3. Analízis modell kidolgozása 1

54 – *Override*

Konzulens:
dr. László Zoltán

Csapattagok:

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

2011. március 2.

Tartalomjegyzék

Ábrák jegyzéke

3. Analízis modell kidolgozása 1

3.1. Objektum katalógus

3.1.1. Parser

Áramkör értelmező objektum, feladata, hogy a paraméterként átadott, illetve fájlban elhelyezett komponenseket értelmezze, a kapcsolatokat feltérképezze, elvégezze az összeköttetéseket, és ezáltal felépítse az áramkört.

3.1.2. Simulation

Szimuláció objektum. A szimulációért felelős. Elindítja a jelgenerátor léptetőt, s utasítja az áramkört több kiértékelési ciklus lefuttatásához, amíg az áramkörben van változás. Ha a változás megadott lépésen belül nem áll meg, tájékoztatja a felhasználót, hogy nincs stacionárius állapot. Amikor leállítódik, a jelgenerátor-léptetőt is leállítja.

3.1.3. Circuit

Az áramkör objektum. Ezen objektum feladata a jelgenerátor léptető kérésére a jelgenerátorok léptetése, az áramkörben található komponensek utasítása arra, hogy töröljék a "már kiértékelve" flaget egy adott kiértékelési ciklus előtt, hogy ezáltal a ciklusban minden kimenet értéke frissülhessen. Továbbá feladata a kiértékelés elindítása az összes kijelzőre, mert a rendszer kiértékelése a kijelzők kiértékelésével kezdődik.

3.1.4. SequenceGeneratorStepper

Jelgenerátor léptető objektum. Feladata, hogy az áramkört utasítsa a jelgenerátorok léptesésére.

3.1.5. SequenceGenerator

Jelgenerátor, az áramkört felépítő egyik alapelem, kiértékelési kezdeményezés hatására az előre betáplált jelsorozatot soron következő elemét állítja be aktuális értékként, így azon komponensek melyek bemenetére a Jelgenerátor van kötve, elérik aktuális értékét. Bemenete nem komponens jellegű így nem kezel más komponenseket. Mikor az áramkör kéri tőle, hogy lépjen, akkor a bitsorozat következő elemére lép.

3.1.6. AndGate

ÉS kapu, az áramkör egyik alapeleme. Bemeneteire kötött komponensek kiértékelését kezdeményezi, s a kapott értékek logikai ÉS kapcsolatát valósítja meg, amit a kimenetén kiad.

3.1.7. OrGate

VAGY kapu, az áramkör egyik alapeleme. Bemeneteire kötött komponensek kiértékelését kezdeményezi, s a kapott értékek logikai VAGY kapcsolatát valósítja meg, amit a kimenetén kiad.

3.1.8. Inverter

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

3.1.9. Gnd

Föld, az áramkört felépítő egyik elem, állandó értéke logikai hamis. Bemenete nem létezik, így nem kezdeményez további kiértékeléseket.

3.1.10. Vcc

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

3.1.11. Led

Egy kijelző az áramkör alapeleme, bemenetére kötött komponens kiértékelését kezdeményezi, és ezáltal az aktuális értékét egy a felhasználó számára érzékelhető módon kijelzi.

3.1.12. Toggle

Kapcsoló, az áramkört felépítő elem, felhasználói interakciót követően, az aktuális értékét lehet állítani. Komponens bemenete nincs, így nem kezel további komponenseket.

3.1.13. Value

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

3.1.14. 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.

3.1.15. FlipFlopJK

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

3.1.16. Mpx

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

3.1.17. SevenSegmentDisplay

Hétszegmenses kijelző objektuma. Minden bemenete egy-egy szegmensért felelős, melyek 8-as alakban helyezkednek el.

3.1.18. SourceWriter

Ez az objektum végzi el a konfigurálható elemek beállításainak fájlba írását a felhasználó kérésére.

3.1.19. SourceReader

Ez az objektum végzi el a konfigurálható elemek beállításainak fájlból beolvasását a felhasználó kérésére.

3.2. Osztályok leírása

3.2.1. Circuit

- Felelősség
  Feladata a jelgenerátor léptető kérésére a jelgenerátorok léptetése, a feldolgozó által létrehozott komponensek felvétele az áramkörbe, illetve ezek utasítása arra, hogy töröljék a "már kiértékelve" flaget egy adott kiértékelési ciklus előtt, hogy ezáltal a ciklusban minden kimenet értéke frissülhessen. Továbbá feladata a kiértékelés elindítása az összes kijelzőre, mert a rendszer kiértékelése a kijelzők kiértékelésével kezdődik.
- Ősosztályok: Object → Circuit.
- Interfészek: (nincs)
- Attribútumok
  - `private HashMap componentMap` Komponenseket tartalmazó HashMap
  - `private List displays` Megjelenítő típusú komponensek
  - `private Simulation simulation` Áramkört éppen szimuláló objektum
  - `private List sources` Jelforrás típusú komponensek
  - `private boolean stable` Áramkör stabilitása
- Metódusok
  - `public AbstractComponent addComponent(AbstractComponent component)`: Komponens hozzáadása az áramkörhöz.
  - `public void doEvaluationCycle()`: Egy kiértékelési ciklus lefuttatása. Az áramkörtől ezután lekérdezhető, hogy stabil (nem változott semelyik komponens kimenete az utolsó futtatás óta) vagy instabil állapotban van-e.
  - `public AbstractComponent getComponentByName(String name)`: Lekérünk egy komponenst az áramkörtől a neve alapján.
  - `public List getDisplays()`: Megjelenítő típusú komponeseket adja vissza.
  - `public List getSources()`: Jelforrás típusú komponenseket adja vissza.
  - `public boolean isStable()`: Áramkör stacionárius állapotának lekérdezése.
  - `public int loadSources(String fileName)`: Fájlból betölti a jelforrások állapotát.
  - `public int saveSources(String fileName)`: Elmenti fájlba a jelforrások állapotát.
  - `public void setSimulation(Simulation simulation)`: Szimuláció beállítása. Ez a szimuláció létrejöttekor hívódik meg.
  - `public void setStable(boolean stable)`: Áramkör stabilitásának beállítása.
  - `public void simulationShouldBeWorking()`: Jelzi a szimuláció felé, hogy új ciklust kell indítani. Ezt egy jelforrás beállítása után hívjuk meg.
  - `public void stepGenerators()`: Jelgenerátorok a szimuláció szemszögéből nézve, egyszerre történő léptetése.

3.2.2. SequenceGeneratorStepper

- Felelősség
  Jelgenerátor-léptető szál. Feladata, hogy az áramkört utasítsa a jelgenerátorok lépteésére a felhasználó által konfigurálható időközönként.
- Ősosztályok: Object → Thread → SequenceGeneratorStepper.
- Interfészek: (nincs)
- Attribútumok
  - `private long pause` Szünet két léptetés között – felhasználó által konfigurálható
  - `private boolean shouldRun` A futási ciklus változója; ha ez hamis lesz, akkor leáll a szál
  - `private Simulation simulation` A szimuláció, aki számára lépteti a jelgenerátorokat
- Metódusok
  - `public void run()`: Megadott időközönként az áramkörön meghívjuk a stepGenerators() metódust.
  - `public void setPause(long pause)`: Két léptetés között eltelt idő állítása (léptetési sebesség)

3.2.3. Simulation

- Felelősség
  Egy szimulációt reprezentáló objektum. Futásakor elindítja a jelgenerátor léptetőt, s utasítja az áramkört több kiértékelési ciklus lefuttatásához, amíg az áramkörben van változás. Ha a változás megadott lépésen belül nem áll meg, tájékoztatja a felhasználót, hogy nincs stacionárius állapot. Amikor leállítódik, a jelgenerátor-léptetőt is leállítja. A szál természetéből adódóan többet nem indítható el, új szimulációhoz új példányt kell létrehozni.
- Ősosztályok: Object → Thread → Simulation.
- Interfészek: (nincs)
- Attribútumok
  - `private Circuit circuit` Szimulált áramkör
  - `private int counter` ciklusszámláló, amely ha eléri a *cycleLimit*-et, akkor leáll a szimuláció és jelezzük a felhasználónak.
  - `private State currentState` Szimuláció jelenlegi állapota
  - `private static final int cycleLimit` cikluslimit
  - `private final Object lock` segéd lock objektum
  - `private SequenceGeneratorStepper seqGenStepper` Jelgenerátor-léptető, amit elindítunk, ha elindul a szimuláció.
  - `private boolean shouldRun` A futási ciklus változója; ha ez hamis lesz, akkor leáll a szál
  - `private final Object synchObj` segéd sync objektum, ami a szál altatásához kell.
- Metódusok
  - `public Circuit getCircuit()`: Szimulált áramkör lekérdezése
  - `public Object getLock()`: Egy lock objektum, a szálak egymást kizáráshoz kell.

- `public void run()`: A szál futása közben történő dolgokat valósítjuk meg. Lásd **??** és **??** ábra.
- `public void setState(State state)`: Állapot beállítása.

3.2.4. Simulation.State

- Felelősség
  Szimuláció állapotait írja le
- Ősosztályok: Object → Enum → Simulation.State.
- Interfészek: (nincs)
- Attribútumok
  - `public static final State PAUSED` Szimuláció szüneteltetve van, a következő jelforrás változásig.
  - `public static final State STOPPED` Szimuláció leállt, ahhoz, hogy bármi történjen az áramkörre újra kell indítani.
  - `public static final State WORKING` Szimuláció éppen dolgozik, egy konkrét jelforráskombinációt alkalmazva dolgoztatja az áramkört
- Metódusok
  - `public static State valueOf(String name):`
  - `public static State[] values():`

3.2.5. Value

- Felelősség
  Az áramkörben előfordulható értéket reprezentál.
- Ősosztályok: Object → Enum → Value.
- Interfészek: (nincs)
- Attribútumok
  - `public static final Value FALSE`
  - `public static final Value TRUE`
- Metódusok
  - `public Value invert()`: Invertálja az adott értéket. Ennek addig van értelme, amíg 2 féle állapot fordulhat elő a rendszerben.
  - `public static Value valueOf(String name):`
  - `public static Value[] values():`

3.2.6. 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 (összekötés, bemenetek kiértékelése stb.)

- Ősosztályok: Object → AbstractComponent.
- Interfészek: Component.
- Attribútumok
  - `protected boolean alreadyEvaluated` "Kiértékelt" flag, ha ez be van billenve, akkor nem számolunk újra, csak visszaadjuk az előzőleg kiszámolt értéket.
  - `protected Circuit circuit` Őt tartalmazó áramkör
  - `protected Value[] currentValue` Jelenlegi (számolás közben) érték, ezt csak rövid ideig tároljuk, ahhoz kell, hogy tudjuk változik-e a lastValue-hoz képest.
  - `protected int[] indices` Itt tároljuk, hogy melyik bemenetre, az adott komponens melyik kimenetét kötöttük.
  - `protected AbstractComponent[] inputs` Az adott bemenetekre kötött komponensek.
  - `protected Value[] lastValue` Kimenetek tényleges értékei, számolás után ide rögtön visszaírjuk. Ez kérdezhető le a felhasználó által.
  - `protected String name` Komponens neve (változó neve, ahogy a leíróban azonosítjuk)
- Metódusok
  - `public void clearEvaluatedFlag()`: Töröljük a komponens "kiértékelt" flagjét.
  - `public Value evaluate()`: Lekérjük a 0. kimenetén lévő értéket.
  - `public Value evaluate(int outputPin)`: Komponens kimeneteinek kiértékelése (ha még nem volt) és a megadott kimeneti lábon lévő érték visszaadása.
  - `public String getName()`: Név lekérdezése
  - `public Value getValue()`: 0-ás kimeneti lábon lévő értéke lekérdezése.
  - `public Value getValue(int idx)`: Adott kimeneti lábon lévő értéke lekérdezése.
  - `public void setCircuit(Circuit parent)`: Szülő beállítása
  - `public void setInput(int inputSlot, AbstractComponent component)`: Shortcut a másik setInput()-hoz, outputPin = 0-val.
  - `public void setInput(int inputPin, AbstractComponent component, int outputPin)`: Beállítunk egy bemenetet.
  - `public void setInputPinsCount(int inputPinsCount)`: Bemenetek számának beállítása
  - `public void setName(String name)`: Név beállítása (változó, amivel azonosítjuk)

3.2.7. Component

Interfész.

- Felelősség
  Komponens interfész, ebből származik az IsDisplay és az IsSource interfész. Legalapvetőbb dolgokat írja le (minden komponensnek van neve és értékei).
- Ősosztályok: (nincs)
- Interfészek: (nincs)
- Metódusok

  - `public String getName()`: Név lekérdezése.
  - `public Value getValue()`: Értéke lekérdezése a 0. kimeneten.
  - `public Value getValue(int idx)`: Érték lekérdezése az adott kimeneten.
  - `public void setName(String name)`: Név beállítása.

3.2.8. FlipFlop

Absztrakt osztály.

- Felelősség
  Flipflopok ősosztálya, minden flipflop 0. bemenete az órajel!
- Ősosztályok: Object → AbstractComponent → FlipFlop.
- Interfészek: (nincs)
- Attribútumok
  - `private boolean active` Ebben tároljuk, hogy a FF számolhat-e vagy sem. (felfutó él)
  - `protected static final int CLK` Fixen a 0. bemenet az órajel
- Metódusok
  - `public boolean isActive()`: Számolhat-e az FF? Ezt hívja meg az FF-ek onEvaluation() metódusa, mielőtt bármit is csinálnának.
  - `public void setActive(boolean active)`: Felfutó élnél a SequenceGenerator-nak meg kell hívni ezt a hozzá kötött FlipFlopokra, egyéb esetben törölnie az active flaget. Így tudja az FF, hogy mikor kell ténylegesen számolnia.
  - `public void setInput(int inputPin, AbstractComponent component, int outputPin)`: Ősosztály implementációjának meghívása, illetve ha egy SequenceGeneratort kötünk éppen a CLK bemenetre, akkor az addFlipFlop() metódus meghívása rajta.

3.2.9. IsDisplay

Interfész.

- Felelősség
  Megjelenítő típusú komponenst reprezentál. Ezt kell implementálnia a megjelenítőknek.
- Ősosztályok: (nincs).
- Interfészek: Component.
- Metódusok
  - (nincs)

3.2.10. IsSource

Interfész.

- Felelősség
  Jelforrás típusú komponenst reprezentál. Ezt kell implementálnia a jelforrásoknak.
- Ősosztályok: (nincs).

- Interfészek: Component.
- Metódusok
  - `public Value[] getValues()`: Lekérhetjük a jelforrás értékeit, hogy el tudjuk menteni.
  - `public void setValues(Value[] values)`: Beállítjuk a jelforrás értékét. Kapcsoló esetén csak 1 elemű tömb adható paraméterként!

3.2.11. 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
  - (nincs)

3.2.12. FlipFlopD

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

3.2.13. 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
  - `private static final int J`: J bemenet lábának a száma
  - `private static final int K`: K bemenet lábának a száma
- Metódusok
  - (nincs)

3.2.14. Gnd

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

3.2.15. Inverter

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

3.2.16. Led

- Felelősség
  Egy LED-et reprezentál, mely világít, ha bemenetén igaz érték van. 3 féle színe lehet, ezeket a Color enumeráció határozza meg.
- Ősosztályok: Object → AbstractComponent → Led.
- Interfészek: IsDisplay.
- Attribútumok
  - `private Color color`: Led jelenlegi színe
- Metódusok
  - `public Color getColor()`: Lekérdezzük a LED színét
  - `public void setColor(Color color)`: Beállítjuk a LED színét

3.2.17. Led.Color

- Felelősség
  LED-színt reprezentáló enum.
- Ősosztályok: Object → Enum → Led.Color.
- Interfészek: (nincs)
- Attribútumok
  - `public static final Color BLUE`
  - `public static final Color RED`
  - `public static final Color YELLOW`
- Metódusok
  - `public static Color valueOf(String name):`
  - `public static Color[] values():`

3.2.18. 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
  - `private static final int DATA0`
  - `private static final int DATA1`
  - `private static final int DATA2`
  - `private static final int DATA3`
  - `private static final int SEL0`
  - `private static final int SEL1`
- Metódusok
  - (nincs)

3.2.19. OrGate

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

  - (nincs)

- Metódusok
  - (nincs)

3.2.20. SequenceGenerator

- Felelősség
  Jelgenerátort reprezentál, amely a beállított bitsorozatot adja ki. A SequenceGeneratorStepper feladata, hogy a step() metódust meghívja ezen osztály példányain. Azokat a FF-eket vezérli, melyek CLK bemenetére ez a komponens van kötve, vagyis ha éppen felfutó él jön, akkor ezeket engedélyezi különben nem.
- Ősosztályok: Object → AbstractComponent → SequenceGenerator.
- Interfészek: IsSource.
- Attribútumok
  - `private List ffList`: Azon FF-ek listája, melyekre ez a jelgenerátor van bekötve a CLK bemenetre.
  - `private int idx`: Bitsorozat egy indexe, ez határozza meg, hogy éppen melyik értéket adja ki.
  - `private Value[] sequence`: Tárolt bitsorozat
- Metódusok
  - `public void addFlipFlop(FlipFlop ff)`: A flipflop-ot feliratkoztatjuk a jelgenerátorhoz, így ha felfutó él lesz, akkor tudunk neki jelezni.
  - `public Value[] getValues()`: Jelgenerátor bitsorozatának lekérdezése
  - `public void setValues(Value[] values)`: Jelgenerátor bitsorozatának beállítása
  - `public void step()`: A jelgenerátor lép, a bitsorozat következő elemére ugrik. A következő léptetésig ez kerül kiadásra a kimeneteken.

3.2.21. 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 → SevenSegmentDisplay.
- Interfészek: IsDisplay.
- Attribútumok
  - (nincs)
- Metódusok
  - (nincs)

3.2.22. Toggle

- Felelősség
  Kapcsoló jelforrás, melyet a felhasználó szimuláció közben kapcsolgathat.
- Ősosztályok: Object → AbstractComponent → Toggle.
- Interfészek: IsSource.
- Attribútumok
  - (nincs)
- Metódusok
  - `public Value[] getValues()`: Lekérjük a kapcsoló értékét (1 elemű tömb)
  - `public void setValues(Value[] values)`: Kapcsoló állapotának változtatása, csak 1 elemű tömböt kaphat paraméterül.
  - `public void toggle()`: Kapcsoló állapotát megváltoztathatjuk. Kényelmesebb, mint a setValues() hívása

3.2.23. Vcc

- Felelősség
  A tápfeszültés komponens, ami konstans igaz értéket ad. Nincs bemenete.
- Ősosztályok: Object → AbstractComponent → Vcc.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - (nincs)

3.2.24. 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. Fontos, hogy egy ilyen objektum csak egyszer használható, új áramkörhöz, újat kell létrehozni.
- Ősosztályok: Object → Parser.
- Interfészek: (nincs)
- Attribútumok
  - `private static final HashMap availableComponents`: Feldolgozó által ismert komponensek listája.
  - `private Circuit circuit`: A leíróból létrehozott áramkör.
  - `private static Pattern componentPattern`: Regex minta egy leíró-sor feldolgozásához.

  - `private int constComps`: Egy számláló, hogy a vcc és gnd komponenseknek eltérő változónevet tudjunk adni.
  - `private static Pattern inputPattern`: Regex minta egy komponens bemeneteinek a feldolgozásához.
  - `private HashMap inputs`: Minden komponens-névhez eltároljuk a bemeneteket, későbbi feldolgozás miatt.

- Metódusok
  - `public Circuit parse(File file)`: Létrehoz egy áramkört a megadott fájlból
  - `public Circuit parse(String[] content)`: Létrehoz egy áramkört az argumentumokban megadott komponensekből (olyan, mintha mindegyik paraméter egy leíró fájl egy sora lenne)

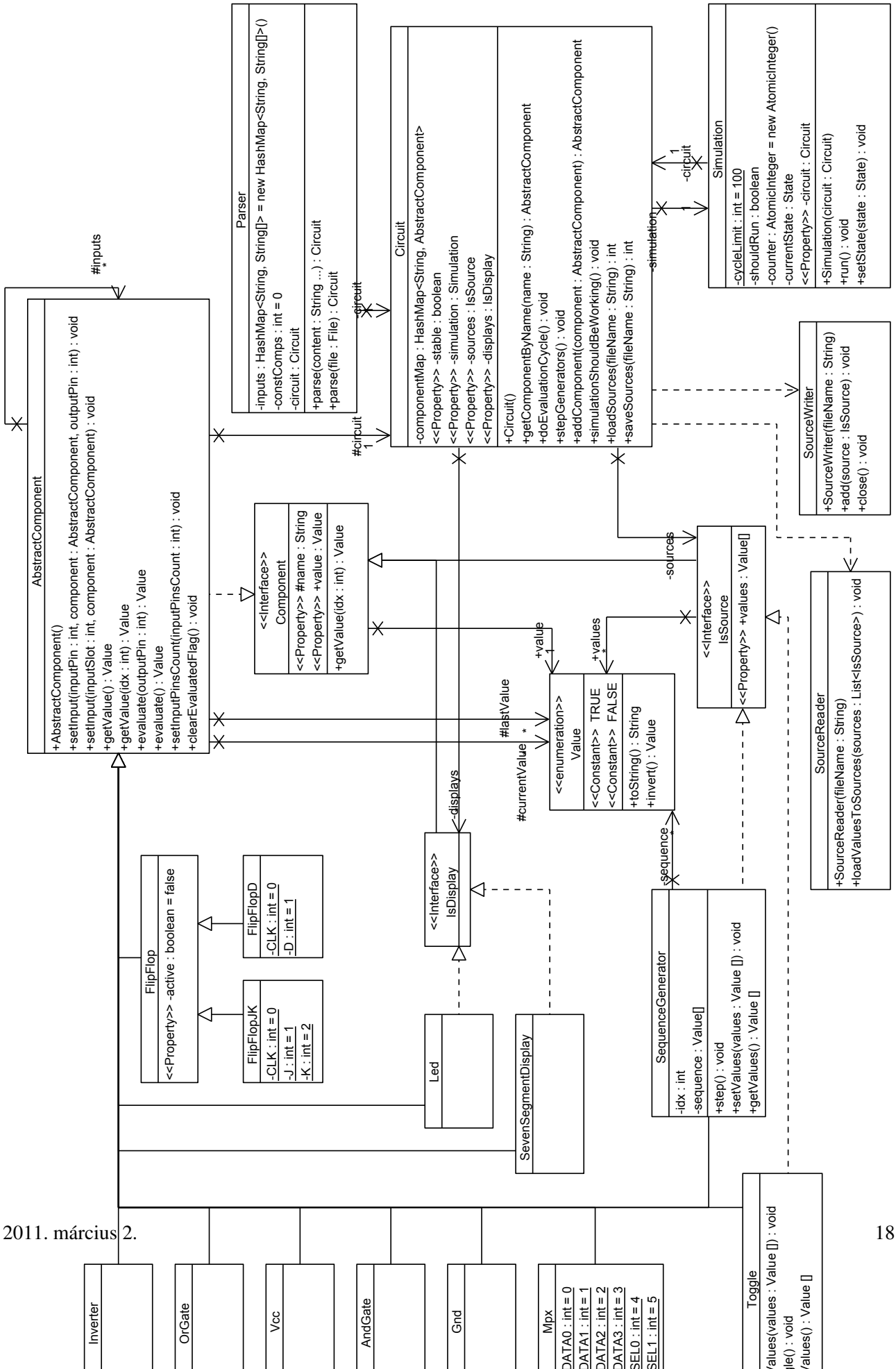
3.2.25. SourceReader

- Felelősség
  A jelforrások értékeinek fájlból történő beolvasására szolgáló osztály.
- Ősosztályok: Object → SourceReader.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - `public void loadValuesToSources(List sources)`: A paraméterben kapott jelforrások értékeit megpróbálja a fájlban található értékekkel feltölteni.

3.2.26. SourceWriter

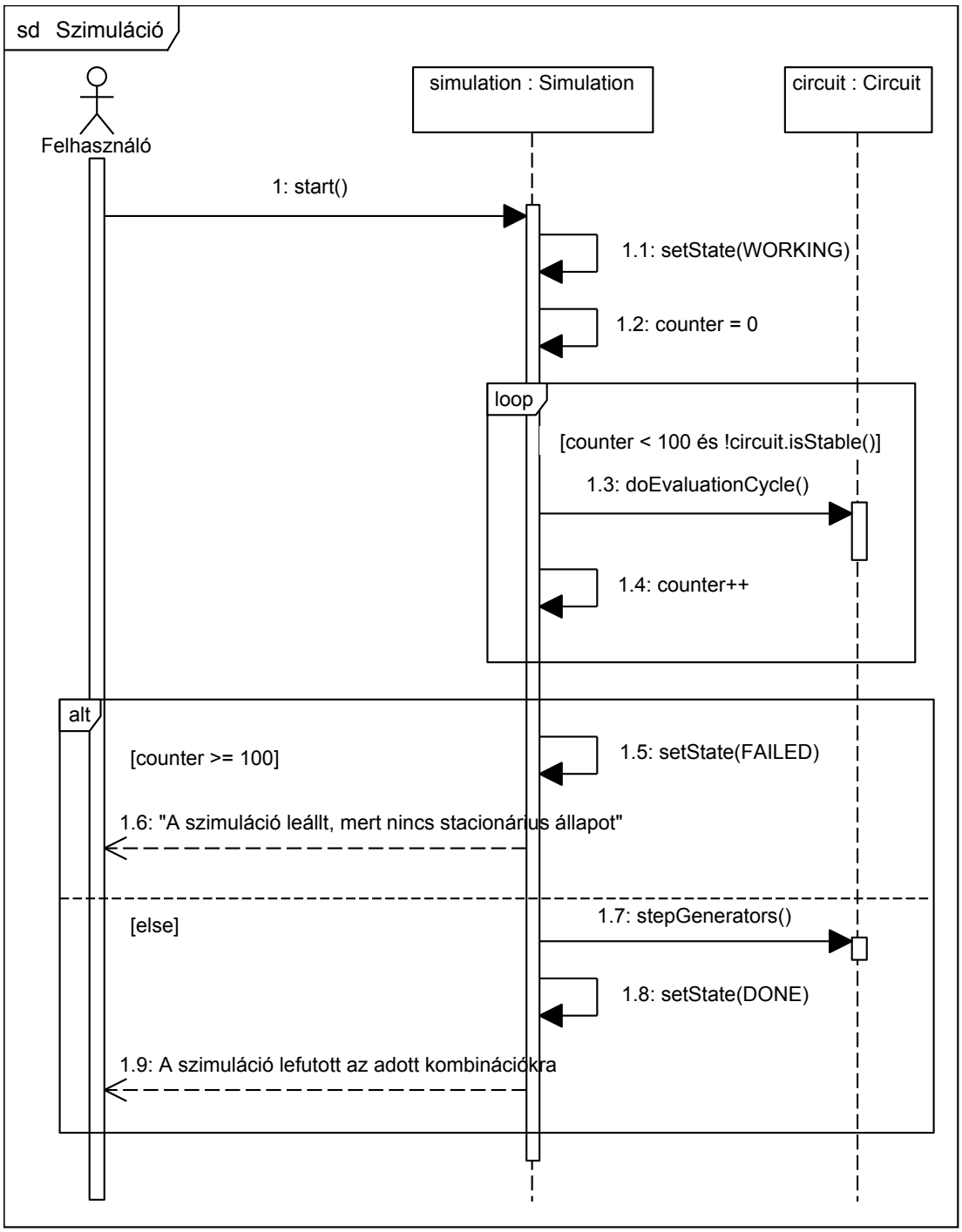
- Felelősség
  Segítségével kiírhatjuk egy fájlba a jelforrások értékeit.
- Ősosztályok: Object → SourceWriter.
- Interfészek: (nincs)
- Attribútumok
  - (nincs)
- Metódusok
  - `public void add(IsSource source)`: Hozzáadjuk a fájlhoz az adott jelforrás beállítását
  - `public void close()`: Bezárjuk a fájlt, ha végeztünk, ezt meg kell hívnunk.

3.3. Statikus struktúra diagramok

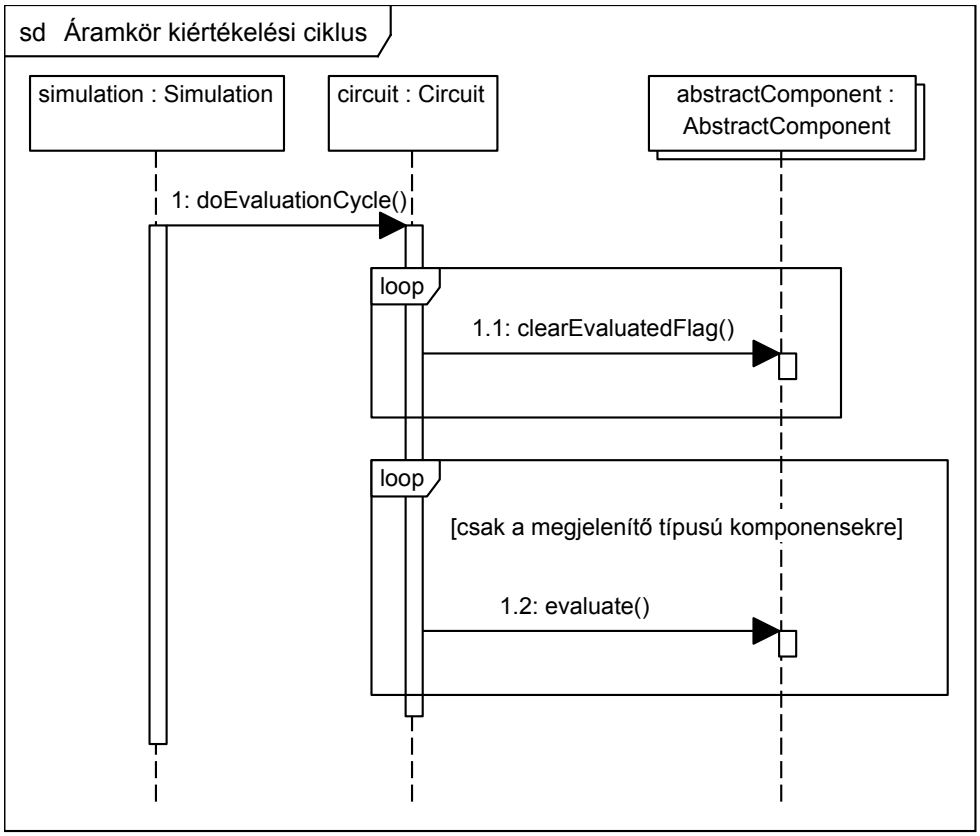


3.4. Szekvencia diagramok

Inicializálásra nem készült szekvencia diagram, mert amikor a rendszer elindul tulajdonképpen csak egy felhasználónk van. Amint az betölt egy áramkört lesz áramkör, és ha indít szimulációt akkor lesz szimuláció. Addig gyakorlatilag semmi se jön létre.

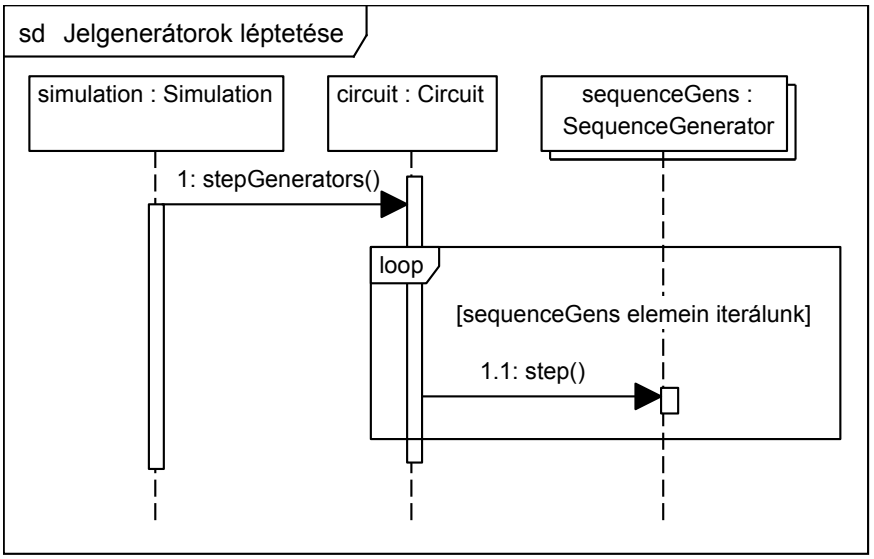


3.2. ábra. Szimuláció futás közben



3.3. ábra. Áramkör kiértékelési ciklus

3.4. ábra. Komponens kiértékelése



3.5. ábra. Jelgenerátorok léptetése

3.5. State-chartok

3.6. Napló

Kezdet	Időtartam	Résztvevők	Leírás
2011.02.23. 10:00	30 perc	**Kriván B.** **Dévényi A.** **Péter T.** **Apagyi G.** **Jákli G.**	Analízis modell kezdeti lépéseinek megbeszélése
2011.02.23. 15:00	30 perc	**Péter T.**	LaTeX és Visual Paradigm for UML szoftverek beállítása.
2011.02.25. 22:00	2 óra	**Péter T.**	Objektum katalógus kitöltése.
2011.02.26. 15:00	1 óra	**Apagyi G.** **Péter T.**	Objektum katalógus elemeinek átbeszélése
2011.02.26. 16:00	1 óra	**Apagyi G.**	Átállás a megbeszélt programokra (LaTeX, VP)
2011.02.26. 17:00	2 óra	**Apagyi G.**	Sequence diagram (start/stop) szerkesztése
2011.02.26. 16:00	1 óra	**Kriván B.**	Szimuláció futtatás közben
2011.02.26. 17:00	1 óra	**Jákli G.**	Áramkör betöltése fájlból
2011.02.26. 17:00	1 óra	**Dévényi A.**	Jelforrások betöltése
2011.02.26. 18:00	1 óra	**Kriván B.**	Áramkör betöltése fájlból - folytatás
2011.02.26. 19:00	1 óra	**Dévényi A.**	Jelforrások mentése
2011.02.26. 19:00	1 óra	**Kriván B.**	Szimuláció kiértékelés
2011.02.27. 10:00	1 óra	**Kriván B.**	Jelforrások betöltése - folytatás
2011.02.27. 11:00	1 óra	**Kriván B.**	Jelforrások mentése - folytatás
2011.02.27. 11:00	1 óra	**Dévényi A.**	FF-ok vezérlése
2011.02.27. 11:00	1 óra	**Jákli G.**	Áramkör választás
2011.02.27. 13:00	1 óra	**Kriván B.**	FF-ok vezérlése - folytatás
2011.02.27. 14:00	1 óra	**Kriván B.**	Áramkör választás - folytatás
2011.02.27. 14:00	1 óra	**Jákli G.**	Jelforrások módosítása
2011.02.27. 15:00	1 óra	**Kriván B.**	Szimuláció állapotváltozásai
2011.02.27. 15:00	20 perc	**Jákli G.**	Szimuláció állapotai
2011.02.27. 16:00	1 óra	**Kriván B.**	Jelgenerátor léptetése
2011.02.27. 16:00	20 perc	**Jákli G.**	Szimuláció start/stop
2011.02.27. 19:00	1 óra	**Dévényi A.**	Class Diagram elkészítése
2011.02.27. 19:00	1 óra	**Kriván B.**	Dokumentáció 3.2-es pontjának véglegesítése
2011.02.27. 20:00	30 perc	**Kriván B.**	Class Diagram méretre vágása, beleillesztése a dokumentációba