Logikai hálózat szimulálórendszer

Készítette Doxygen 1.13.2

Falucskai András
A2PT9J

1. fejezet

Felhasználói dokumentáció

1.0.1. Program leírása

A program egy logikai hálózatot modellez. Ebben komponensek vannak, amiknek több bemenete, de csak egy kimenete van (például AND kapu). A be és kimenetek vezetékre csatlakoznak, amik a jelet ütemezve továbbítják a többi komponens felé.

A programban megvalósuló osztályok:

- AND, XOR, OR, NOT bináris kapuk
- Wire: Komponenseket összekötő vezeték
- INP: Konstans jelet adó komponens
- STD_INP: Olyan INP, ami a jelet első frissítéskor a standard bemenetről olvassa be
- LogicNetwork: A vezetékeket és komponenseket egybefoglaló logikai hálózat osztály
- LogicNetworkConfigurer: Olyan logikai hálózat osztály, aminek a felépítését fájlból lehet beolvasni és visszaírni

1.0.2. Szimuláció működése

Az időbeli szimulálás miatt a vezetékeknek két része van. A vezeték egyes részeire akárhány komponens csatlakozhat, így csomópontként is funkcionál. A vezeték „elejére” csak írni lehet, a „végéről” pedig csak olvasni lehet a jelet.

A hálózat frissítése („órajele”) két fázisból áll. Első lépésben a komponensek leolvassák a bemenetükre kötött vezetékek végéről az adatot. Az eredmény kirakják a rájuk kötött vezetékek elejére. A nagyobb jelszint felülírja a kisebbet.

A második fázisban minden vezeték átrakja a jelet az elejéről a végére. Így a frissítés sorrendjétől független lesz a hálózat viselkedése.

Ennek megfelelően egy egész logikai hálózat frissítése először az összes komponens, majd az összes vezeték frissítéséből áll. (Ezt valósítja meg a logikai hálózat osztály update() függvénye)

A logikai hálózatnak van bulk_update() függvénye, ami egymás után több frissítést végez el. Az egyes frissítések kezdetét a kimeneten jelzi.

1.0.3. Hálózat létrehozása

Hálózatot a main.cpp fájl példái alapján lehet létrehozni kódban a logikai hálózatok tagfüggvényeivel, vagy pedig szöveges fájl alapján is. A fájl szerinti beolvasásra is van példa a main.cpp-ben.

Hálózat létrehozásakor meg kell adni a kábelek számát, és az adatfolyamot ahova a frissítéseket kezdetét jelző szöveget írni fogja. Fájlból olvasásnál a PRINT komponensek is ide írnak.

A fájlból beolvasásnál az első sorban a kábelek, majd a komponensek számát kell megadni, utána soronként az egyes komponenseket a nekik megfelelő formátumban. A bemenetek a kábel sorszámát jelzik a hálózatban (0-tól indexelve):

- AND / OR / XOR [1. bemenet] [2. bemenet] [kimenet]
- NOT [bemenet] [kimenet]
- INP [kimenet] [kimeneti jel]
- STD_INP [kimenet] [tetszőleges címke]
- PRINT [bemenet] [tetszőleges címke]

1.1. Programozói dokumentáció

1.1.1. Hibák, kivételek

- A logikai hálózat beolvasása ha nem sikerült, szövegesen eldobja, hol érzékelt hibát.
- A logikai hálózatban a vezeték lekérésekor szöveges hibát dob, ha nincs elég kábel.
- STD_INP komponens szöveges hibát dob, ha a bejövő jel nem 0 vagy 1. Bár a program tudna más jeleket is kezelni, a megvalósított komponensek csak ezt a két értéket értelmezik.

1.1.2. Tesztesetek

A main.cpp fájlban vannak a gtest_lite-al megvalósított tesztek. Az utolsó teszteset standard bemenetről olvas, amihez példabemenetet kap a Jporta.

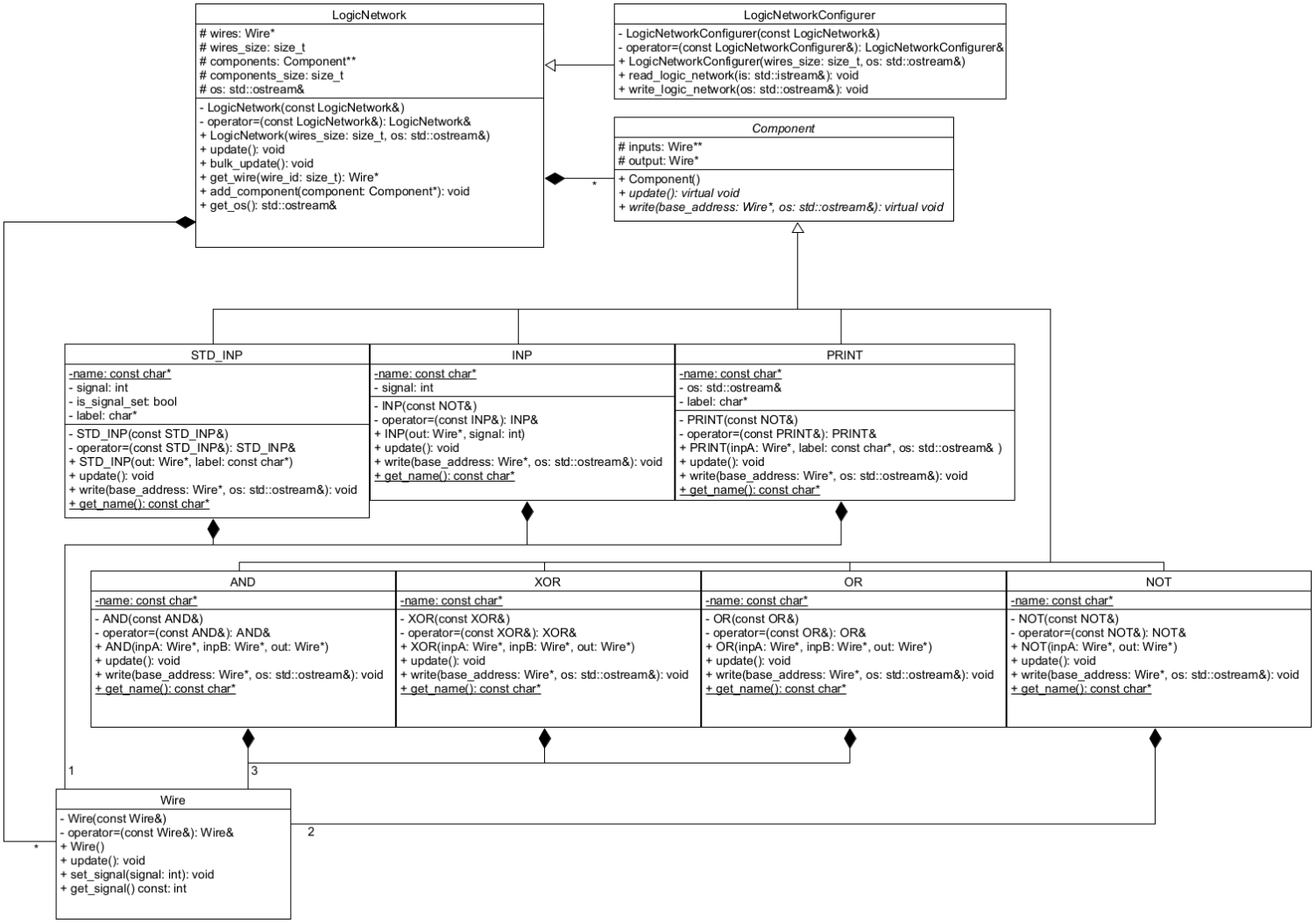
1.1.3. Egyéb megjegyzések

A komponensek mutatókkal tárolják, melyik vezetékhez kapcsolódnak. Emiatt minden osztálynál le van tiltva a másoló konstruktor és az értékadó operátor, mivel nem lehet / nincs értelme ugyan olyan objektumokat létrehozni.

1.1.4. Új komponens felvétele

Az új komponens felvételéhez az osztálynak a többi komponenshez hasonlóan a Component leszármazottjának kell lennie. Ezen felül tetszőlegesen megvalósítható a beolvasás / kiírás. A beolvasást a LogicNetworkConfigurer read_logic_network() függvényében kell megvalósítani a többihez hasonlóan. A komponens nevében nem szerepelhet whitespace, és legfeljebb 100 karakter hosszú lehet.

1.2. UML diagram



2. fejezet

Hierarchikus mutató

2.1. Osztályhierarchia

Majdnem (de nem teljesen) betűrendbe szedett leszármazási lista:

3. fejezet

Osztálymutató

3.1. Osztálylista

Az összes osztály, struktúra, unió és interfész listája rövid leírásokkal:

4. fejezet

Fájlmutató

4.1. Fájllista

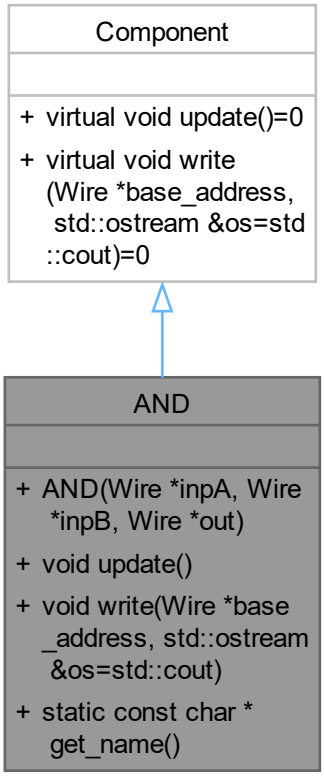
Az összes dokumentált fájl listája rövid leírásokkal:

5. fejezet

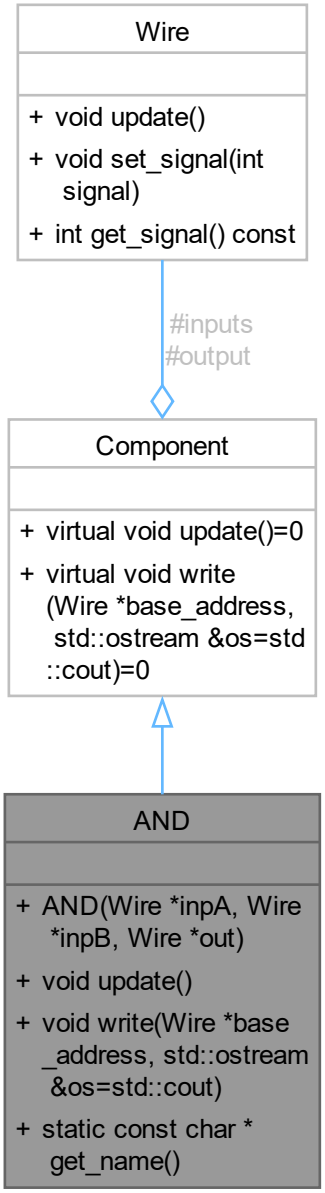
Osztályok dokumentációja

5.1. AND osztályreferencia

Az AND osztály származási diagramja:



Az AND osztály együttműködési diagramja:



Publikus tagfüggvények

- AND (Wire ∗inpA, Wire ∗inpB, Wire ∗out)

  AND komponens konstruktora.
- void **update** ()

  Kimeneti jel a bemeneteken végzett AND eredménye.
- void write (Wire ∗base_address, std::ostream &os=std::cout)

  Komponens kiírása a fájlba mentéshez.

Statikus publikus tagfüggvények

- static const char * get_name ()

  Visszaadja a komponens elmentésekor használt nevet.

5.1.1. Konstruktorok és destruktorok dokumentációja

5.1.1.1. AND()

```
AND::AND (
            Wire * inpA,
            Wire * inpB,
            Wire * out)
```

AND komponens konstruktora.

Paraméterek

inpA	Első bemenet
inpB	Második bemenet
out	Kimenet

5.1.2. Tagfüggvények dokumentációja

5.1.2.1. get_name()

```
static const char * AND::get_name ()  [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

Visszatérési érték

A komponens neve

5.1.2.2. write()

```
void AND::write (
            Wire * base_address,
            std::ostream & os = std::cout)  [virtual]
```

Komponens kiírása a fájlba mentéshez.

Paraméterek

base_address	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
os	A kimeneti adatfolyam, amire a kapu kiírja magát

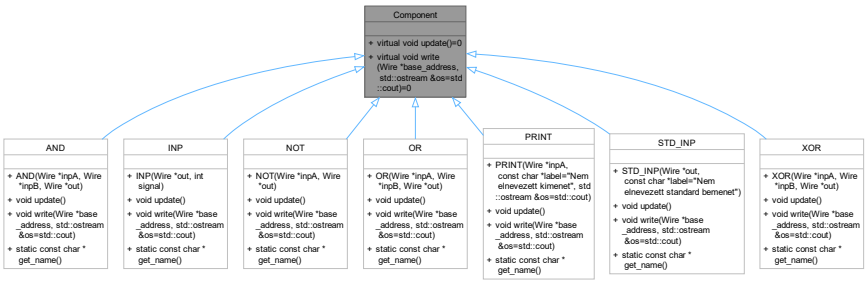
Megvalósítja a következőket: Component.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

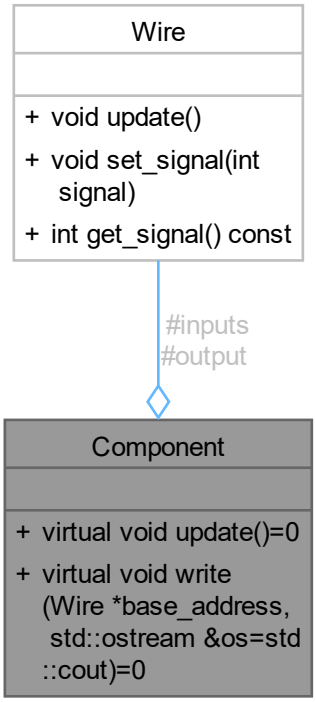
- AND.h
- AND.cpp

5.2. Component osztályreferencia

A Component osztály származási diagramja:



A Component osztály együttműködési diagramja:



Publikus tagfüggvények

- virtual void **update** ()=0

  Kaputól függően a frissítés elvégzése:
  Bementi jelekből a kimeneti jelek kiszámítása / egyéb műveletek végrehajtása.
- virtual void write (Wire *base_address, std::ostream &os=std::cout)=0

  Komponens kiírása a fájlba mentéshez.

5.2.1. Tagfüggvények dokumentációja

5.2.1.1. write()

```
virtual void Component::write (
            Wire * base_address,
            std::ostream & os = std::cout)  [pure virtual]
```

Komponens kiírása a fájlba mentéshez.

Paraméterek

base_address	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
os	A kimeneti adatfolyam, amire a kapu kiírja magát

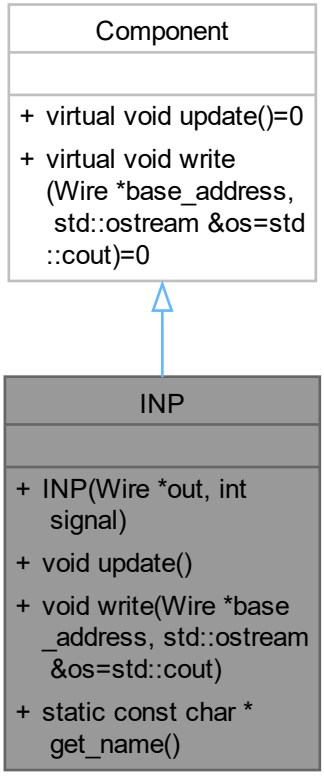
Megvalósítják a következők: AND, INP, NOT, OR, PRINT, STD_INP és XOR.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

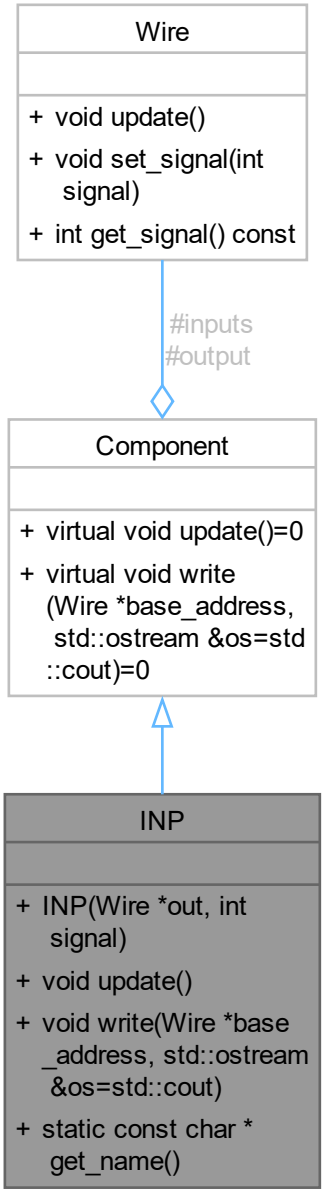
- component.h
- component.cpp

5.3. INP osztályreferencia

Az INP osztály származási diagramja:



Az INP osztály együttműködési diagramja:



Publikus tagfüggvények

- INP (Wire ∗out, int signal)

  INP komponens konstruktora.
- void **update** ()

  A belső jelet kirakja a kimeneti kábelre.
- void write (Wire ∗base_address, std::ostream &os=std::cout)

  Komponens kiírása a fájlba mentéshez.

Statikus publikus tagfüggvények

- static const char * get_name ()

  Visszaadja a komponens elmentésekor használt nevet.

5.3.1. Konstruktorok és destruktorok dokumentációja

5.3.1.1. INP()

```
INP::INP (
            Wire * out,
            int signal)
```

INP komponens konstruktora.

Paraméterek

out	Kimenet

5.3.2. Tagfüggvények dokumentációja

5.3.2.1. get_name()

```
static const char * INP::get_name ()  [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

Visszatérési érték

A komponens neve

5.3.2.2. write()

```
void INP::write (
            Wire * base_address,
            std::ostream & os = std::cout)  [virtual]
```

Komponens kiírása a fájlba mentéshez.

Paraméterek

base_address	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
os	A kimeneti adatfolyam, amire a kapu kiírja magát

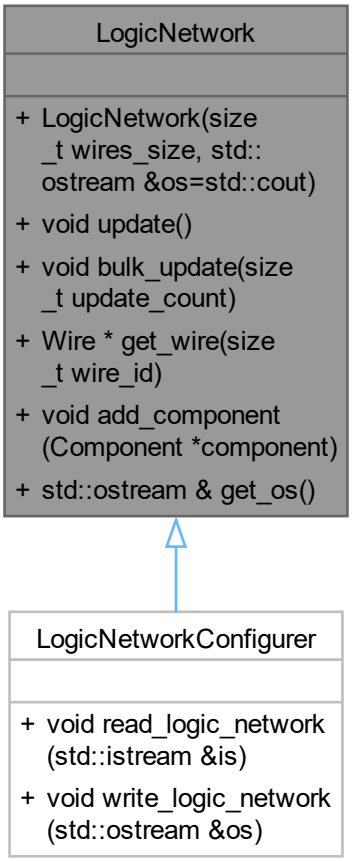
Megvalósítja a következőket: Component.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

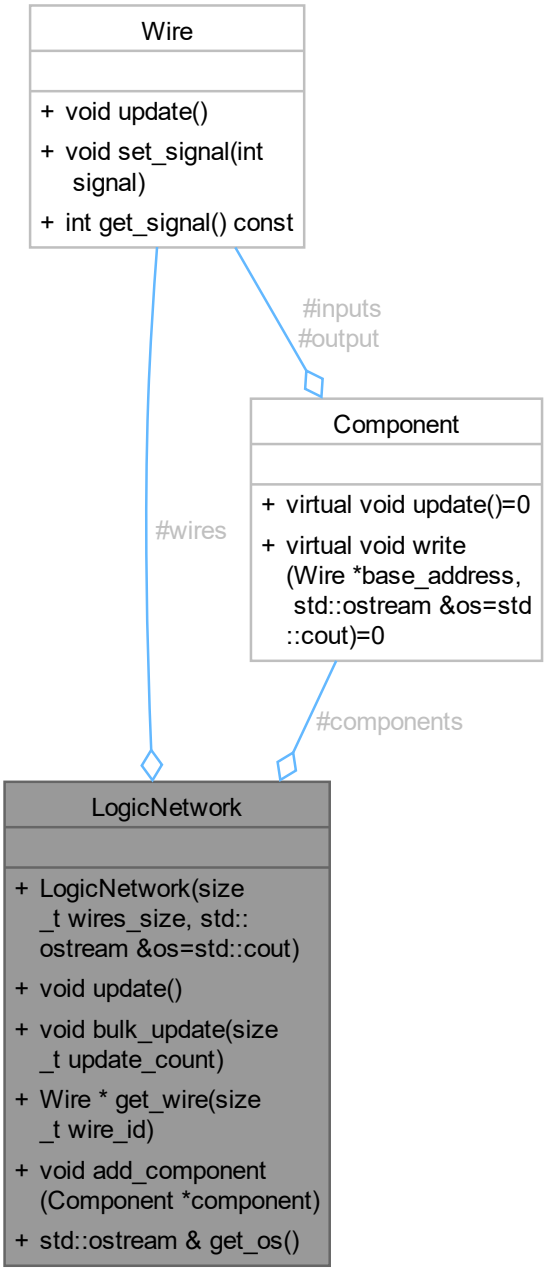
- INP.h
- INP.cpp

5.4. LogicNetwork osztályreferencia

A LogicNetwork osztály származási diagramja:



A LogicNetwork osztály együttműködési diagramja:



Publikus tagfüggvények

- LogicNetwork (size_t wires_size, std::ostream &os=std::cout)

  Logikai hálózat konstruktora.
- void **update** ()

  A hálózat elemeinek frissítése: először kábelek, majd komponensek sorrendben.
- void bulk_update (size_t update_count)

Egyszerre több frissítés futtatása. Kiírja az egyes frissítések kezdetét az adatfolyamra.

- Wire * get_wire (size_t wire_id)

  Visszaad egy mutatót az eltárolt vezetékekre sorszám alapján.

- void add_component (Component *component)

  Hozzáad egy komponenst a hálózathoz. A komponenst a hálózat fogja felszabadítani.

- std::ostream & get_os ()

  Visszaadja a kimeneti adatfolyamot, amire a frissítések kezdetét jelző szöveget írja ki.

5.4.1. Konstruktorok és destruktorok dokumentációja

5.4.1.1. LogicNetwork()

```
LogicNetwork::LogicNetwork (
            size_t wires_size,
            std::ostream & os = std::cout)
```

Logikai hálózat konstruktora.

Paraméterek

wires_size	A hálózatban lévő kábelek maximális száma
os	Az adatfolyam, ahova a frissítéseket elválasztó üzeneteket írja

5.4.2. Tagfüggvények dokumentációja

5.4.2.1. add_component()

```
void LogicNetwork::add_component (
            Component * component)
```

Hozzáad egy komponenst a hálózathoz. A komponenst a hálózat fogja felszabadítani.

Paraméterek

component	Az új komponens mutatóval, amit hozzáad

5.4.2.2. bulk_update()

```
void LogicNetwork::bulk_update (
            size_t update_count)
```

Egyszerre több frissítés futtatása. Kiírja az egyes frissítések kezdetét az adatfolyamra.

Paraméterek

update_count	A frissítések száma

5.4.2.3. get_os()

```
std::ostream & LogicNetwork::get_os ()  [inline]
```

Visszaadja a kimeneti adatfolyamot, amire a frissítések kezdetét jelző szöveget írja ki.

Visszatérési érték

std::ostream&

5.4.2.4. get_wire()

```
Wire * LogicNetwork::get_wire (
            size_t wire_id)
```

Visszaad egy mutatót az eltárolt vezetékekre sorszám alapján.

Kivételek

const_char∗	Ha nincsen wire_id számú kábel, "Nincs elég kábel!" kivételt dob

Paraméterek

wire↩ _id	Vezeték sorszáma

Visszatérési érték

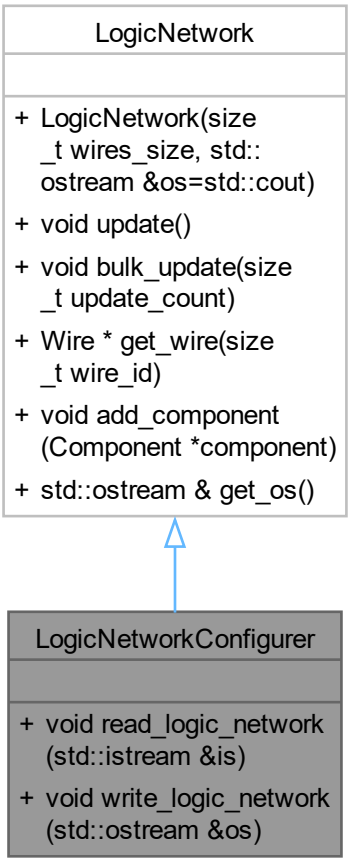
Wire∗

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

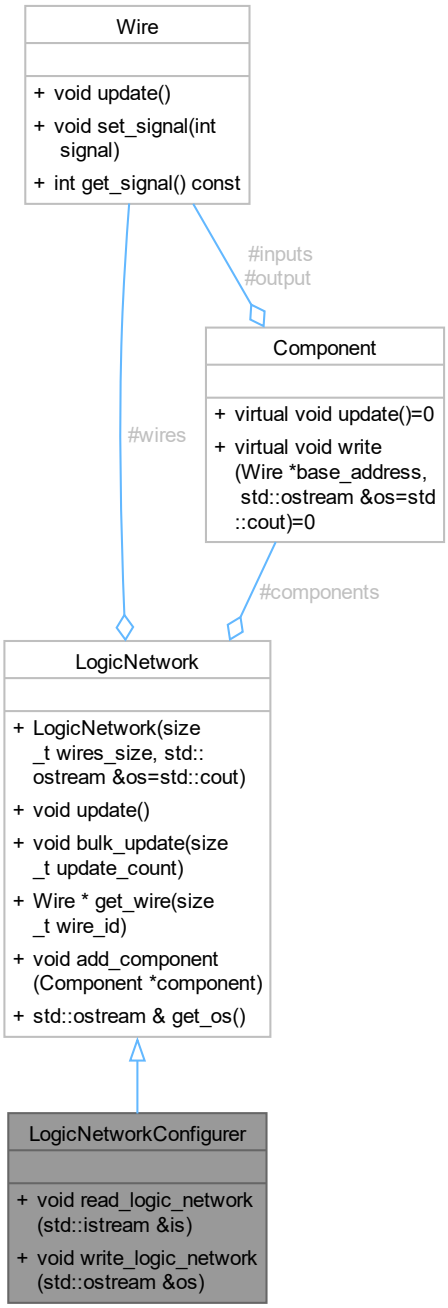
- logic_network.h
- logic_network.cpp

5.5. LogicNetworkConfigurer osztályreferencia

A LogicNetworkConfigurer osztály származási diagramja:



A LogicNetworkConfigurer osztály együttműködési diagramja:



Publikus tagfüggvények

- void read_logic_network (std::istream &is)

  Beolvassa a kapott bemeneti adatfolyamról a hálózatot.
- void write_logic_network (std::ostream &os)

  Kiírja egy kimeneti adatfolyamra a hálózatot.

Publikus tagfüggvények a(z) LogicNetwork osztályból származnak

- LogicNetwork (size_t wires_size, std::ostream &os=std::cout)

  Logikai hálózat konstruktora.
- void **update** ()

  A hálózat elemeinek frissítése: először kábelek, majd komponensek sorrendben.
- void bulk_update (size_t update_count)

  Egyszerre több frissítés futtatása. Kiírja az egyes frissítések kezdetét az adatfolyamra.
- Wire * get_wire (size_t wire_id)

  Visszaad egy mutatót az eltárolt vezetékekre sorszám alapján.
- void add_component (Component *component)

  Hozzáad egy komponenst a hálózathoz. A komponenst a hálózat fogja felszabadítani.
- std::ostream & get_os ()

  Visszaadja a kimeneti adatfolyamot, amire a frissítések kezdetét jelző szöveget írja ki.

5.5.1. Tagfüggvények dokumentációja

5.5.1.1. read_logic_network()

```
void LogicNetworkConfigurer::read_logic_network (
            std::istream & is)
```

Beolvassa a kapott bemeneti adatfolyamról a hálózatot.

Kivételek

*const_char**	A különböző hibaeseteket leíró cstringet dob, ha nem sikerült a beolvasás.

Paraméterek

is	A bemeneti adatfolyam

5.5.1.2. write_logic_network()

```
void LogicNetworkConfigurer::write_logic_network (
            std::ostream & os)
```

Kiírja egy kimeneti adatfolyamra a hálózatot.

Paraméterek

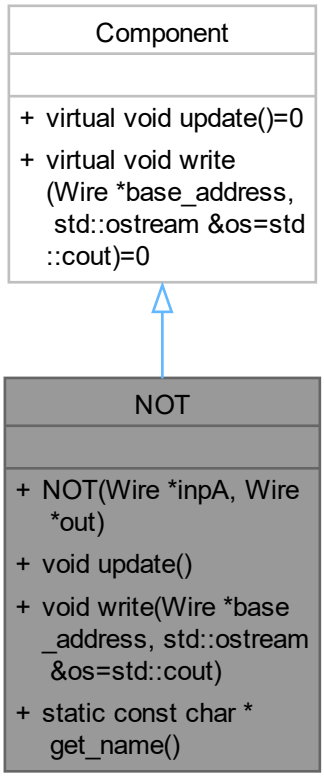
os	A kimeneti adatfolyam

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

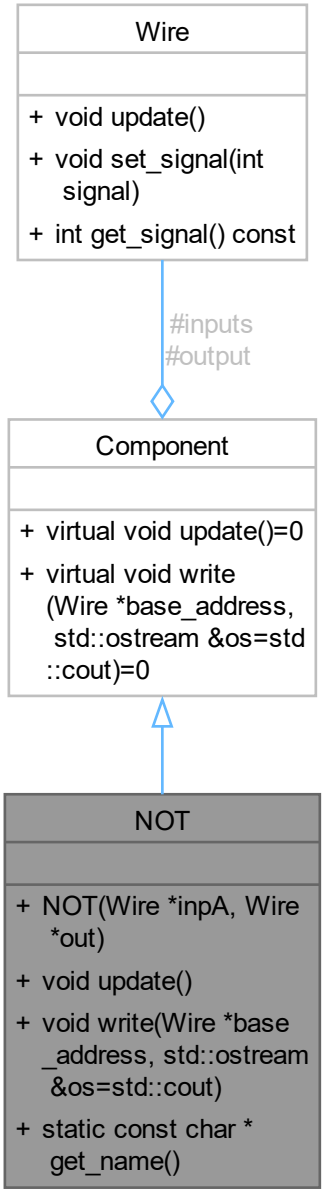
- logic_network_configurer.h
- logic_network_configurer.cpp

5.6. NOT osztályreferencia

A NOT osztály származási diagramja:



A NOT osztály együttműködési diagramja:



Publikus tagfüggvények

- NOT (Wire ∗inpA, Wire ∗out)

  NOT komponens konstruktora.
- void **update** ()

  Kimeneti jel a bemeneten végzett NOT eredménye.
- void write (Wire ∗base_address, std::ostream &os=std::cout)

  Komponens kiírása a fájlba mentéshez.

Statikus publikus tagfüggvények

- static const char * get_name ()

  Visszaadja a komponens elmentésekor használt nevet.

5.6.1. Konstruktorok és destruktorok dokumentációja

5.6.1.1. NOT()

```
NOT::NOT (
            Wire * inpA,
            Wire * out)
```

NOT komponens konstruktora.

Paraméterek

inpA	Bemenet
out	Kimenet

5.6.2. Tagfüggvények dokumentációja

5.6.2.1. get_name()

```
static const char * NOT::get_name ()  [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

Visszatérési érték

A komponens neve

5.6.2.2. write()

```
void NOT::write (
            Wire * base_address,
            std::ostream & os = std::cout)  [virtual]
```

Komponens kiírása a fájlba mentéshez.

Paraméterek

base_address	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
os	A kimeneti adatfolyam, amire a kapu kiírja magát

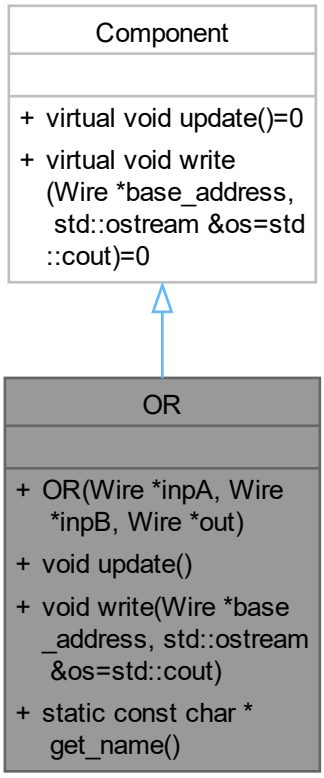
Megvalósítja a következőket: Component.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

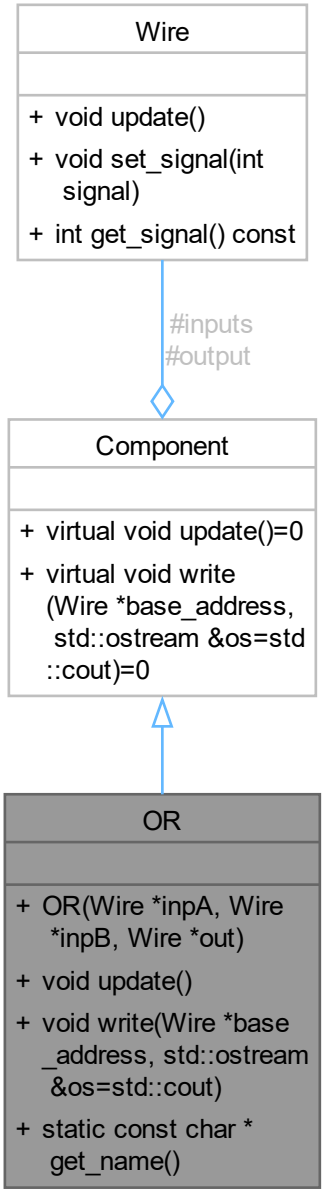
- NOT.h
- NOT.cpp

5.7. OR osztályreferencia

Az OR osztály származási diagramja:



Az OR osztály együttműködési diagramja:



Publikus tagfüggvények

- OR (Wire ∗inpA, Wire ∗inpB, Wire ∗out)

  OR komponens konstruktora.
- void **update** ()

  Kimeneti jel a bemeneteken végzett OR eredménye.
- void write (Wire ∗base_address, std::ostream &os=std::cout)

  Komponens kiírása a fájlba mentéshez.

Statikus publikus tagfüggvények

- static const char * get_name ()

  Visszaadja a komponens elmentésekor használt nevet.

5.7.1. Konstruktorok és destruktorok dokumentációja

5.7.1.1. OR()

```
OR::OR (
            Wire * inpA,
            Wire * inpB,
            Wire * out)
```

OR komponens konstruktora.

Paraméterek

inpA	Első bemenet
inpB	Második bemenet
out	Kimenet

5.7.2. Tagfüggvények dokumentációja

5.7.2.1. get_name()

```
static const char * OR::get_name ()  [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

Visszatérési érték

A komponens neve

5.7.2.2. write()

```
void OR::write (
            Wire * base_address,
            std::ostream & os = std::cout)  [virtual]
```

Komponens kiírása a fájlba mentéshez.

Paraméterek

base_address	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
os	A kimeneti adatfolyam, amire a kapu kiírja magát

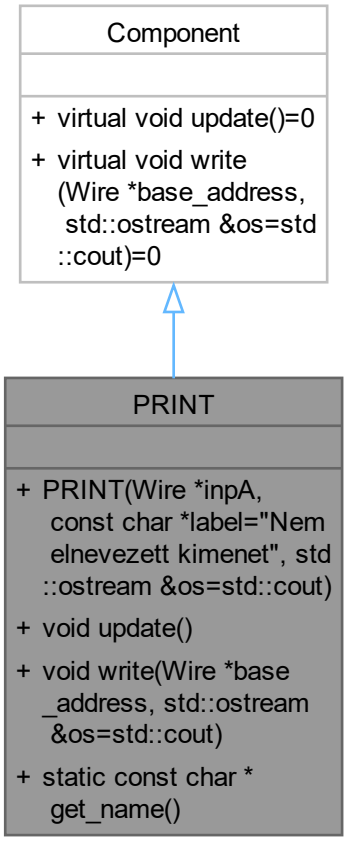
Megvalósítja a következőket: Component.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

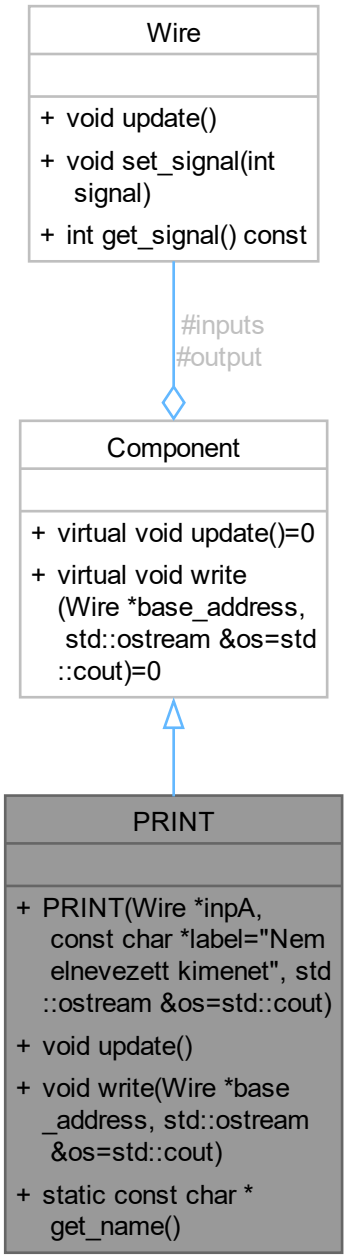
- OR.h
- OR.cpp

5.8. PRINT osztályreferencia

A PRINT osztály származási diagramja:



A PRINT osztály együttműködési diagramja:



Publikus tagfüggvények

- PRINT (Wire ∗inpA, const char ∗label="Nem elnevezett kimenet", std::ostream &os=std::cout)

  PRINT komponens konstruktora.
- void **update** ()

  Kiírja a bemeneti kábel értékét a megadott adatfolyamra.
- void write (Wire ∗base_address, std::ostream &os=std::cout)

  Komponens kiírása a fájlba mentéshez.

Statikus publikus tagfüggvények

- static const char * get_name ()

  Visszaadja a komponens elmentésekor használt nevet.

5.8.1. Konstruktorok és destruktorok dokumentációja

5.8.1.1. PRINT()

```
PRINT::PRINT (
            Wire * inpA,
            const char * label = "Nem elnevezett kimenet",
            std::ostream & os = std::cout)
```

PRINT komponens konstruktora.

Paraméterek

inpA	Bemeneti kábel
label	A címke, ami alapján meg lehet különböztetni a kimeneteket
os	Az adatfolyam, amire a kimenetét írja

5.8.2. Tagfüggvények dokumentációja

5.8.2.1. get_name()

```
static const char * PRINT::get_name ()  [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

Visszatérési érték

A komponens neve

5.8.2.2. write()

```
void PRINT::write (
            Wire * base_address,
            std::ostream & os = std::cout)  [virtual]
```

Komponens kiírása a fájlba mentéshez.

Paraméterek

base_address	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
os	A kimeneti adatfolyam, amire a kapu kiírja magát

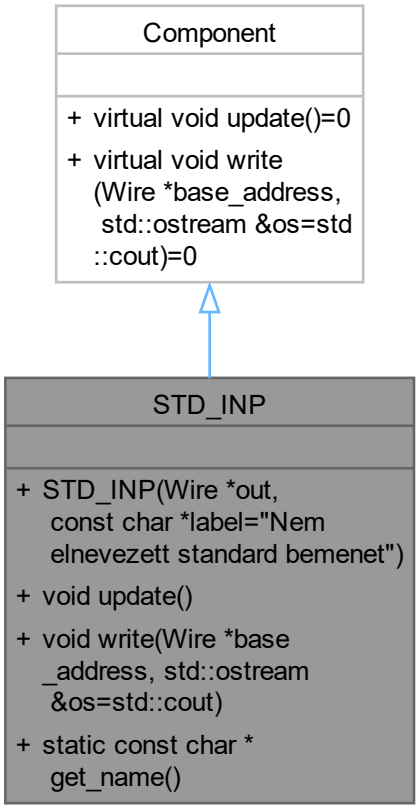
Megvalósítja a következőket: Component.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

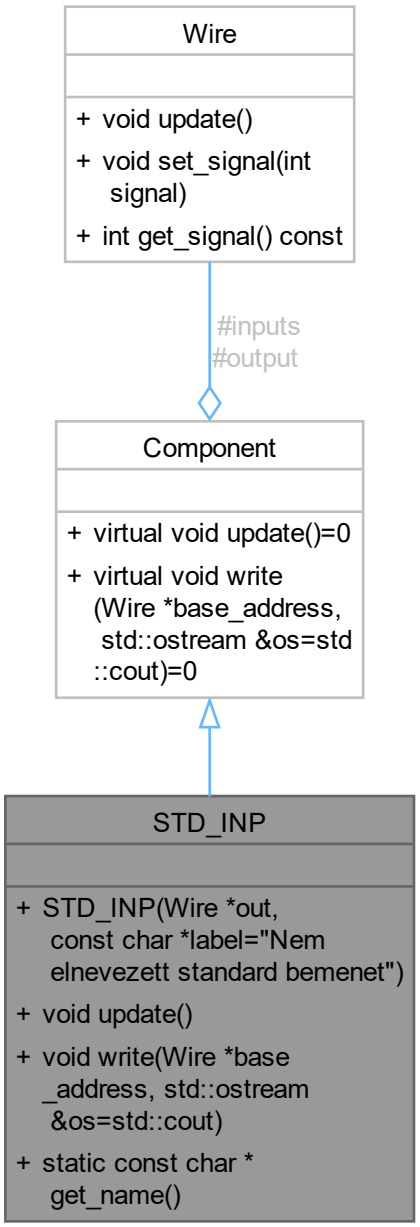
- PRINT.h
- PRINT.cpp

5.9. STD_INP osztályreferencia

A STD_INP osztály származási diagramja:



A STD_INP osztály együttműködési diagramja:



Publikus tagfüggvények

- STD_INP (Wire ∗out, const char ∗label="Nem elnevezett standard bemenet")

  STD_INP komponens konstruktora.
- void update ()

  Első frissítéskor elkéri, hogy mit rakjon a kimenetre. Ez után ilyen értékű INP kapuként működik.
- void write (Wire ∗base_address, std::ostream &os=std::cout)

  Komponens kiírása a fájlba mentéshez.

Statikus publikus tagfüggvények

- static const char * get_name ()

  Visszaadja a komponens elmentésekor használt nevet.

5.9.1. Konstruktorok és destruktorok dokumentációja

5.9.1.1. STD_INP()

```
STD_INP::STD_INP (
            Wire * out,
            const char * label = "Nem elnevezett standard bemenet")
```

STD_INP komponens konstruktora.

Paraméterek

out	Kimenet
label	A bemenetek megkülönböztetésére használt címke

5.9.2. Tagfüggvények dokumentációja

5.9.2.1. get_name()

```
static const char * STD_INP::get_name ()  [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

Visszatérési érték

A komponens neve

5.9.2.2. update()

```
void STD_INP::update ()  [virtual]
```

Első frissítéskor elkéri, hogy mit rakjon a kimenetre. Ez után ilyen értékű INP kapuként működik.

Kivételek

*const_char**	Első frissítésnél "Hibás bemenet egy STD_INP kapunak" kivételt dob, ha a kapott érték nem 0 vagy 1

Megvalósítja a következőket: Component.

5.9.2.3. write()

```
void STD_INP::write (
            Wire * base_address,
            std::ostream & os = std::cout)  [virtual]
```

Komponens kiírása a fájlba mentéshez.

Paraméterek

base_address	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
os	A kimeneti adatfolyam, amire a kapu kiírja magát

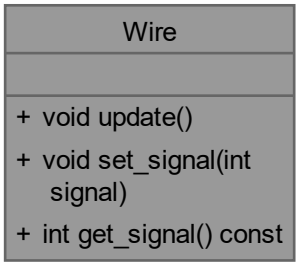
Megvalósítja a következőket: Component.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- STD_INP.h
- STD_INP.cpp

5.10. Wire osztályreferencia

A Wire osztály együttműködési diagramja:



Publikus tagfüggvények

- void **update** ()

  A vezeték elejéről a végére rakja a jelet, elejét 0-ra állítja.
- void set_signal (int signal)

  Beállítja a bemenetén a jelet, a kapott és a már ott lévő érték közül a nagyobbra.
- int get_signal () const

  Visszaadja a kimenetén lévő jelet.

5.10.1. Tagfüggvények dokumentációja

5.10.1.1. get_signal()

```
int Wire::get_signal () const  [inline]
```

Visszaadja a kimenetén lévő jelet.

Visszatérési érték

int

5.10.1.2. set_signal()

```
void Wire::set_signal (
            int signal)
```

Beállítja a bemenetén a jelet, a kapott és a már ott lévő érték közül a nagyobbra.

Paraméterek

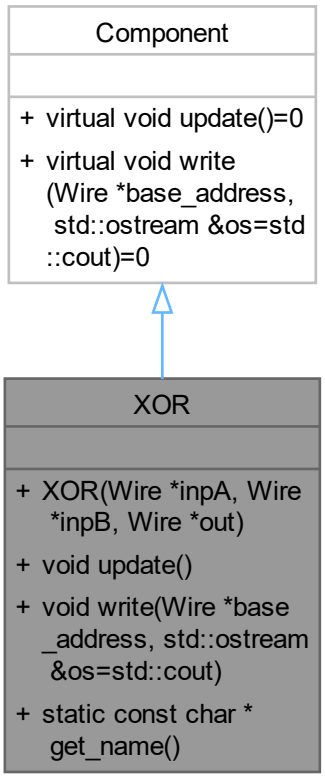
signal	

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

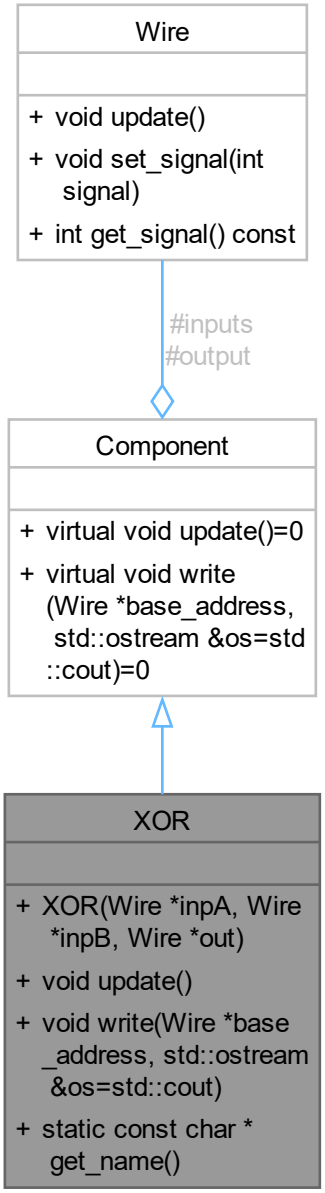
- wire.h
- wire.cpp

5.11. XOR osztályreferencia

A XOR osztály származási diagramja:



A XOR osztály együttműködési diagramja:



Publikus tagfüggvények

- XOR (Wire ∗inpA, Wire ∗inpB, Wire ∗out)

  XOR komponens konstruktora.
- void **update** ()

  Kimeneti jel a bemeneteken végzett XOR eredménye.
- void write (Wire ∗base_address, std::ostream &os=std::cout)

  Komponens kiírása a fájlba mentéshez.

Statikus publikus tagfüggvények

- static const char * get_name ()

  Visszaadja a komponens elmentésekor használt nevet.

5.11.1. Konstruktorok és destruktorok dokumentációja

5.11.1.1. XOR()

```
XOR::XOR (
            Wire * inpA,
            Wire * inpB,
            Wire * out)
```

XOR komponens konstruktora.

Paraméterek

inpA	Első bemenet
inpB	Második bemenet
out	Kimenet

5.11.2. Tagfüggvények dokumentációja

5.11.2.1. get_name()

```
static const char * XOR::get_name ()  [inline], [static]
```

Visszaadja a komponens elmentésekor használt nevet.

Visszatérési érték

A komponens neve

5.11.2.2. write()

```
void XOR::write (
            Wire * base_address,
            std::ostream & os = std::cout)  [virtual]
```

Komponens kiírása a fájlba mentéshez.

Paraméterek

base_address	A logikai hálózat vezetékeket tároló tömbjének kezdőcíme
os	A kimeneti adatfolyam, amire a kapu kiírja magát

Megvalósítja a következőket: Component.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- XOR.h
- XOR.cpp

6. fejezet

Fájlok dokumentációja

6.1. basic_components.h

```
00001 #ifndef BASIC_COMPONENTS_H
00002 #define BASIC_COMPONENTS_H
00003 // alkatalógusból importálja az egyszerű komponenseket
00004 #include "basic_components/AND.h"
00005 #include "basic_components/INP.h"
00006 #include "basic_components/NOT.h"
00007 #include "basic_components/OR.h"
00008 #include "basic_components/PRINT.h"
00009 #include "basic_components/STD_INP.h"
00010 #include "basic_components/XOR.h"
00011 #endif
```

6.2. AND.h

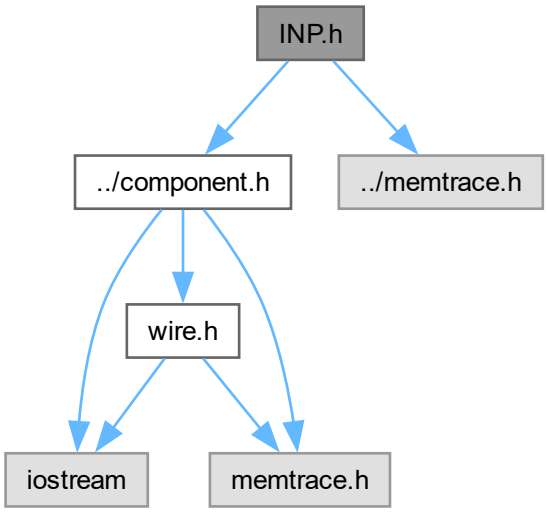
```
00001
00002 #ifndef AND_H
00003 #define AND_H
00004
00005 #include "../component.h"
00006 #include "../memtrace.h"
00007
00008 class AND : public Component {
00009     static const char* name;
00010     // copy constructor és értékadó operátor letiltása
00011     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00012     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00013     AND(const AND&);
00014     AND& operator=(const AND&);
00015
00016    public:
00017    public:
00025     AND(Wire* inpA, Wire* inpB, Wire* out);
00029     void update();
00030     void write(Wire* base_address, std::ostream& os = std::cout);
00036     static const char* get_name() {
00037         return name;
00038     }
00039 };
00040 #endif
```

6.3. INP.h fájlreferencia

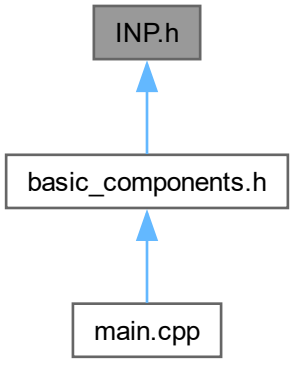
Kábelre konstans értéket író komponens.

```
#include "../component.h"
#include "../memtrace.h"
```

Az INP.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class INP

6.3.1. Részletes leírás

Kábelre konstans értéket író komponens.

6.4. INP.h

Ugrás a fájl dokumentációjához.

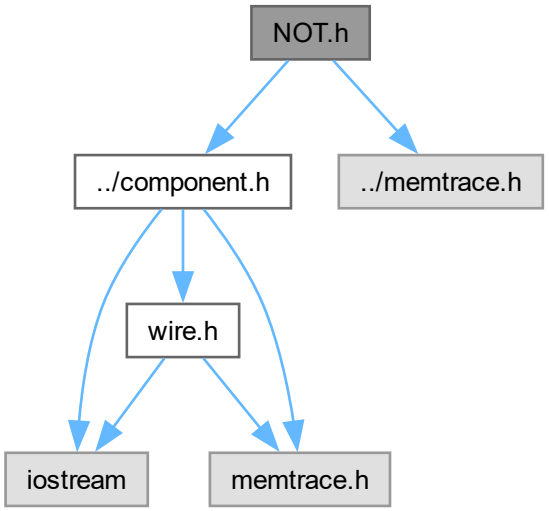
```
00001
00005 #ifndef INP_H
00006 #define INP_H
00007
00008 #include "../component.h"
00009 #include "../memtrace.h"
00010
00011 class INP : public Component {
00012     static const char* name;
00013     int signal;
00014     // copy constructor és értékadó operátor letiltása
00015     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00016     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00017     INP(const INP&);
00018     INP& operator=(const INP&);
00019
00020   public:
00026     INP(Wire* out, int signal);
00030     void update();
00031     void write(Wire* base_address, std::ostream& os = std::cout);
00037     static const char* get_name() {
00038         return name;
00039     }
00040 };
00041
00042 #endif
```

6.5. NOT.h fájlreferencia

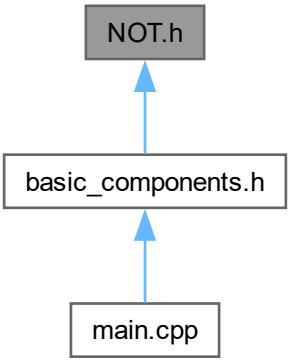
NOT kaput megvalósító komponens.

```
#include "../component.h"
#include "../memtrace.h"
```

A NOT.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class NOT

6.5.1. Részletes leírás

NOT kaput megvalósító komponens.

6.6. NOT.h

Ugrás a fájl dokumentációjához.

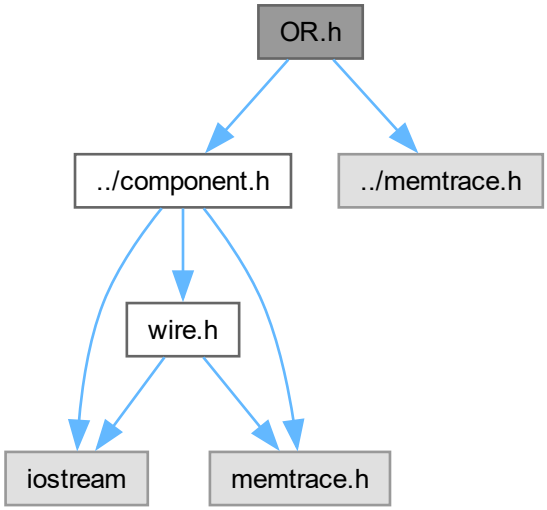
```
00001
00005 #ifndef NOT_H
00006 #define NOT_H
00007
00008 #include "../component.h"
00009 #include "../memtrace.h"
00010
00011 class NOT : public Component {
00012     static const char* name;
00013
00014     // copy constructor és értékadó operátor letiltása
00015     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00016     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00017     NOT(const NOT&);
00018     NOT& operator=(const NOT&);
00019
00020   public:
00027     NOT(Wire* inpA, Wire* out);
00031     void update();
00032     void write(Wire* base_address, std::ostream& os = std::cout);
00038     static const char* get_name() {
00039         return name;
00040     }
00041 };
00042
00043 #endif
```

6.7. OR.h fájlreferencia

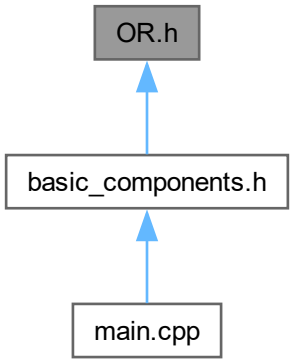
OR kaput megvalósító komponens.

```
#include "../component.h"
#include "../memtrace.h"
```

Az OR.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class OR

6.7.1. Részletes leírás

OR kaput megvalósító komponens.

6.8. OR.h

Ugrás a fájl dokumentációjához.

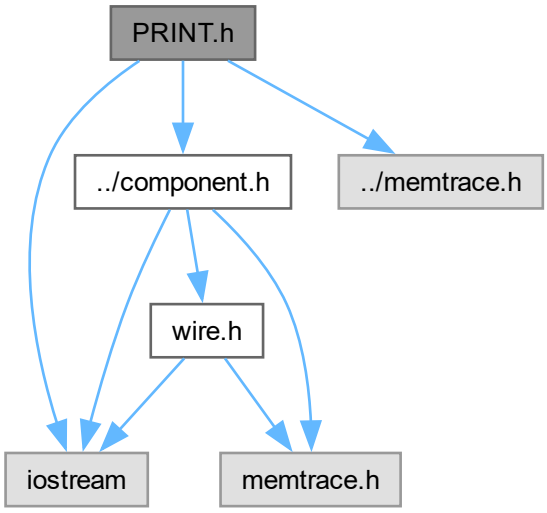
```
00001
00005
00006 #ifndef OR_H
00007 #define OR_H
00008 #include "../component.h"
00009 #include "../memtrace.h"
00010
00011 class OR : public Component {
00012     static const char* name;
00013     // copy constructor és értékadó operátor letiltása
00014     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00015     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00016     OR(const OR&);
00017     OR& operator=(const OR&);
00018
00019    public:
00027     OR(Wire* inpA, Wire* inpB, Wire* out);
00031     void update();
00032     void write(Wire* base_address, std::ostream& os = std::cout);
00038     static const char* get_name() {
00039         return name;
00040     }
00041 };
00042
00043 #endif
```

6.9. PRINT.h fájlreferencia

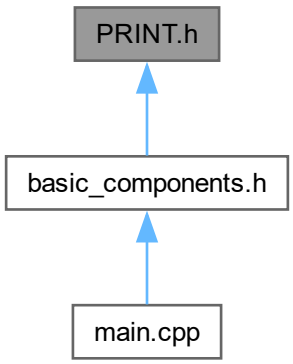
A kábel értéket kimenetre író komponens.

```
#include <iostream>
#include "../component.h"
#include "../memtrace.h"
```

A PRINT.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class PRINT

6.9.1. Részletes leírás

A kábel értéket kimenetre író komponens.

6.10. PRINT.h

Ugrás a fájl dokumentációjához.

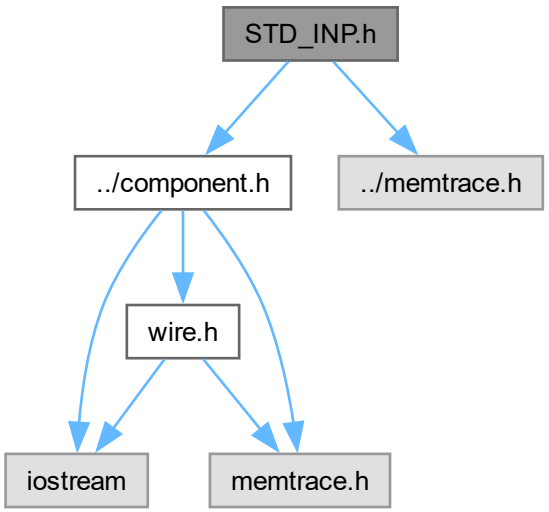
```
00001
00007 #ifndef PRINT_H
00008 #define PRINT_H
00009
00010 #include <iostream>
00011
00012 #include "../component.h"
00013 #include "../memtrace.h"
00014 class PRINT : public Component {
00015     static const char* name;
00016     std::ostream& os;
00017     char* label;
00018
00019     // copy constructor és értékadó operátor letiltása
00020     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00021     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00022     PRINT(const PRINT&);
00023     PRINT& operator=(const PRINT&);
00024
00025   public:
00033     PRINT(Wire* inpA, const char* label = "Nem elnevezett kimenet", std::ostream& os = std::cout);
00038     void update();
00039     void write(Wire* base_address, std::ostream& os = std::cout);
00045     static const char* get_name() {
00046         return name;
00047     }
00048     ~PRINT();
00049 };
00050
00051 #endif
```

6.11. STD_INP.h fájlreferencia

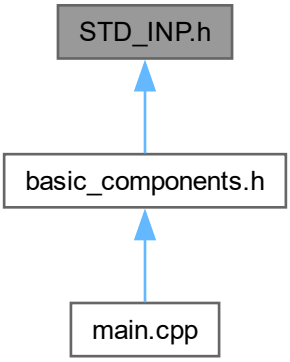
Olyan INP komponens, aminek az értékét a standard bemeneten lehet megadni.

```
#include "../component.h"
#include "../memtrace.h"
```

A STD_INP.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class STD_INP

6.11.1. Részletes leírás

Olyan INP komponens, aminek az értékét a standard bemeneten lehet megadni.

6.12. STD_INP.h

Ugrás a fájl dokumentációjához.

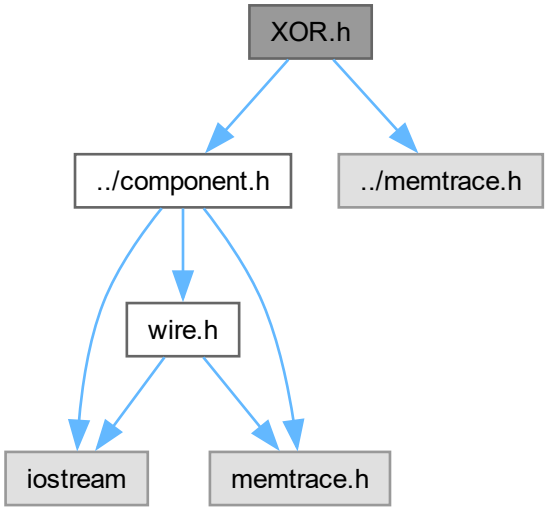
```
00001
00005
00006 #ifndef STD_INP_H
00007 #define STD_INP_H
00008
00009 #include "../component.h"
00010 #include "../memtrace.h"
00011
00012 // mindig a standard bemenetről olvassa be, hogy mit adjon ki
00013 class STD_INP : public Component {
00014     static const char* name;
00015     int signal;
00016     bool is_signal_set;
00017     char* label;
00018
00019     // copy constructor és értékadó operátor letiltása
00020     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00021     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00022     STD_INP(const STD_INP&);
00023     STD_INP& operator=(const STD_INP&);
00024
00025   public:
00032     STD_INP(Wire* out, const char* label = "Nem elnevezett standard bemenet");
00037     void update();
00038     void write(Wire* base_address, std::ostream& os = std::cout);
00044     static const char* get_name() {
00045         return name;
00046     }
00047     ~STD_INP();
00048 };
00049
00050 #endif
```

6.13. XOR.h fájlreferencia

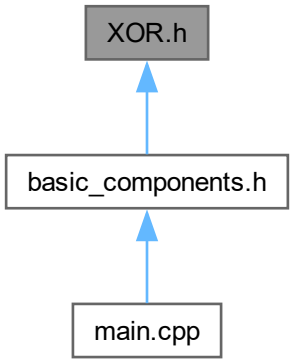
XOR kaput megvalósító komponens.

```
#include "../component.h"
#include "../memtrace.h"
```

A XOR.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class XOR

6.13.1. Részletes leírás

XOR kaput megvalósító komponens.

6.14. XOR.h

Ugrás a fájl dokumentációjához.

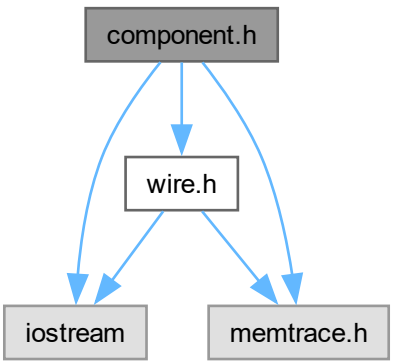
```
00001
00005 #ifndef XOR_H
00006 #define XOR_H
00007
00008 #include "../component.h"
00009 #include "../memtrace.h"
00010
00011 class XOR : public Component {
00012     static const char* name;
00013
00014     // copy constructor és értékadó operátor letiltása
00015     // mert memóriahibát okozna, és nincs értelme ugyan azt a komponenst létrehozni
00016     // pontosan ugyan azt csinálná két ugyan oda kötött komponens, mint egy
00017     XOR(const XOR&);
00018     XOR& operator=(const XOR&);
00019
00020   public:
00028     XOR(Wire* inpA, Wire* inpB, Wire* out);
00032     void update();
00033     void write(Wire* base_address, std::ostream& os = std::cout);
00039     static const char* get_name() {
00040         return name;
00041     }
00042 };
00043
00044 #endif
```

6.15. component.h fájlreferencia

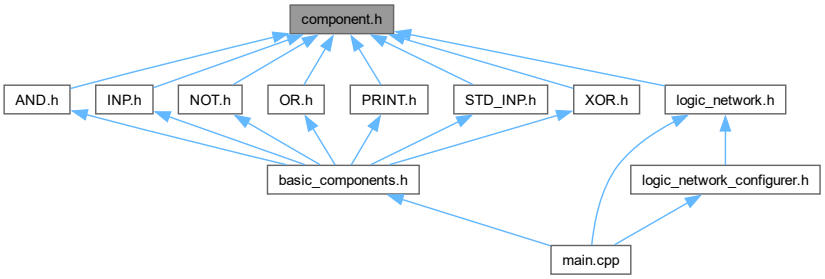
A komponensek absztrakt bázisosztálya.

```
#include <iostream>
#include "memtrace.h"
#include "wire.h"
```

A component.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class Component

6.15.1. Részletes leírás

A komponensek absztrakt bázisosztálya.

6.16. component.h

Ugrás a fájl dokumentációjához.

```
00001
00005
00006 #ifndef COMPONENT_H
00007 #define COMPONENT_H
00008 #include <iostream>
00009
00010 #include "memtrace.h"
00011 #include "wire.h"
00012
00013 class Component {
00014     // a leszármazottak állítják be, hogy mik legyenek a be és kimenetek
00015   protected:
00016     Wire** inputs;
00017     Wire* output;
00018
00019   public:
00020     Component() : inputs(nullptr), output(nullptr) {};
00025     virtual void update() = 0;
00032     virtual void write(Wire* base_address, std::ostream& os = std::cout) = 0;
00033     virtual ~Component();
00034 };
00035 #endif
```

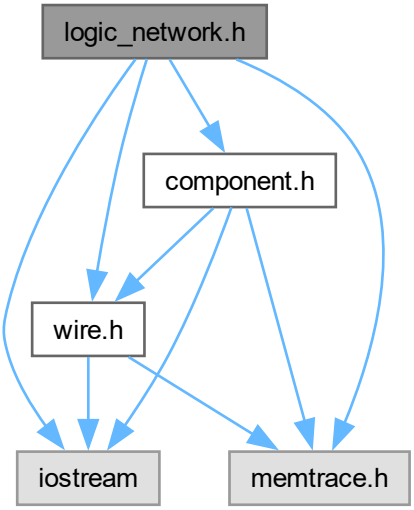
6.17. logic_network.h fájlreferencia

Logikai hálózatot megvalósító osztály. Eltárolja a neki átadott komponenseket, használat után törli azokat.

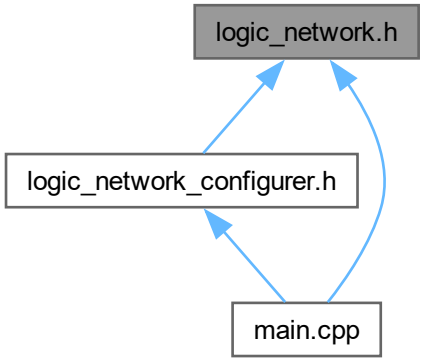
```
#include <iostream>
#include "component.h"
#include "memtrace.h"
```

```
#include "wire.h"
```

A logic_network.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class LogicNetwork

6.17.1. Részletes leírás

Logikai hálózatot megvalósító osztály. Eltárolja a neki átadott komponenseket, használat után törli azokat.

6.18. logic_network.h

Ugrás a fájl dokumentációjához.

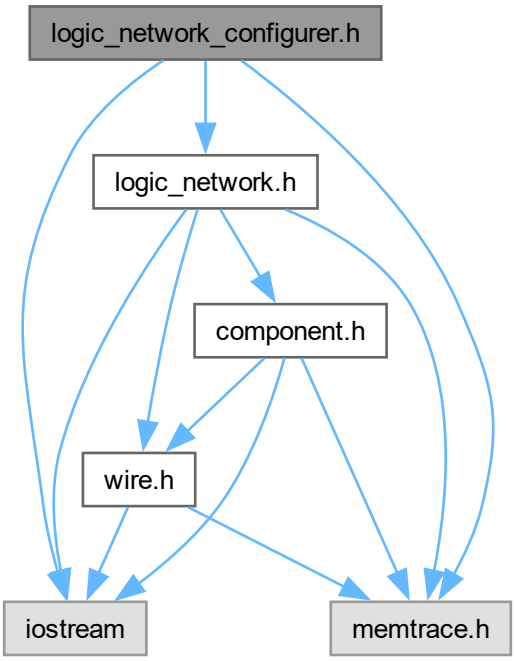
```
00001
00005 #ifndef LOGIC_NETWORK_H
00006 #define LOGIC_NETWORK_H
00007 #include <iostream>
00008
00009 #include "component.h"
00010 #include "memtrace.h"
00011 #include "wire.h"
00012 class LogicNetwork {
00013     // ne lehessen lemásolni, mert nincs sok értelme
00014     // és lehetetlen lenne lemásolni a mutatók miatt, valójában kirakná egy streamre és visszaolvasná
00015     // automatikusan a configurert se lehet lemásolni
00016     LogicNetwork(const LogicNetwork&);
00017     LogicNetwork& operator=(const LogicNetwork&);
00018
00019    protected:
00020     // UML-ben frissíteni, ez wire tömböt tárol, nem Wire*[]-t
00021     // mert asszem végül nem lehet felüldefelni a wire-t?
00022     Wire* wires;
00023     size_t wires_size;
00024
00025     Component** components;
00026     size_t components_size;
00027     // protecteden, mert nem kell tudnia a felhasználónak, hogy mennyi van a komponensekből, meg hogy
      mik azok
00028     // viszont ki szeretném írni
00029
00030     std::ostream& os;
00031
00032    public:
00039     LogicNetwork(size_t wires_size, std::ostream& os = std::cout);
00043     void update();
00049     void bulk_update(size_t update_count);
00050
00059     Wire* get_wire(size_t wire_id);
00065     void add_component(Component* component);
00066
00072     std::ostream& get_os() { return os; }
00073     virtual ~LogicNetwork();
00074 };
00075 #endif
```

6.19. logic_network_configurer.h fájlreferencia

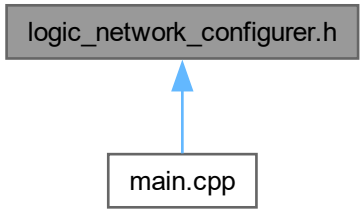
Fájlba menthető és fájlból beolvasható logikai hálózat osztály.

```
#include <iostream>
#include "logic_network.h"
#include "memtrace.h"
```

A logic_network_configurer.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class LogicNetworkConfigurer

6.19.1. Részletes leírás

Fájlba menthető és fájlból beolvasható logikai hálózat osztály.

6.20. logic_network_configurer.h

Ugrás a fájl dokumentációjához.

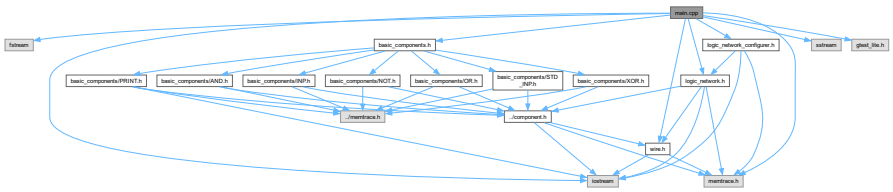
```
00001
00005 #ifndef LOGIC_NETWORK_CONFIGURER_H
00006 #define LOGIC_NETWORK_CONFIGURER_H
00007
00008 #include <iostream>
00009
00010 #include "logic_network.h"
00011 #include "memtrace.h"
00012 class LogicNetworkConfigurer : public LogicNetwork {
00013    public:
00014     LogicNetworkConfigurer(size_t wires_size = 0, std::ostream& os = std::cout) :
      LogicNetwork(wires_size, os) {}
00022     void read_logic_network(std::istream& is);
00028     void write_logic_network(std::ostream& os);
00029 };
00030
00031 #endif
```

6.21. main.cpp fájlreferencia

Main file az összes teszttel Minden teszt gtest_lita-al írt. Az utolsó teszteset standard bemenetről olvas.

```
#include <fstream>
#include <iostream>
#include <sstream>
#include "basic_components.h"
#include "gtest_lite.h"
#include "logic_network.h"
#include "logic_network_configurer.h"
#include "memtrace.h"
#include "wire.h"
```

A main.cpp definíciós fájl függési gráfja:



6.21.1. Részletes leírás

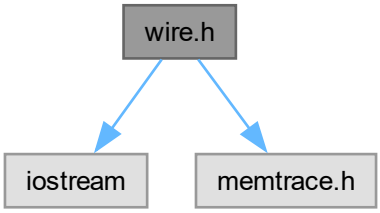
Main file az összes teszttel Minden teszt gtest_lita-al írt. Az utolsó teszteset standard bemenetről olvas.

6.22. wire.h fájlreferencia

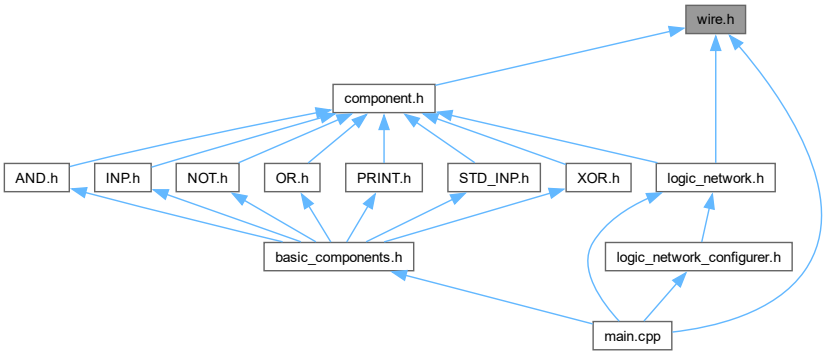
Vezetéket megvalósító osztály.

```
#include <iostream>
#include "memtrace.h"
```

A wire.h definíciós fájl függési gráfja:



Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:



Osztályok

- class Wire

6.22.1. Részletes leírás

Vezetéket megvalósító osztály.

6.23. wire.h

Ugrás a fájl dokumentációjához.

```
00001
00005 #ifndef WIRE_H
00006 #define WIRE_H
00007 #include <iostream>
```

```
00008
00009 #include "memtrace.h"
00010 class Wire {
00011     int input;
00012     int output;
00013     Wire(const Wire&);
00014     Wire& operator=(const Wire&);
00015
00016   public:
00017     Wire() : input(0), output(0) {}
00022     void update();
00028     void set_signal(int signal);
00034     int get_signal() const {
00035         return output;
00036     }
00037 };
00038
00039 #endif
```

Tárgymutató