DOI 10.26886/2520-7474.6(50)2021.3

UDC: 51-37

AN ADVANCE OF THE THEORY OF THE WIDENED LONG FLIP-FLOP: PROPOSED IS FORMULATING IN THE LANGUAGE VERILOG

Mykola Stukach

https://orcid.org/0000-0002-2001-6937

e-mail: mykola.stukach@npp.nau.edu.ua

National Aviation University, Ukraine, Kyiv

The Widened long flip-flop in a quality of a circuit of a finite-state automaton is attractive as a solution with constant asymptotic complexity for the task of checking multiple stack-at faults that has an exponential complexity.

The article formulates the theory of Widened long flip-flop in the language of Verilog that provides an easy way to make all sorts of experiments with that theory, starting from validity check with respect to every product one plans to base on this theory. Besides, it reduces the cost of an outlay for designing and "making of metal" for any such product "to a very small amount".

Key words: finite-state automaton, multiple stuck-at faults, mission-critical application, formulating theory in Verilog, experiments with theory

Introduction. An interest to the "Theory of widened long flip-flop" can be formulated as the following: "It is obviously that there is NEVER the absolute certainty that a complicated digital apparatus works properly. So there is ALWAYS a desire to increase such certainty. ESPECIALLY when it is related to 'a critical-mission application'.

It must be said that ensuring testability of digital devices used to be devoted a lot of attention to. When the solution was developed, the interest to it deteriorated. No doubt, the latter solution has serious drawbacks.

These are generalities, in particulars there is a necessity to know the following: what is the attractiveness of Widened long flip-flop as a circuit of a finite-state automaton?

We can check easily (with 36-bit test, including all stimulus and responses) that our circuit of a finite-state automaton doesn't have any possible stuck-at faults (included multiple) which are not protected with a two-rail code itself. So, in the work mode the two-rail code will work properly. One may ask why it is so complicated. Instead one must ask why it is so simple. Because in fact we have got a solution to a constant asymptotic complexity for the task of checking multiple stack-at faults, itself having the exponential complexity.

Let's define what exactly that result was achieved due to:

1. A bizarre circuit was used: it is a "Long flip-flop", that has a bizarre behavior.
2. The inverters-less circuitry (when used are AND and OR gates but NOT gates are not) and the two-rail code was put to use.
3. Viewing windows were used for the purpose of observing logical variables. These windows make it possible to test delivery tracks for output vectors of the finite-state automaton.

Here are the details if required:

1. While looking for easy-to-test circuits the author of this article accidentally invented a "Long flip-flop" that is characterized by rare testability with respect to multiple stack-at faults.
2. After adding groups 4 and 5 of gates and some inputs to gates of group 1 and using defined restrictions to possible internal links of the

resulting circuit, the author successfully built the “Widened long flip-flop” that is capable of implementing a finite-state automaton (see Method IV [2, p.16]) and characterized by rare testability with respect to multiple stack-at faults (see Theorem 2 [2, p.11]).

3. In one doesn’t like that not all faults are detected during the test mode, whilst the rest faults are detected during the work mode (just when they appear)? One should build the finite-state automaton circuit not by Method IV, but by Method V [2, p. 27]). In than case the test will be somewhat larger and the circuit will have somewhat more gates, but all faults will be detectable during the test mode.

The theory of a Widened long flip-flop was set forth in details in [2] in the year 2007. It fitted all the criteria of academic style of formulating a theory. Unfortunately it didn’t have resonance with engineers and scientists [3]. For example, there was no mathematician to conform all the mathematics in my theory (i.e. 2 lemmas, 3 theorems and 2 methods of designing a circuit of a finite-state automaton).

The author of the present article reached a conclusion that the writing style he used when setting forth the teory of Widened long flip-flop is too hard to comprehend.

As a possible solution the theory could be formulated in language of Verilog. This language is clear for programs used for designing, modeling, testing and eventually manufacturing digital apparatuses. In fact it means turning from an academic formulating of the theory to an engineering one.

There are two tasks here:

THE FIRST TASK. To prove the theory of Widened long flip-flop without taking on hire an expensive professional mathematician, using the interactive program prover [3] instead.

THE SECOND TASK. To formulate the theory of Widened long flip-flop (or at least its key elements) in the language of Verilog [4; 5]. It at once

provides a way to make all sorts of experiments with our theory, starting from validity check with respect to every product one bases on our theory. Besides, the cost of an outlay for engineering and “making of metal” any apparatus will be appreciably reduced.

The present article is devoted to solving the second task. Three specific examples will be provided (in **Addenda**) to make everything explicit.

Further research. Firstly, to conform the mathematics of our theory with the help of the interactive program-prover of Coq or similar. Secondly, to make necessary experiments to find out scopes of effective practical application for the Widened long flip-flop. Fortunately, after formulating our theory in Verilog, it is both simple and convenient.

Essential feature of that solution (see [2, p.34, fig.V-5]) is using in the Widened long flip-flop only two-input gates AND. Due to this not only circuit solutions [2, pp.22-24] were changed to [2, pp.31-33], but also for the purpose of implementing multiplace AND instead of gates of groups 4 and 5 an adequate number of Long flip-flops [2, c.9, fig.II-4] were used , whilst emerged problem of convolution of results of testing of a great number of added Long flip-flops caused the need to use 4 additional Long flip-flops [2, p.9, fig.II-3].

The language of Verilog is powerful and convenient, but the task of formulating he Theory of a Widened Long Flip-Flop based on it is not trivial. On the whole, writing this article took one year. The 3 examples given here are sufficient for a lot of technical specialists to understand the substance of a proposed solution and to move on on their own. In addition, further examples are planned to be given as they are ready.

(*To be continued*)

Addenda

Example 1: “The Long Flip-Flop in the most common case”

```
/*****************************************************
A program in Verilog for “Figure II-1”. The latter is from
“https://arxiv.org/ftp/arxiv/papers/0808/0808.2602.pdf”
We propose using the statement of “defparam” to exchange the default
values of parameters
*****************************************************/
module AND2 #(parameter F_i1=2’b00, F_i2=2’b00, F_o=2’b00) (output
wire o, input wire i1, i2);
assign #1 o = (F_o[1]==1’b1)? F_o[2] : ((F_i1[1]==1’b1)? F_i1[2] : i1) &
((F_i2[1]==1’b1)? F_i2[2] : i2);
endmodule
module OR2 #(parameter F_i1=2’b00, F_i2=2’b00, F_o=2’b00) (output wire
o, input wire i1, i2);
assign #1 o = (F_o[1]==1’b1)? F_o[2] : ((F_i1[1]==1’b1)? F_i1[2] : i1) |
((F_i2[1]==1’b1)? F_i2[2] : i2);
endmodule
macromodule LFF #(parameter L=3, T_F=2’b00, iB_F=2’b00) (input wire
[1:N] iL, input wire [1:N] iR, input wire iT, iB, output wire oT, oB);
genvar N=L, Nplus1=L+1, i;
wire [1:N] o_g3, o_g2; wire [1:N+1] o_g1;
generate
// slice 1 of gates
AND2 g1_1(.o(o_g1[1]), .i1(iT), .i2(o_g3[1]));
assign #1 oT = (F_oT[1]==1’b1)? (F_oT[2] : o_g1[1]);
OR2 g3_1(.o(o_g3[1]), .i1(o_g1[2]), .i2(iL[1]));
OR2 g2_1(.o(o_g2[1]), .i1(o_g1[1]), .i2(iR[1]));
// slice i of gates
for (i=2; i<=N; i=i+1) begin: slice
AND2 g1_i(.o(o_g1[i]), .i1(o_g2[i-1]), .i2(o_g3[i]));
```

```
OR2 g3_i(.o(o_g3[i]), .i1(o_g1[i+1]), .i2(iL[i]));
OR2 g2_i(.o(o_g2[i]), .i1(o_g1[i]), .i2(iR[i]));
end
// slice Nplus1 of gates
AND2 g1_Nplus1(.o(o_g1[N+1]), .i1(o_g2[N]), .i2(iB));
assign #1 oB = (F_oB[1]==1'b1)? (F_oB[2] : o_g1[N+1]);
endgenerate
endmodule
```

Example 2: "The Widened Long Flip-Flop in the most common case"

```
/*****************************************************
A Verilog program for "Figure III-1" (as well as for "Figure III-2"). The last
two are from "https://arxiv.org/ftp/arxiv/papers/0808/0808.2602.pdf"
We propose using the statement of "defparam" to exchange the default
values of parameters
*****************************************************/
module AND_5 #(parameter slice, delay, F_i1, msb_i2, lsb_i2, parameter
_i2 [msb_i2: lsb_i2], parameter [1:2] F_i2 [msb_i2: lsb_i2], parameter _i3
[1:K], parameter [1:2] F_i3 [1: K], parameter F_o=2'b00) (output wire o,
input wire i1, input wire [msb_i2: lsb_i2] i2, input wire[1:K] i3);
wire R2=1'b1, R3=1'b1;
genvar j, k;
generate
for (j= msb_i2; j<= lsb_i2; j=j+1) begin: block i2
if(_i2[j] == 1'b1)
assign R2 = R2 & (F_i2[j][1]==1'b1)? F_i2[j][2] : i2[j]);
end
endgenerate
for (k= 1; k<= K; k=k+1) begin: block i3
```

```
if(_i3[j] == 1’b1)
assign R3 = R3 & (F_i3[k][1]==1’b1)? F_i3[k][2] : i3[k]);
end
endgenerate
assign #(delay) o=(F_o[1]=1’b1)? F_o[2] : & {i1, R2, R3};
endmodule
//
module AND_1 #(parameter slice, delay, F_i1, msb_i2, lsb_i2, parameter
_i2 [msb_i2: lsb_i2], parameter [1:2] F_i2 [msb_i2: lsb_i2], parameter _i3
[1:K], parameter [1:2] F_i3 [1:K], pRmeter F_o=2’b00) (output wire o, input
wire i1, input wire i2, input wire[1:K] i3);
wire R3=1’b1;
genvar k;
generate
for (k= 1; k<= K; k=k+1) begin: block i3
if(_i3[j] == 1’b1)
assign R3 = R3 & (F_i3[k][1]==1’b1)? F_i3[k][2] : i3[k]);
end
endgenerate
assign #(delay) o=(F_o[1]=1’b1)? F_o[2] : & {i1, i2, R3};
endmodule
//
module AND_4 #(
parameter slice, delay, F_i1, msb_i2, lsb_i2, parameter _i2 [msb_i2: lsb_i2],
parameter [1:2] F_i2 [msb_i2: lsb_i2], parameter _i3 [1:K], parameter [1:2]
F_i3 [1:K], parameter F_o=2’b00) (output wire o, input wire i1, input wire
[msb_i2: lsb_i2] i2, input wire[1:K] i3);
wire R2=1’b1, R3=1’b1;
genvar j, k;
```

```
generate
for (j= msb_i2; j<= lsb_i2; j=j+1) begin: block i2
if(_i2[j] == 1'b1)
assign R2 = R2 & (F_i2[j][1]==1'b1)? F_i2[j][2] : i2[j]);
end
endgenerate
for (k= 1; k<= K; k=k+1) begin: block i3
if(_i3[j] == 1'b1)
assign R3 = R3 & (F_i3[k][1]==1'b1)? F_i3[k][2] : i3[k]);
end
endgenerate
assign #(delay) o=(F_o[1]=1'b1)? F_o[2] : & {i1, R2, R3};
endmodule
module OR2 #(parameter F_i1=2'b00, F_i2=2'b00, F_o=2'b00) (output wire
o, input wire i1, i2);
assign #1 o = (F_o[1]==1'b1)? F_o[2] : ((F_i1[1]==1'b1)? F_i1[2] : i1) |
((F_i2[1]==1'b1)? F_i2[2] : i2);
endmodule
//
module VW #(parameter group=7, slice=i, case=1'b0, parameter [1:2]
fault=b'00) (output wire o, input wire i1); /*a Viewing Window for watching a
logical variable.
In case that parameter "case" equals 1'b1, this module is just a short wire,
otherwise this module is a three-parted chain consisting of a logic-input
amplifier, a light-emitting diod, and a logic-output amplifier. This chain is
perceived by a surrounding part of the circuit like a repeater of a logical
signal. Besides, it's light-emitting diode generates light in the event that the
repeating logic signal equals 'true'. */
generate
```

```
case(case)
1’b0: assign o= i1;
1’b1: assign #5 o = (fault[1]==1’b1)? fault[2] : i1;
endcase
endgenerate
endmodule
//
module Finpt #(parameter F_o=2’b00) (output wire o, input wire i1);
assign #1 o = (F_o[1]==1’b1)? F_o[2] : i1;
endmodule
module FinptK #(parameter [2:2*K] F_o ={K{2’b00}}) (output wire [1:K] o,
input wire [1:K] i1);
genvar k;
generate
for (k=1; k<=K; k=k+1)
assign #1 o[k] = (F_o[2*k-1]==1’b1)? F_o[2*k] : i1[k];
endgenerate
endmodule
//
macromodule #(parameter K=11, N=7, parameter [1:2] F_6 [1:K]={K{2’b0}},
parameter _vw_7[1:N]={N{1’b0}}, parameter [1:2] F_vw_7[1:N]={N{2’b0}},
parameter _vw_8[1:N]={N{1’b0}}, parameter [1:2] F_vw_8[1:N]={N{2’b0}},
F_OUTPUT13=2’b00, F_OUT14=2’b00) WLFF(input wire [1:K ]IN6, input
wire IN9, IN10, IN11, IN12, output wire OUT13, OUT14);
wire o_g5[1:N], o_g3[1:N], o_g2[1:N], o_4[1:N], o_v7[1:N], o_v8[1:N]; //1-bit
out
wire o_g1[1:N+1]; //1-bit out
wire [1:K] o_IN6; //K-bit out
genvar k, n, j, i, Nplus1=N+1;
```

```
assign #1 OUT13 = ((F_OUT13[1]==1'b1)? (F_OUT13[2] : o_g1[1]);
assign #1 OUT14 = ((F_OUT14[1]==1'b1)? (F_OUT14[2] : o_g1[N+1]);
Finpt F_IN_9 (.o(F_IN9)), .i1(IN9)); Finpt F_IN_10 (.o(F_IN10)), .i1(IN10));
Finpt F_IN_11 (.o(F_IN11)), .i1(IN11); Finpt F_IN_12 (.o(F_IN12)),
.i1(IN12));
FinpK FinpK_6(.o(F_IN6[1:K]), .i1(o_IN6[1:K]));
//SLICE 1 of gates OR2 g3_1(.o(o_g3[1]), .i1(o_g1[2]), .i2(o_g5[1]));
VW #(.group(8), .slice(1), .case(_vw_8[1]), .fault(F_vw_8[1]))
v8_1(.o(o_v8[1]), .i1(o_g3[1]));
OR2 g2_1(.o(o_g2[1]), .i1(o_g1[1]), .i2(o_g4[1]));
VW #(.group(7), .slice(1), .case(_vw_7[1]), .fault(F_vw_7[1])
v7_1(.o(o_v7[1]), .i1(o_g2[1]));
AND_5 #(.slice(1), .delay(log2(1+K)), .F_i1(2'b00), .msb_i2(0), .lsb_i2(0),
._i2(0), .F_i2(0), ._i3(K{1'b1}}), .F_i3({K{2'b00}}), .F_o(2'b00)) g5_1
(.o(o_g5[1]), .i1(F_IN12), .i3(F_IN6[1:K]));
AND_1 #(.slice(1), .delay(log2(2+K)), .F_i1(2'b00), .F_i2(2'b00),
._i3(K{1'b1}}), .F_i3({K{2'b00}}), .F_o(2'b00)) g1_1 (.o(o_g1[1]), .i1(F_IN9),
.i2(o_v8[1]), .i3(F_IN6[1:K]));
AND_4 #(.slice(1), .delay(log2(1+N-1+K)), .F_i1(2'b00), .msb_i2(2),
.lsb_i2(N), ._i2({N-1{1'b1}}), .F_i2({N-1{2'b00}}), .msb_i3(1), .lsb_i3(K),
._i3(K{1'b1}}), .F_i3({K{2'b00}}), F_o=2'b00) g4_1 (.o(o_g4[1]), .i1(F_IN11),
.i2(o_g2[2:N]), .i3(F_IN6[1:K]));
//SLICE i of gates
generate
for (i=2; i<N; i=i+1) begin: block
OR2 g3_i(.o(o_g3[i]), .i1(o_g1[i+1]), .i2(o_g5[i]));
VW #(.group(8), .slice(i), .case(_vw_8[i]), .fault(F_vw_8[i])) v8_i(.o(o_v8[i]),
.i1(o_g3[i]));
OR2 g2_i(.o(o_g2[i]), .i1(o_g1[i]), .i2(o_g4[i]));
```

```
VW #(.group(7), .slice(i), .case(_vw_7[i]), .fault(F_vw_7[i]) v7_i(.o(o_v7[i]),
.i1(o_g2[i]));
AND_5 #(.slice(i), .delay(log2(1+ i-1+K)), .F_i1(2'b00), .msb_i2(1), .lsb_i2(i-
1), ._i2({i-1{1'b1}}), .F_i2({i-1{2'b00}}), .msb_i3(1), .lsb_i3(K), ._i3(K{1'b1}}),
.F_i3({K{2'b00}}), .F_o(2'b00)) g5_i (.o(o_5[i]), .i1(F_IN12), .i2(o_g3[1:i-1]),
.i3(F_IN6[1:K]));
AND_1 #(.slice(i), .delay(log2(2+K)), .F_i1(2'b00), .F_i2(2'b00), .msb_i3(1),
.lsb_i3(K), ._i3(K{1'b1}}), .F_i3({K{2'b00}}), .F_o(2'b00)) g1_i (.o(o_g1[i]),
.i1(o_v7[i-1]), .i2(o_v8[i]), .i3(F_IN6[1:K]));
AND_4 #(.slice(i), .delay(log2(1+N-i+K)), .F_i1(2'b00), .msb_i2(i+1),
.lsb_i2(N), ._i2({N-i{1'b1}}), .F_i2({N-i{2'b00}}), .msb_i3(1), .lsb_i3(K),
._i3(K{1'b1}}), .F_i3({K{2'b00}}), F_o=2'b00) g4_i (.o(o_g4[i]), .i1(F_IN11),
.i2(o_g2[i+1:N]), .i3(F_IN6[1:K]));
end
endgenerate
//SLICE N of gates
OR2 g3_N(.o(o_g3[N]), .i1(o_g1[N]), .i2(o_g5[N]));
VW #(.group(8), .slice(N), .case(_vw_8[N]), .fault(F_vw_8[N]))
v8_N(.o(o_v8[N]), .i1(o_g3[N]));
OR2 g2_N(.o(o_g2[N]), .i1(o_g1[N]), .i2(o_g4[N]));
VW #(.group(7), .slice(N), .case(_vw_7[N]), .fault(F_vw_7[N])
v7_N(.o(o_v7[N]), .i1(o_g2[N]));
AND_5 #(.slice(N), .delay(log2(1+N-1+K)), .F_i1(2'b00), .msb_i2(1),
.lsb_i2(N-1), ._i2({N-1{1'b1}}), .F_i2({N-1{2'b00}}), .msb_i3(1), .lsb_i3(K),
._i3(K{1'b1}}), .F_i3({K{2'b00}}), .F_o(2'b00)) g5_N (.o(o_g5[N]),
.i1(F_IN12), .i2(o_g3[1:N-1]), .i3(F_IN6[1:K]));
AND_1 #(.slice(N), .delay(log2(2+K)), .F_i1(2'b00), .F_i2(2'b00),
.msb_i3(1), .lsb_i3(K), ._i3(K{1'b1}}), .F_i3({K{2'b00}}), .F_o(2'b00)) g1_N
(.o(o_g1[N]), .i1(o_v7[N-1]), .i2(o_v8[N]), .i3(F_IN6[1:K]));
```

```
AND_4 #(.slice(N), .delay(log2(1+K)), .F_i1(2'b00), .msb_i2(0), .lsb_i2(0),
._i2(0), .F_i2(0), .msb_i3(1), .lsb_i3(K), ._i3(K{1'b1}}), .F_i3({K{2'b00}}),
F_o=2'b00) g4_N (.o(o_g4[N]), .i1(F_IN11), .i3(F_IN6[1:K]));
//SLICE N+1 of gates
AND_1 #(.slice(N+1), .delay(log2(2+K)), .F_i1(2'b00), .F_i2(2'b00),
.msb_i3(1), .lsb_i3(K), ._i3(K{1'b1}}), .F_i3({K{2'b00}}), .F_o(2'b00))
g1_Nplus1 (.o(o_g1[N+1]), .i1(o_v7[N]), .i2(F_IN10), .i3(F_IN6[1:K]));
endmodule
```

Example 3: "The Widened Long Flip-Flop that is a result of Method IV"

```
/*****************************************************
A program in Verilog for "Figure IV-8b". The latter is from
"https://arxiv.org/ftp/arxiv/papers/0808/0808.2602.pdf"
We propose using the statement of "defparam" to exchange the default
values of parameters
*****************************************************/
module AND1 #(parameter F_i1=2'b00, F_o=2'b00) (output wire o, input
wire i1);
assign #1 o = (F_o[1]==1'b1)? F_o[2] : ((F_i1[1]==1'b1)? F_i1[2] : i1;
endmodule
module AND2 #(parameter F_i1=2'b00, F_i2=2'b00, F_o=2'b00) (output
wire o, input wire i1, i2);
assign #1 o = (F_o[1]==1'b1)? F_o[2] : ((F_i1[1]==1'b1)? F_i1[2] : i1) &
((F_i2[1]==1'b1)? F_i2[2] : i2);
endmodule
module AND3 #( parameter F_i1=2'b00, F_i2=2'b00, parameter [1:2] F_i3
[1:1]=2'b00, parameter F_o=2'b00) (output wire o, input wire i1, i2, input
wire i3 [1:1]);
```

```
assign #1 o = (F_o[1]==1'b1)? F_o[2] : ((F_i1[1]==1'b1)? F_i1[2] : i1) &
((F_i2[1]==1'b1)? F_i2[2] : i2) & ((F_i3[1][1]==1'b1)? F_i3[1][2] : i3[1]);
endmodule
module AND4 #( parameter F_i1=2'b00, F_i2=2'b00, parameter [1:2] F_i3
[1:2]={2 {2'b000}}, F_o=2'b00) (output wire o, input wire i1, i2, input wire
[1:2] i3);
assign #1 o = (F_o[1]==1'b1)? F_o[2] : ((F_i1[1]==1'b1)? F_i1[2] : i1) &
((F_i2[1]==1'b1)? F_i2[2] : i2) & ((F_i3[1][1]==1'b1)? F_i3[1][2] : i3[1]) &
((F_i3[2][1]==1'b1)? F_i3[2][2] : i3[2]);
endmodule
module OR2 #(parameter F_i1=2'b00, F_i2=2'b00, F_o=2'b00) (output wire
o, input wire i1, i2);
assign #1 o = (F_o[1]==1'b1)? F_o[2] : ((F_i1[1]==1'b1)? F_i1[2] : i1) |
((F_i2[1]==1'b1)? F_i2[2] : i2);
endmodule
module VW #(parameter F_i1=2'b00, F_o=2'b00) (output wire o, input wire
i1);
assign #5 o = (F_o[1]==1'b1)? F_o[2] : i1;
endmodule
module Finpt #(parameter F_o=2'b00) (output wire o, input wire i1);
assign #1 o = (F_o[1]==1'b1)? F_o[2] : i1;
endmodule
macromodule WLFF_ FigIV8b #(parameter N=23, K=6, parameter
F_OUTPUT13=2'b00, F_OUTPUT14=2'b00) (input wire [1:K] IN6, input
wire IN9, IN10, IN11, IN12, output wire OUT13, OUT14);
wire [1:N] o_g5, o_g3, o_g2, o_4, o_v7, o_v8; wire [1:N+1] o_g1;
Finpt F_IN6_1 (.o(F_IN6[1])), .i1(IN6[1]); //to feed "const 0" in the work
mode
Finpt F_IN6_2 (.o(F_IN6[2])), .i1(IN6[2]); //to feed "-c[1]" in the work mode
```

Finpt F_IN6_3 (.o(F_IN6[3])), .i1(IN6[3]); //to feed “c[1]” in the work mode
Finpt F_IN6_4 (.o(F_IN6[4])), .i1(IN6[4]); //to feed “-x[2]” in the work mode
Finpt F_IN6_5 (.o(F_IN6[5])), .i1(IN6[5]); //to feed “x[2]” in the work mode
Finpt F_IN6_6 (.o(F_IN6[6])), .i1(IN6[6]); //to feed “-x[1]” in the work mode
Finpt F_IN6_7 (.o(F_IN6[7])), .i1(IN6[7]); //to feed “x[1]” in the work mode
Finpt F_IN6_8 (.o(F_IN6[8])), .i1(IN6[8]); //to feed “-c[2]” in the work mode
Finpt F_IN6_9 (.o(F_IN6[9])), .i1(IN6[9]); //to feed “c[2]” in the work mode
Finpt F_IN_9 (.o(F_IN9)), .i1(IN9); //to feed “const 1” in the work mode
Finpt F_IN_10 (.o(F_IN10)), .i1(IN10); //to feed “const 1” in the work mode
Finpt F_IN_11 (.o(F_IN11)), .i1(IN11); //to feed “const 1” in the work mode
Finpt F_IN_12 (.o(F_IN12)), .i1(IN12); //to feed “const 1” in the work mode
AND1 g5_1(.o(o_g5[1]), .i1(IN12)); OR2 g3_1(.o(o_g3[1]), .i1(o_g1[2]), .i2(o_g5[1])); AND3 g1_1(.o(o_g1[1]), .i1(IN9), .i2(o_g3[1]), .i3(IN6[8]));
assign #1 OUT13 = ((F_OUT13[1]==1’b1)? (F_OUT13[2] : o_g1[1]);
OR2 g2_1(.o(o_g2[1]), .i1(o_g1[1]), .i2(o_g4[1])); AND2 g4_1(.o(o_g4[1]), .i1(IN11), .i2(o_g2[18]));
AND2 g5_2(.o(o_g5[2]), .i1(IN12), .i2(IN6[1])); OR2 g3_2(.o(o_g3[2]), .i1(o_g1[3]), .i2(o_g5[2])); VW v8_2(.o(o_v8[2]), .i1(o_g3[2])); /**********to watch “y1” in the work mode**********/ AND2 g1_2(.o(o_g1[2]), .i1(o_g2[1]), .i2(o_v8[2])); OR2 g2_2(.o(o_g2[2]), .i1(o_g1[2]), .i2(o_g4[2])); AND3 g4_2(.o(o_g4[2]), .i1(IN11), .i2(o_g2[18]), .i3(IN6[9]));
AND1 g5_3(.o(o_g5[3]), .i1(IN12)); OR2 g3[3](.o(o_g3[3]), .i1(o_g1[4]), .i2(o_g5[3])); AND2 g1_3(.o(o_g1[3]), .i1(o_g2[2]), .i2(o_g3[3])); OR2 g2_3(.o(o_g2[3]), .i1(o_g1[3]), .i2(o_g4[3])); AND1 g4_3(.o(o_g4[3]), .i1(IN11));
AND1 g5_4(.o(o_g5[4]), .i1(IN12)); OR2 g3_4(.o(o_g3[4]), .i1(o_g1[5]), .i2(o_g5[4])); AND3 g1_4(.o(o_g1[4]), .i1(o_g2[3]), .i2(o_g3{4]), .i3(IN6[8]));
OR2 g2_4(.o(o_g2[4]), .i1(o_g1[4]), .i2(o_g4[4])); AND2 g4_4(.o(o_g4[4]), .i1(IN11), .i2(o_g2[21]));

AND2 g5_5(.o(o_g5[5]), .i1(IN12), .i2(IN6[1])); OR2 g3_5(.o(o_g3[5]), .i1(o_g1[6]), .i2(o_g5[5])); AND2 g1_5(.o(o_g1[5]), .i1(o_g2[4]), .i2(o_g3[5])); OR2 g2_5(.o(o_g2[5])), .i1(o_g1[5]), .i2(o_g4[5])); AND3 g4_5(.o(o_g4[5]), .i1(IN11), .i2(o_g2[21]), .i3(IN6[9]));
AND1 g5_6(.o(o_g5[6]), .i1(IN12)); OR2 g3_6(.o(o_g3[6]), .i1(o_g1[7]), .i2(o_g5[6])); AND2 g1_6(.o(o_g1[6]), .i1(o_g2[5]), .i2(o_g3[6])); OR2 g2_6(.o(o_g2[6]), .i1(o_g1[6]), .i2(o_g4[6])); AND1 g4_6(.o(o_g4[6]), .i1(IN11));
AND3 #(.K(2)) g5_7(.o(o_g5[7]), .i1(IN12), .i2(o_g3[5]), .i3({IN6[5], IN6[6]})); OR2 g3_7(.o(o_g3[7]), .i1(o_g1[8]), .i2(o_g5[7])); AND3 g1_7(.o(o_g1[7]), .i1(o_g2[6]), .i2(o_g3[7]), .i3(IN6[1])); OR2 g2_7(.o(o_g2[7]), .i1(o_g1[7]), .i2(o_g4[7])); AND1 g4_7(.o(o_g4[7]), .i1(IN11));
AND2 g5_8(.o(o_g5[8]), .i1(IN12), .i2(IN6[7])); OR2 g3_8(.o(o_g3[8]), .i1(o_g1[9]), .i2(o_g5[8])); AND3 g1_8(.o(o_g1[8]), .i1(o_g2[7]), .i2(o_g3[8]), .i3(IN6[1])); OR2 g2_8(.o(o_g2[8]), .i1(o_g1[8]), .i2(o_g4[8])); AND1 g4_8(.o(o_g4[8]), .i1(IN11));
AND2 g5_9(.o(o_g5[9]), .i1(IN12), .i2(IN6[4])); OR2 g3_9(.o(o_g3[9]), .i1(o_g1[10]), .i2(o_g5[9])); AND2 g1_9(.o(o_g1[9]), .i1(o_g2[8]), .i2(o_g3[9])); OR2 g2_9(.o(o_g2[9]), .i1(o_g1[9]), .i2(o_g4[9])); AND1 g4_9(.o(o_g4[9]), .i1(IN11));
AND2 g5_10(.o(o_g5[10]), .i1(IN12), .i2(o_g3[2])); OR2 g3_10(.o(o_g3[10]), .i1(o_g1[11]), .i2(o_g5[10])); AND2 g1_10(.o(o_g1[10]), .i1(o_g2[9]), .i2(o_g3[10])); OR2 g2_10(.o(o_g2[10]), .i1(o_g1[10]), .i2(o_g4[10])); AND1 g4_10(.o(o_g4[10]), .i1(IN11));
AND2 g5_11(.o(o_g5[11]), .i1(IN12), .i2(o_g3[7])); OR2 g3_11(.o(o_g3[11]), .i1(o_g1[12]), .i2(o_g5[11])); VW v8_11(.o(o_v8[11]), .i1(o_g3[11]));
/**********to watch “y2” in the work mode**********/ AND3 g1_11(.o(o_g1[11]), .i1(o_g2[10]), .i2(o_v8[11]), .i3(IN6[1])); OR2

g2_11(.o(o_g2[11]), .i1(o_g1[11]), .i2(o_g4[11])); AND1 g4_11(.o(o_g4[11]), .i1(IN11));
AND3 g5_12(.o(o_g5[12]), .i1(IN12), .i2(IN6[7]) .i3(IN6[5])); OR2 g3_12(.o(o_g3[12]), .i1(o_g1[13]), .i2(o_g5[12])); AND2 g1_12(.o(o_g1[12]), .i1(o_g2[11]), .i2(o_g3[12])); OR2 g2_12(.o(o_g2[12]), .i1(o_g1[12]), .i2(o_g4[12])); AND1 g4_12(.o(o_g4[12]), .i1(IN11));
AND3 g5_13(.o(o_g5[13]), .i1(IN12), .i2(o_g3[8]), .i3(IN6[4])); OR2 g3_13(.o(o_g3[13]), .i1(o_g1[14]), .i2(o_g5[13])); AND3 g1_13(.o(o_g1[13]), .i1(o_g2[12]), .i2(o_g3[13]), .i3(IN6[1])); OR2 g2_13(.o(o_g2[13]), .i1(o_g1[13]), .i2(o_g4[13])); AND1 g4_13(.o(o_g4[13]), .i1(IN11));
AND3 g5_14(.o(o_g5[14]), .i1(IN12), .i2(o_g3[8]), .i3(IN6[6])); OR2 g3_14(.o(o_g3[14]), .i1(o_g1[15]), .i2(o_g5[14])); AND2 g1_14(.o(o_g1[14]), .i1(o_g2[13]), .i2(o_g3[14])); OR2 g2_14(.o(o_g2[14]), .i1(o_g1[14]), .i2(o_g4[14])); AND1 g4_14(.o(o_g4[14]), .i1(IN11));
AND3 g5_15(.o(o_g5[15])), .i1(IN12), .i2(o_g3[5]), .i3(o_g3[2])); OR2 g3_15(.o(o_g3[15]), .i1(o_g1[16]), .i2(o_g5[15])); VW v8_15(.o(o_v8[15]), .i1(o_g3[15])); /**********to watch “y3” in the work mode**********/ AND3 g1_15(.o(o_g1[15]), .i1(o_g2[14]), .i2(o_v8[15]), .i3(IN6[1])); OR2 g2_15(.o(o_g2[15]), .i1(o_g1[15]), .i2(o_g4[15])); AND1 g4_15(.o(o_g4[15]), .i1(IN11));
AND3 g5_16(.o(o_g5[16]), .i1(IN12), .i2(o_g3[13]), .i3(o_g3[11])); OR2 g3_16(.o(o_g3[16]), .i1(o_g1[17]), .i2(o_g5[16])); AND2 g1_16(.o(o_g1[16]), .i1(o_g2[15]), .i2(o_g3[16])); OR2 g2_16(.o(o_g2[16]), .i1(o_g1[16]), .i2(o_g4[16])); AND1 g4_16(.o(o_g4[16]), .i1(IN11));
AND1 g5_17(.o(o_g5[17]), .i1(IN12)); OR2 g3_17(.o(o_g3[17]), .i1(o_g1[18]), .i2(o_g5[17])); AND3 g1_17(.o(o_g1[17]), .i1(o_g2[16]), .i2(o_g3[17]), .i3(IN6[1])); OR2 g2_17(.o(o_g2[17]), .i1(o_g1[17]), .i2(o_g4[17])); AND1 g4_17(.o(o_g4[17]), .i1(IN11));

AND3 g5_18(.o(o_g5[18]), .i1(IN12), .i2(o_g3[7]), .i3(IN6[3])); OR2 g3_18(.o(o_g3[18]), .i1(o_g1[19]), .i2(o_g5[18])); AND2 g1_18(.o(o_g1[18]), .i1(o_g2[17]), .i2(o_g3[18])); OR2 g2_18(.o(o_g2[18]), .i1(o_g1[18]), .i2(o_g4[18])); AND2 g4_18(.o(o_g4[18]), .i1(IN11), .i2(IN6[1]));
AND2 g5_19(.o(o_g5[19]), .i1(IN12), .i2(o_g3[7])); OR2 g3_19(.o(o_g3[19]), .i1(o_g1[20]), .i2(o_g5[19])); AND2 g1_19(.o(o_g1[19]), .i1(o_g2[18]), .i2(o_g3[19])); OR2 g2_19(.o(o_g2[19]), .i1(o_g1[19]), .i2(o_g4[19])); AND1 g4_19(.o(o_g4[19]), .i1(IN11));
AND1 g5_20(.o(o_g5[20]), .i1(IN12)); OR2 g3_20(.o(o_g3[20]), .i1(o_g1[21]), .i2(o_g5[20])); AND3 g1_20(.o(o_g1[20]), .i1(o_g2[19]), .i2(o_g3[20]), .i3(IN6[2])); OR2 g2_20(.o(o_g2[20]), .i1(o_g1[20]), .i2(o_g4[20])); AND1 g4_20(.o(o_g4[20]), .i1(IN11));
AND3 g5_21(.o(o_g5[21]), .i1(IN12), .i2(o_g3[8]), .i3(IN6[3])); OR2 g3_21(.o(o_g3[21]), .i1(o_g1[22]), .i2(o_g5[21])); AND2 g1_21(.o(o_g1[21]), .i1(o_g2[20]), .i2(o_g3[21])); OR2 g2_21(.o(o_g2[21]), .i1(o_g1[21]), .i2(o_g4[21])); AND2 g4_21(.o(o_g4[21]), .i1(IN11), .i2(IN6[1]));
AND2 g5_22(.o(o_g5[22]), .i1(IN12), .i2(o_g3[8])); OR2 g3_22(.o(o_g3[22]), .i1(o_g1[23]), .i2(o_g5[22])); AND2 g1_22(.o(o_g1[22]), .i1(o_g2[21]), .i2(o_g3[22])); OR2 g2_22(.o(o_g2[22]), .i1(o_g1[22]), .i2(o_g4[22])); AND1 g4_22(.o(o_g4[22]), .i1(IN11));
AND3 g1_23(.o(o_g1[23]), .i1(o_g2[22]), .i2(IN10), .i3(IN6[2]));
assign #1 OUT14 = ((F_OUT14[1]==1'b1)? (F_OUT14[2] : o_g1[23]);
endmodule

References:

1. Boundary scan. Retrieved from https://en.wikipedia.org/wiki/Boundary_scan (2021, December, 21).

2. Stukach, Nick (2008). Easily testable logical networks based on a "widened long flip-flop". Retrieved from

https://arxiv.org/ftp/arxiv/papers/0808/0808.2602.pdf. (2021, December, 21).

3. Stukach N.D. (2019). Primenenie interaktivnoy programmy-pruvera tipa Coq mozhet stat' novym standartom aprobatsii prakticheski vazhnykh teoreticheskikh rabot. Retrieved from https://naukajournal.org/index.php/Paradigm/article/download/1784/1839 (2021, December, 21). [in Russian].

4. Michael D. Chiletti (2003). Advanced digital design with the Verilog HDL. – New Delhi: Prentice-Hall of India, 2005, ISBN-81-203-2756-X, www.phindia.com.

5. Soloviov V.V. (2014). Osnovy yazyka proektirovaniia tsyfrovoi apparatury Verilog. – Moskva: Goriachaia liniia – Telekom, 2014, ISBN 978-5-9912-0353-1, www.techbook.ru [in Russian]

Citation: Mykola Stukach (2021). AN ADVANCE OF THE THEORY OF THE WIDENED LONG FLIP-FLOP: PROPOSED IS FORMULATING IN THE LANGUAGE VERILOG. Frankfurt. TK Meganom LLC. Paradigm of knowledge. 6(50). doi: 10.26886/2520-7474.6(50)2021.3