

Аутоматика у паметним
стамбено-пословним објектима

ФБД програмски језик – Основе

Борис Јеличић

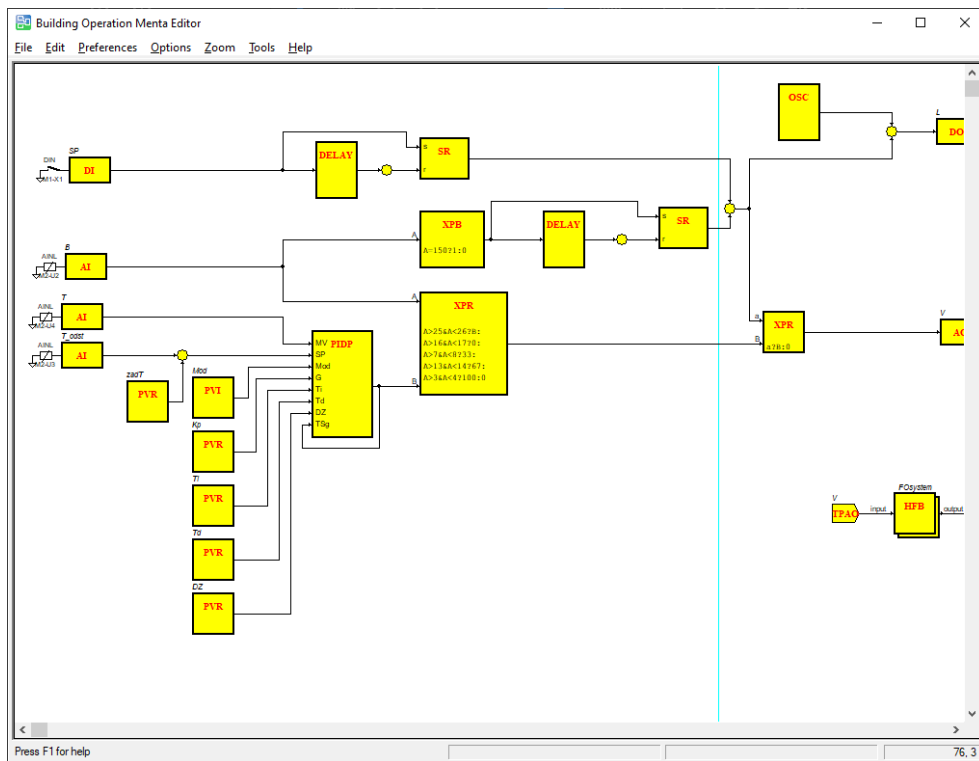
Садржај

- ▶ Function Block editor - опис корисничког окружења
- ▶ ФБД програмски језик
 - ▶ Увод у ФБД
 - ▶ Функцијски блокови
- ▶ Примери са упутством корак по корак
 - ▶ Креирање програма
 - ▶ Логичке операције
 - ▶ Аритметичке операције
 - ▶ Логичке функције
 - ▶ Тајмери
 - ▶ Бројачи
 - ▶ ПИД регулатори
 - ▶ Организација програмског кода
- ▶ Референце

Function Block editor

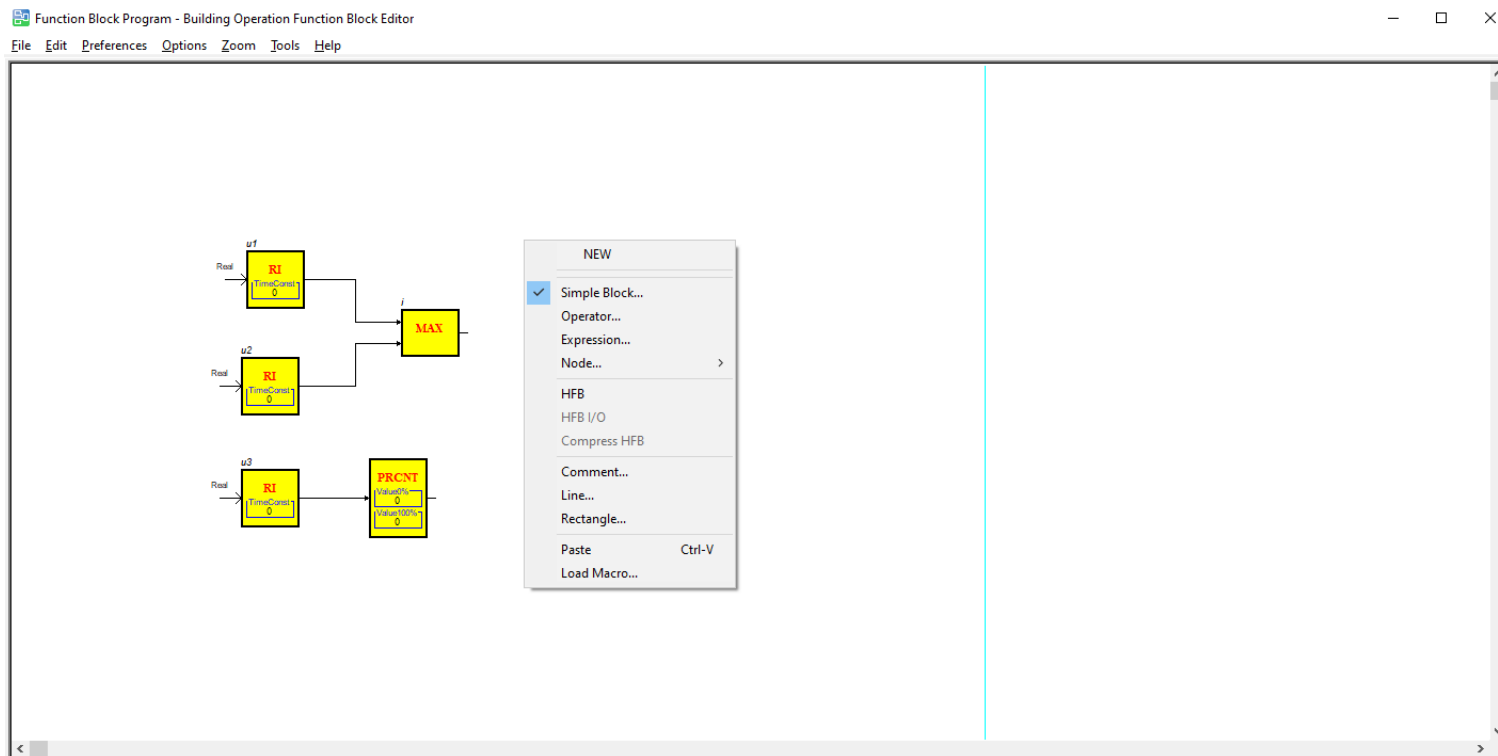
Function Block Editor

- ▶ Function Block Editor је софтверски алат за креирање и уређивање програма у ФБД (функцијски блок дијаграм) програмском језику
- ▶ ФБД (функцијски блок дијаграм) је графички оријентисан програмски језик
- ▶ Инсталација едитора је укључена у инсталациони пакет WorkStation



Режими рада (1/3)

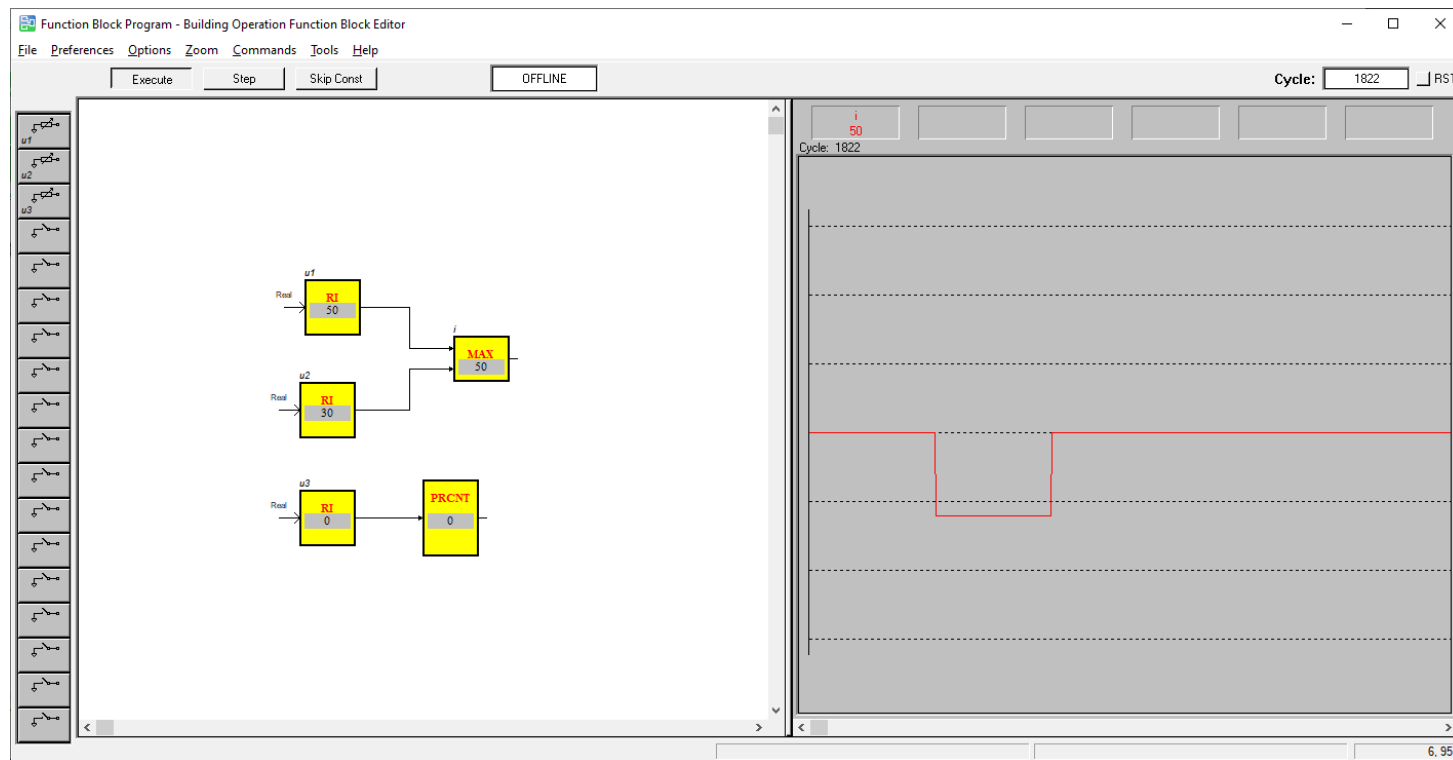
► Режим за измену програма



Режими рада (2/3)

► Режим за тестирање програма

► F12 или *options/simulate*



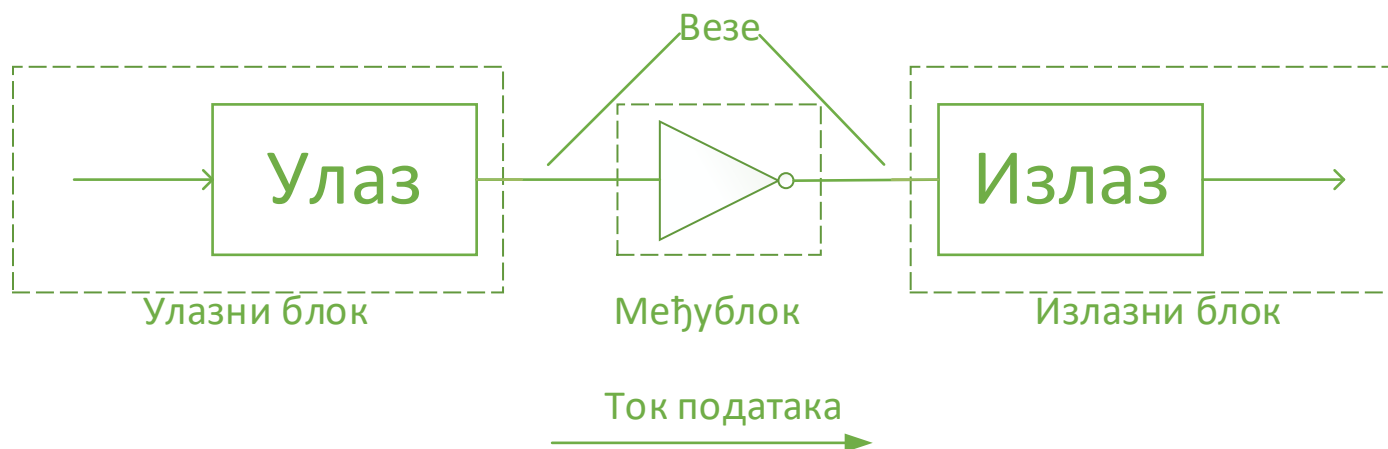
Режими рада (3/3)

- ▶ *Execute* (F2 или *commands/execute*) - укључује се извршавање програма
- ▶ *Step* (F3 или *commands/step*) - извршава се један програмски циклус
- ▶ *Edit* (F12 или *options/edit*) - Повратак на режим за измену програм

Увод у ФБД

ОСНОВЕ

- ▶ Функцијски блок дијаграм (ФБД) је графички програмски језик
- ▶ Постоје два основна елемента:
 - ▶ Везе - преносе податке (сигнале)
 - ▶ Функцијски блокови - врше обраду података (сигнала)



- ▶ Ток података је са леве на десну страну осим у случају повратне спреге
- ▶ У току сваког програмског циклуса промене на улазним блоковима пропадају ка излазним преко међублокова који одређују логику рада

Везе

► Постоје три типа везе:

- Целобројна: означени целобројни 16-битни број (опсег: од -32767 до 32767)
- Реална: означени реални 32-битни број (*IEEE* формат) прецизности у 6 децимала (опсег: од $3,4 \times 10^{-38}$ до $3,4 \times 10^{38}$)
- Бинарна: један бит представља бинарну вредност (0/1 = *FALSE/TRUE*)



Бинарна веза



Целобројна
или реална веза

- Аналогни сигнали се преносе целобројном или реалном везом
- Дигитални сигнали се преносе бинарном везом

Функцијски блокови

- ▶ Постоје четири типа функцијских блокова:
 - ▶ Стандардни блокови
 - ▶ Операторски блокови
 - ▶ Блокови за изразе
 - ▶ Хијерархијски блокови

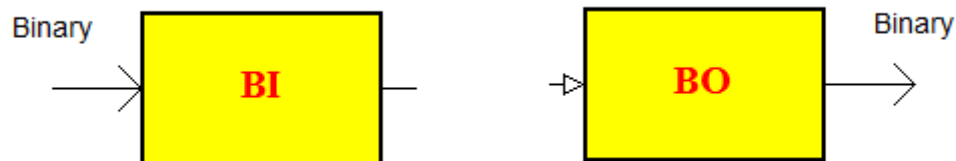
Функцијски блокови

Стандардни блокови (1/5)

► Могу се поделити у следеће групе:

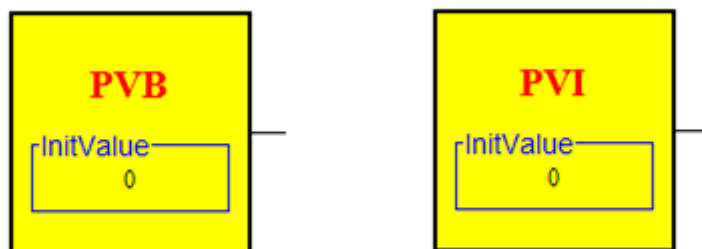
► Блокови за повезивање

- Бинарни улази и излази, реални улази и излази итд.



► Генератори сигнала

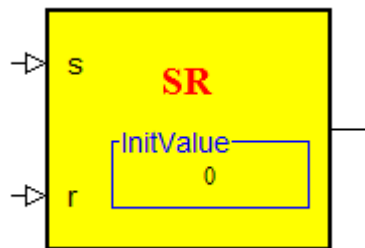
- Бинарни, целобројни и реални генератор вредности итд.



Стандардни блокови (2/5)

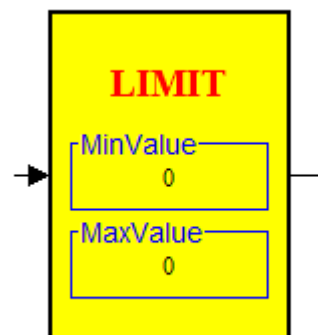
► Логичке функције

► Флип-флоп итд.



► Нелинеарне функције

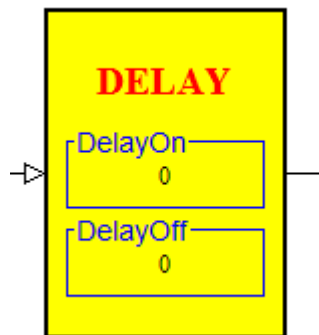
► минимум, максимум, ограничавач итд.



Стандардни блокови (3/5)

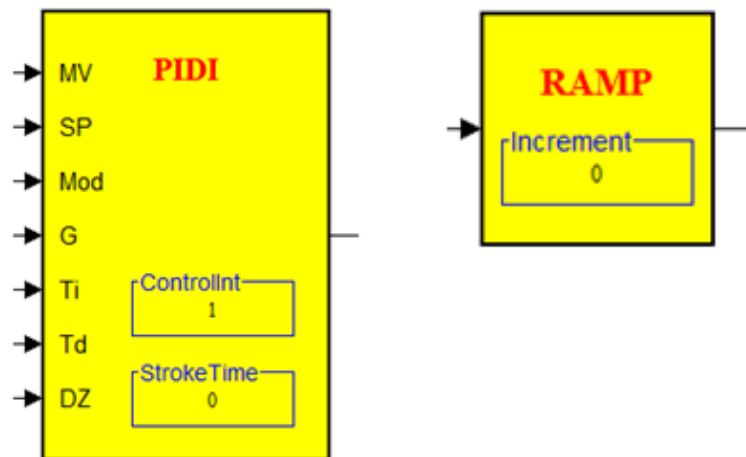
► Временска кашњења

- Кашњење бинарне, целобројне и реалне вредности итд.



► Регулатори и филтери

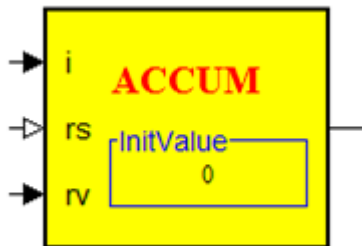
- *PID*, рампа филтер итд.



Стандардни блокови (4/5)

► Акумулатори

- Акумулатор, интегратор итд.



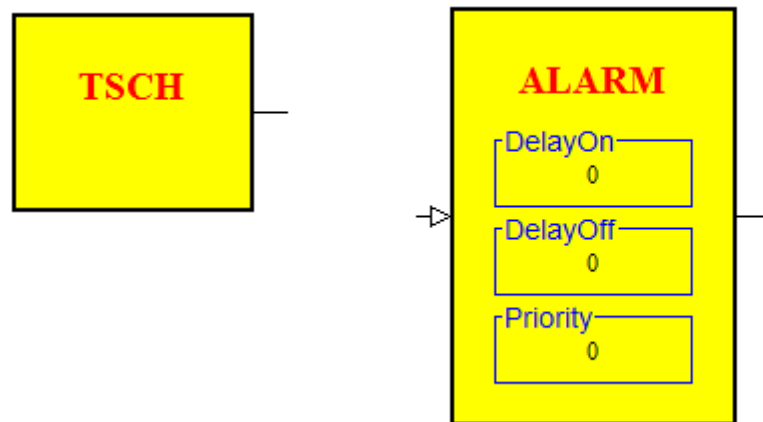
► Системске променљиве

- Датум, време, дужина циклуса итд.



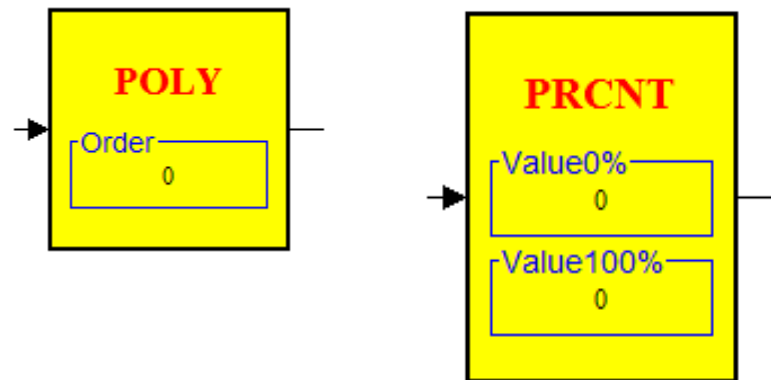
Стандардни блокови (5/5)

► Временски распореди и аларми



► Трансформационе функције

► Полином, линеарна трансформација...

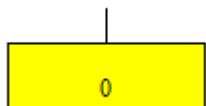


Операторски блокови (1/2)

► Могу се поделити у следеће групе:

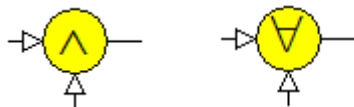
► Константе

- бинарна, целобројна, реална константа



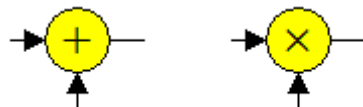
► Логички оператори

- И, ИЛИ, Екс ИЛИ итд.



► Аритметички оператори

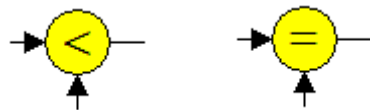
- сабирање, одузимање, множење итд.



Операторски блокови (2/2)

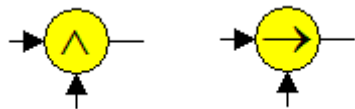
► Упоредни оператори

- мање, веће, једнако итд.



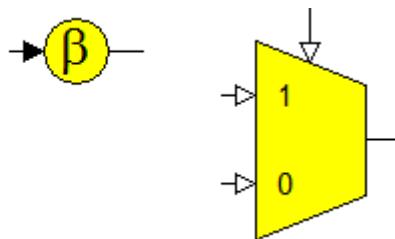
► Бит оператори

- И, ИЛИ, померање и итд.



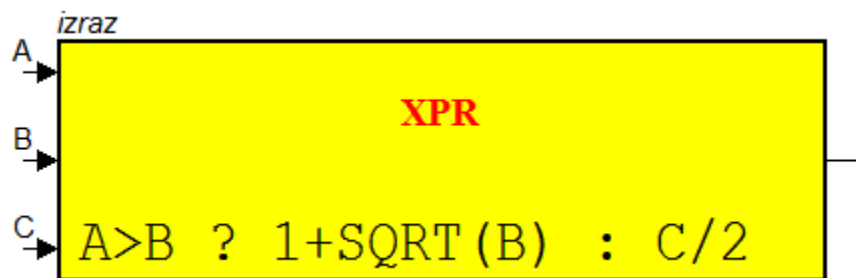
► Други оператори

- АД/ДА конвертори, мултиплексери итд.



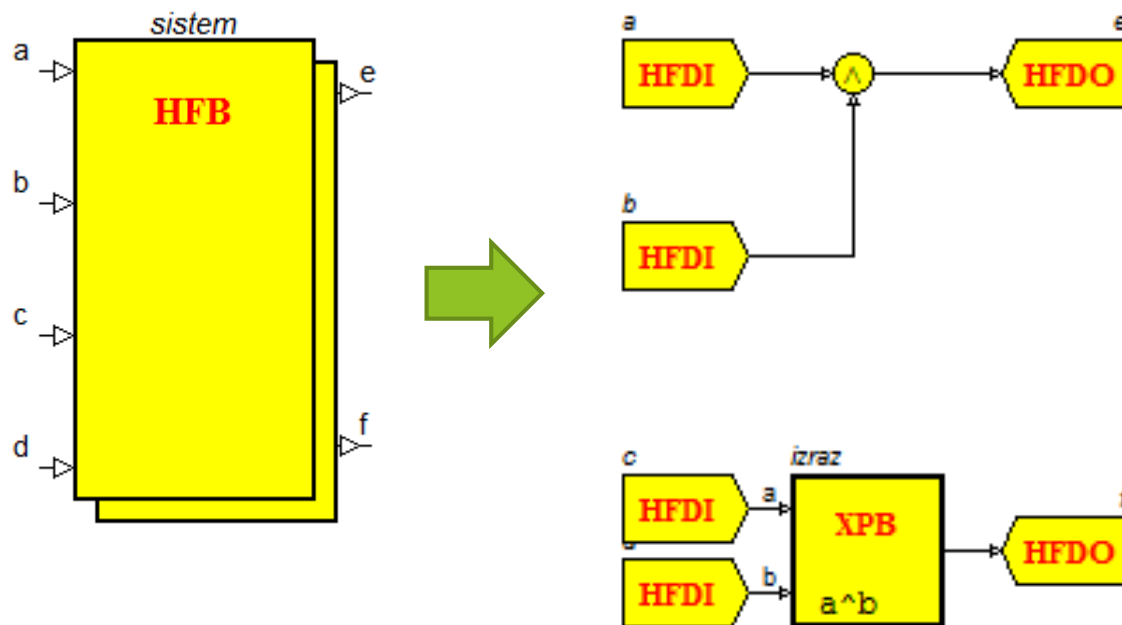
Блокови за изразе

- ▶ Произвољан израз типа:
 - ▶ $USLOV ? IZRAZ1 : IZRAZ2$



Хијерархијски блокови

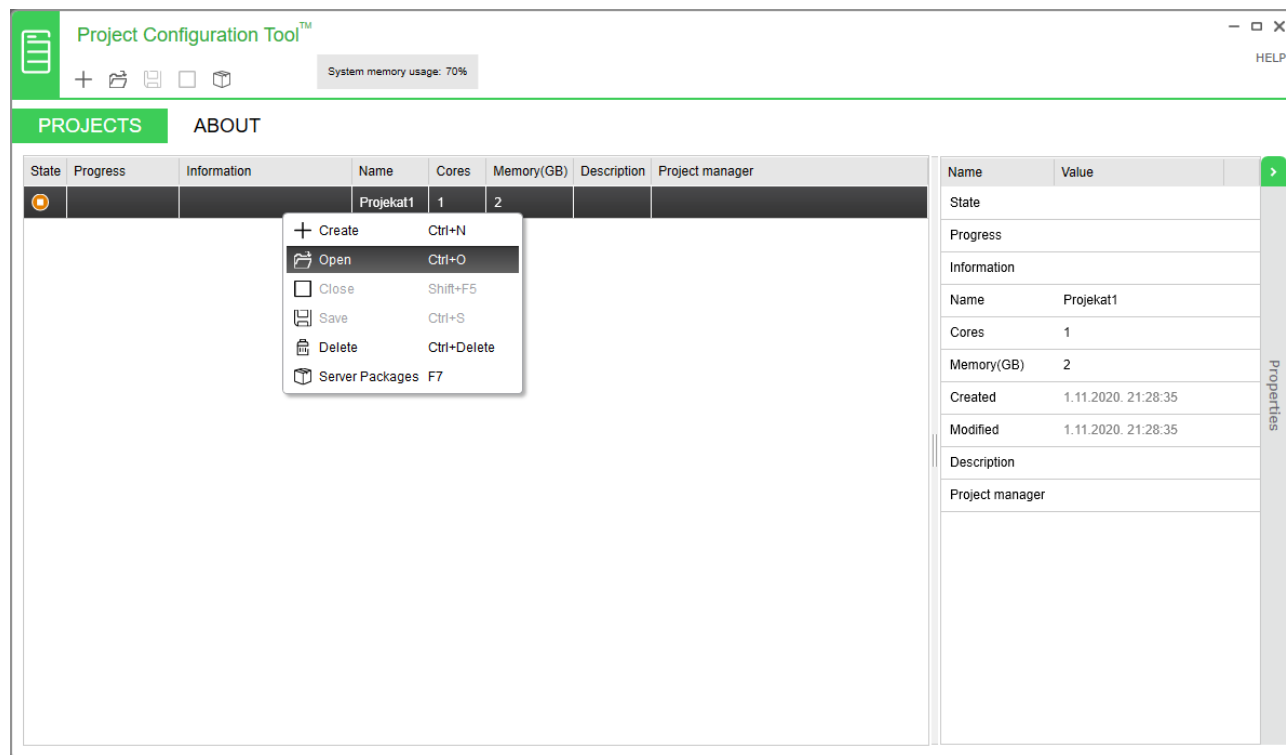
- Структурирана организација програма



Креирање програма

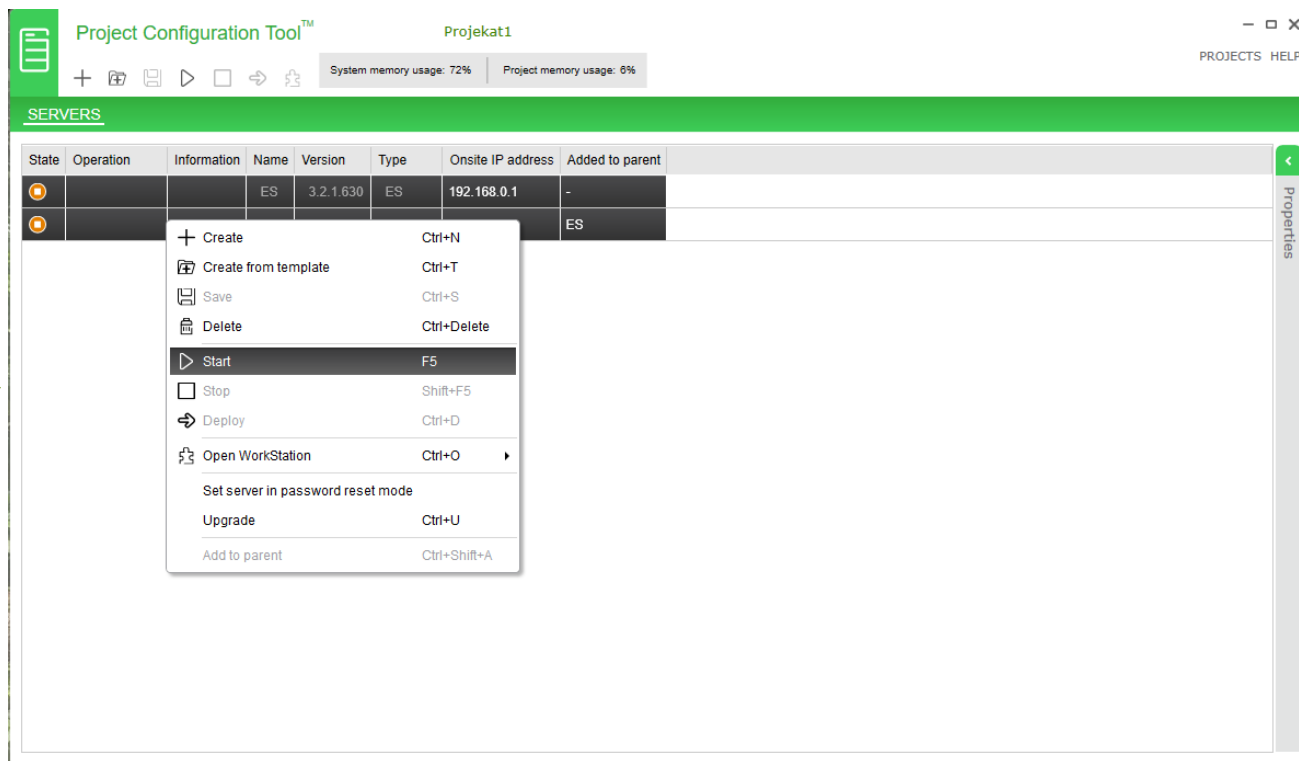
Креирање програма (1/9)

- ▶ Отворити пројекат Projekat1 кликом на иконицу за отварање пројекта која се налази у оквиру траке са алатима или кликом на десни тастер миша па Open



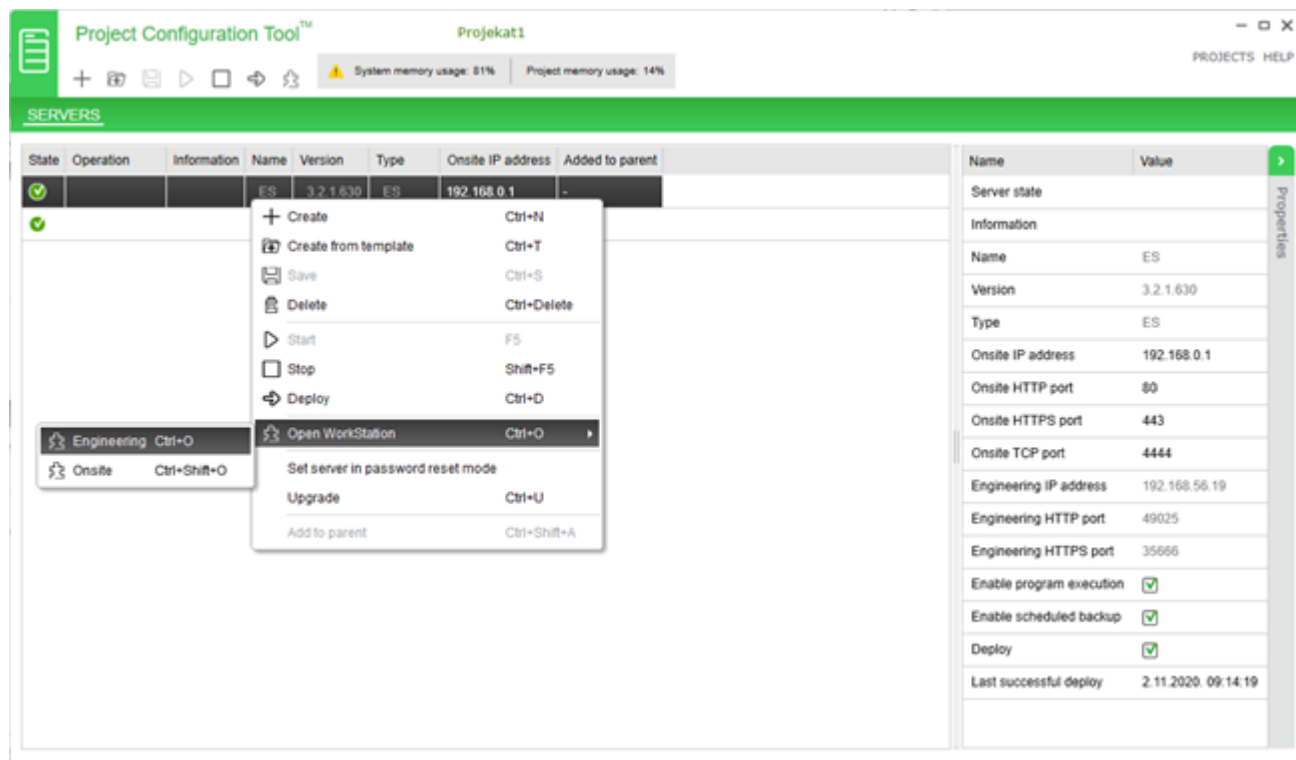
Креирање програма (2/9)

- Покренути оба сервера кликом на иконицу за покретање сервера која се налази у оквиру траке са алатима или кликом на десни тастер миша па Start



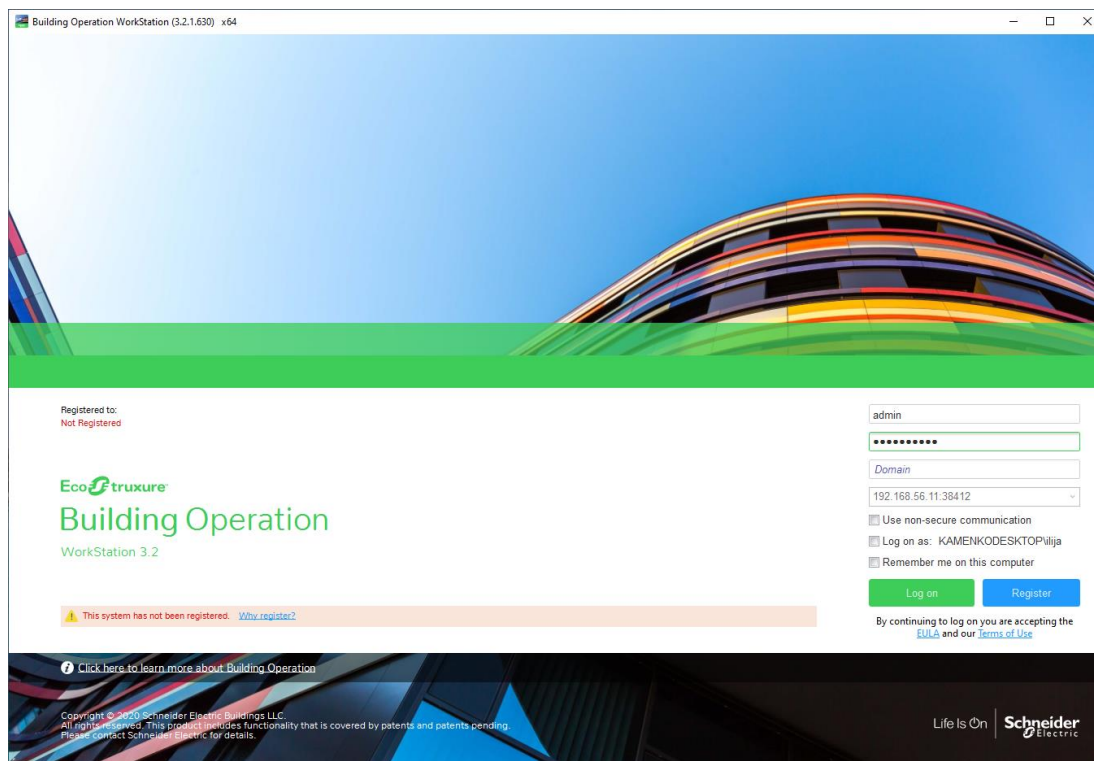
Креирање програма (3/9)

- Означити ES сервер па отворити Workstation кликом на иконицу за отварање која се налази у оквиру траке са алатима или кликом на десни тастер миша па Open Workstation/Engineering



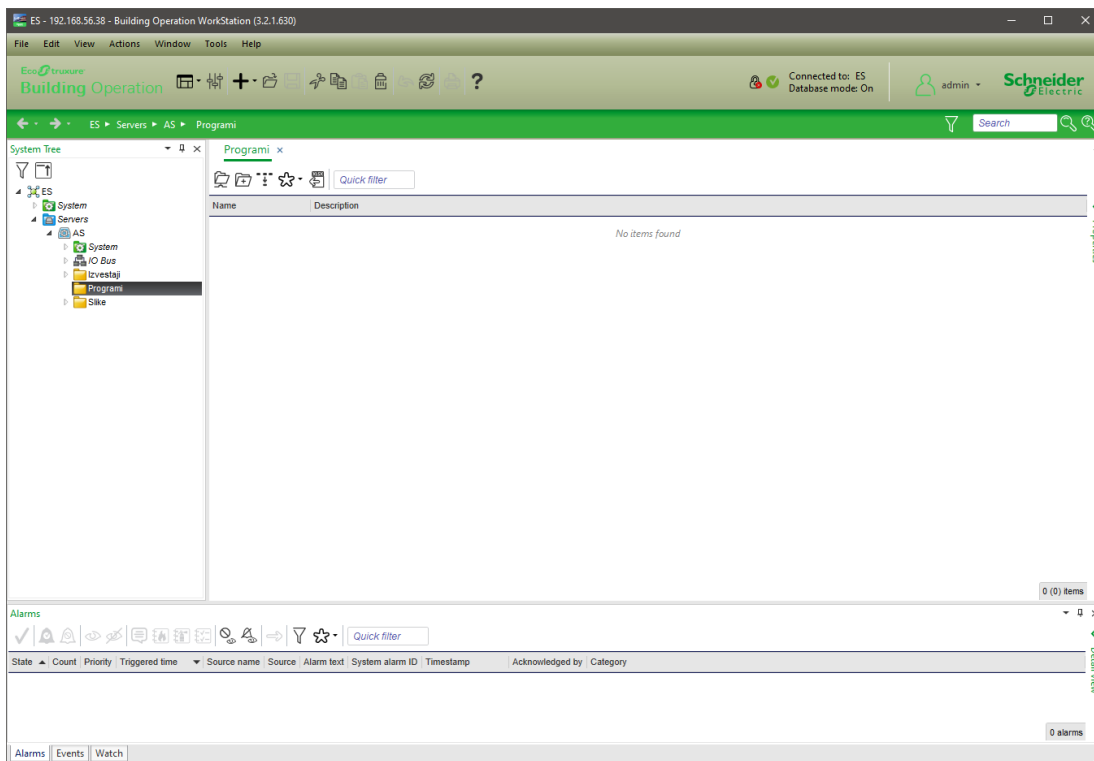
Креирање програма (4/9)

- ▶ Унети креденцијале подешене у току креирања ES сервера:
 - ▶ Корис. име: admin
 - ▶ Лозинка: Admin!2020



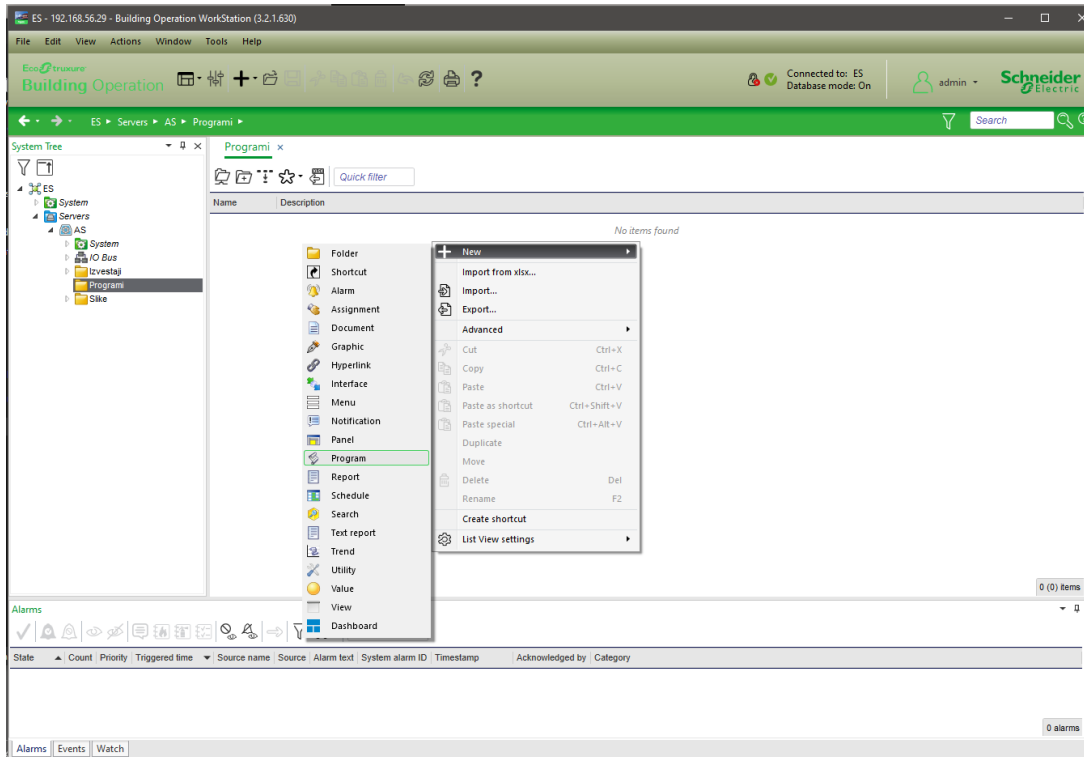
Креирање програма (5/9)

- Позиционирати се на фолдер Programi у оквиру AS сервера



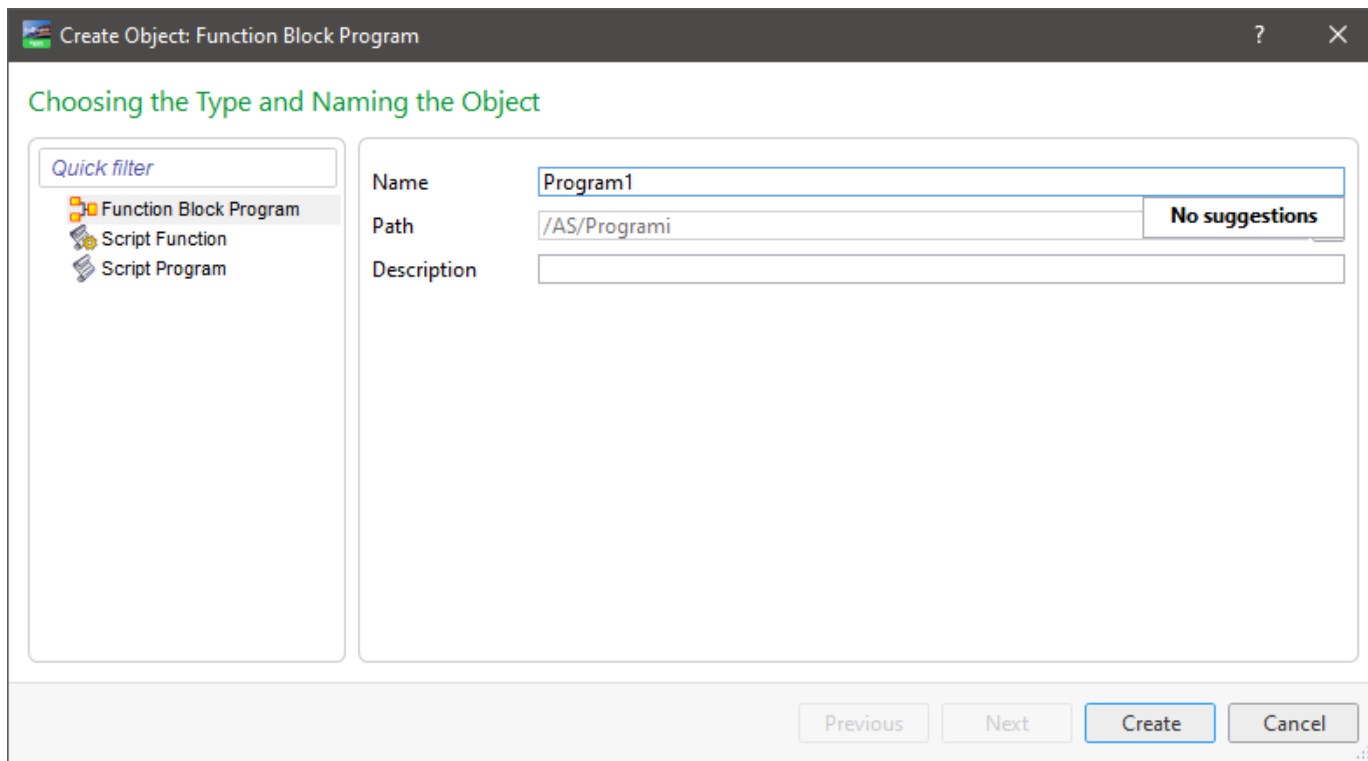
Креирање програма (6/9)

- Програм се креира кликом на десни тастер миша па New/Program



Креирање програма (7/9)

- За назив програма унети Program1



Create Object: Function Block Program

Choosing the Type and Naming the Object

Quick filter

- Function Block Program
- Script Function
- Script Program

Name: Program1

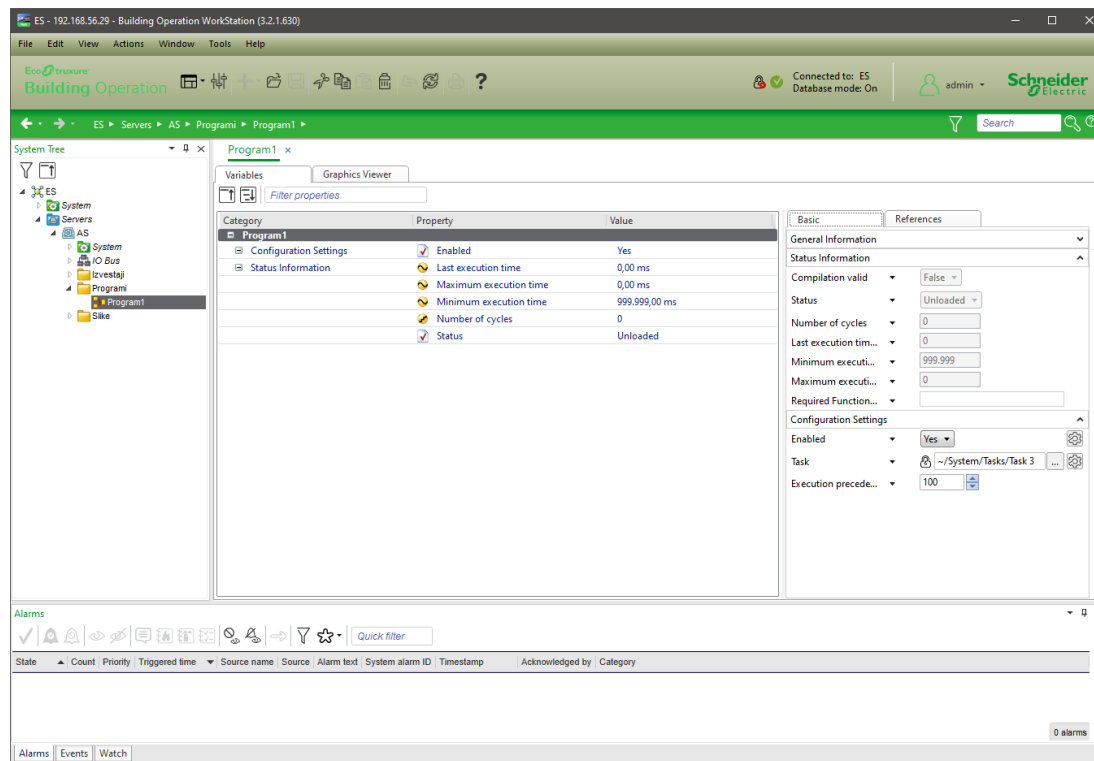
Path: /AS/Program1 No suggestions

Description:

Previous Next Create Cancel

Креирање програма (8/9)

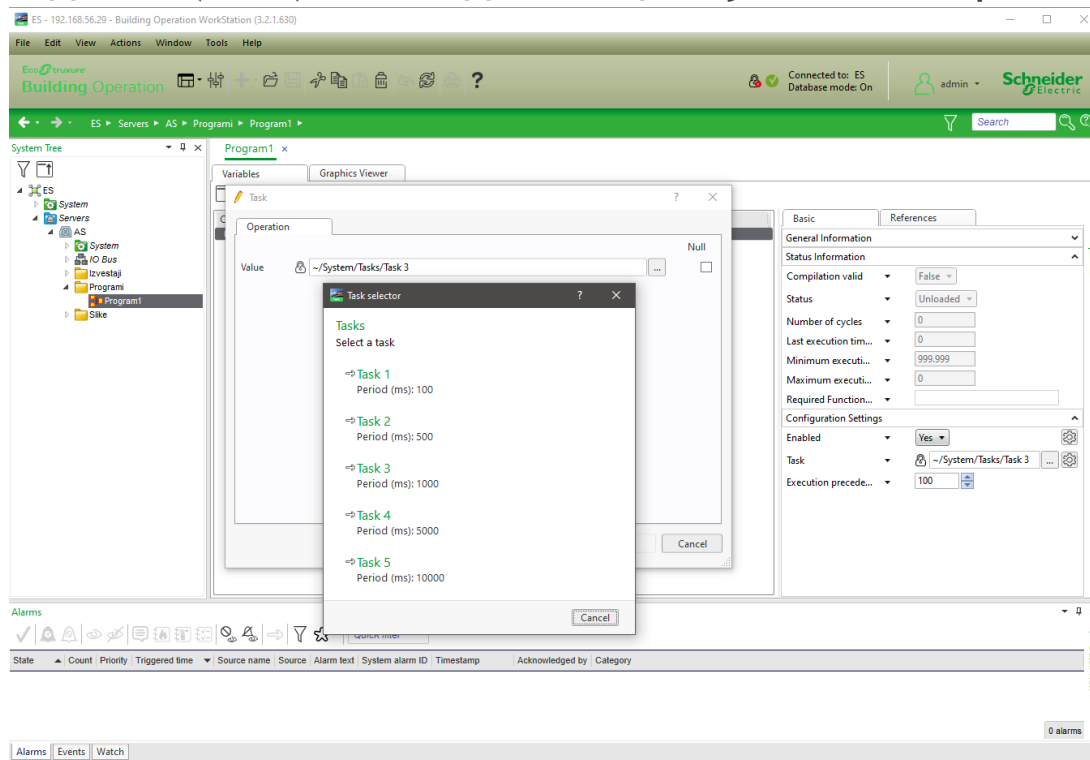
- ▶ На крају добијамо креиран програм



Креирање програма (9/9)

► Могуће је изабрати задатак (task) са следећим циклусом скенирања:

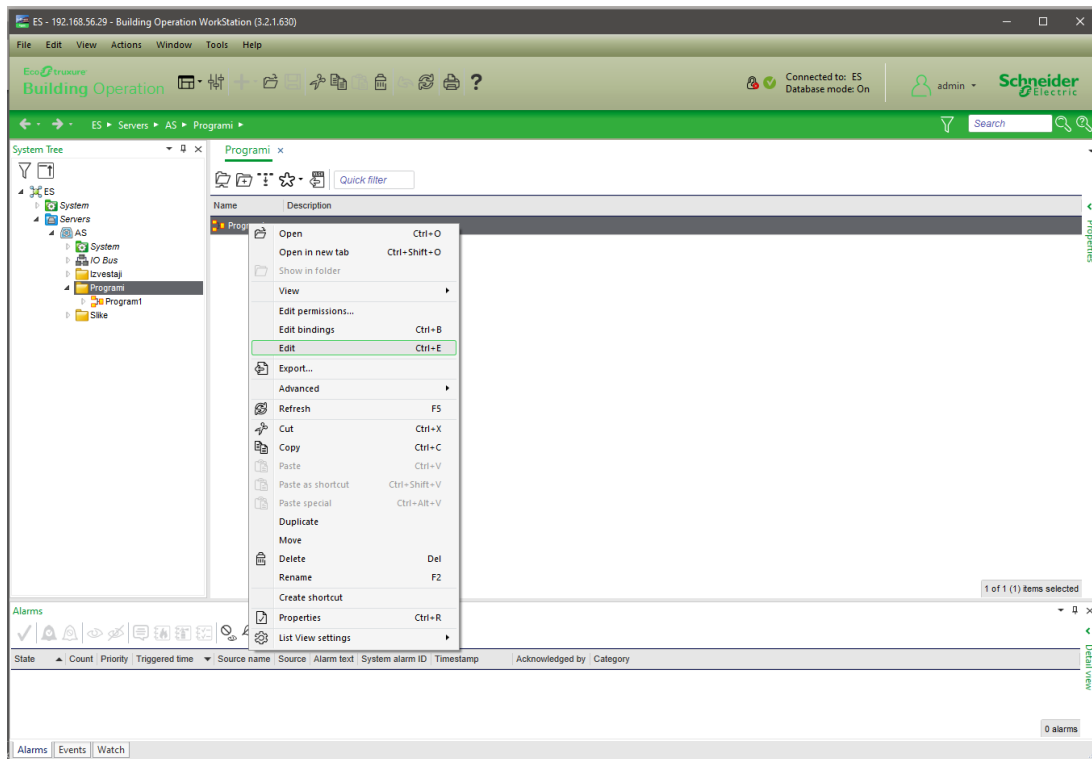
- Task 1 - 100ms
- Task 2 - 500ms
- Task 3 - 1000ms
- Task 4 - 5000ms
- Task 5 - 10000ms



- Програм је подразумевано повезан са задатком (task) број 3 са 1000s циклусом скенирања
- Подразумеван приоритет извршавања је 100

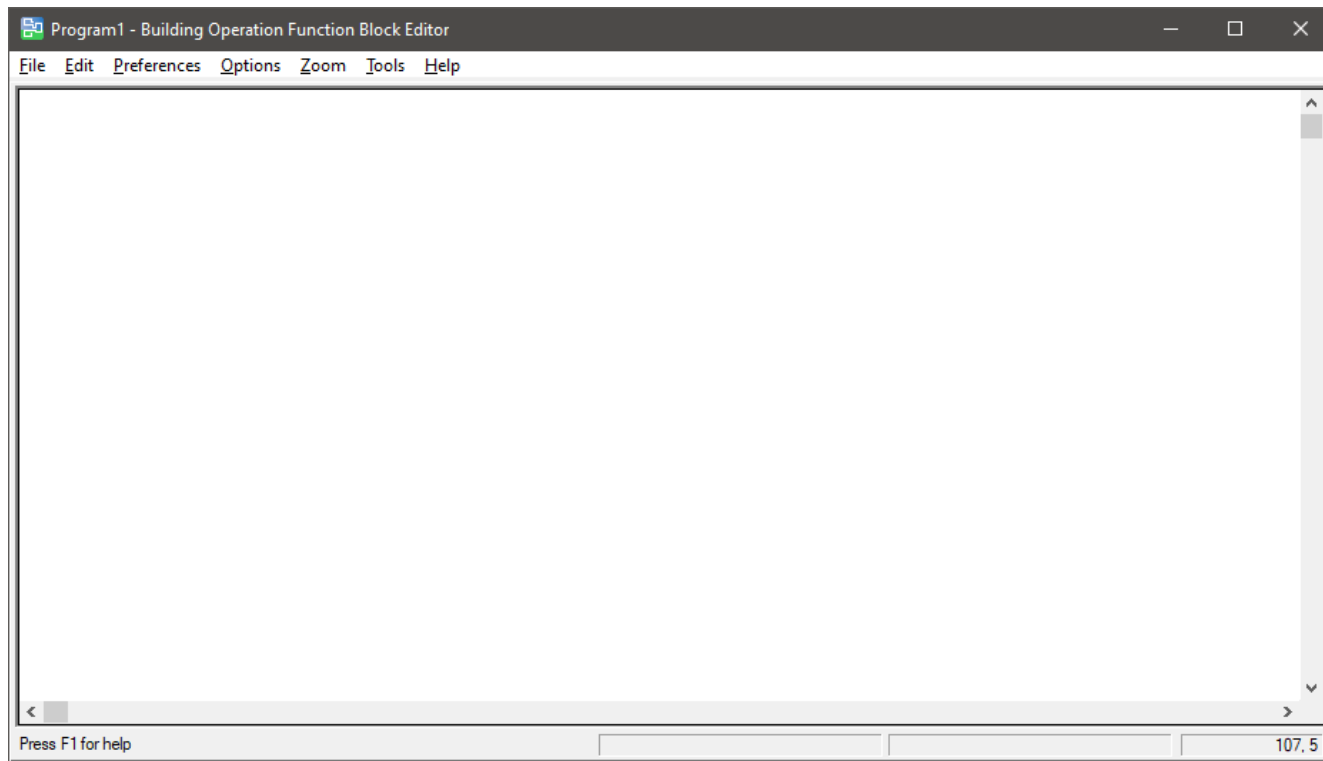
Отварање програма (1/2)

- Програм се отвара у режиму измени кликом на десни тастер миша па Edit



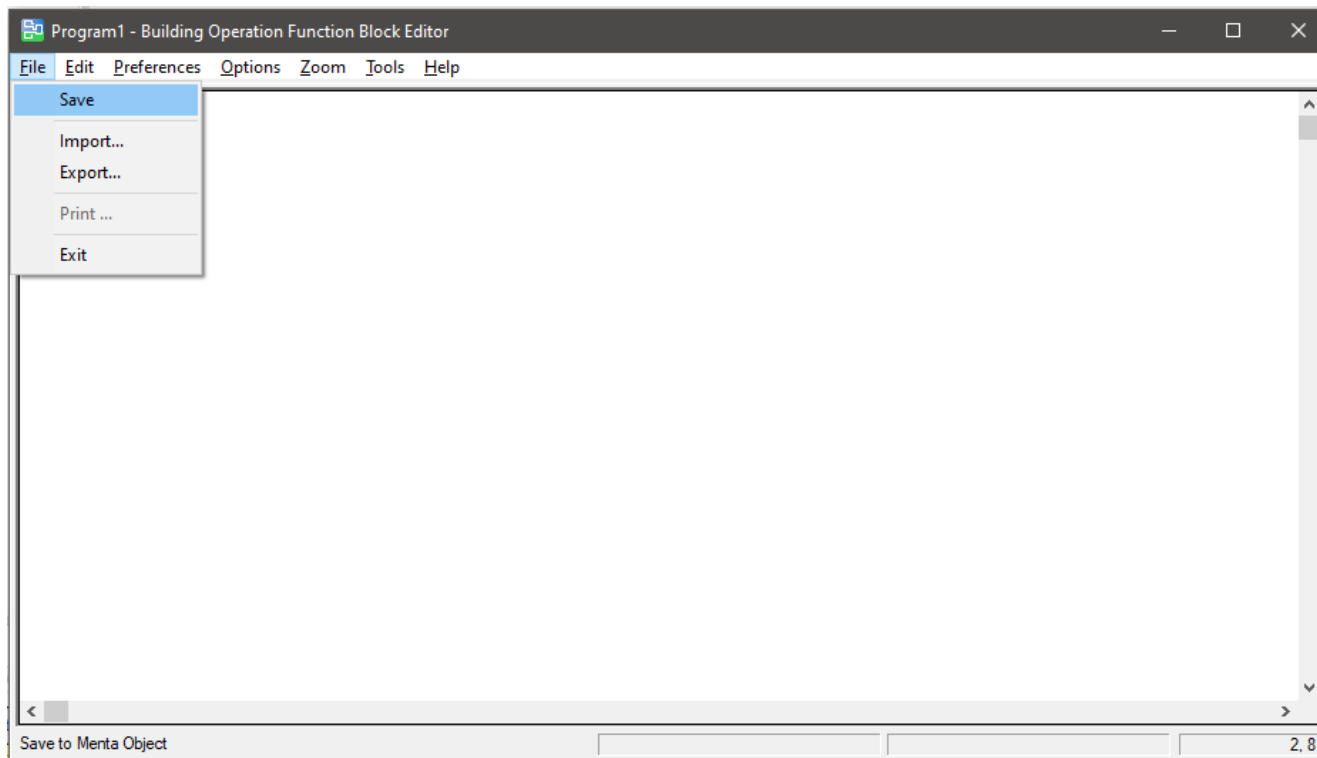
Отварање програма (2/2)

- Програм у режиму измене (Function Block editor)



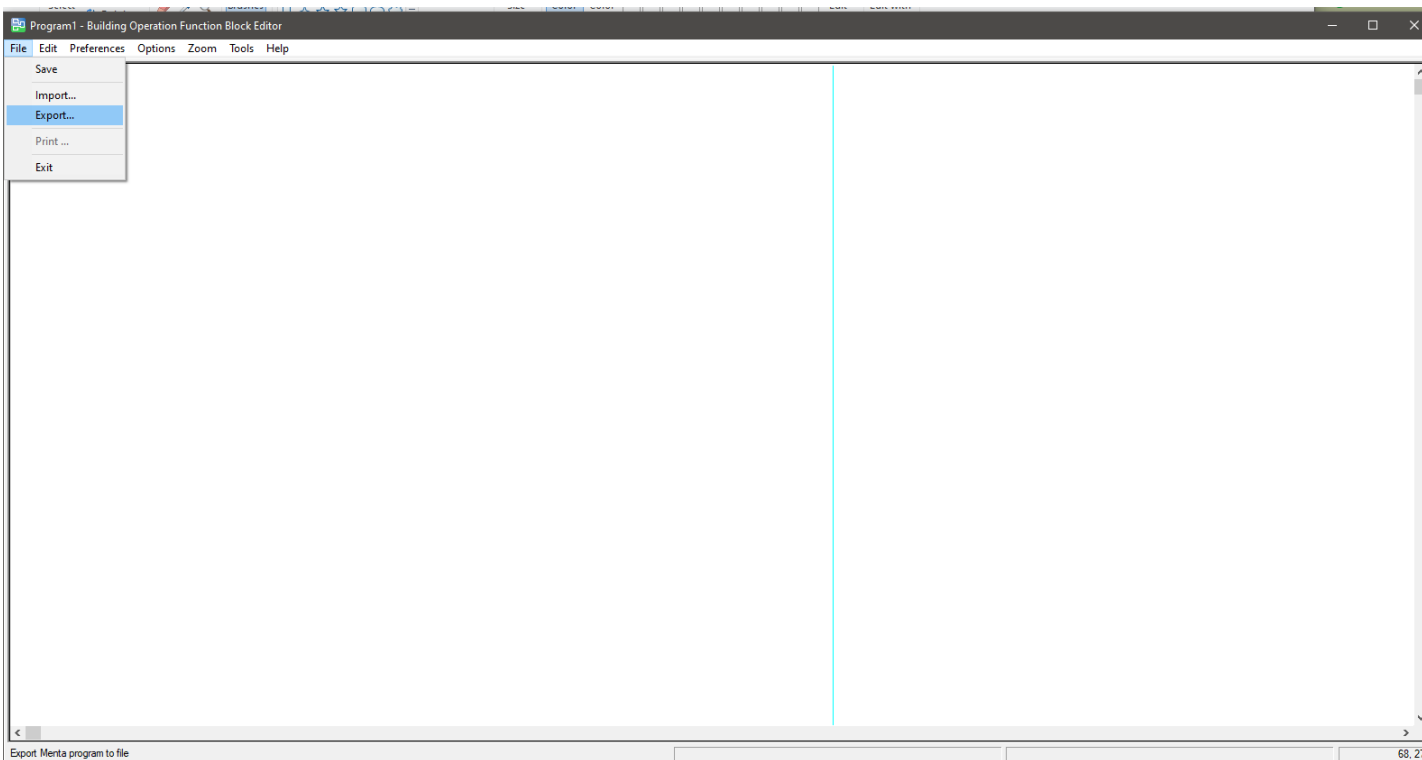
Затварање програма

- Измене се могу сачувати избор File/Save



Креирање резервне копије програма

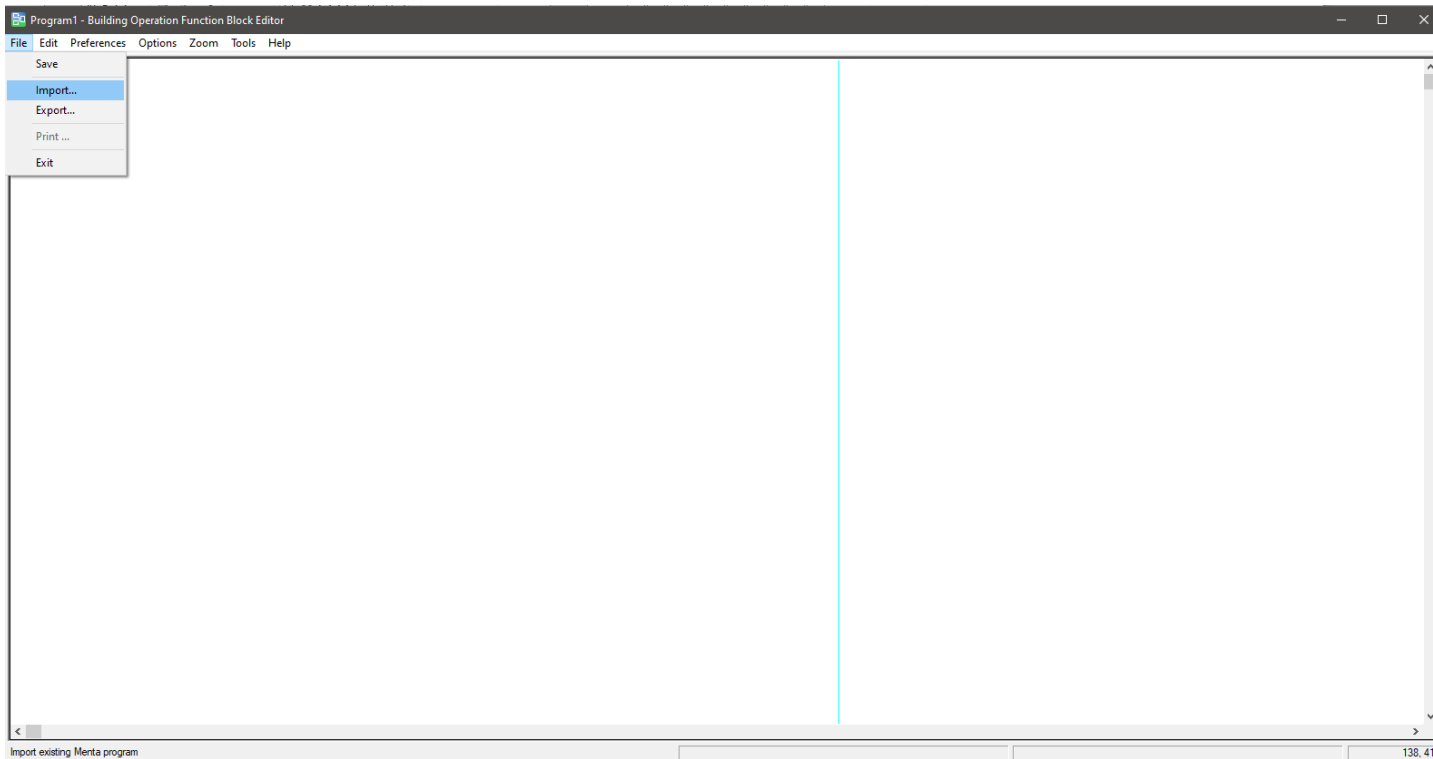
- Резервна копија програма се може креирати кликом на File/Export...



- Програм се на овај начин чува у посебној датотеци са екстензијом .aut и може се независно уређивати

Враћање резервне копије програма

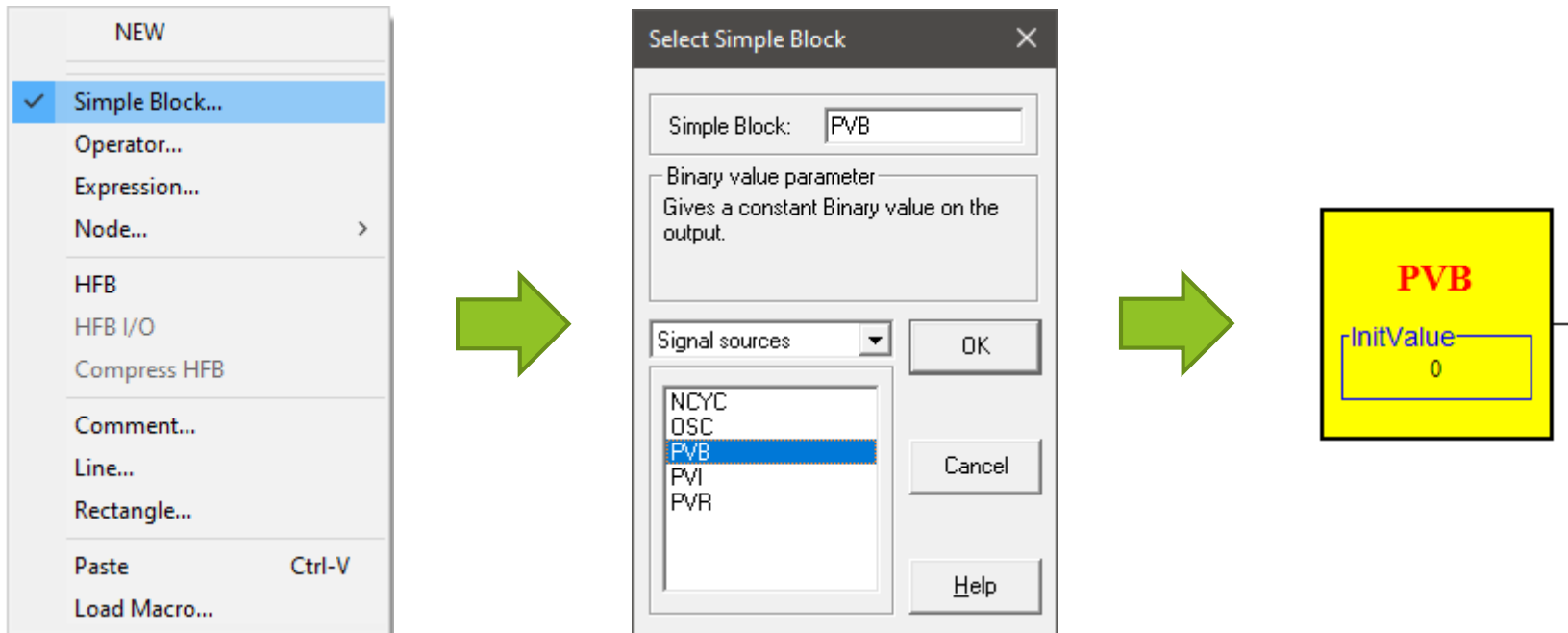
- ▶ Резервна копија програма се може вратити кликом на File/Import...
- ▶ Потребно је селектовати датотеку са екстензијом .aut у којој се налази резервна копија



Логичке операције

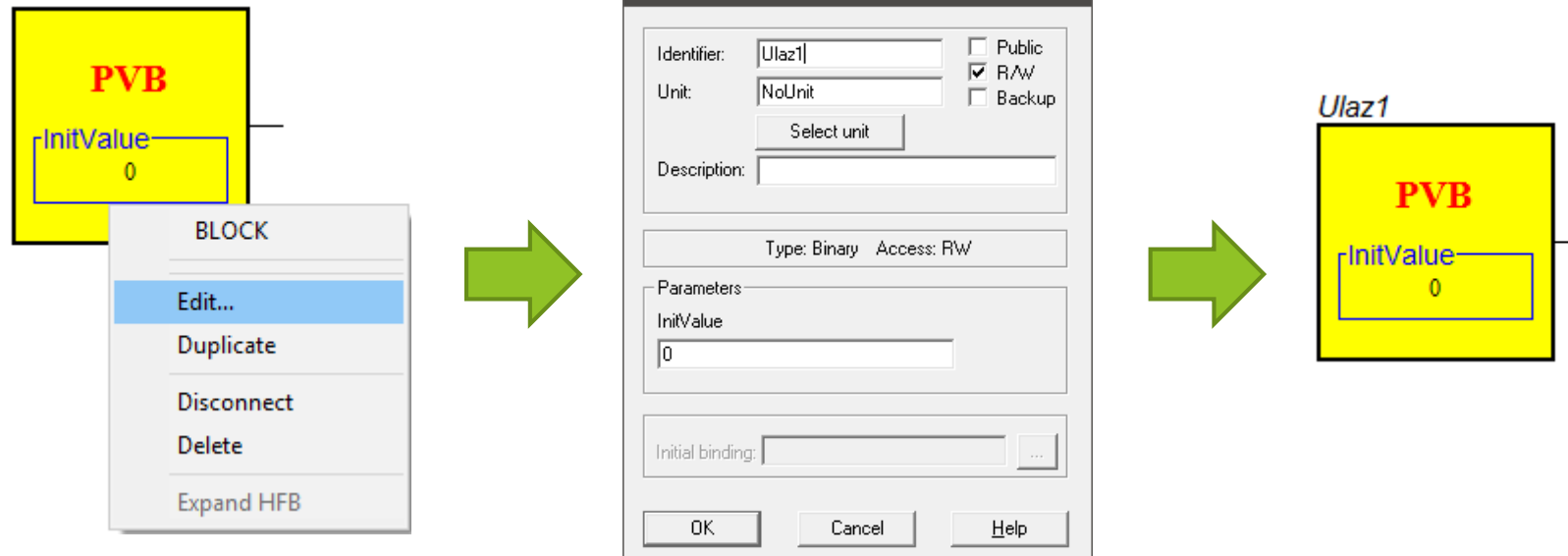
Додавање бинарних константи (1/2)

- Додати константу бинарног типа

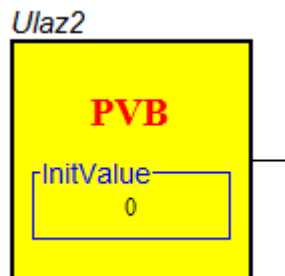


Додавање бинарних константи (2/2)

- Унети за назив блока 'Ulaz1'

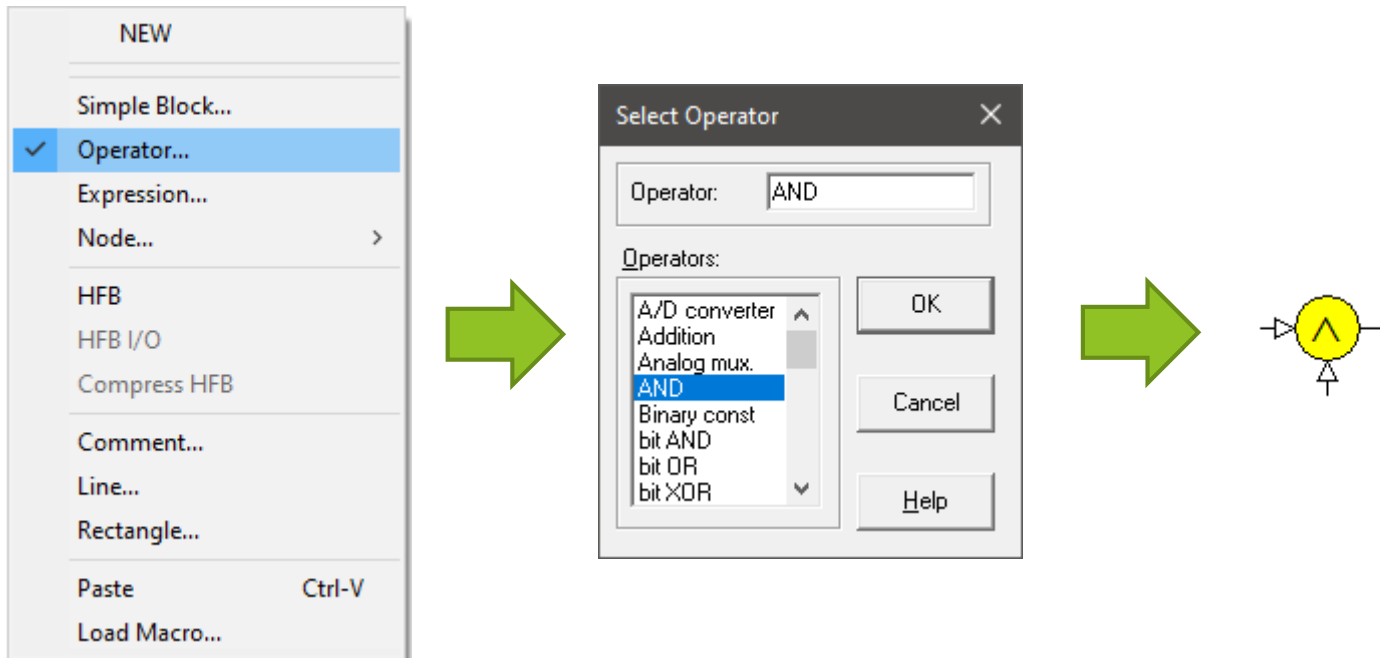


- Додати још једну константу бинарног типа и за назив блока унети 'Ulaz2'



Додавање логичких оператора

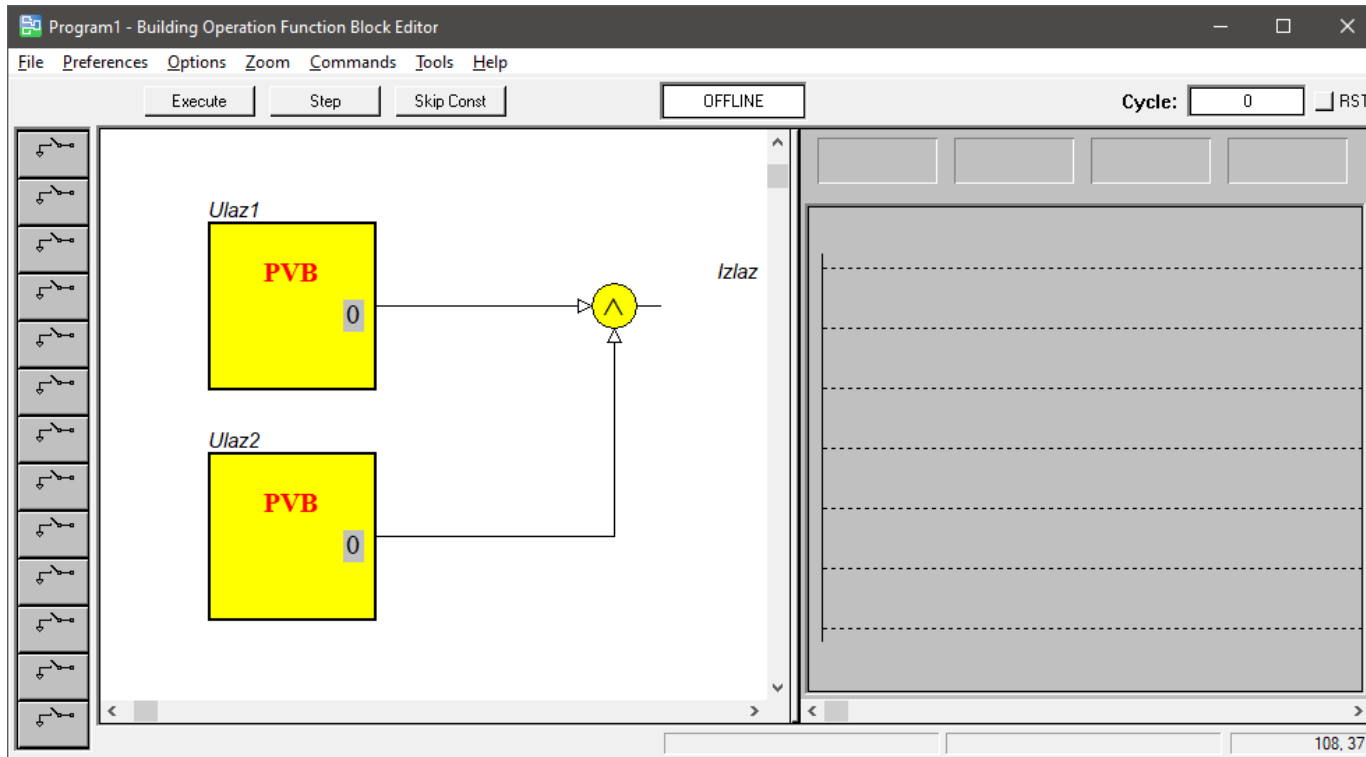
- Додати оператор логичке операције И (AND)



- За назив блока унети 'Izlaz'

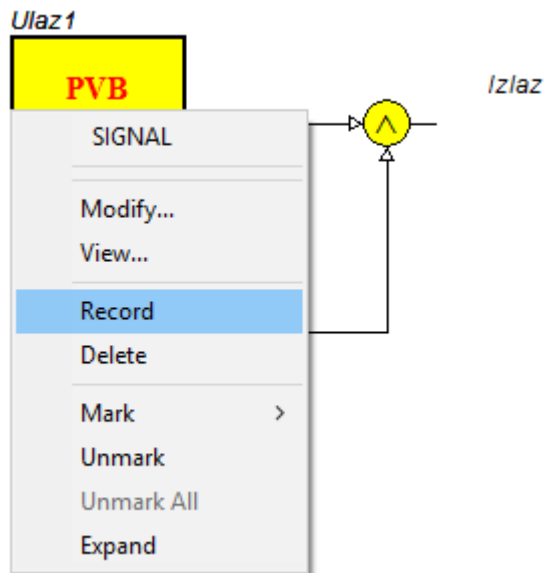
Тестирање и симулација (1/3)

- Повезати блокове и укључити режим за симулацију (F12)



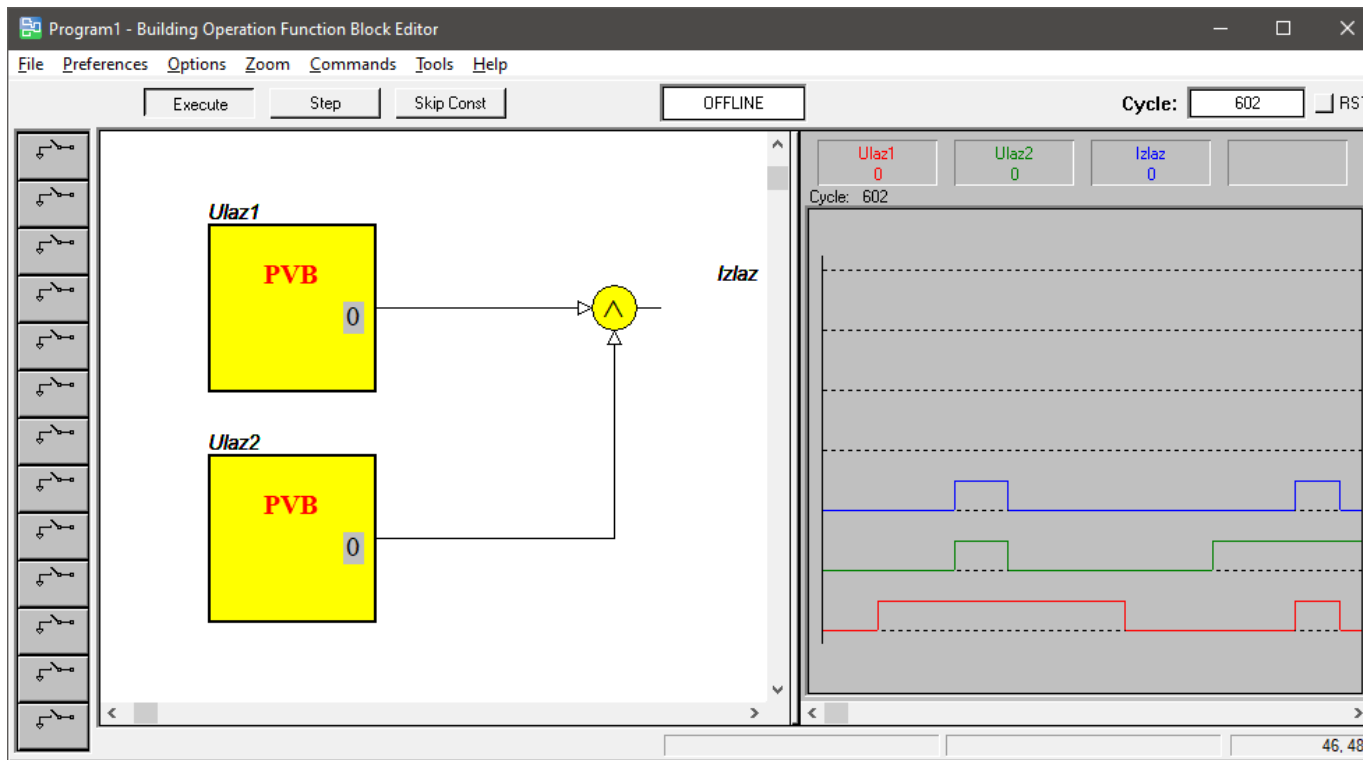
Тестирање и симулација (2/3)

- Додати графике за посматрање величина у реалном времену за све три величине



Тестирање и симулација (3/3)

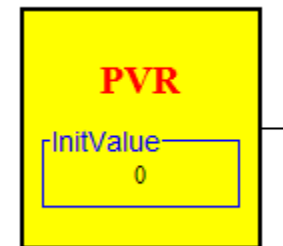
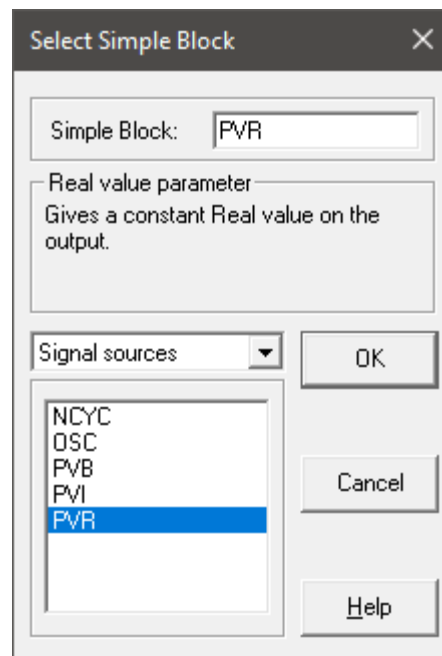
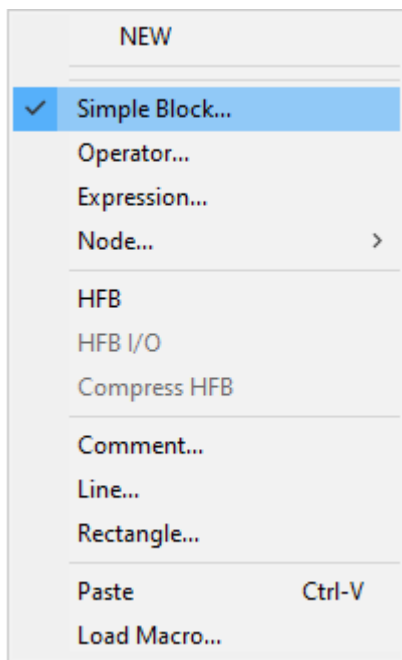
- Покренути извршавање програма (*Execute*)



Аритметичке операције

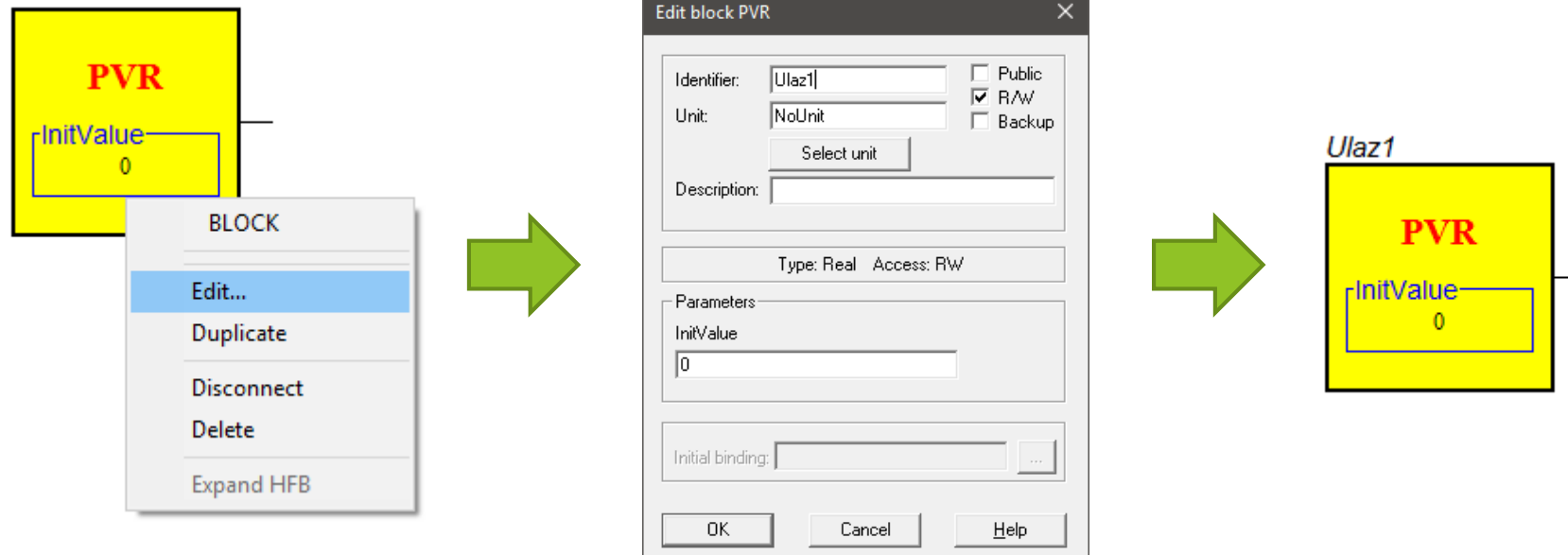
Додавање реалних константи (1/2)

- Додати константу реалног типа

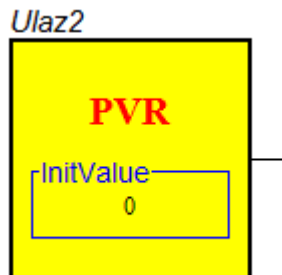


Додавање реалних константи (2/2)

- Унети за назив блока 'Ulaz1'

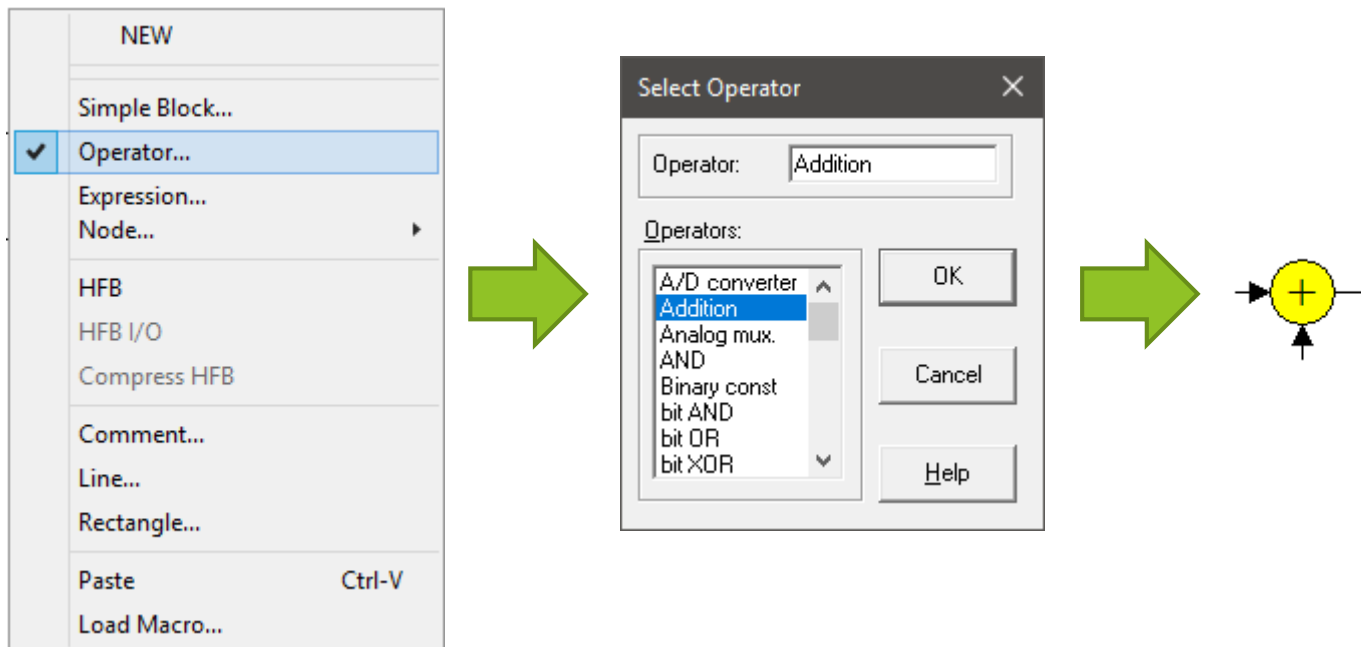


- Додати још једну константу реалног типа и за назив блока унети 'Ulaz2'



Додавање аритметичких оператора

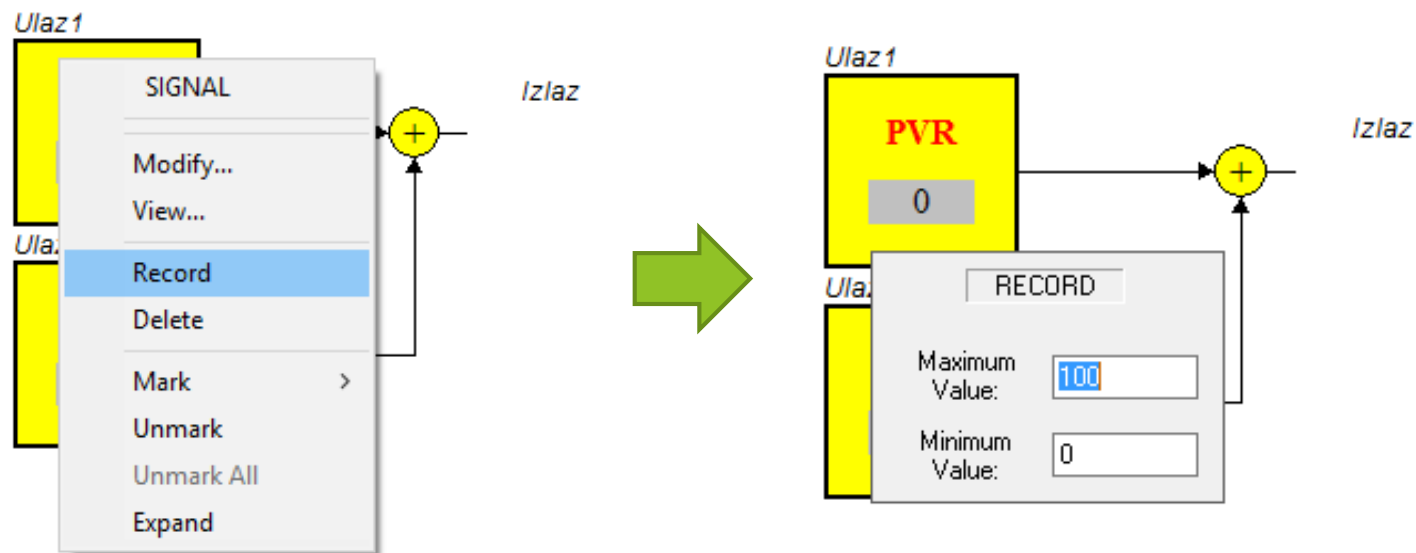
- Додати оператор операције сабирања (*Addition*)



- За назив блока унети 'Izlaz'

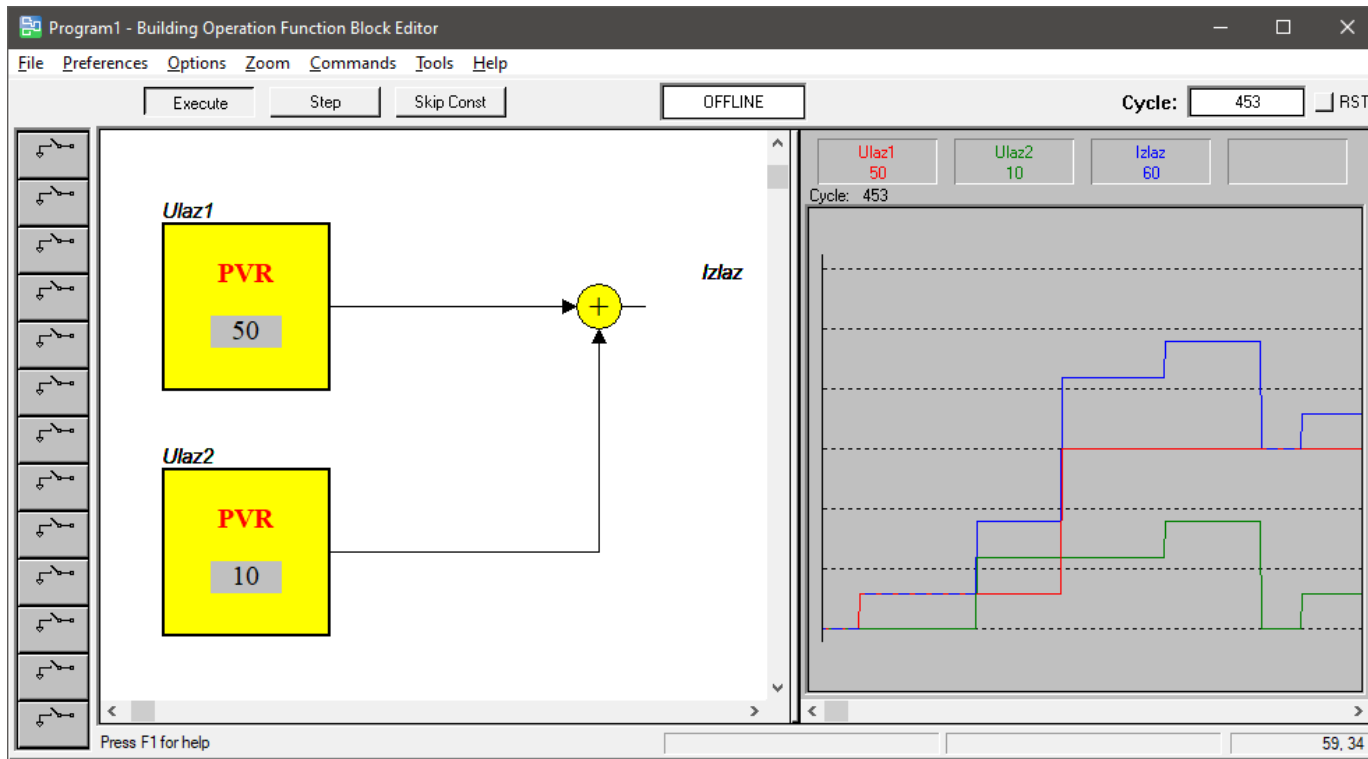
Тестирање и симулација (1/2)

- Повезати блокове и укључити режим за симулацију
- Додати графике за посматрање величина у реалном времену за све три величине нпр. са ограничењем од 0 до 100



Тестирање и симулација (2/2)

► Покренути извршавање програма



Блок за изразе

Опис (1/2)

- ▶ Произвољан израз типа:
 - ▶ USLOV ? IZRAZ1 : IZRAZ2
- ▶ Претходни израз се може описати псеудо кодом

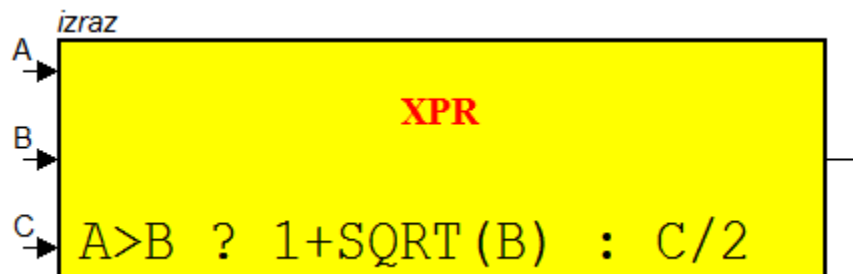
IF USLOV=**TRUE** THEN

IZRAZ1

ELSE

IZRAZ2

END

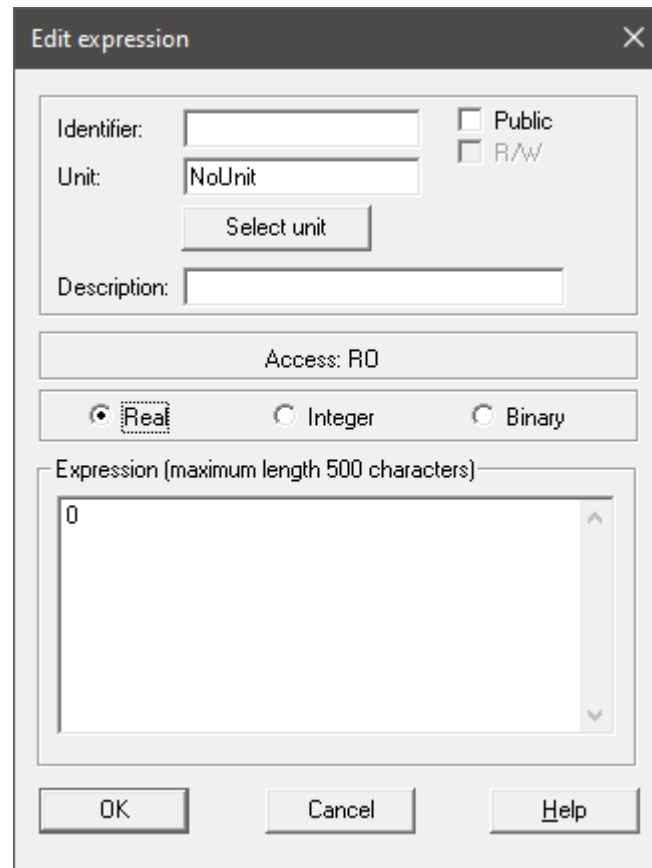
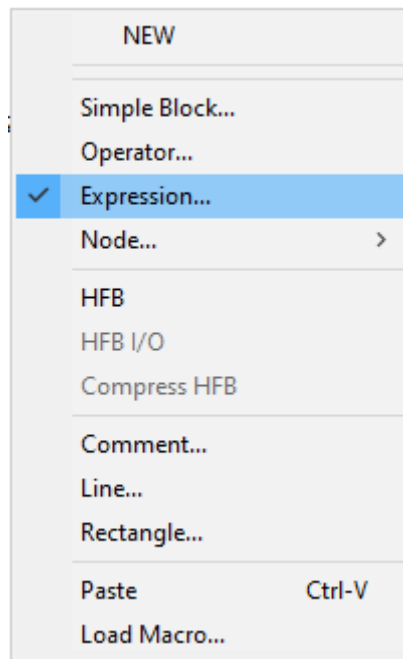


Опис (2/2)

- ▶ Делови блока за изразе:
 - ▶ Операнди
 - ▶ Улази, константе итд. (A, B, C - реални улази)
 - ▶ Оператори
 - ▶ Логички и аритметички (>, +, /)
 - ▶ IF-THEN-ELSE (? :)
 - ▶ Аритметичке функције
 - ▶ синус, косинус, логаритам итд. (SQRT)
 - ▶ Излази
 - ▶ Бинарни, целобројни, реални (XPR - реални излаз)

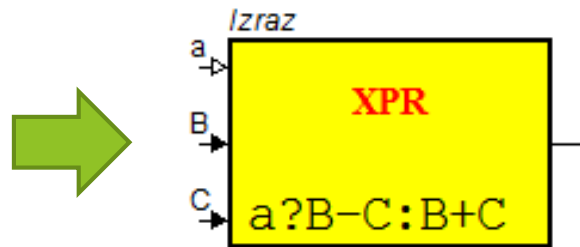
Додавање блока за изразе (1/2)

- ▶ Додати израз чији ће резултат бити збир односно разлика два улаза у зависности од вредности трећег улаза
- ▶ Додати блок за изразе



Додавање блока за изразе (2/2)

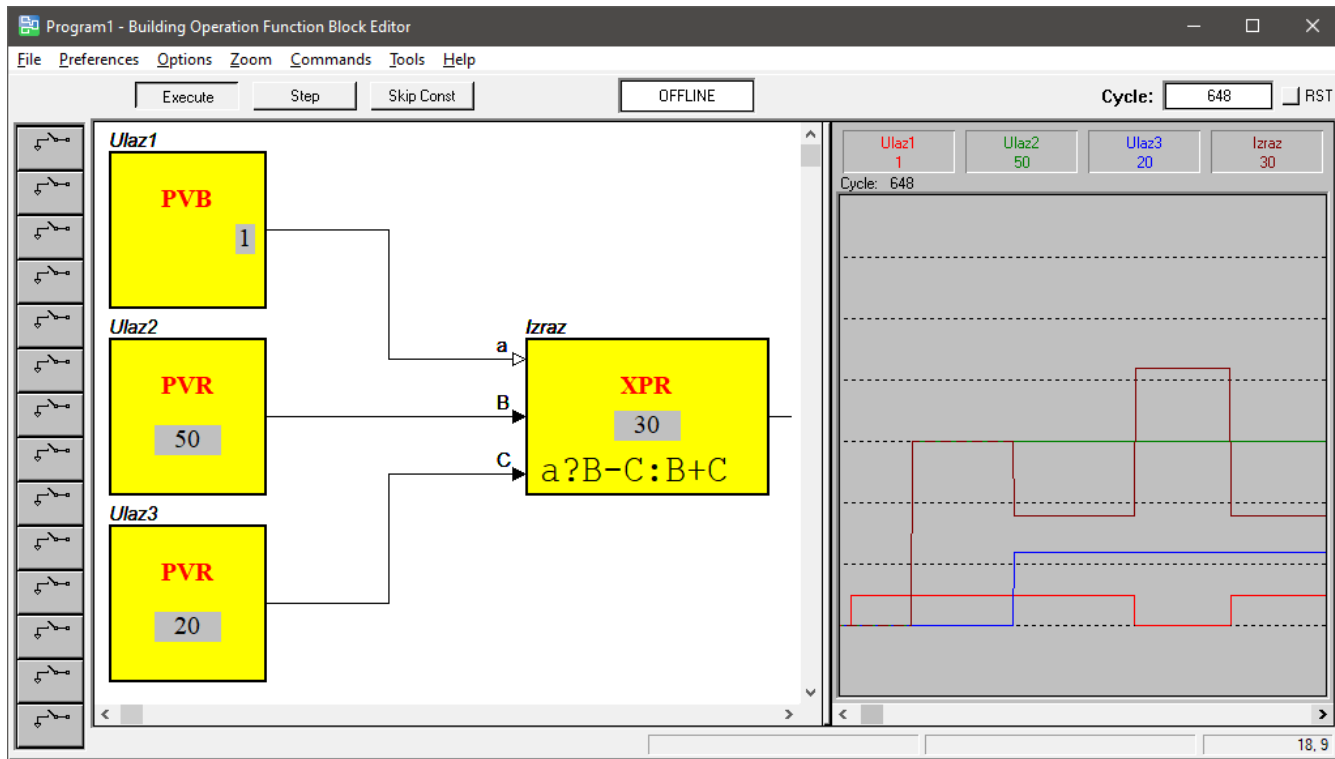
- За назив блока унети 'Izraz'
- За излаз блока изабрати реални излаз (*Real*)
- У пољу за израз унети 'a?B-C:B+C'
 - Мала слова означавају бинарне, а велика слова целобројне и реалне вредности



- Додати бинарну константу *Ulaz1* и две реалне константе *Ulaz2* и *Ulaz3*

Тестирање и симулација

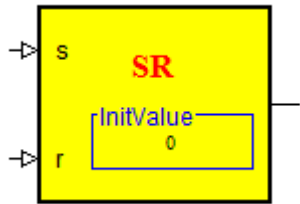
► Покренути извршавање програма



Логичке функције

SR флип-фоп (1/3)

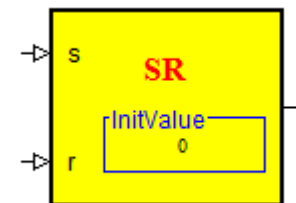
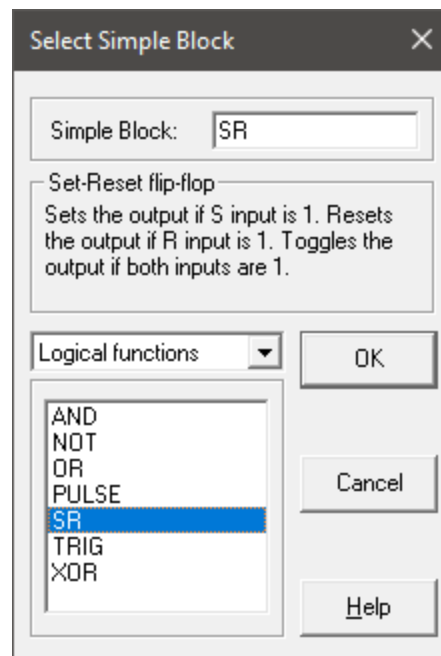
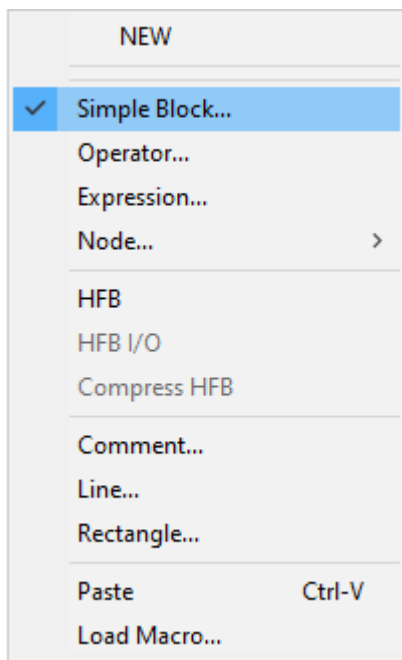
- SR флип-фоп је бистабилни елемент тј. поседује два стабилна стања излаза (логичка 0 и логичка 1)



- Растућа ивица сигнала на S улазу поставља вредност излаза из блока на логичку јединцу
- Даље промене сигнала на S улазу не утичу на стање излаза које остаје логичка јединца
- Растућа ивица сигнала на R улазу поставља вредност излаза из блока на логичку нулу.
- Даље промене сигнала на R улазу не утичу на стање излаза које остаје логичка нула
- Када истовремено дођу растуће ивице сигнала на оба улаза, стање излаза је недефинисано тј. наизменично прелази из једног у друго стање

SR флип-фоп (2/3)

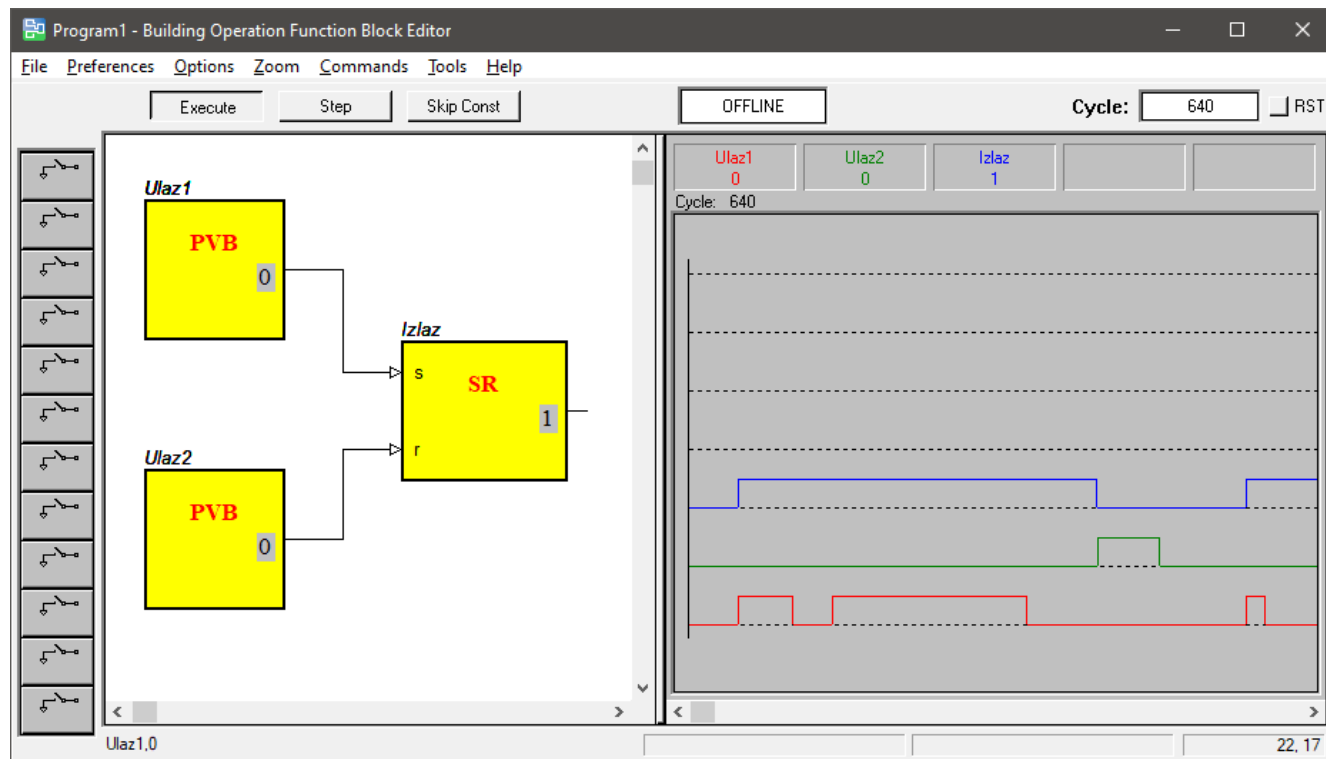
- Додати SR флип-фоп



- За назив блока унети 'Izlaz'
- Додати бинарну константу и за назив блока унети 'Ulaz1'
- Додати бинарну константу и за назив блока унети 'Ulaz2'

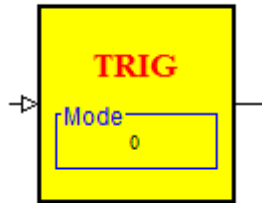
SR флип-фоп (3/3)

► Тестирање и симулација



Детекција ивице сигнала (1/3)

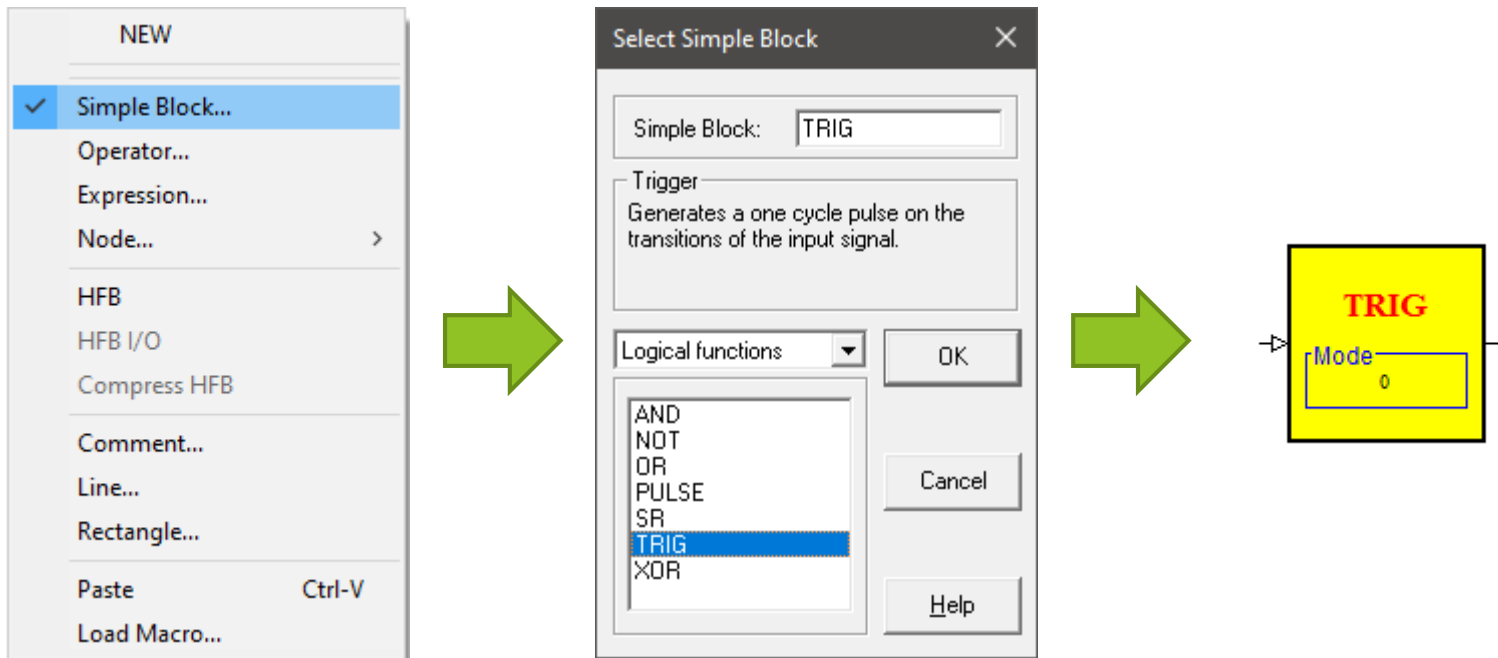
- ▶ *TRIG* блок врши детекцију примене сигнала са улаза



- ▶ Ако се промена десила излаз блока постаје логичка јединица у трајању од једног програмског циклуса
- ▶ Поседује следеће режиме рада (*Mode*)
 - ▶ 0 и 1: растућа ивица сигнала (0->1)
 - ▶ 2: опадајућа ивица сигнала (1->0)
 - ▶ 3: растућа и опадајућа ивица сигнала (1->0 и 0->1)

Детекција ивице сигнала (2/3)

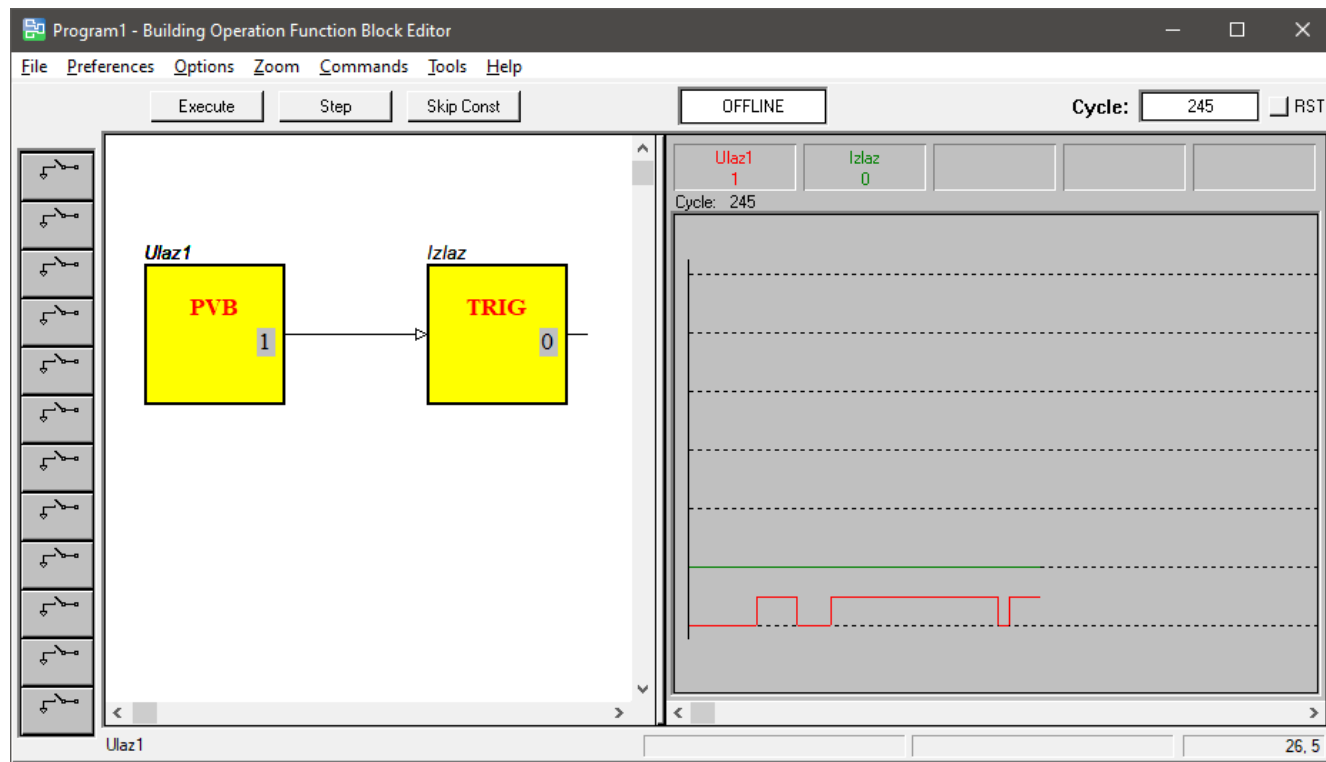
- Додати блок за детекцију сигнала (*TRIG*)



- За назив блока унети 'Izlaz'
- Додати бинарну константу и за назив блока унети 'Ulaz'

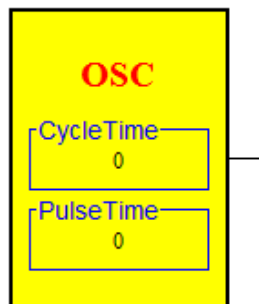
Детекција ивице сигнала (3/3)

► Тестирање и симулација



Генерисање правоугаоног сигнала (1/3)

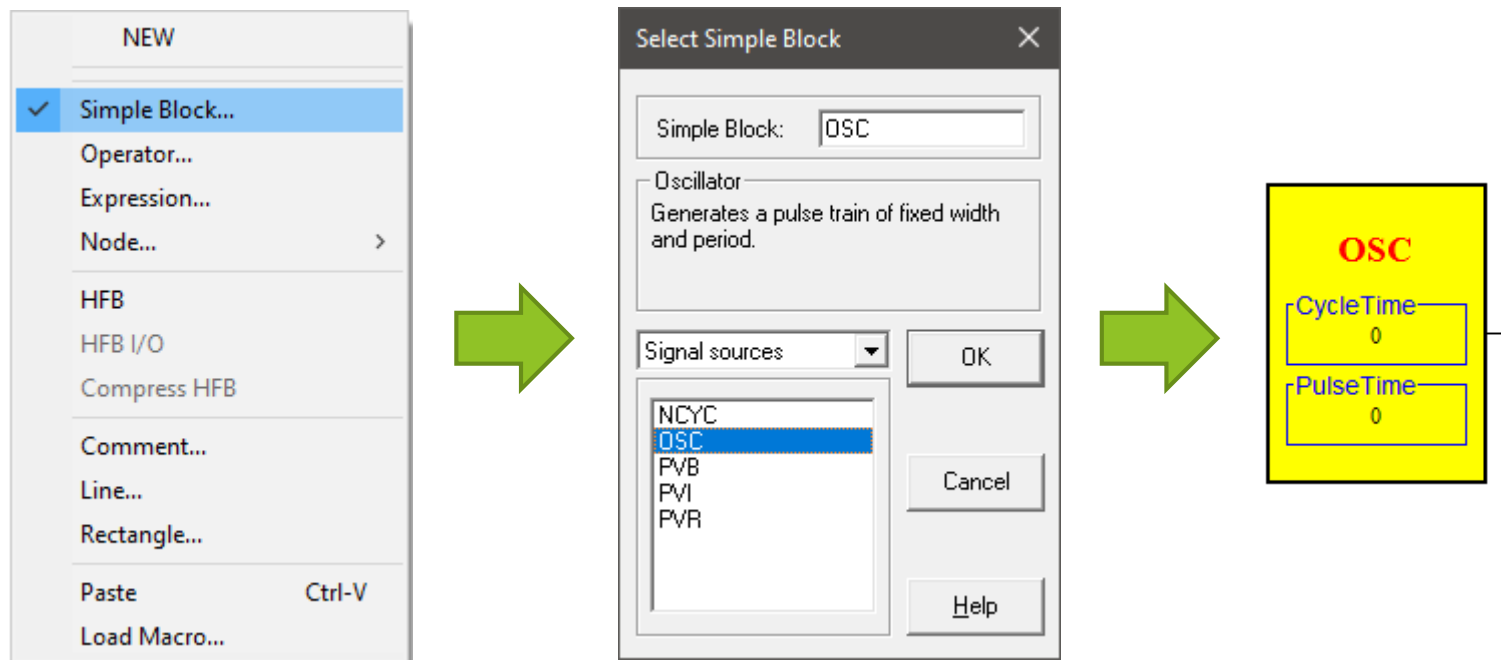
- OSC блок врши генерисање правоугаоног сигнала одређене фреквенције и одређеног односа између логичке нуле и логичке јединице у једној периоди



- Поседује следеће параметре
 - *Cycle Time*: време трајања периоде
 - *Pulse Time*: време трајања логичке јединице у периоди

Генерисање правоугаоног сигнала (2/3)

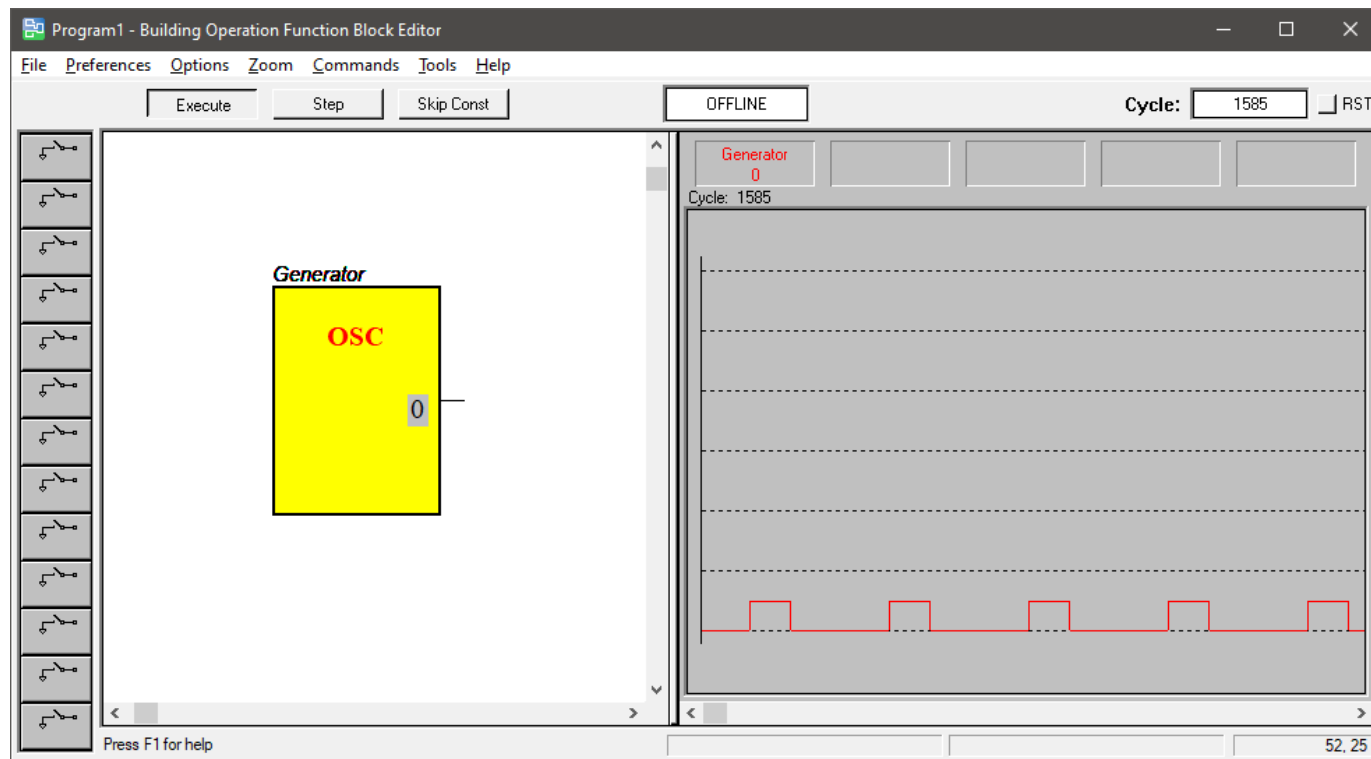
- Додати блок за генерисање правоугаоног сигнала (OSC)



- За назив блока унети 'Generator'
- За параметар *Cycle Time* унети 100
- За параметар *Pulse Time* унети 30

Генерисање правоугаоног сигнала (3/3)

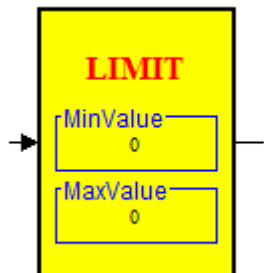
► Тестирање и симулација



Линеарне и нелинеарне функције

Блок за ограничење сигнала (1/3)

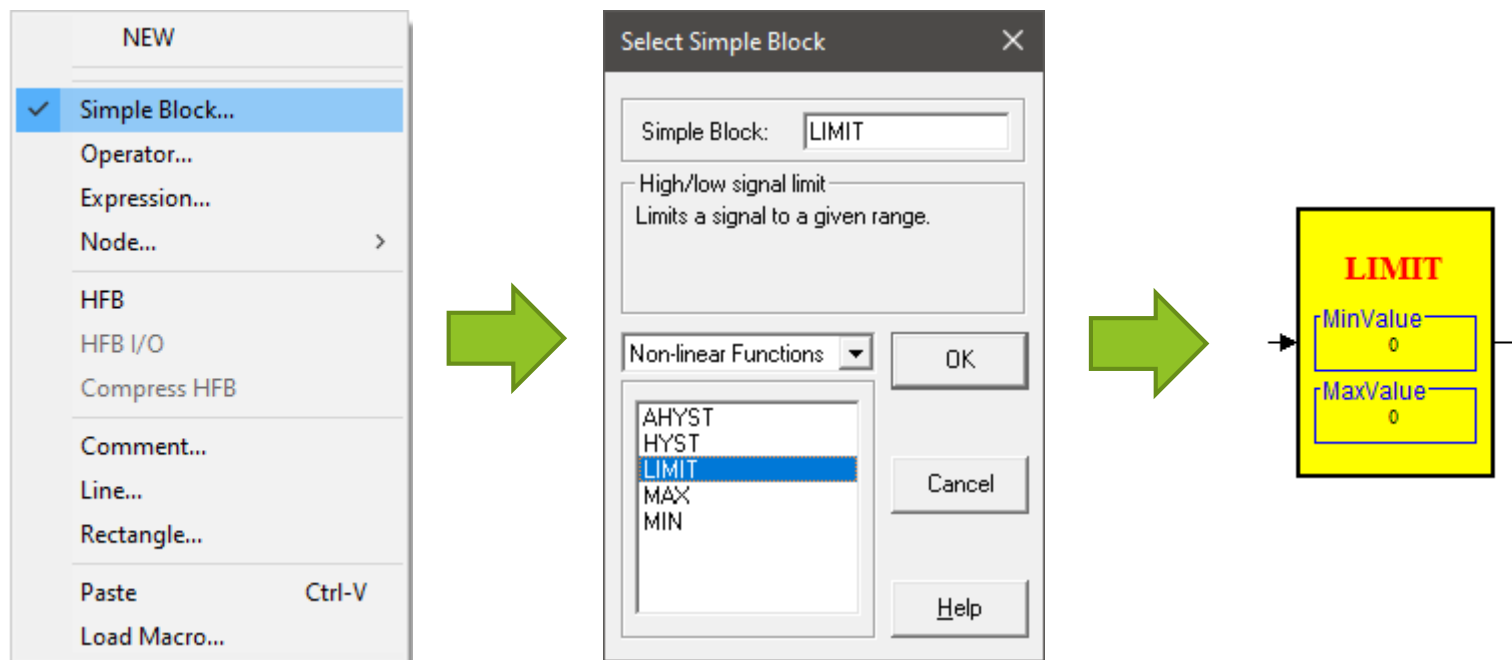
- ▶ Блок *Limit* врши одсецање сигнала са улаза који је изван задатих граница



- ▶ Параметар *MinValue* доња граница
- ▶ Параметар *MaxValue* горња граница
- ▶ Ако сигнал на улазу има већу вредност од горње границе сигнал на излазу узима вредност горње границе
- ▶ Ако сигнал на улазу има мању вредност од доње границе сигнал на излазу узима вредност доње границе
- ▶ Ако је сигнал на улазу има вредност између ове две границе, сигнал се пропушта на излаз без промене

Блок за ограничење сигнала (2/3)

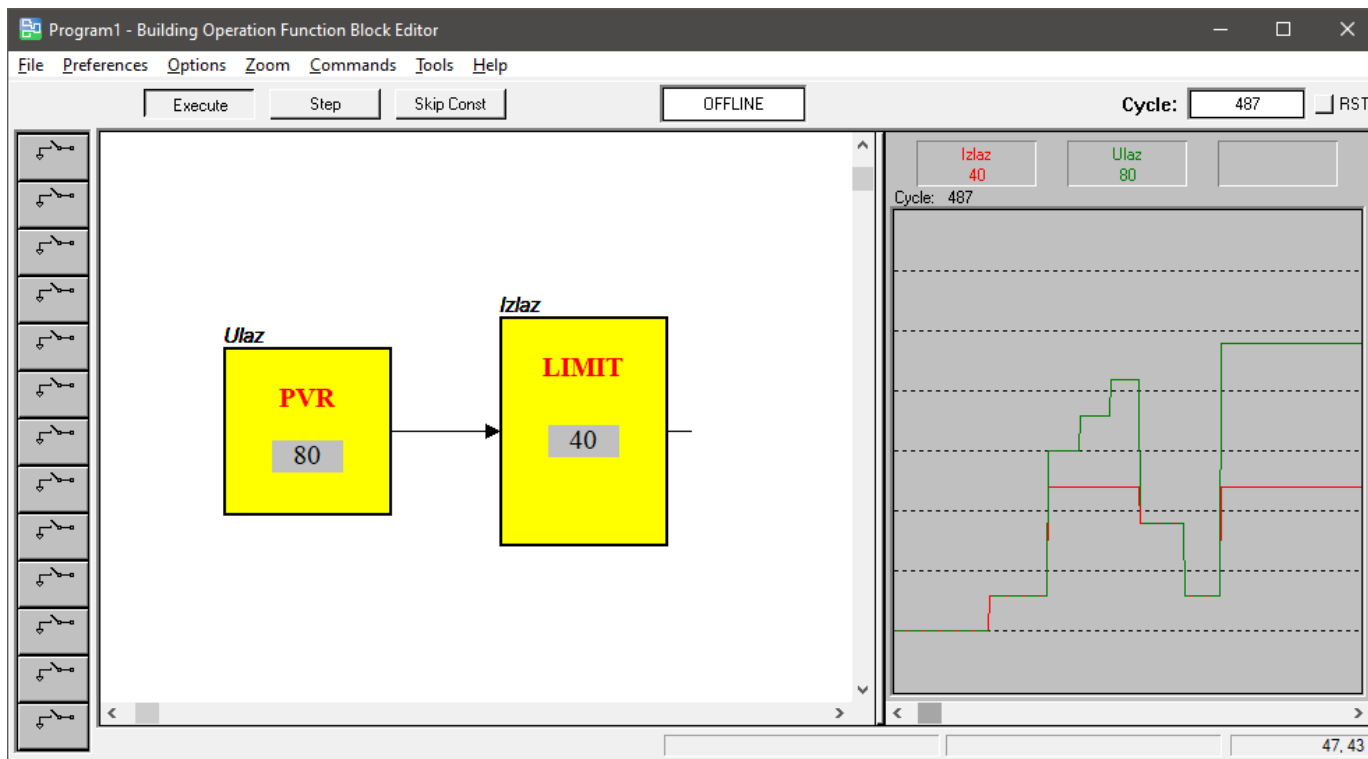
- Додати блок за ограничење сигнала (*LIMIT*)



- За назив блока унети 'Izlaz'
- За параметар *MinValue* унети 0, а за параметар *MaxValue* унети вредност 40
- Додати реалну константу и за назив блока унети 'Ulaz'

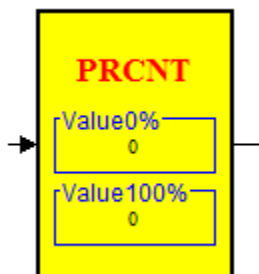
Блок за ограничење сигнала (3/3)

► Симулација и тестирање



Блок за трансформацију сигнала (1/3)

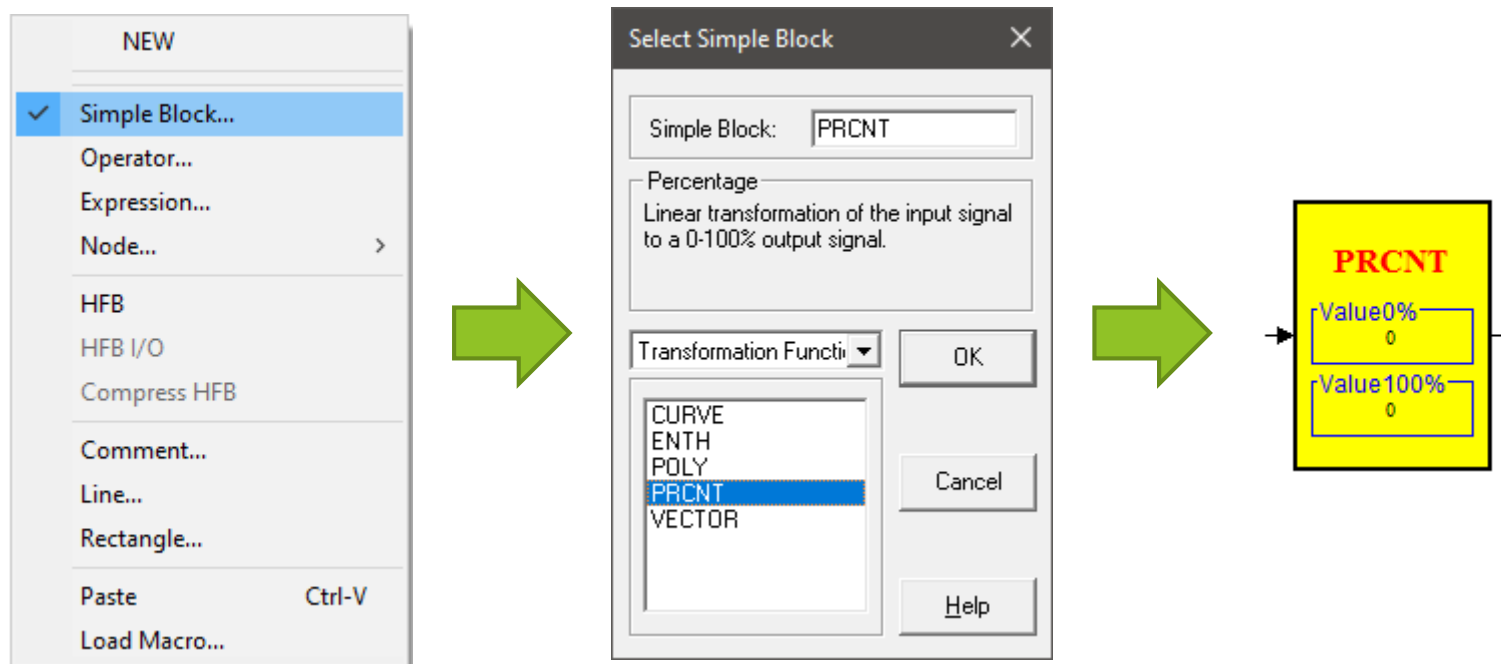
- ▶ Блок *Percent* врши линеарну трансформацију сигнала са улаза у сигнал на излазу у опсегу од 0 до 100 посто



- ▶ Параметар *Value0%* - вредност улазног сигнала који одговара 0%
- ▶ Параметар *Value100%* - вредност улазног сигнала који одговара 100%
- ▶ Ако сигнал на улазу има већу вредност од дефинисане, сигнал на излазу узима вредност 100%
- ▶ Ако сигнал на улазу има мању вредност од дефинисане, сигнал на излазу узима вредност 0%
- ▶ У осталим случајевима врши се линеарна трансформација улазног сигнала из опсега *Value0%* - *Value100%* у опсег 0% - 100%

Блок за трансформацију сигнала (2/3)

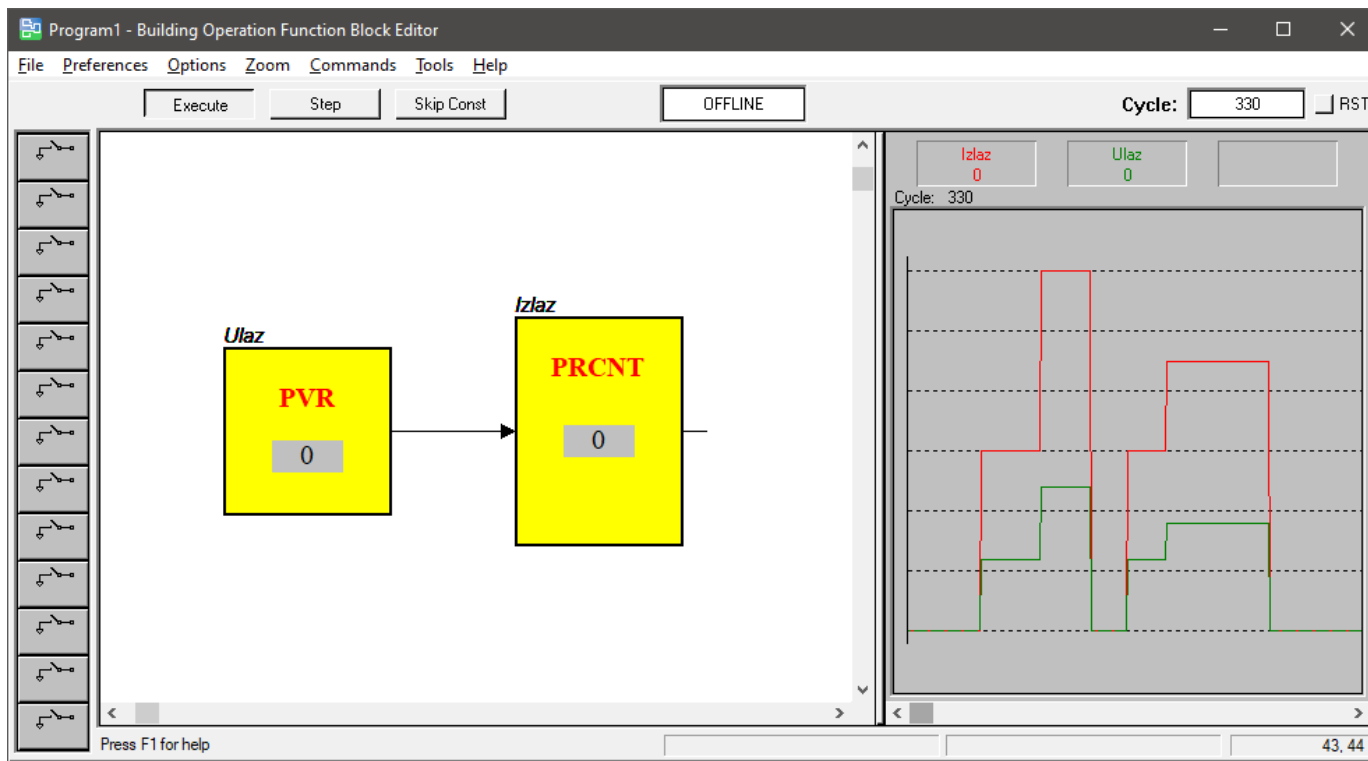
- Додати блок за линеарну трансформацију сигнала (*PRCNT*)



- За назив блока унети 'Izlaz'
- За параметар *Value0%* унети 0, а за параметар *Value100%* унети 40
- Додати реалну константу и за назив блока унети 'Ulaz'

Блок за трансформацију сигнала (3/3)

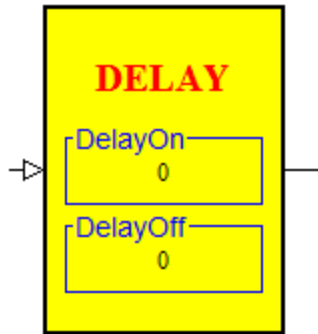
► Симулација и тестирање



Тајмери

Тајмери по укључењу и искључењу (1/3)

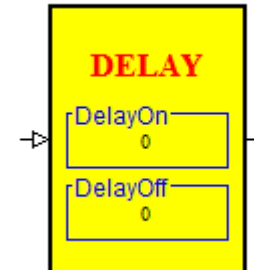
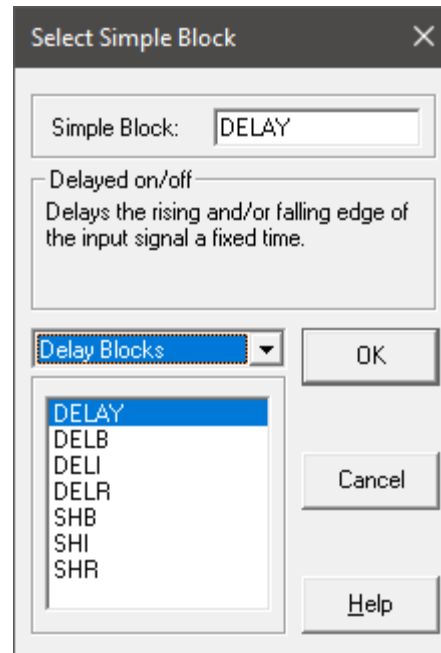
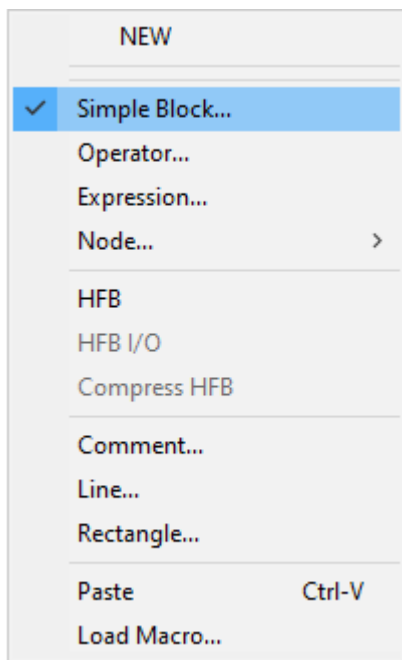
- Тајмер по укључењу задржава растућу ивицу сигнала са улаза за задати број секунди док тајмер по искључењу задржава опадајућу ивицу сигнала са улаза за задати број секунди



- Улаз је бинарног типа
- Излаз је бинарног типа
- *DelayOn* параметар - време одлагања по укључењу
- *DelayOff* параметар - време одлагања по искључењу

Тајмери по укључењу и искључењу (2/3)

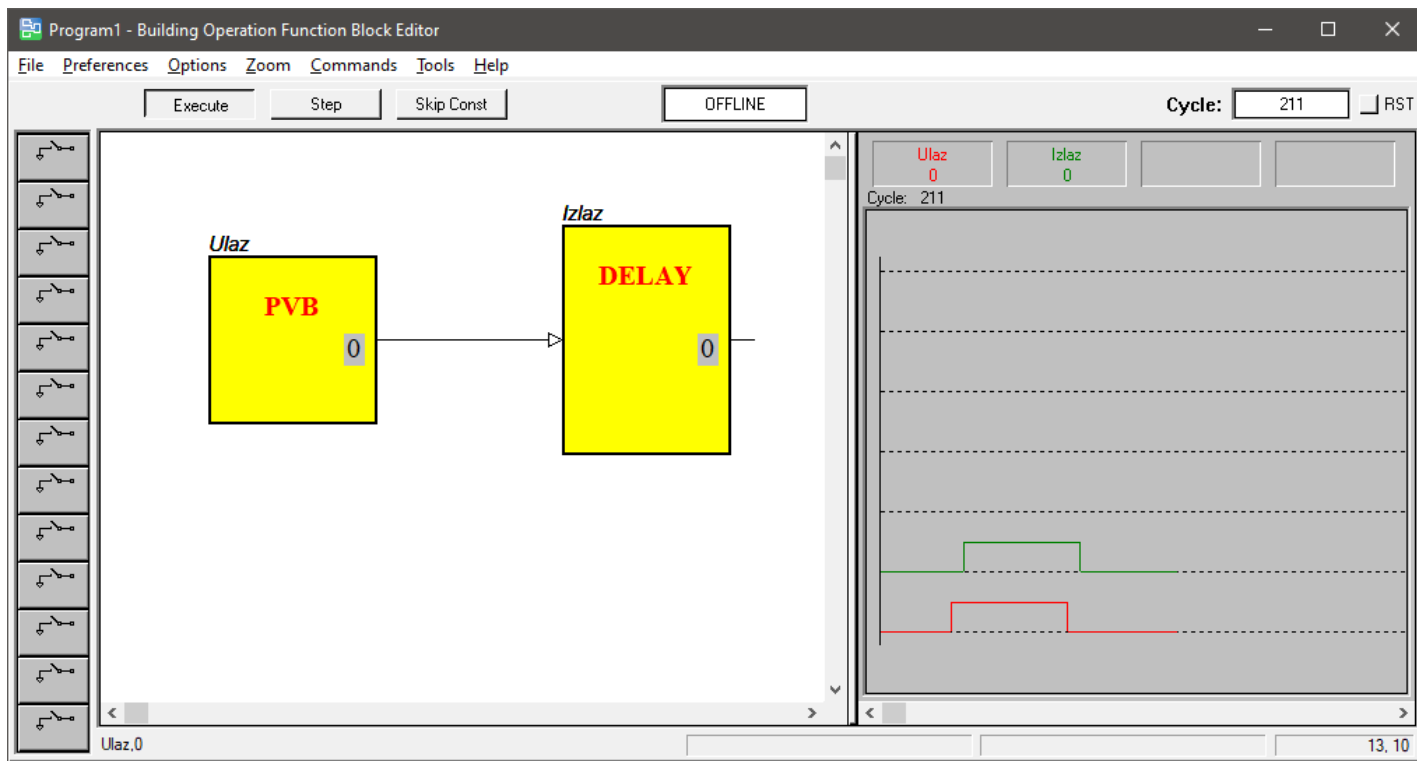
- Додати тајмер по укључењу/по искључењу



- За назив блока унети 'Izlaz'
- Унети за параметре *DelayOn* и *DelayOff* вредност 10
- Додати константу бинарног типа и за назив блока унети 'Ulaz'

Тајмери по укључењу и искључењу (3/3)

► Тестирање и симулација



Пулсни тајмер (1/3)

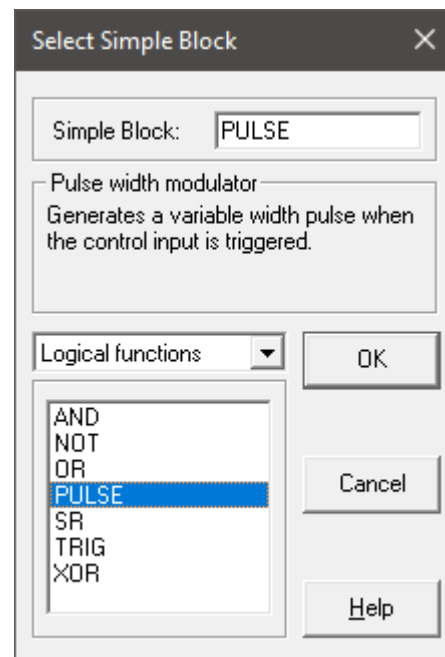
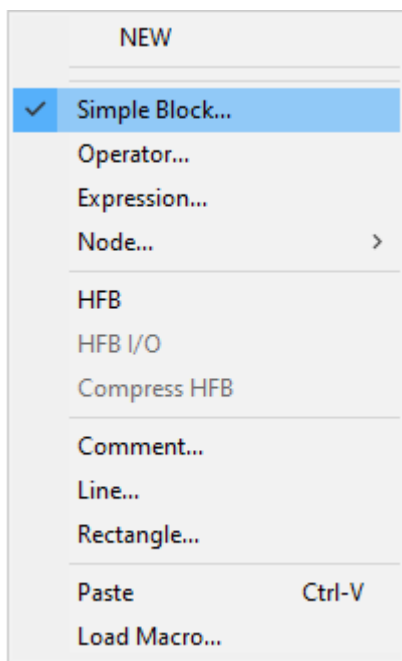
- Излаз из блока добија вредност логичке јединице након растуће ивице сигнала са улаза и задржава је у задатом времену



- t - бинарни улаз
 - Улаз од интереса чија се промена посматра
- pl - реални улаз
 - Задато време трајања логичке јединице на излазу

Пулсни тајмер (2/3)

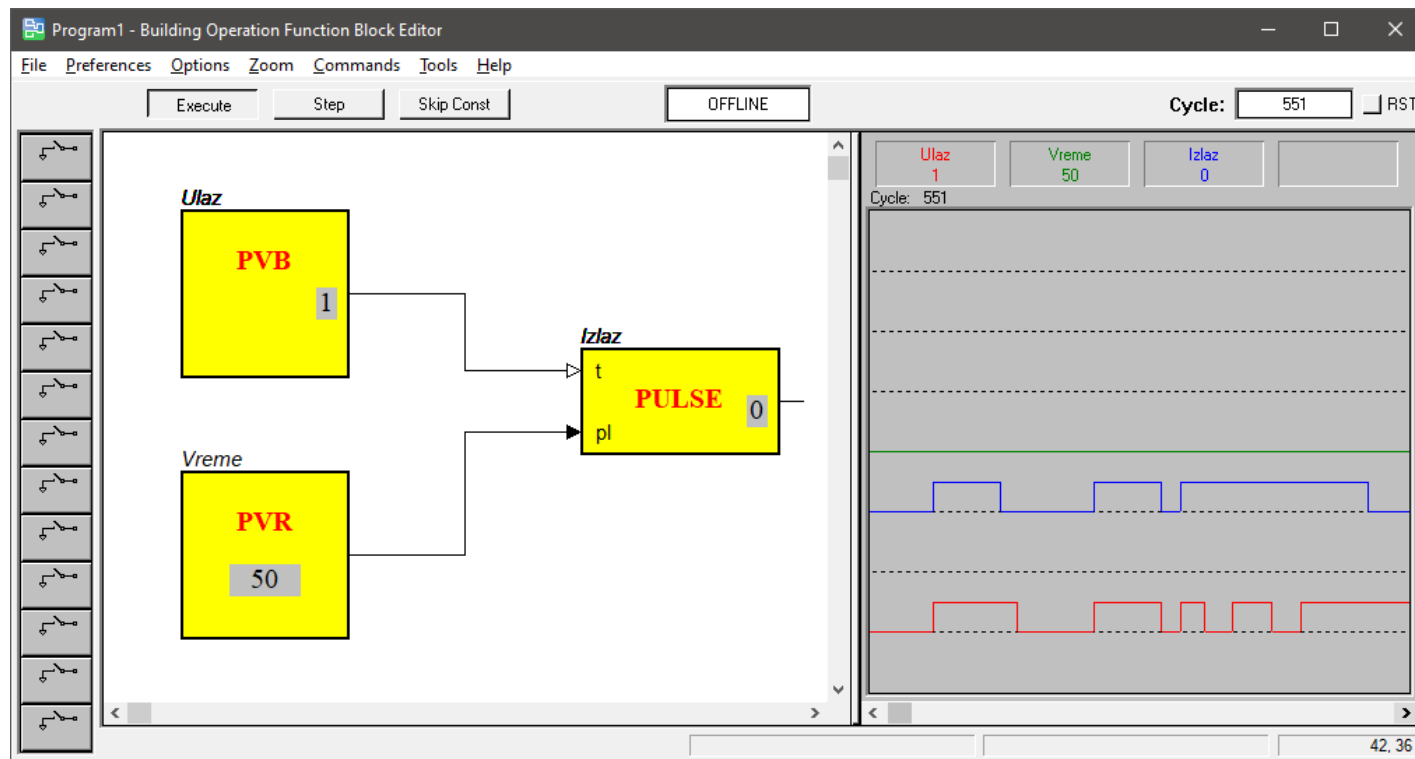
- Додати пулсни тајмер



- За назив блока унети 'Izlaz'
- Додати константу бинарног типа и за назив блока унети 'Ulaz'
- Додати константу реалног типа и за назив блока унети 'Vreme' а за почетну вредност унети 50

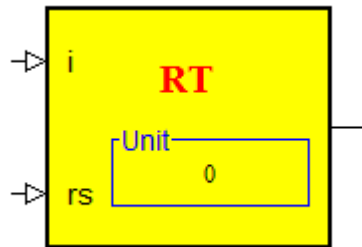
Пулсни тајмер (3/3)

► Тестирање и симулација



Мерач времена (1/3)

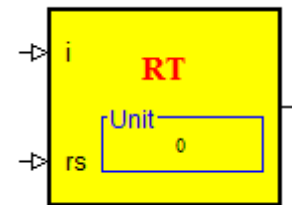
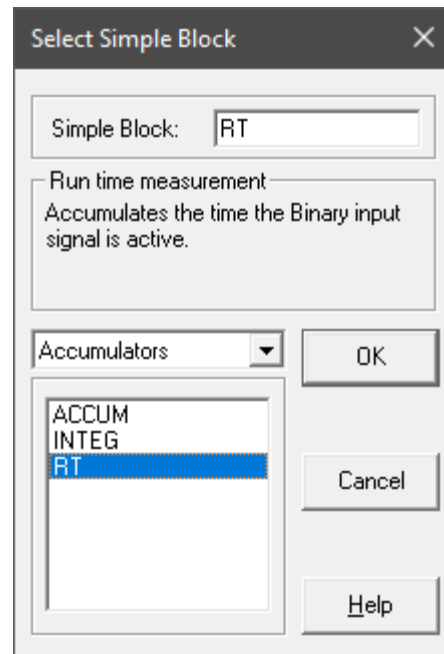
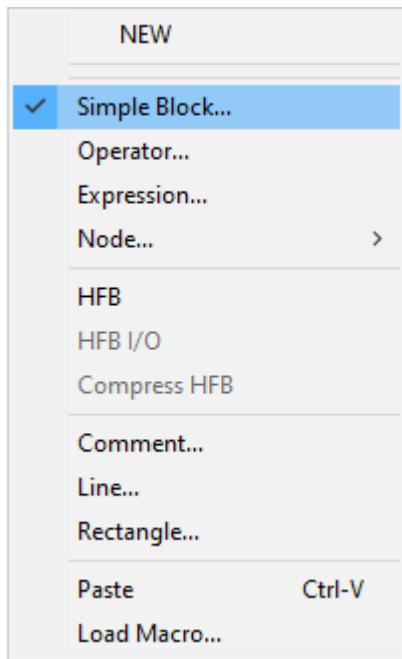
- Мери се време у којем сигнал има вредност логичке јединице



- ***i*** - бинарни улаз
 - Улаз од интереса за мерење времена
- ***rs*** - бинарни улаз
 - Кад је улаз једнак логичкој јединици ресетује се измерено време
- ***unit*** - параметар
 - 0 - сати
 - 1 - минуте
 - 2 - секунде
- Излаз из блока представља измерено време

Мерач времена (2/3)

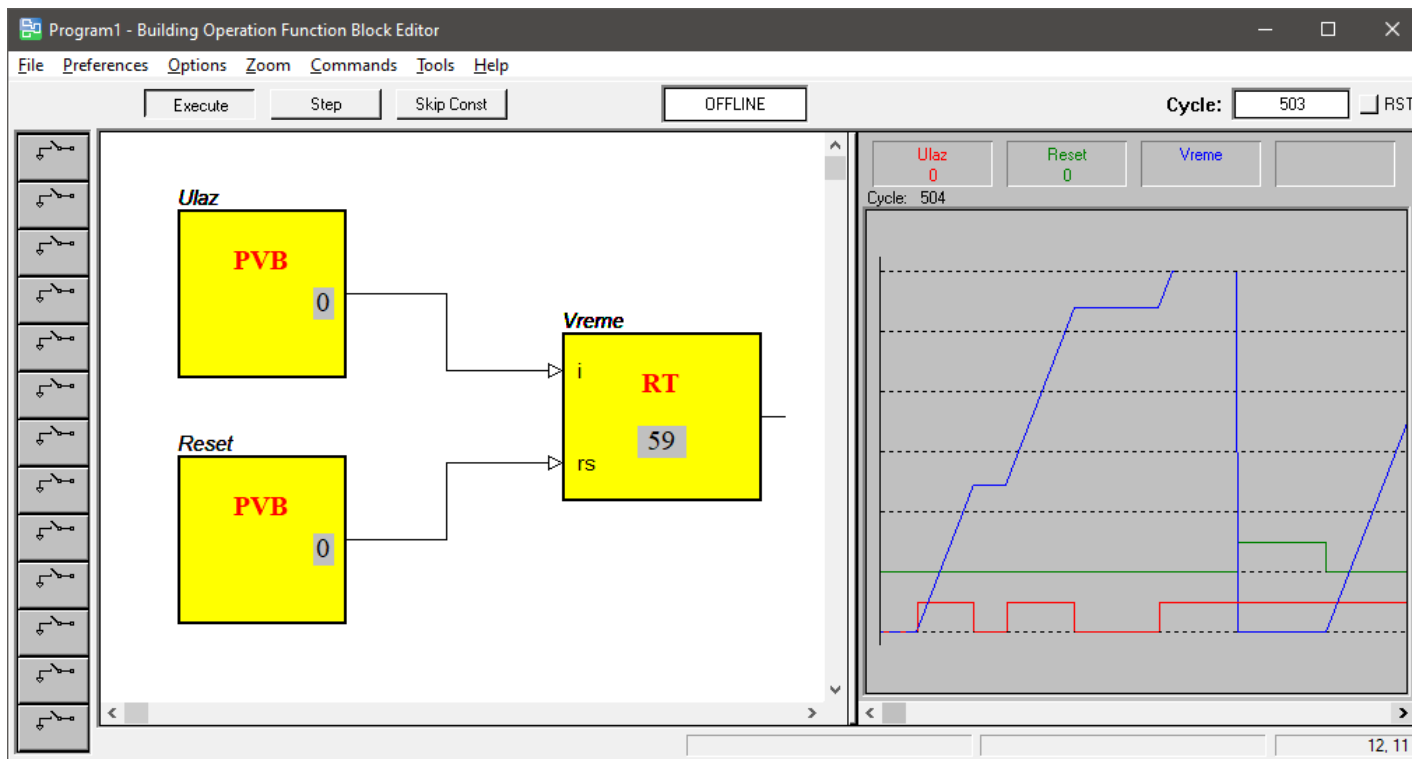
► Додати мерач времена



- За назив блока унети 'Izlaz'
- Како би мерач бројао секунде унети за параметар *Unit* вредност 2
- Додати константу бинарног типа и за назив блока унети 'Ulaz'
- Додати константу бинарног типа и за назив блока унети 'Reset'

Мерач времена (3/3)

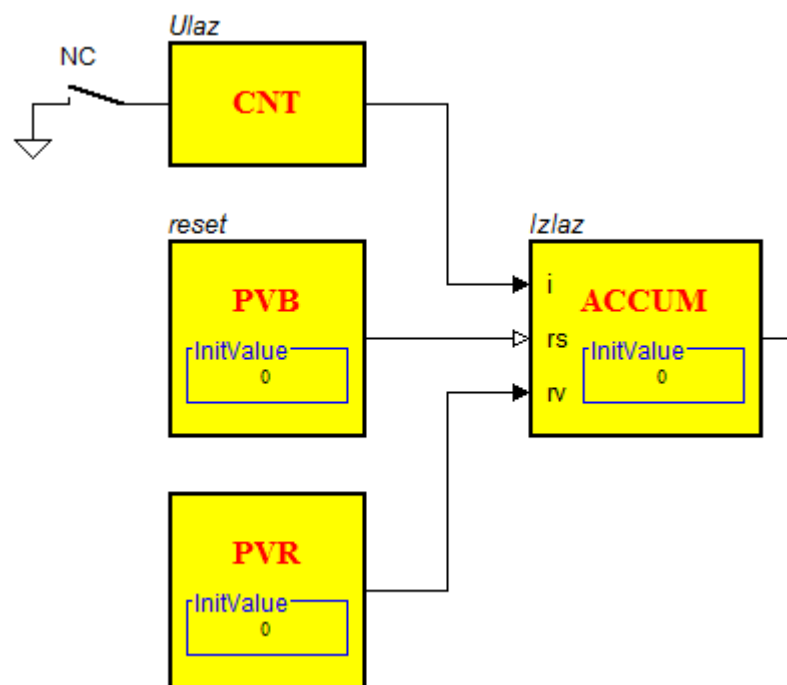
► Тестирање и симулација



Бројачи

Наменски бројач

- ▶ ФБД не поседује класичне бројаче као у другим програмским језицима
- ▶ Постоји наменски бројач за бројање импулса са посебног бројачког улаза који се може користити само за ту намену
- ▶ CNT блок броји импулсе у једном циклусу скенирања док блок ACCUM акумулира вредности из сваког циклуса



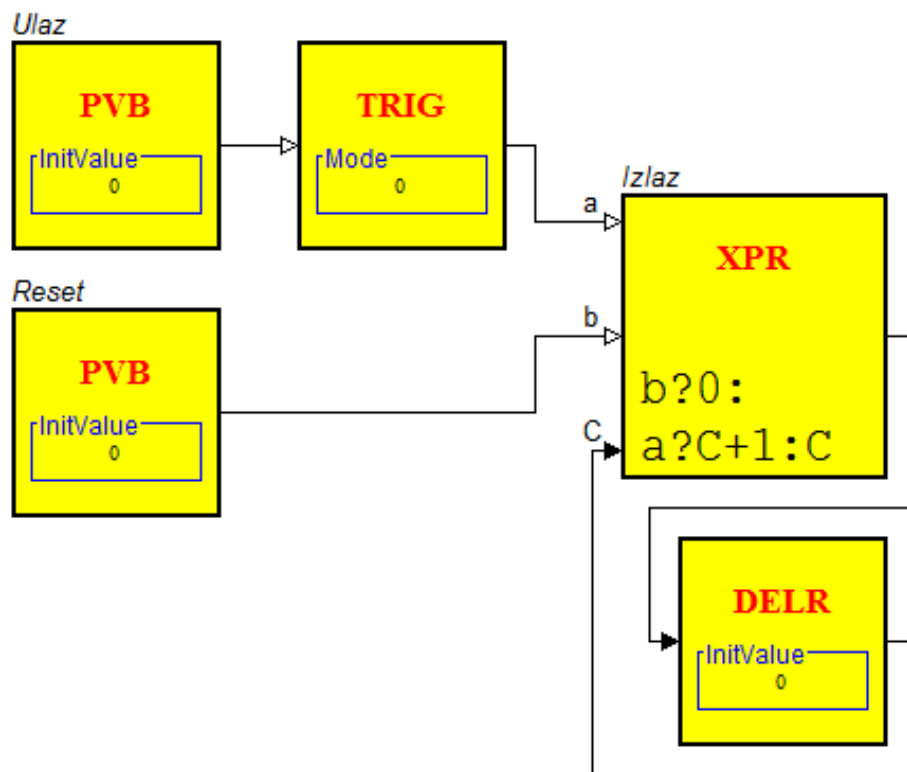
Произвољни бројач (1/3)

- ▶ Бројач има једноставну функцију да на сваку растућу ивицу улазног сигнала претходна вредност бројача се увећа за један
- ▶ Пошто ФБД не поседује механизам чувања вредности променљивих, чување се мора одрадiti помоћу повратне спреге али обавезно са колом задршке од једног програмског циклуса
- ▶ Бројач мора да поседује улаз за ресетовање како би се вредност бројања анулирала
- ▶ Бројач мора да поседује детекцију растуће ивице сигнала са улаза чије се импулси броје
- ▶ Псеудо код једног бројача

```
IF Reset==true THEN
    Brojac=0
ELSE
    IF trig(Ulaz) THEN
        Brojac=Brojac+1
    END
END
```

Произвољни бројач (2/3)

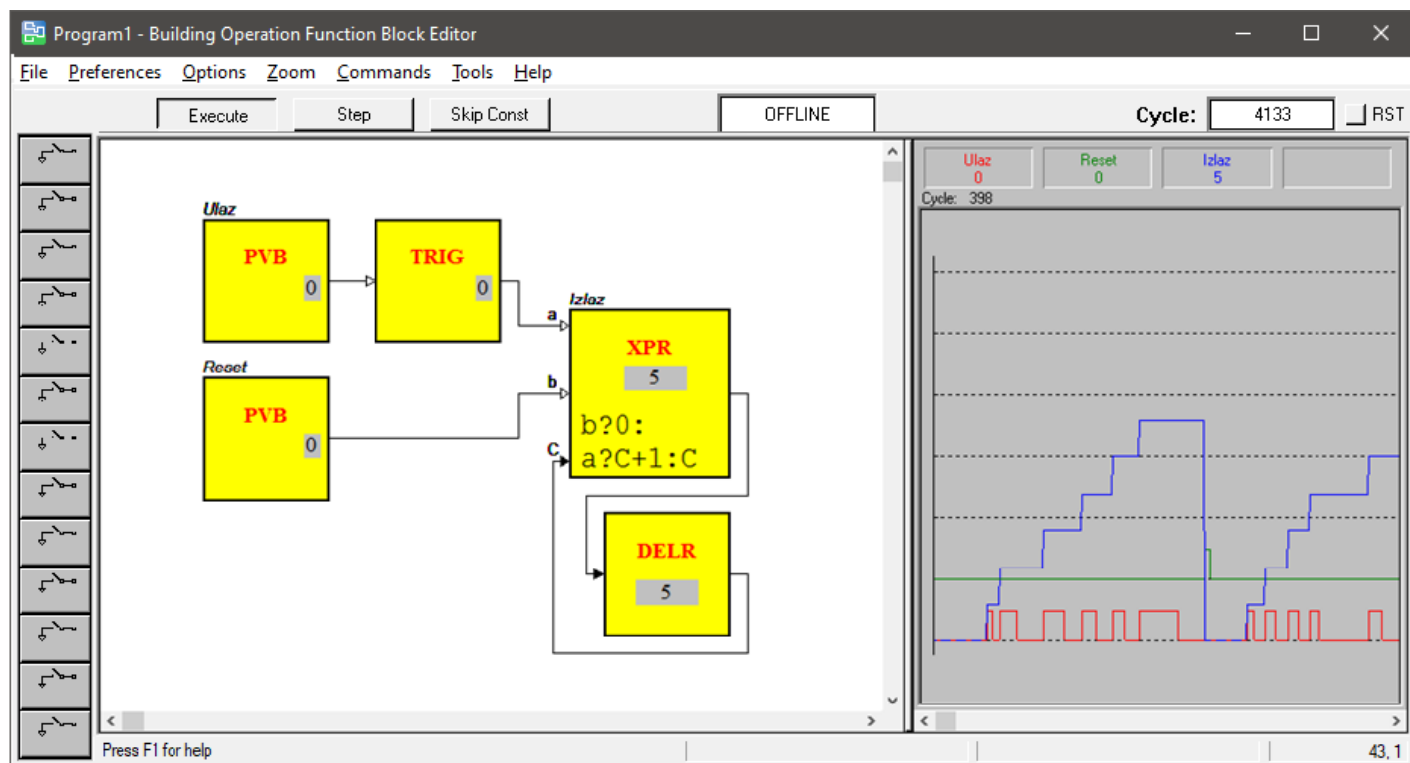
- Предлог реализације једног бројача



- Додати блокове и повезати их тако да буду у складу са предлогом решења

Произвољни бројач (3/3)

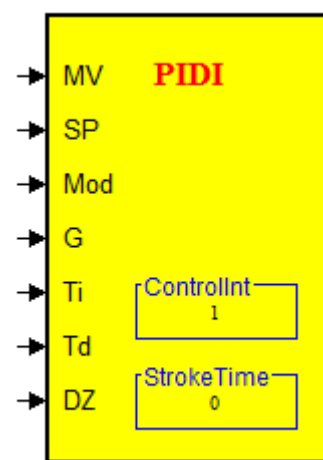
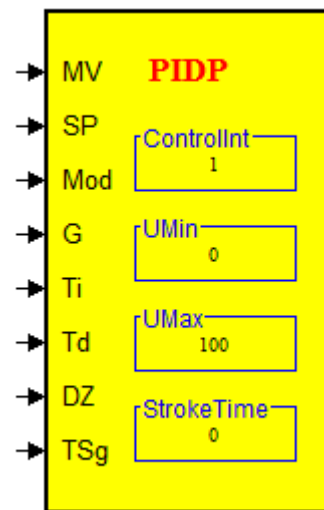
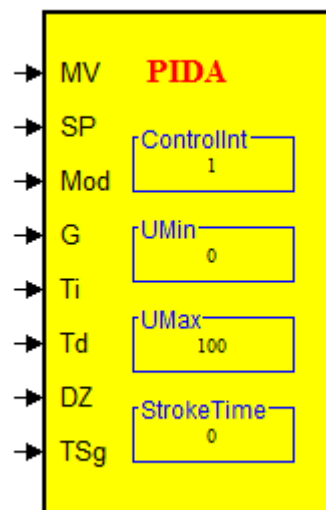
► Тестирање и симулација



ПИД контролер

УВОД

- ▶ ПИД регулатор је један од најпознатијих и најприменљивијих алгоритама управљања у индустријској и неиндустријској примени
- ▶ У ФБД-у постоје три врсте ПИД регулатора
 - ▶ *PIDA* - класичан ПИД регулатор који на излазу даје укупно управљање
 - ▶ *PIDP* - модификована верзија ПИД алгорита са тежинским факторима за задату и тренутну вредност
 - ▶ *PIDI* - регулатор са инкременталним излазом који на излазу даје промену управљања



Опис (1/2)

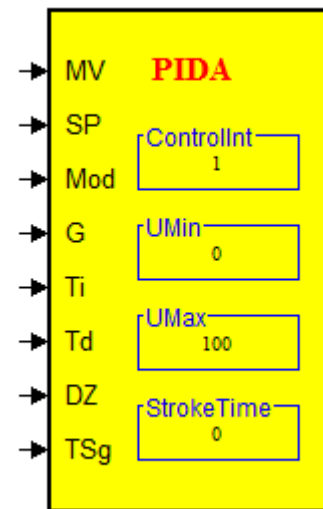
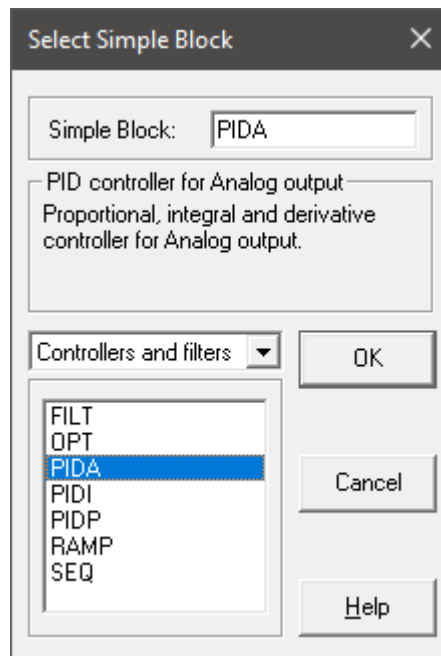
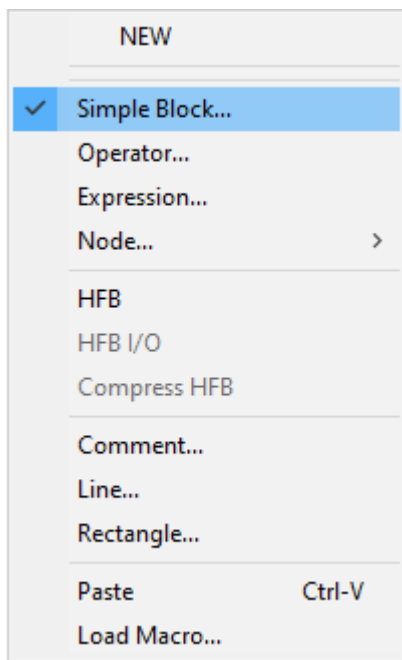
- ▶ Улазни сигнали ПИД регулатора:
 - ▶ *MV* - тренутна вредност (реални тип)
 - ▶ *SP* - задата вредност (реални тип)
 - ▶ *Mod* - режим рада (целобројни тип)
 - ▶ 0 - искључен, ($du = 0$)
 - ▶ 1 - нормални рад
 - ▶ 2 - излаз из регулатора је постављен на *UMax*
 - ▶ 3 - излаз из регулатора је постављен на *UMin*
 - ▶ *G* - пропорционално појачање (реални тип)
 - ▶ *Ti* - време интеграљења у секундама (реални тип)
 - ▶ *Td* - време диференцирања у секундама (реални тип)
 - ▶ *DZ* - опсег неутралности регулатора (реални тип)
 - ▶ *TSg* - повратни управљачки сигнал (реални тип)
 - ▶ *PIDI* нема овај параметар пошто не рачуна укупно управљање

Опис (2/2)

- ▶ Додатни параметри ПИД регулатора:
 - ▶ *ControlInt* - време извршавања ПИД алгоритма у секундама (реални тип)
 - ▶ *StrokeTime* - Време промене стања актуатора у секундама из једне у другу крајњу тачку (реални тип)
 - ▶ *Umin* - Минимална вредност управљачког сигнала (реални тип)
 - ▶ *PIDI* нема овај параметар пошто не рачуна укупно управљање
 - ▶ *Umax* - Максимална вредност управљачког сигнала (реални тип)
 - ▶ *PIDI* нема овај параметар пошто не рачуна укупно управљање

Додавање ПИД блока (1/2)

► Додати ПИД блок (PIDA)



► За назив блока унети 'PID'

Додавање ПИД блока (2/2)

- ▶ Додати 6 реалних константи а за називе и почетне вредности унети следеће:
 - ▶ *TrenVred* - 40
 - ▶ *ZadVred* - 50
 - ▶ *Kp* - 1
 - ▶ *Ti* - 0,1
 - ▶ *Td* - 10
 - ▶ *Neutral* - 0
- ▶ Додати целобројну константу а за назив и почетну вредност унети следеће :
 - ▶ *Rezim* - 1
- ▶ Повезати блокове и излаз из PIDA блока вратити на TSg улаз

ПІД контролер

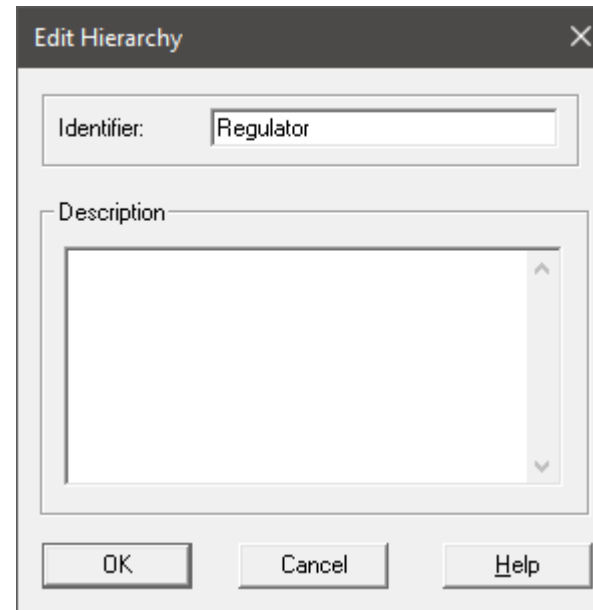
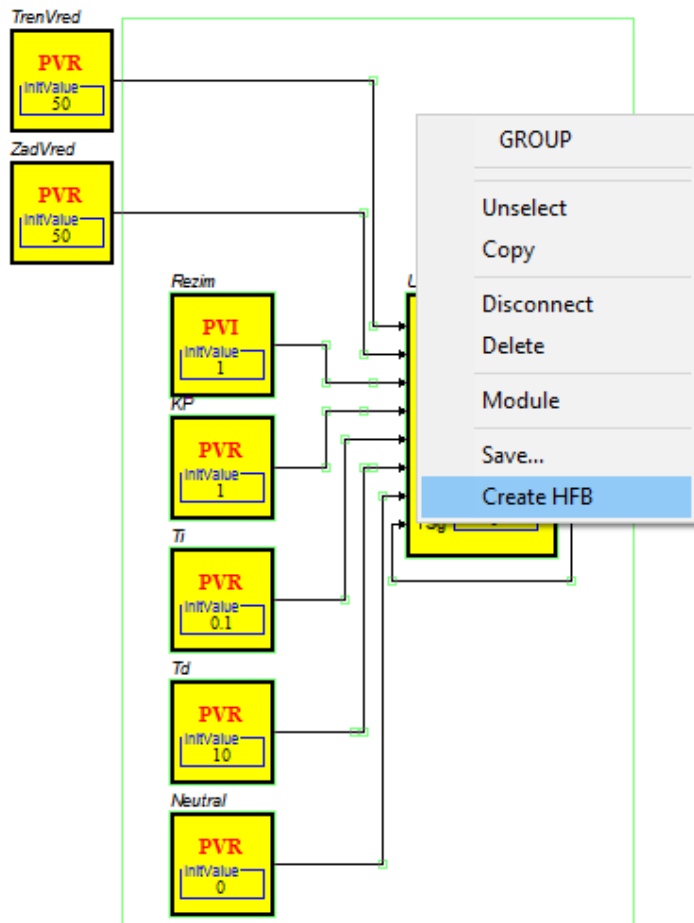
Schneider
Electric



Организација програмског кода

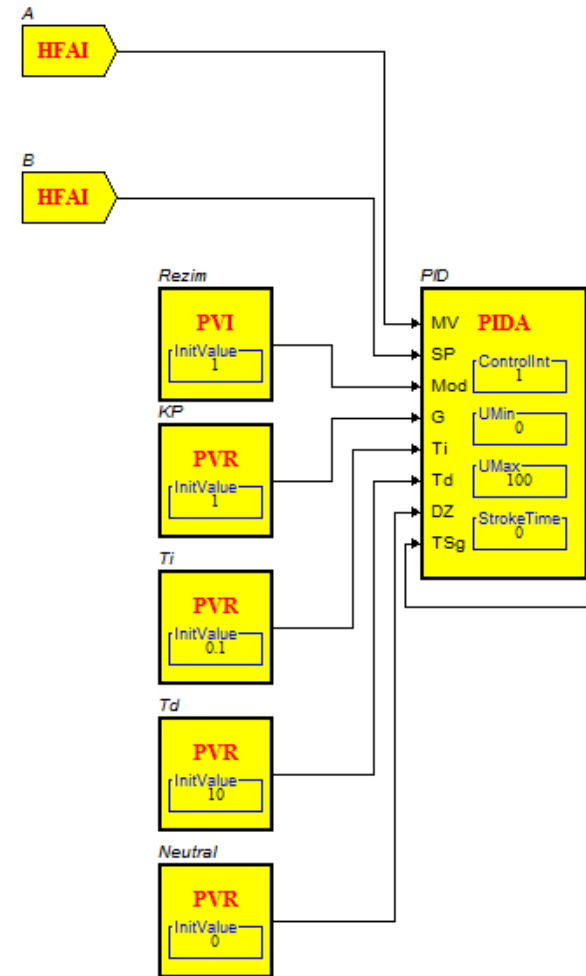
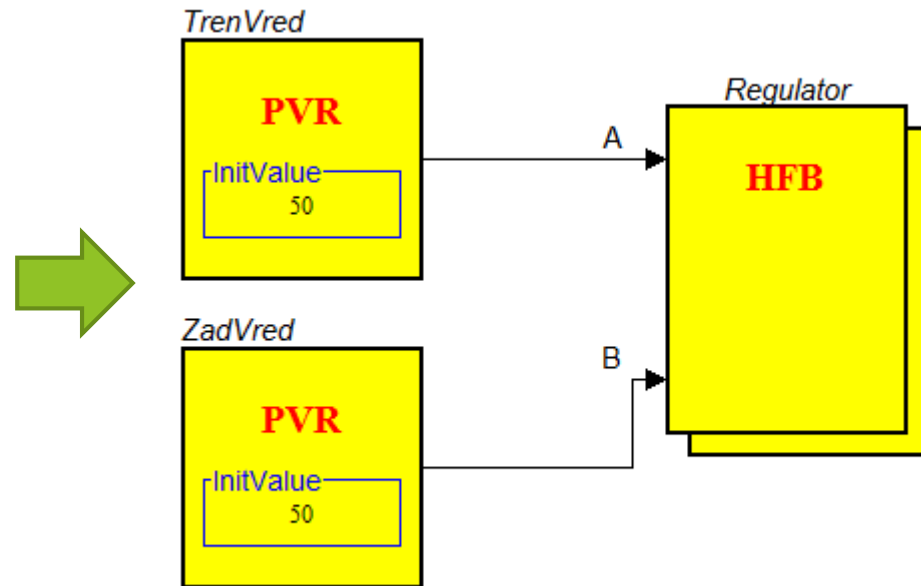
Додавање хијерархијског блока (1/5)

- ▶ Додати хијерархијски блок који ће обухватати блокове оператора и израза
- ▶ Унети за назив блока 'Regulator'



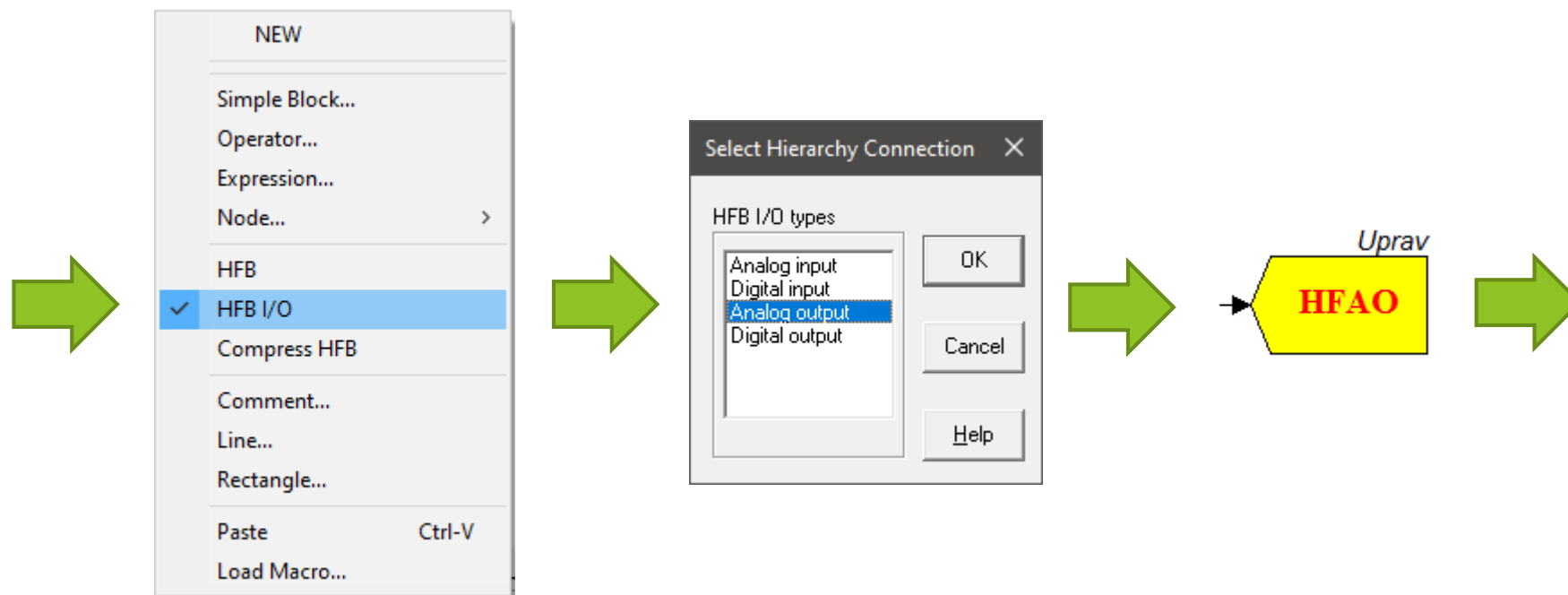
Додавање хијерархијског блока (2/5)

- ▶ Отворити ново формиран блок



Додавање хијерархијског блока (3/5)

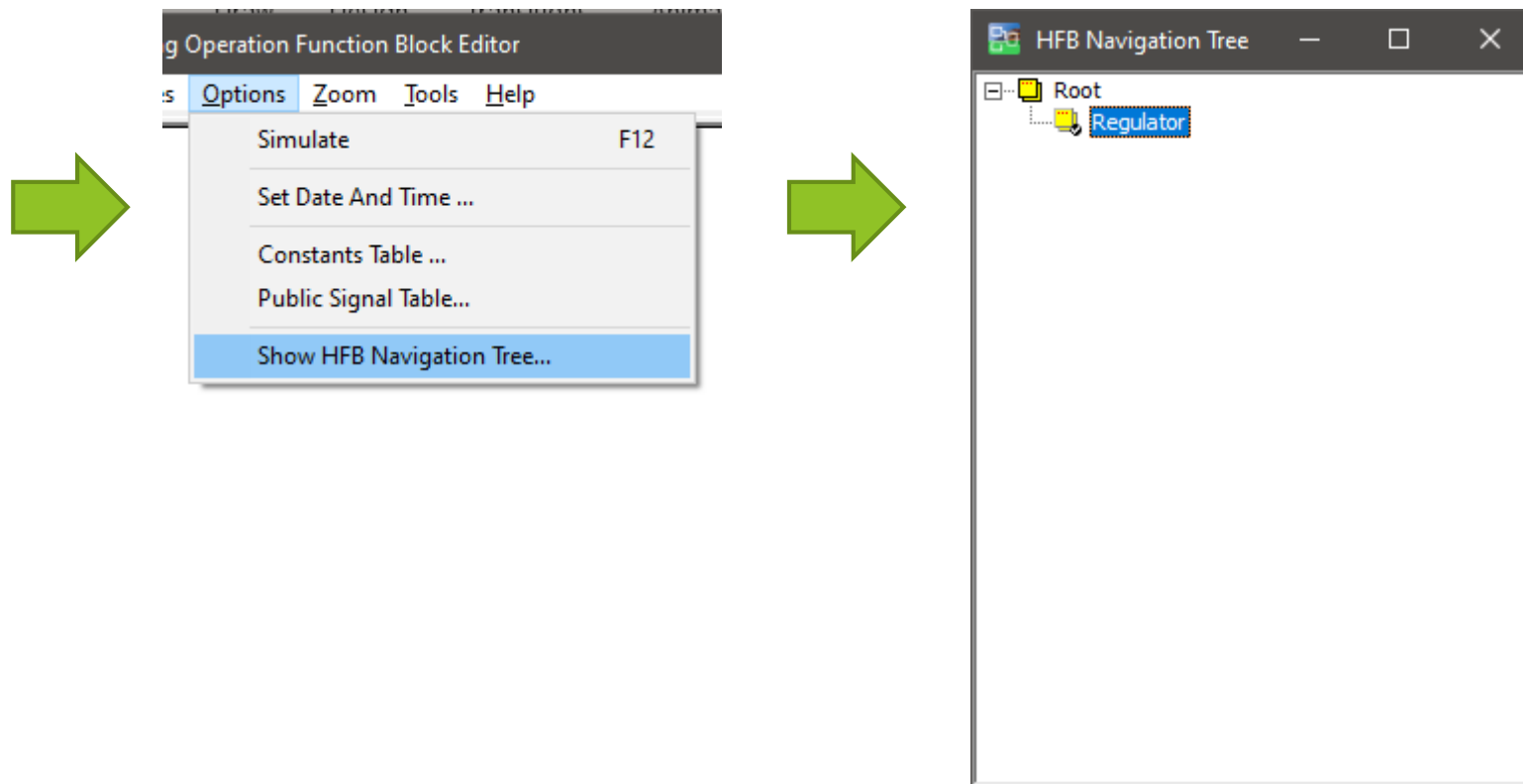
- ▶ Додати аналогни излазни блок у ново формираном хијерархијском блоку и за назив блока унети 'U' а затим га повезати са излазом из ПИД блока



- ▶ Изменити називе улазних аналогних блокова
 - ▶ 'A' - 'TrenVred'
 - ▶ 'B' - 'ZadVred'

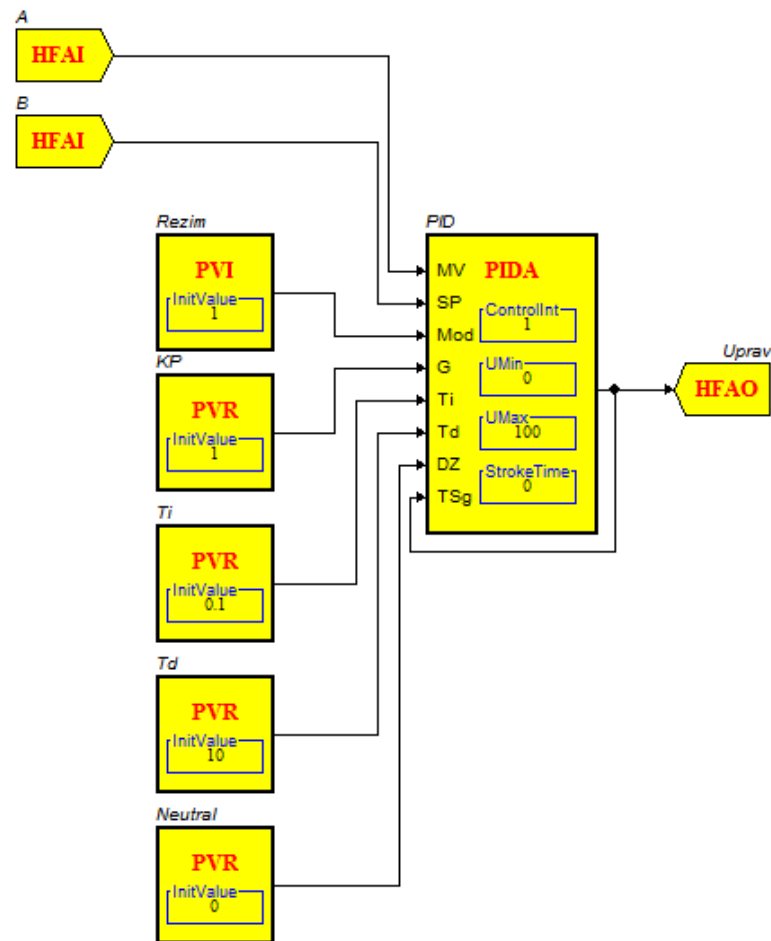
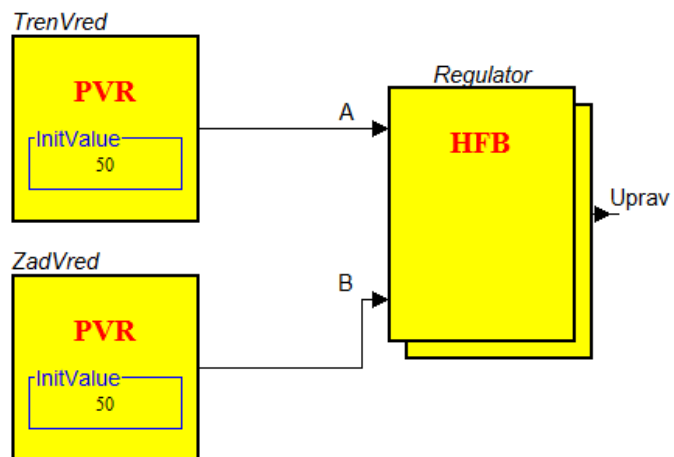
Додавање хијерархијског блока (4/5)

► Преглед хијерархијских блокова



Додавање хијерархијског блока (5/5)

► Хијерархијски блок 'Regulator'



Референце

Референце

- *EcoStruxure Building Operation - Technical Reference Guide - 04-16006-04-en*

