Single Event Effects Mitigation with TMRG Tool

Szymon KULIS, CERN, Geneva, Switzerland (szymon.kulis@cern.ch)

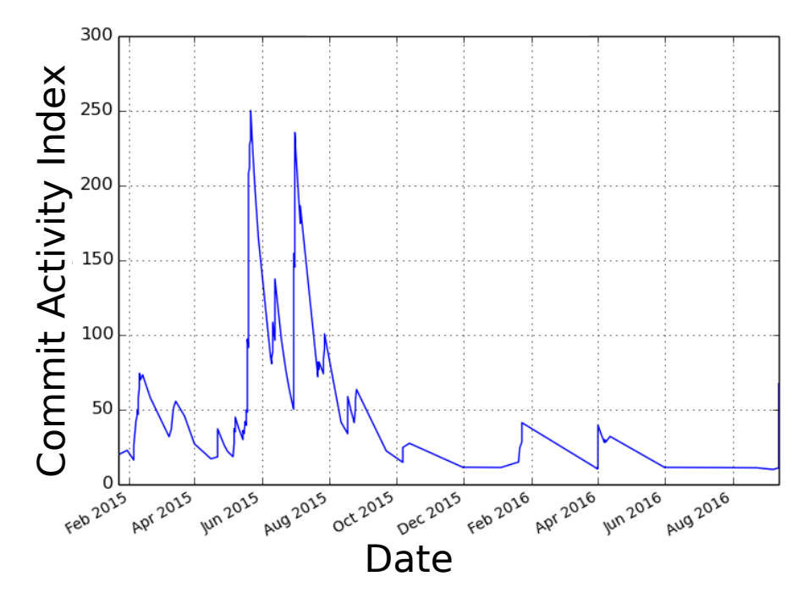
Triple Modular Redundancy Generator
http://cern.ch/tmrg

Motivation & Status

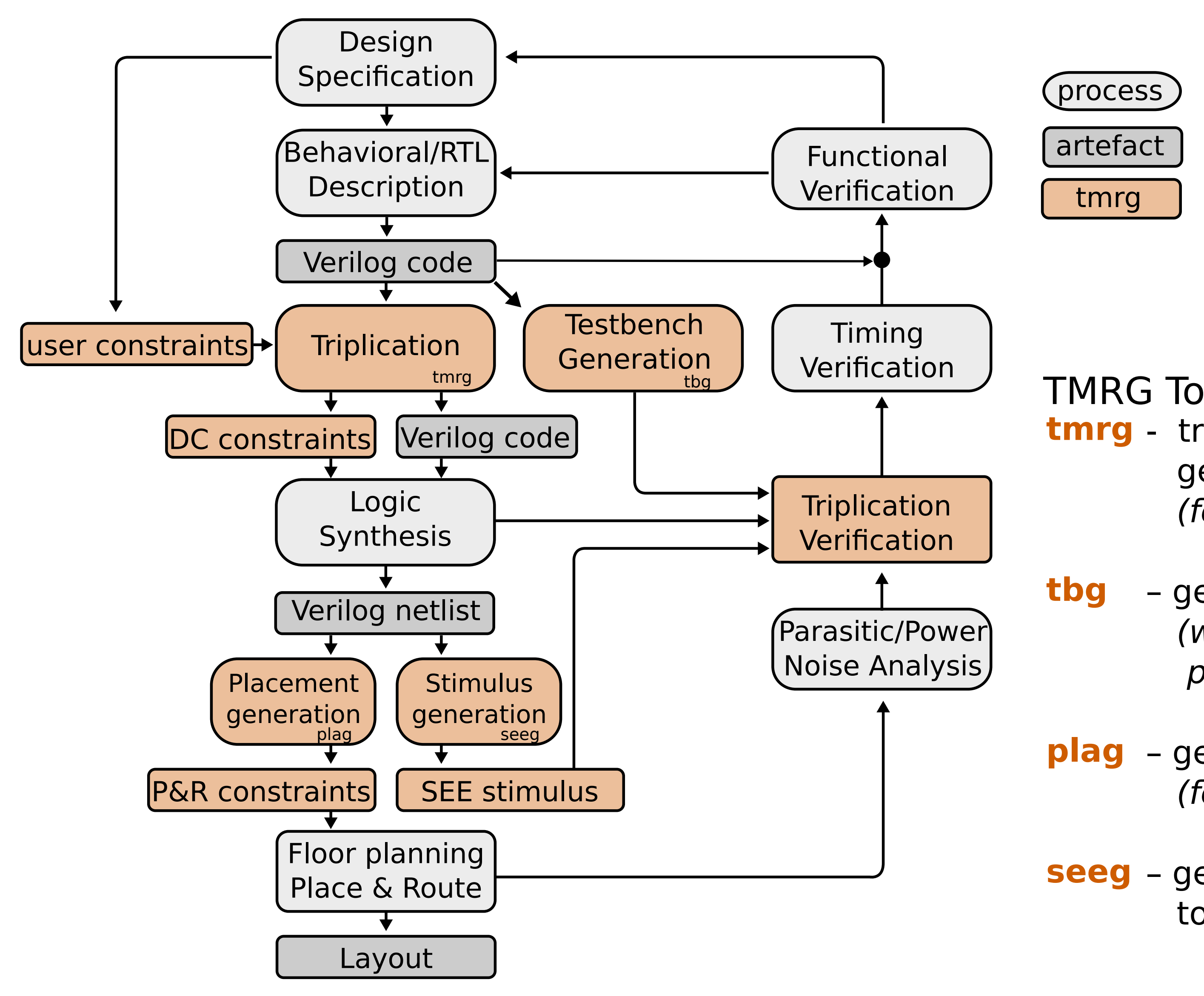
The purpose of the TMRG toolset is to automatize the process of triplicating digital circuits.

The tool:
* is compatible with ASIC design flow used in the HEP (Verilog-95 RTL, Cadence tools)
* does not constrain user's coding style (the source Verilog should be synthesizable)
* allows to obtain various flavors of TMR (registers only, full triplication, logic triplication, clock triplication)
* assists in the physical implementation stage (synthesis, P&R)
* assists the designer in the verification process (generation of SEE)
* can be run in a batch mode (fully automatic flow)

Project started: March 2015
* Project size: >13000 lines of code
* Documentation size: 60 pages (pdf)
* Projects actively using TMRG
  (GBLD10+, LDQ10, VLAD, DRAD, lpGBTX, ePLL-CDR)
* User base: ~5 designers
* Open source, hosted in CERN svn repository
  https://svnweb.cern.ch/cern/wsvn/tmrg (400+ revisions):



TMRG Digital Design Flow



TMRG Toolset:

tmrg - triplicates the Verilog code and generates synthesis constrains *(for the Design Compiler)*

tbg – generates generic test bench template *(with /without TMR, SEE injection, post synthesis, post PNR)*

plag – generates placement directives *(for Encounter)*

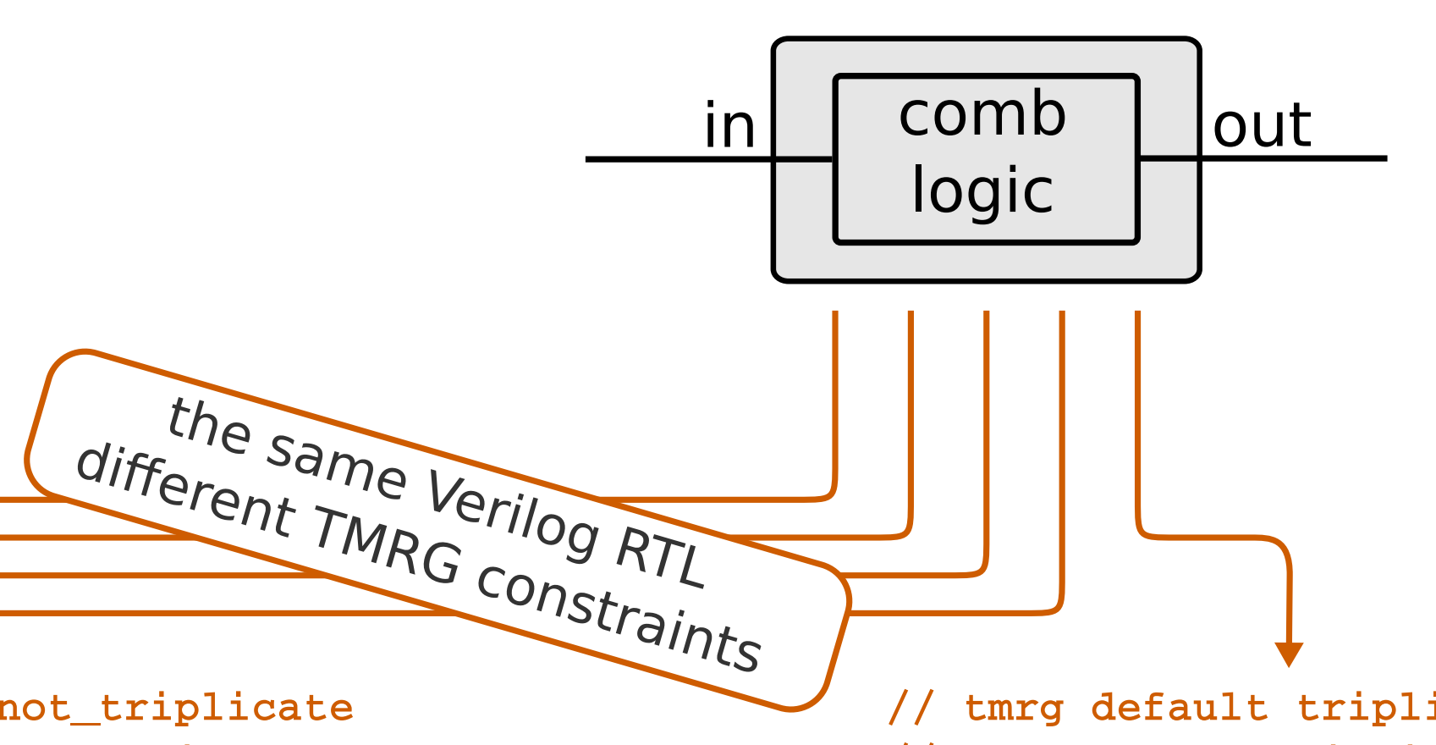
seeg – generates Single Event Effects stimulus to be used for transient simulations

Constraining the design

* designer decides which blocks and signals are to be triplicated by using TMRG directives
  (placed in Verilog code, configuration file, or command line argument)

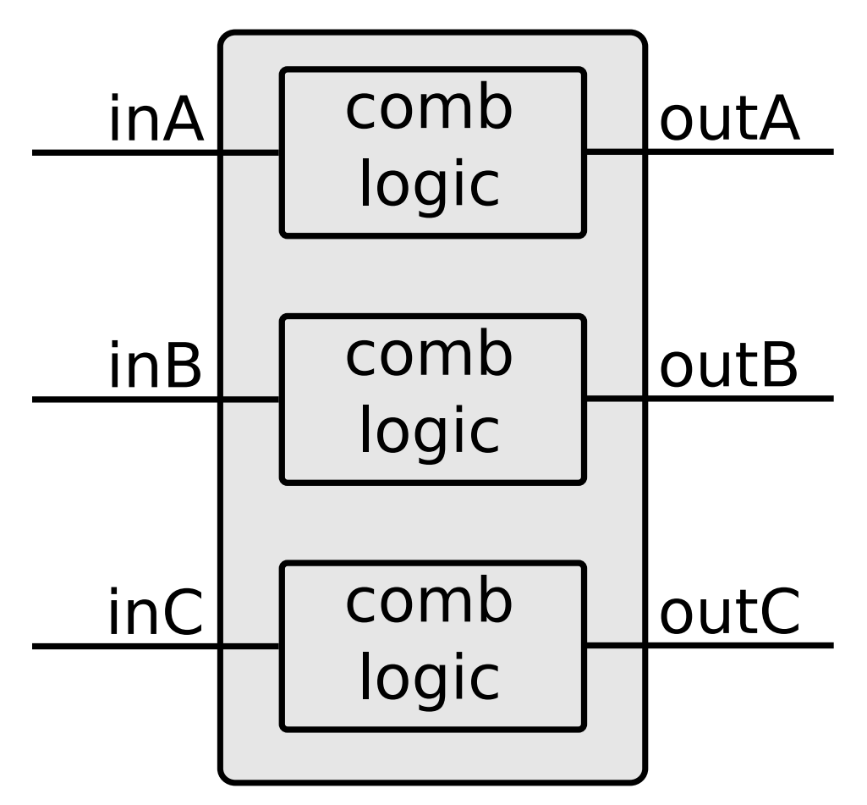
```
// tmrg default [triplicate|do_not_triplicate]
// tmrg triplicate netName
// tmrg do_not_triplicate netName
```

* TMRG automatizes “conversion” between triplicated and not triplicated signals:
  – if non triplicated signal is connected to a triplicated signal a passive fanout is added
  – if a triplicated signal is connected to a non triplicated signal a majority voter is added



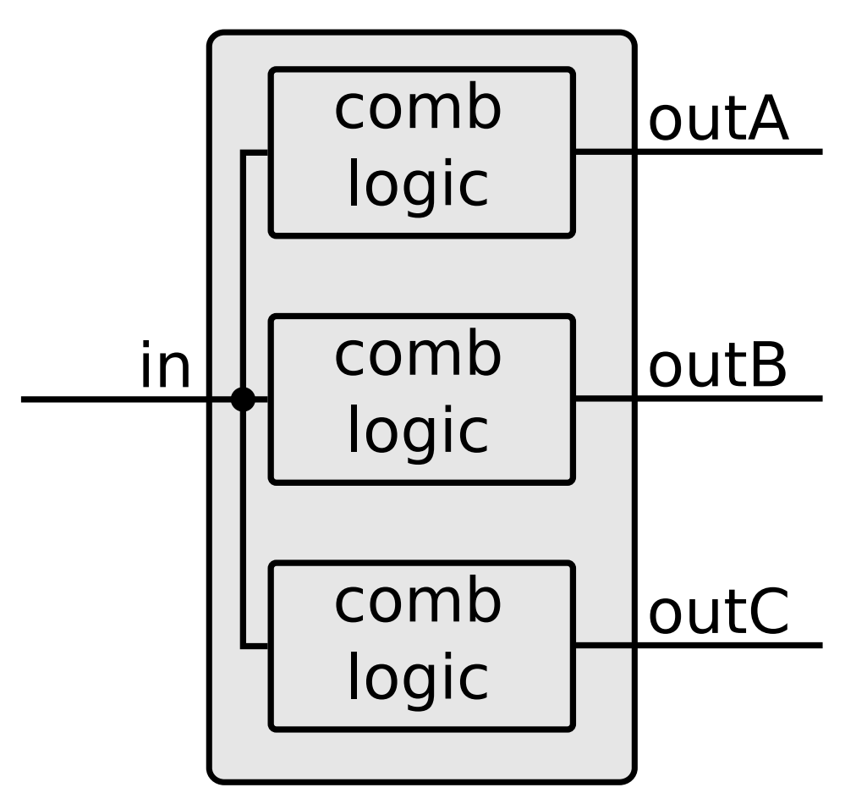
```
module comb (
  input in,
  output out
);
  wire combLogic;
  assign combLogic = ~in;
  assign out = combLogic;
endmodule
```

```
// tmrg default triplicate
```



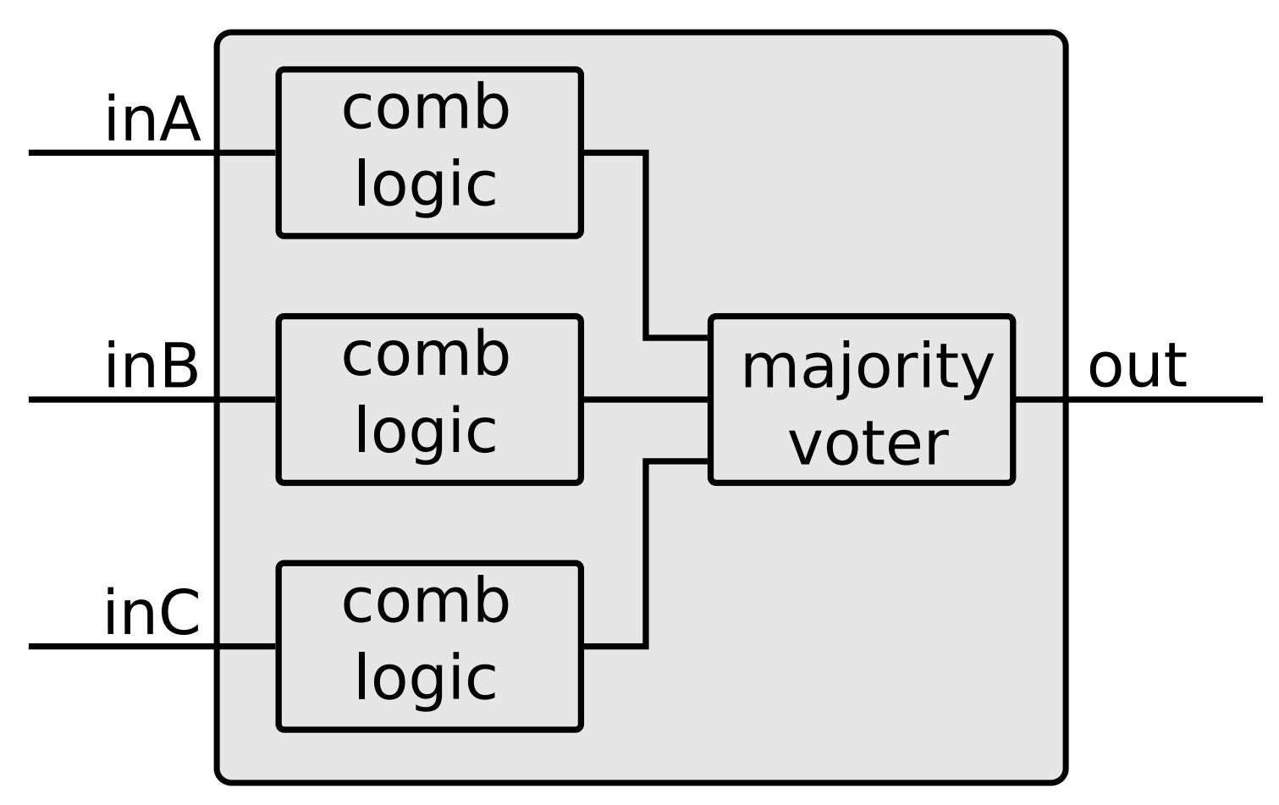
```
module comb02TMR(
  input inA,  inB,  inC,
  output outA, outB, outC
);
wire combLogicA;
wire combLogicB;
wire combLogicC;
assign combLogicA =  ~inA;
assign combLogicB =  ~inB;
assign combLogicC =  ~inC;
assign outA =  combLogicA;
assign outB =  combLogicB;
assign outC =  combLogicC;
endmodule
```

```
// tmrg default triplicate
// tmrg do_not_triplicate in
```



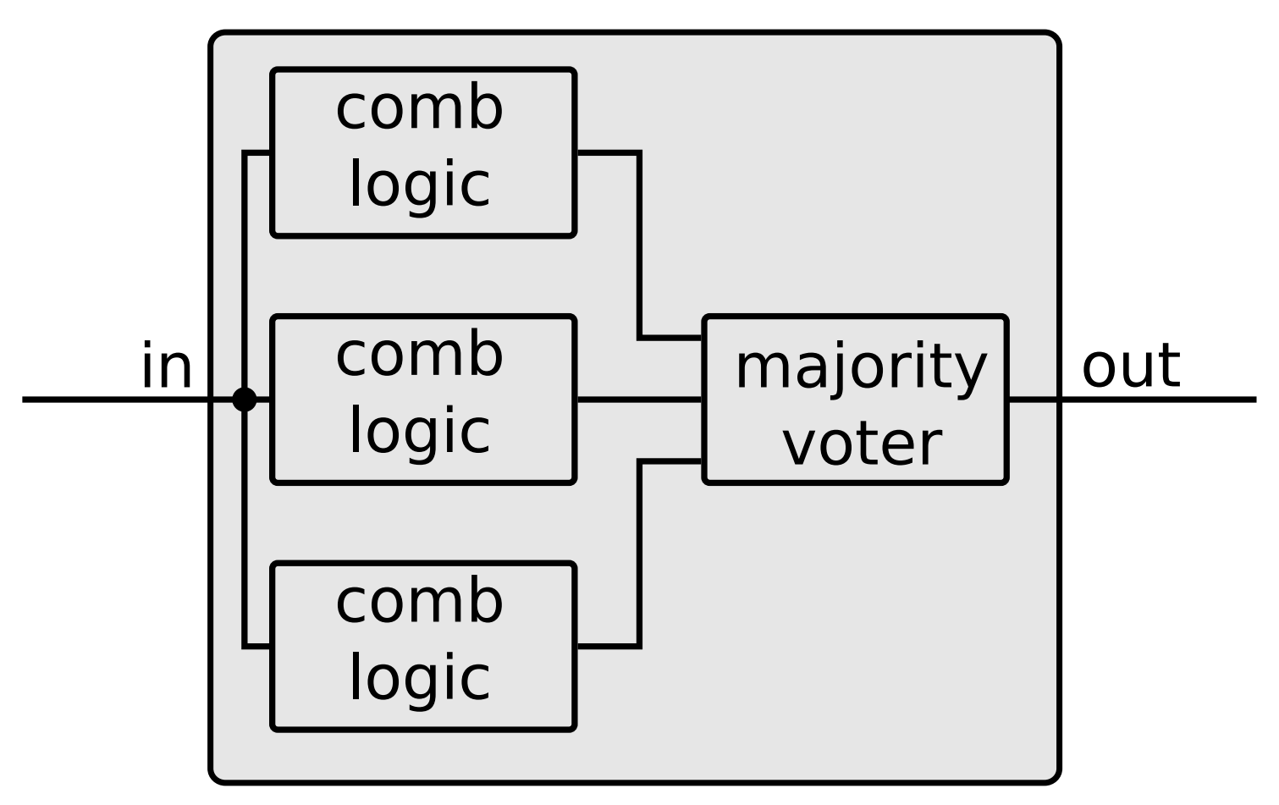
```
module comb03TMR(
  input in,
  output outA,  outB,  outC
);
wire inA,inB,inC;
wire combLogicA;
wire combLogicB;
wire combLogicC;
assign combLogicA =  ~inA;
assign combLogicB =  ~inB;
assign combLogicC =  ~inC;
assign outA =  combLogicA;
assign outB =  combLogicB;
assign outC =  combLogicC;

fanout inFanout (
  .in(in),  .outA(inA),
  .outB(inB),  .outC(inC));
endmodule
```

```
// tmrg default triplicate
// tmrg do_not_triplicate out
```



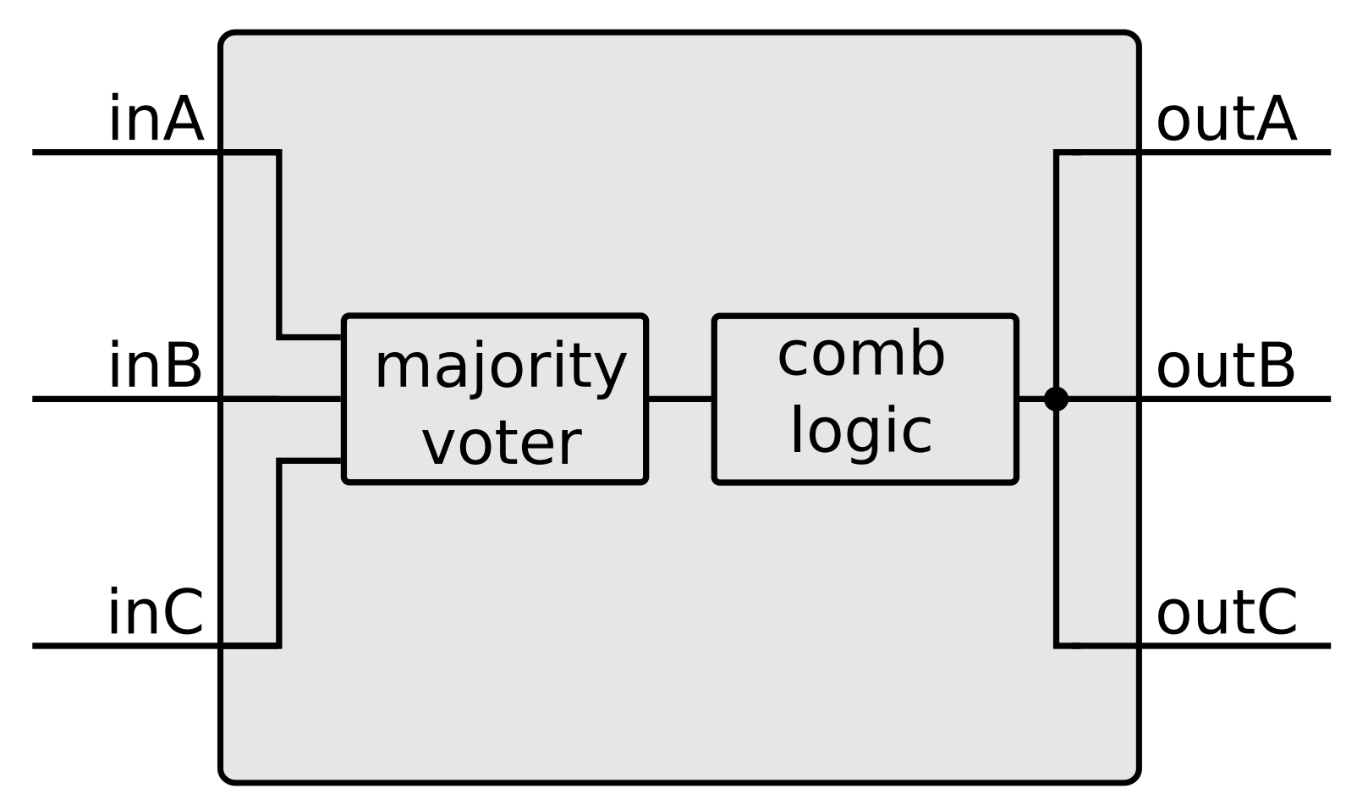
```
module comb04TMR(
  input inA,  inB,  inC,
  output out
);
wire combLogic;
wire combLogicA;
wire combLogicB;
wire combLogicC;
assign combLogicA =  ~inA;
assign combLogicB =  ~inB;
assign combLogicC =  ~inC;
assign out =  combLogic;

majorityVoter combLogicVoter (
  .inA(combLogicA),
  .inB(combLogicB),
  .inC(combLogicC),
  .out(combLogic)
);
endmodule
```

```
// tmrg default do_not_triplicate
// tmrg triplicate combLogic
```



```
module comb05TMR(
  input in,
  output out
);
wire inA,inB,inC;
wire combLogic;
wire combLogicA,combLogicB,combLogicC;
assign combLogicA =  ~inA;
assign combLogicB =  ~inB;
assign combLogicC =  ~inC;
assign out =  combLogic;

majorityVoter combLogicVoter (
  .inA(combLogicA),  .inB(combLogicB),
  .inC(combLogicC),  .out(combLogic));

fanout inFanout (
  .in(in),  .outA(inA),
  .outB(inB),  .outC(inC));
endmodule
```

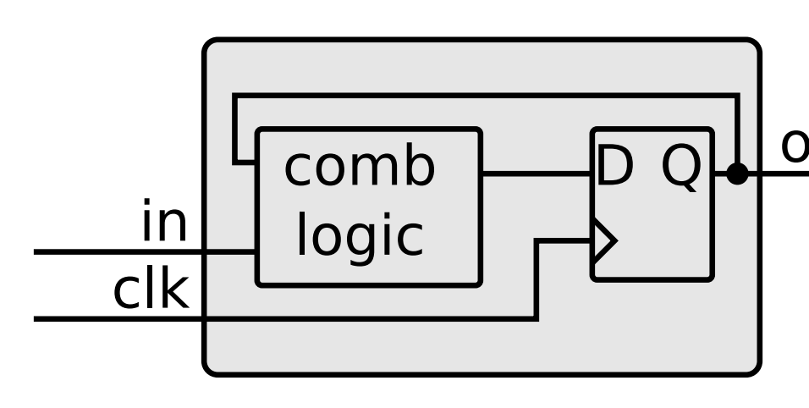
```
// tmrg default triplicate
// tmrg do_not_triplicate combLogic
```



```
module comb06TMR(
  input inA,  inB,  inC,
  output outA, outB, outC
);
wire in;
wire combLogic;
wire combLogicA,combLogicB,combLogicC;
assign combLogic =  ~in;
assign outA =  combLogicA;
assign outB =  combLogicB;
assign outC =  combLogicC;

majorityVoter inVoter (
  .inA(inA),  .inB(inB), .inC(inC),
  .out(in));

fanout combLogicFanout (
  .in(combLogic),  .outA(combLogicA),
  .outB(combLogicB),  .outC(combLogicC));
endmodule
```

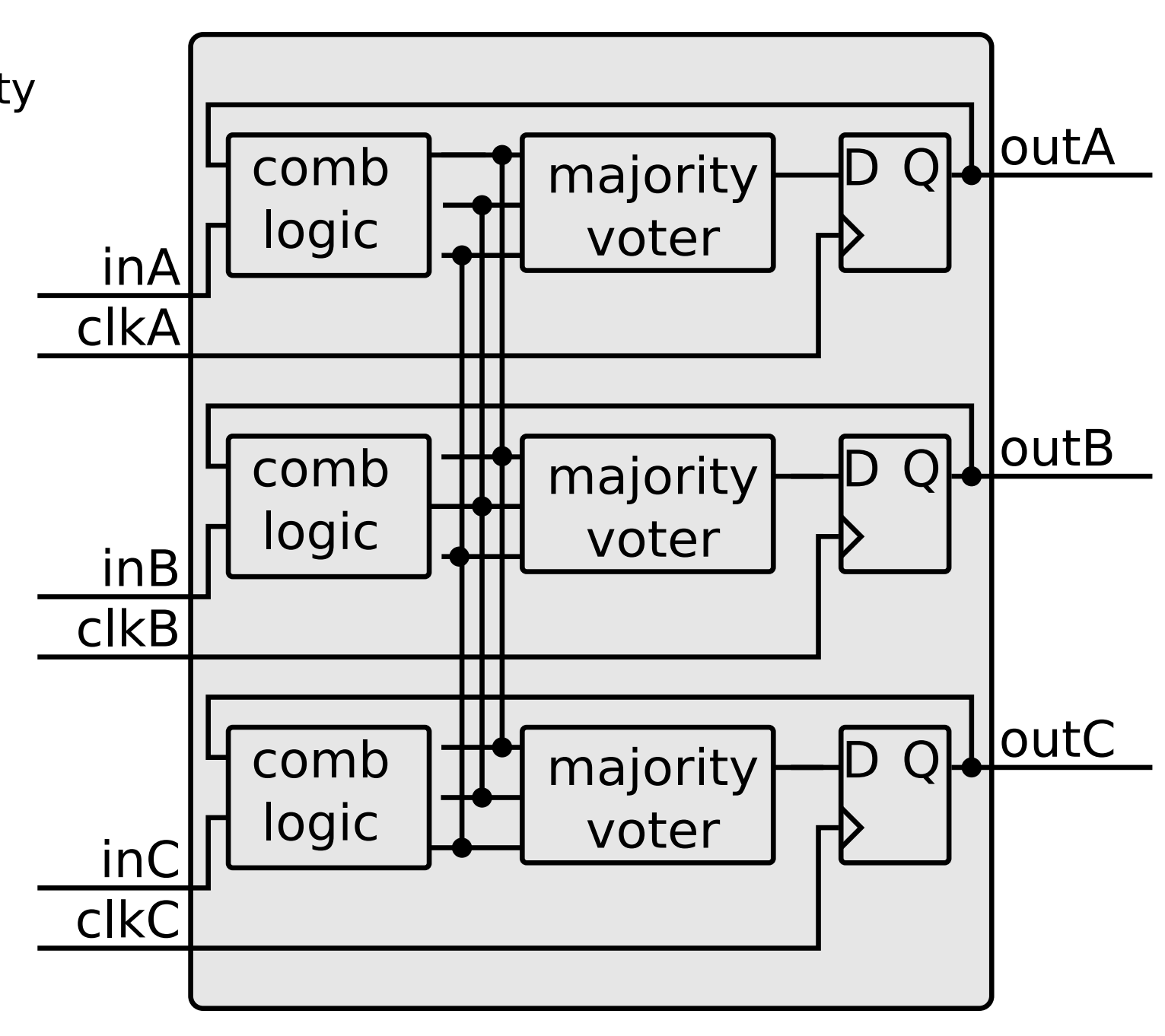
Triplicating Finite State Machine

```
module fsm (
  input in,
  output out,
  input clk);
  reg state;
  reg stateNext;
  assign out=state;

  always @(posedge clk)
    state <= stateNext;

  always @(state or in)
    stateNext = in ^ state;
endmodule
```

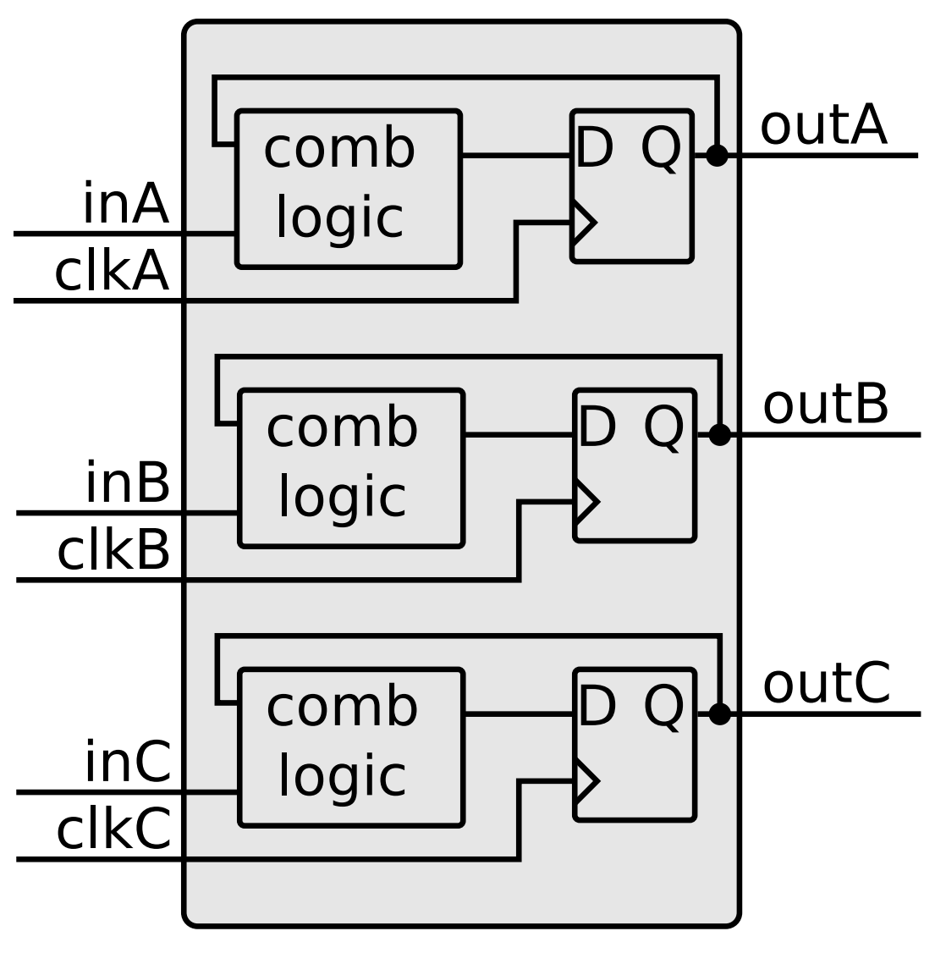


To generate **full TMR** (3 interconnected majority voters) a net declaration with a specific name (**Voted** postfix) has to be used:

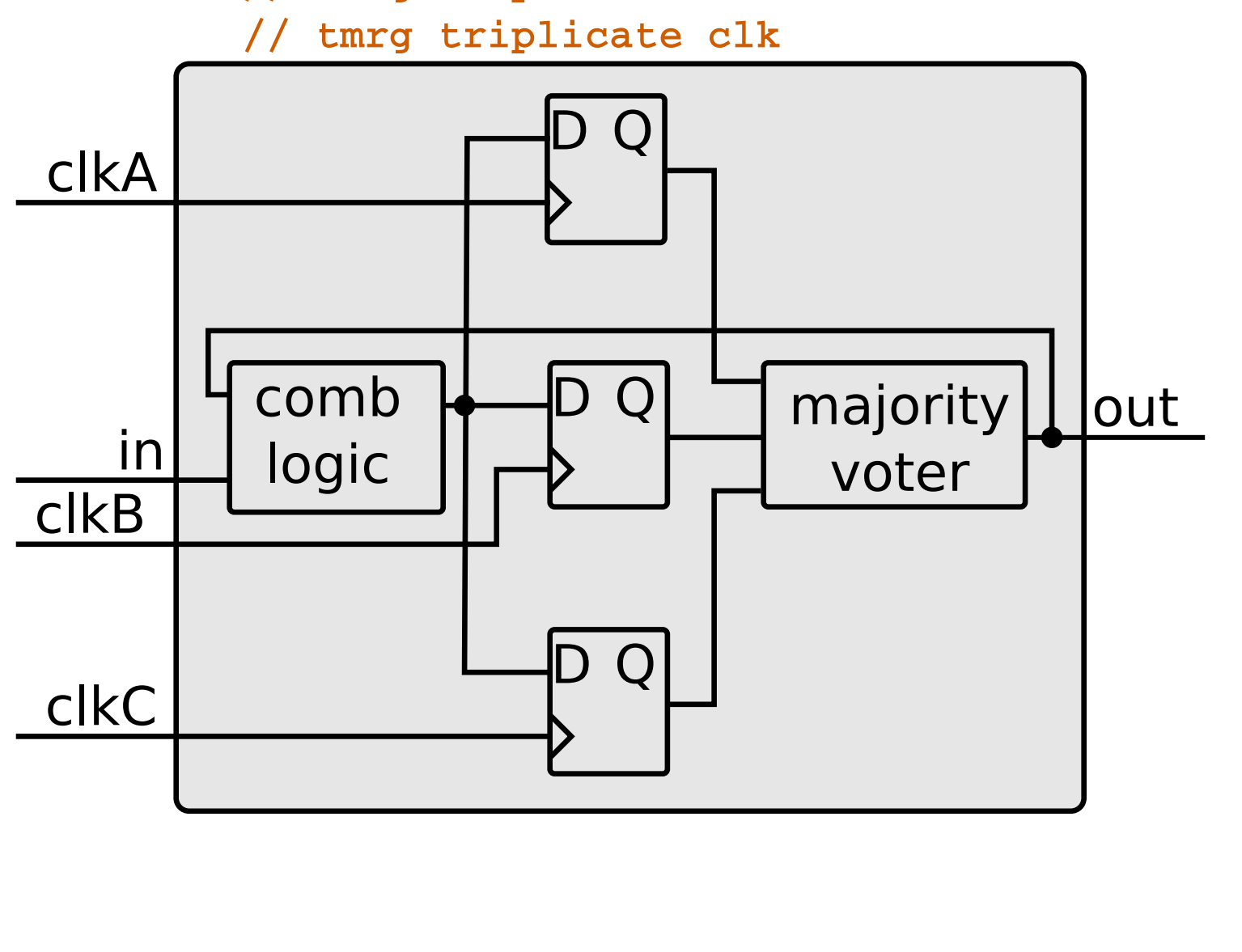
```
module fsm (
  input in,
  output out,
  input clk);
  // tmrg default triplicate
  reg state;
  reg stateNext;
  wire stateNextVoted=stateNext;
  assign out=state;

  always @(posedge clk)
    state <= stateNextVoted;

  always @(state or in)
    stateNext = in ^ state;
endmodule
```



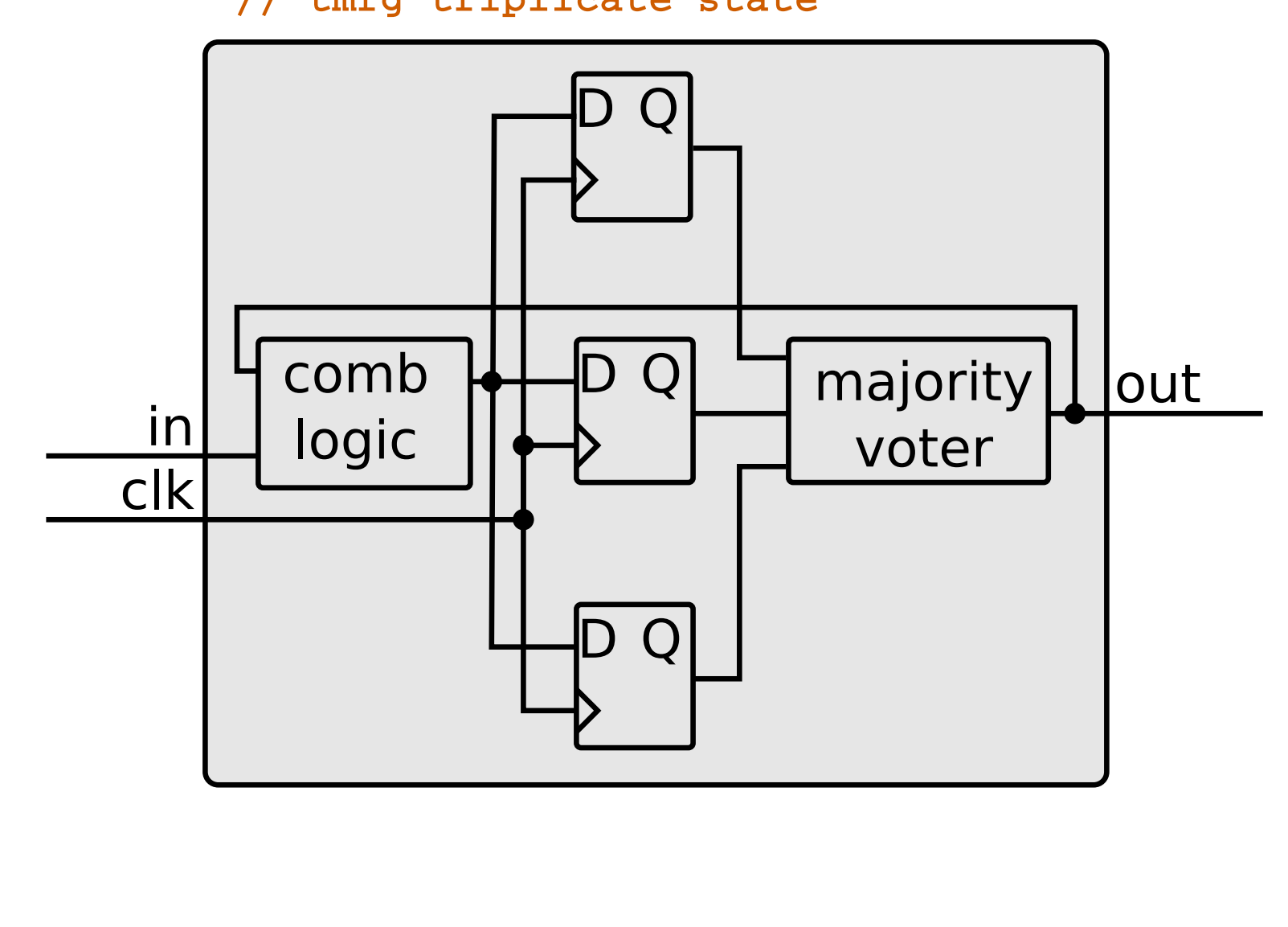
```
// tmrg default triplicate
```



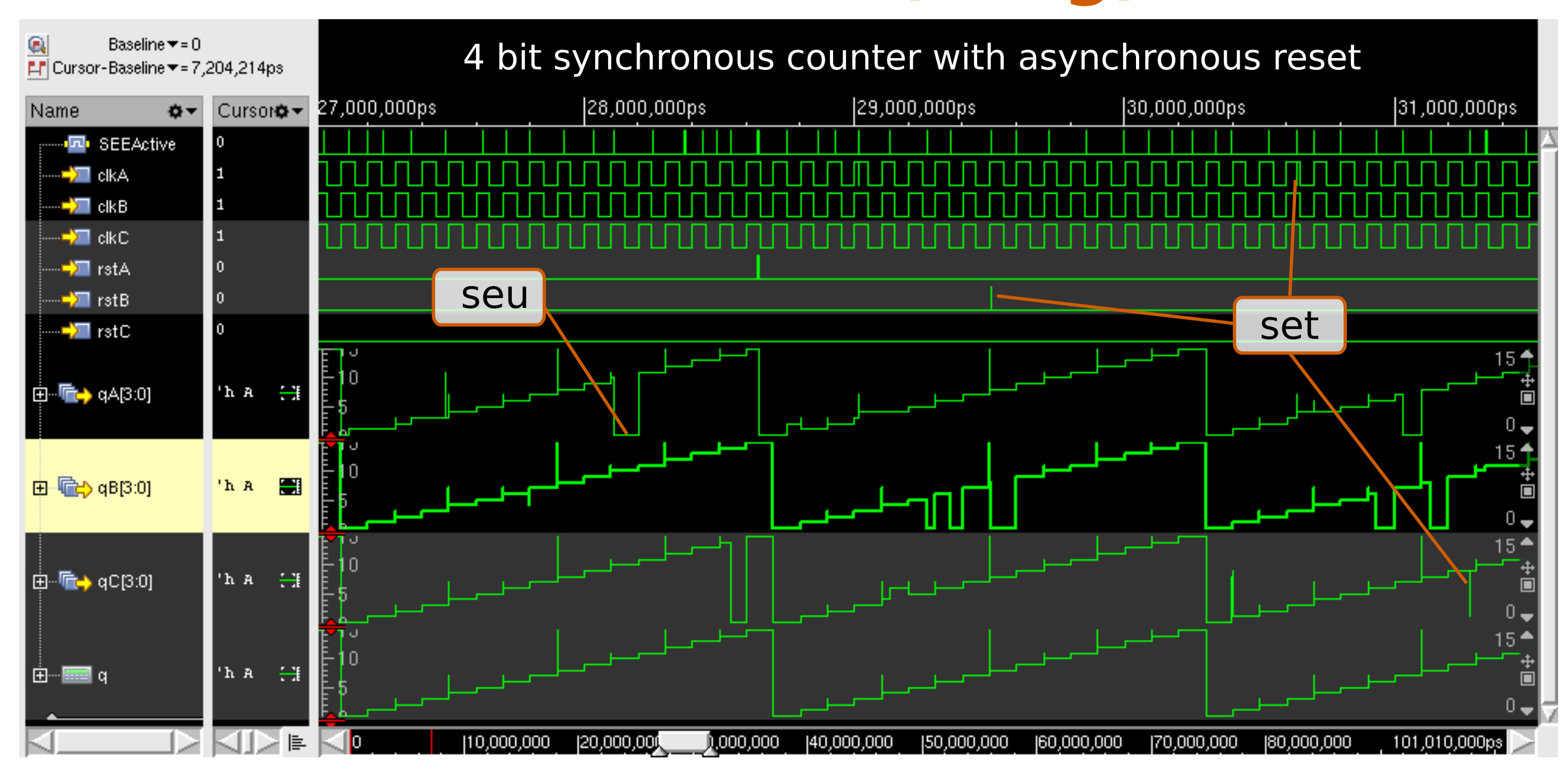
```
// tmrg default do_not_triplicate
// tmrg triplicate state
// tmrg triplicate clk
```



```
// tmrg default do_not_triplicate
// tmrg triplicate state
```

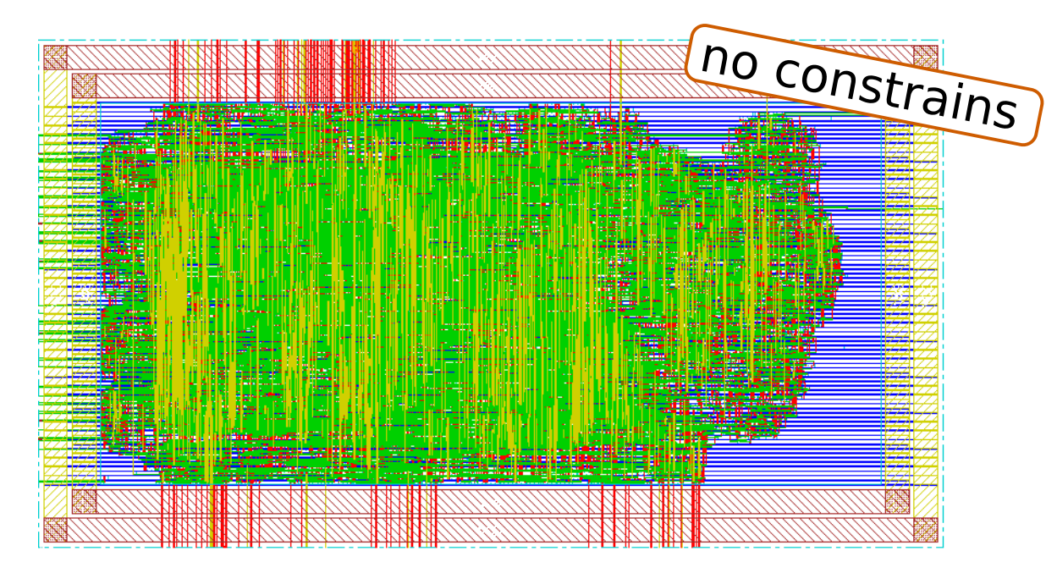


SEU / SET Generator (seeg)

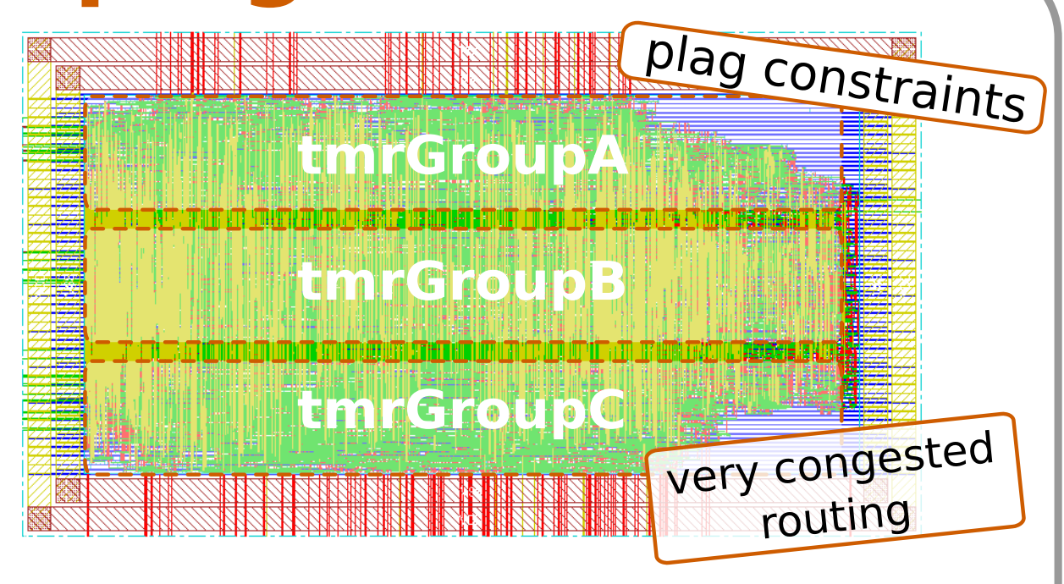


The desiger should verify that the design is immune to SEE. SEEG tool streamlines the verification process by generating a verilog file containing several verilog tasks, which can toggle nets (to simulate SET) or toggle flip-flops state (to simulate SEU).

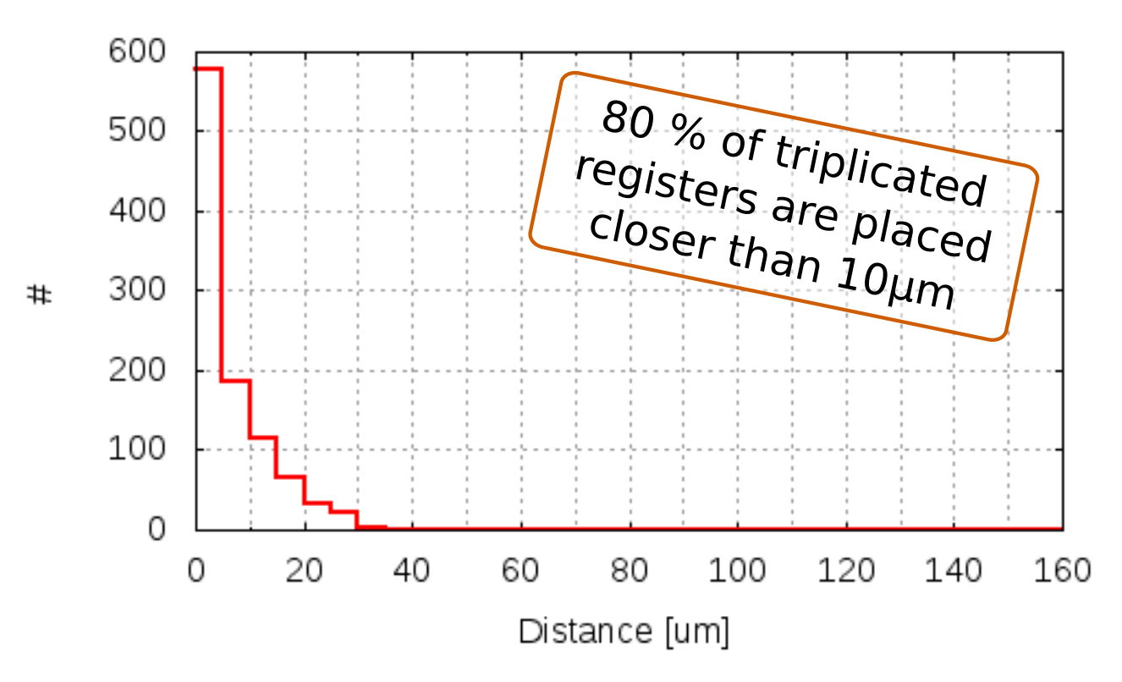
Placement Generator (plag)

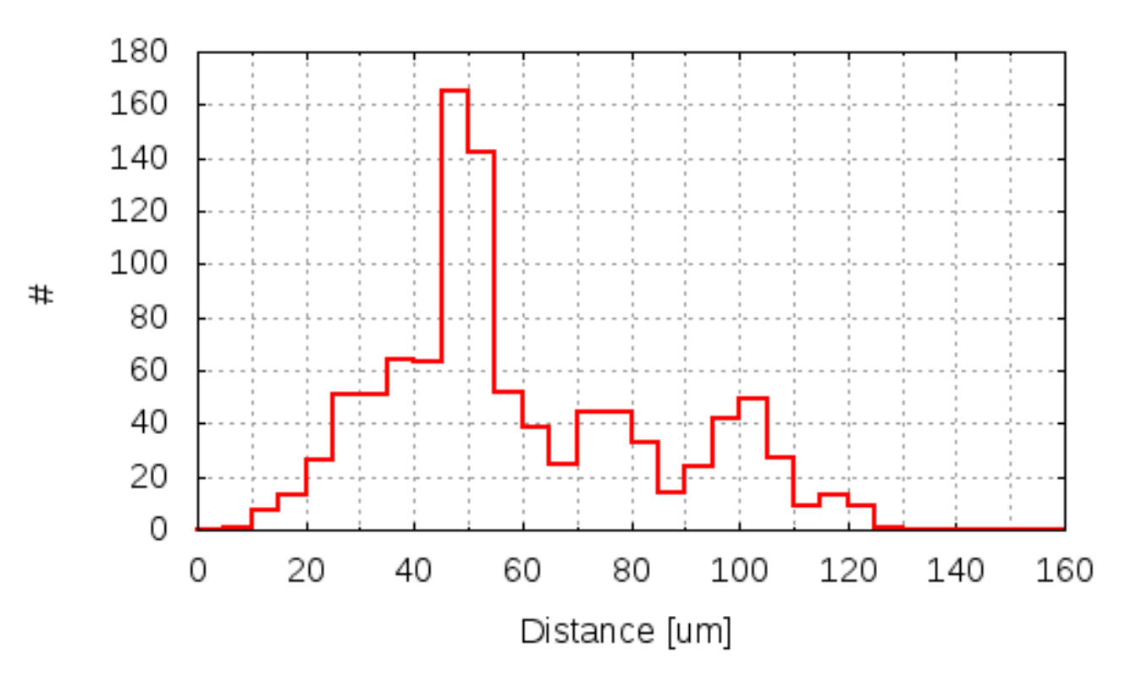


no constrains



plag constraints
tmrGroupA
tmrGroupB
tmrGroupC
very congested routing


80 % of triplicated registers are placed closer than 10µm
#
Distance [um]


#
Distance [um]