájl neve	Méret	Keletkezés ideje	Tartalom
src/logsim/view/component/impl/SevenSegmentDisplayView.java	3749 byte	2011.05.01. 14:33	A 7 szegmenses kijelző alkatrészt megjelenítő osztály
src/logsim/view/component/impl/ToggleView.java	1549 byte	2011.05.01. 14:33	A kapcsolót megjelenítő osztály
src/logsim/view/component/impl/VccView.java	1104 byte	2011.05.01. 14:33	A permanens logikai egyet megjelenítő osztály
compile.bat	183 byte	2011.05.09. 10:48	Fordítást segítő .bat fájl
doc.bat	314 byte	2011.05.09. 10:35	Java dokumentációt generáló .bat fájl
run.bat	111 byte	2011.05.09. 10:56	Futtatást segítő .bat fájl
tesztek/test1.txt	78 byte	2011.04.17. 21:58	Az 1. teszteset áramköre
tesztek/test2.txt	221 byte	2011.04.17. 21:58	A 2. teszteset áramköre
tesztek/test3.txt	83 byte	2011.04.17. 21:58	A 3. teszteset áramköre
tesztek/test4.txt	96 byte	2011.04.17. 21:58	A 4. teszteset áramköre
tesztek/test5.txt	89 byte	2011.04.17. 21:58	Az 5. teszteset áramköre
tesztek/test6.txt	263 byte	2011.04.17. 21:58	A 6. teszteset áramköre
tesztek/test7.txt	161 byte	2011.04.17. 21:58	A 7. teszteset áramköre

### 13.1.2. Fordítás

A hibamentes és minél inkább gördülékeny fordítás érdekében létrehoztunk egy `compile.bat` nevezetű batch fájlt, mely a projekt főkönyvtárban található. Projekt főkönyvtára az, amelyik a batch fájlokat és a "src" nevezetű mappát tartalmazza, melyben a program forráskódja található. Szükség esetén kézzel kell módosítani a batch fájlt

```
set C="C:\Program Files\Java\jdk1.6.0_23\bin\"
```

sorát, attól függően, hogy a gépen éppen melyik Java JDK verzió található és az hová van telepítve!

A `compile.bat` fájl az alábbi parancsokat hajtja végre:

```
@echo off
set C="C:\Program_Files\Java\jdk1.6.0_24\bin\"
mkdir build
cd src
%C%\javac -d ../build logsim\GuiController.java
cd..
if not errorlevel 1 echo Forditas sikeres
pause
```

Ha hibamentes volt a fordítás, a "Fordítás sikeres" kimenettel értesíti a felhasználót.

A fordítás sikeressége után, lehetőség van a dokumentáció legenerálására is. Ehhez felhasználható a főkönyvtárban található `doc.bat` batch fájl. Szükség esetén kézzel kell módosítani a batch file

```
set C="C:\Program Files\Java\jdk1.6.0_23\bin\"
```

sorát, attól függően, hogy a gépen éppen melyik Java JDK verzió található és az hová van telepítve!

A batch fájl az alábbi parancsokat hajtja végre:

```
@echo off
set C="C:\Program_Files\Java\jdk1.6.0_24\bin\"
cd src
%C%\javadoc logsim logsim logsim.model logsim.model.component ^
logsim.model.component.impl logsim.view logsim.view.component ^
logsim.view.component.impl -d ../documents
cd..
if not errorlevel 1 echo Dokumentum generalas sikeres volt.
pause
```



Ha a dokumentum generálás sikeres volt, akkor a documents nevezetű mappában megtalálhatóak a kívánt dokumentumok.

### 13.1.3. Futtatás

A futtatás megkönnyítése érdekében elkészítettük a `run.bat` batch fájlt. Szükség esetén kézzel kell módosítani a `run.bat` batch fájlt

```
set C="C:\Program Files\Java\jdk1.6.0_23\bin\"
```

sorát, attól függően, hogy a gépen éppen melyik Java JDK verzió található és az hová van telepítve!

A `run.bat` fájl az alábbi parancsokat hajtja végre:

```
@echo off
set C="C:\Program_Files\Java\jdk1.6.0_24\bin\"
cd build
%C%\java logsim.GuiController
cd. .
PAUSE
```

A "build" könyvtárból elindítja az előzőleg lefordított programot.

## 13.2. Értékelés

Tag	Munka százalékban	Aláírás
Apagyi Gábor	13 %	
Dévényi Attila	20 %	
Jákli Gábor	21 %	
Kriván Bálint	34 %	
Péter Tamás Pál	13 %	

## 13.3. Napló

Kezdet	Időtartam	Résztevők	Leírás
2011.05.03. 18:00	3 óra	<b>Jákli G.</b>	Formok, panelek és dialógus ablakok implementálása
2011.05.04. 12:00	3 óra	<b>Dévényi A.</b>	Komponensek megjelenítésének implementálása
2011.05.04. 12:00	4 óra	<b>Kriván B.</b>	Parser felkészítése a pozíciók beolvasására és a komponensek összeköttetésekének ábrázolása.
2011.05.07. 10:30	2 óra	<b>Péter T.</b>	GUI részeinek véglegesítése, finomítása.
2011.05.09. 10:00	1 óra	<b>Apagyi G.</b>	Dokumentáció megírása

## 14. Összefoglalás

### 14.1. Projekt összegzés

Tag	Munkaidő (óra)
Apagyí Gábor	23
Dévényi Attila	35
Jákli Gábor	37
Kriván Bálint	61
Péter Tamás Pál	22.5
<b>Összesen:</b>	<b>178.5</b>

Fázis	Forrásor
Szkeleton	1735
Protó	2571
Grafikus	5023

- Mit tanultak a projektből konkrétan és általában?

Végre gyakorlatban is láthattuk azt, amit az előző félévben a Szoftvertechnológia tárgyból tanultunk. Kipróbálhattuk magunkat egy viszonylag hosszabb projektmunkában, megismerhettük egymás erősségeit és gyengeségeit.

- Mi volt a legnehezebb és a legkönnyebb?

Legnehezebb a közös időpontok megszervezése, valamint a közös tervezések közben felmerült problémák megoldásainak közös elfogadása. A legkönnyebb az implementálás volt az elkészült tervek alapján, mely szinte csak gépelésről szólt.

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

Az első részben kicsit tévútra indultunk és ott fölösleges órák mentek el a rossz megoldásra, de ezt a negyedik beadásra sikerült korrigálni. Összességben korrektnek érezzük a kapott pontszámokat.

- Ha nem, akkor hol okozott ez nehézséget?

Ahogy már említettük az első részben az analízis modell rosszra sikeredett, ezt sok idő volt korrigálni, és maga a rossz megoldás is sok időt elvett.

- Milyen változtatási javaslatuk van?

Több évfolyamtárstól hallottuk, hogy labvezérek által számonkért beadandó dokumentumok minőségének a szórása igen nagy. Mi úgy érezzük, hogy beletettünk elég sok munkát és ennek megfelelően jogosnak érezzük az elért eredményeket, pontszámokat, azonban néhány másik csoport ennek töredékéért megkapja ugyanazt, vagy akár többet, annak ellenére, hogy nem biztos, hogy jobb minőségű munkát

adtak ki a kezükből. Ha ezen egy kicsit lehetne javítani, homogenizálni a labvezérek egyéni elvárásait, akkor talán kicsit jobb lenne a több munkát belefektető csapatok hangulata. Továbbá ha megnézzük az eltöltött órák számát, akkor kicsit kevésnek érezzük a tárgyért járó 2 kreditet, 3-nak esetleg 4-nek jobban örülnénk.

- Milyen feladatot ajánlanának a projektre?

Mindenképpen valami ehhez hasonló, tehát a projekt végére egy „használható” termék-szerűség készüljön el, amit akár valódi célokra is fel lehet használni (gondolok itt arra, hogy például a most elkészült programot a Digitális technika I-II. című tárgy keretében akár még használni is lehetne).