8. Részletes tervek

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. április 4.

Tartalomjegyzék

Ábrák jegyzéke

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 kezdethez
- `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()`:

  `- void eval(String s)`: Parancs értelmezése

  `+ static void main(String[] args)`: Program belépési pontja.

  `+ 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 Viewt egy kontrollerrel és a kimenettel, ide fog menni a kimenet.

+ `void newline()`: Új sor a kimeneten

+ `void writeDetails(AbstractComponent ac)`: Kiírunk egy komponenst (be és kimenetek)

+ `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 ac)`: Kiírunk egy scope-ot

+ `void writeScopeValues(Scope scope)`: Kiírja a scope által tárolt értékeket

+ `void writeSequenceGenerator(SequenceGenerator sg)`: Szekvenciagenerátor szekvenciájának kiírása

+ `void writeSequenceGeneratorSequence(SequenceGenerator sg)`: Kiírja a jelgenerátor szekvenciáját

+ `void writeSequenceGeneratorValue(SequenceGenerator sg)`: Kiírja a jelgenerátor éppen kiadott értékét

+ `void writeSevenSegmentDisplayValues(SevenSegmentDisplay seg)`: Kiírja 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 kimenetek)

  + `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 writeSequenceGeneratorSequence(SequenceGenerator sg)`: Kiírja a jelgenerátor szekvenciáját

`+ void writeSequenceGeneratorValue(SequenceGenerator sg)`: Kiírja a jelgenerátor éppen kiadott értékét

`+ void writeSevenSegmentDisplayValues(SevenSegmentDisplay seg)`: Kiírja 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 vezetéket
  + `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()`: Oszcilloszkópok 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

`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 komponenst 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 ősosztálya, 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 komponenst 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 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` 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 Object → AbstractComponent → 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 Object → AbstractComponent → 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 Object → AbstractComponent → DisplayComponent → 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 szekvenciát állítja be.

  + `void addTo(Composite composite)`: Hozzáadás kompozithoz.

  + `SequenceGenerator copy(String newName)`:

  + `Value[] getValues()`: Jelgenerátor bitsorozatának lekérdezése

  # `void onEvaluation()`:

  + `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.

  + `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 setValues(Value[] newValues)`: Kapcsoló állapotának változtatása, csak 1 elemű tömböt kaphat paraméterül.

  + `void writeValueTo(Viewable view):`

8.1.29. Vcc

- Felelősség
  A tápfeszültés komponens, ami konstans igaz értéket ad. Nincs bemenete.
- Ősosztályok Object → AbstractComponent → 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

Bemenet	*Kimenet*

`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

Bemenet	*Kimenet*
loadCircuit test2.ovr switch inmpx1 switch inmpx3 step switch selmpx2 switch seg2 step switch selmpx2 switch selmpx1 step	load successful inmpx1: 1 inmpx3: 1 simulation successful inmpx1: 1 inmpx2: 0 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

Bemenet	*Kimenet*

`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 megfelelelően 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

Bemenet	*Kimenet*

`loadCircuit test5.ovr`
`switch k`
`step`
`step`
`switch j`
`step`
`step`
`switch j`
`switch k`
`step`
`step`
	`load successful`
	
	`k: 1`
	
	`simulation successful`
	`j: 0`
	`k: 1`
	`seqgen: 0`
	`scope: 0`
	
	`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

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

Bemenet	*Kimenet*

`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 bementére 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

Bemenet	*Kimenet*

`loadCircuit test7.ovr`
`step`
`step`
`switch tog`
`step`
`step`
	`load successful`
	
	`simulation successful`
	`tog: 0`
	`led: 1`
	
	`simulation successful`
	`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 tesztesetnél az alábbi lehet:

- "Teszt sikeres!" ha a generált tesztfájl megegyezik a referencia fájllal
- 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	**Péter T.**	Tesztesetek megtervezése, leírása, felépítésük megadása a bemeneti nyelvnek megfelelően
2011.04.02. 10:00	3 óra	**Apagyi G.**	Tesztesetek felhasználói interakciójának, illetve várt kimeneteinek megtervezése.
2011.04.04. 10:00	4 óra	**Kriván B.**	Komponensek implementálása, osztályok leírásának megcsinálása
2011.04.04. 11:00	3 óra	**Jákli G.**	Konzultáció Kriván B-vel.
2011.04.04. 10:00	4 óra	**Dévényi A.**	Tesztesetek ellenőrzése, szépítése, FF logika áttervezése.