Verifikasi Rangkaian Terpadu Pengecek Paritas Menggunakan UVM

by Artikel 1

Verifikasi Rangkaian Terpadu Pengecek Paritas Menggunakan UVM

PENGUSUL

Widianto, ST., MT

D3 TEKNOLOGI ELEKTRONIKA

DIREKTORAT PENDIDIKAN VOKASI

UNIVERSITAS MUHAMMADIYAH MALANG

2023

Kata Pengantar

Alhamdulillah, peneliti mengucapkan syukur kehadhirat Alloh Subhanahu Wa Ta'ala atas limpahan nikmat sehat dhahir dan bathin karena telah menyelesaikan penelitian dengan judul "*testbench* untuk mendeteksi kerusakan hubung singkat di dalam rangkaian terpadu register geser yang didesain menggunakan SystemVerilog".

Kerusakan yang berupa hubung singkat ke suplai tegangan dan ke *ground* bisa terjadi pada input atau output rangkaian penyusun di dalam rangkaian terpadu pengecek paritas. Rangkaian penyusun dari rangkaian terpadu pengecek paritas tediri 12 gerbang inverter, 11 gerbang buffer, 20 gerbang AND, dan 5 gerbang NOR.

Tujuan dalam penelitian ini adalah merancang UVM (*Universal Verification Methology*) yang diajukan untuk memverifikasi kerusakan hubung singkat ke suplai tegangan dan ke ground yang terjadi di dalam rangkaian terpadu pengecek paritas. UVM terdiri dari *testbench* dan DUT (*Design Under Tes*t). *Testbench* tersusun dari beberapa komponen, yaitu: *interface*, *environment*, dan *testbench top*. Sedangkan DUT atau desain yang akan diuji yaitu rangkaian terpadu pengecek paritas. *Testbench* dan DUT dirancang menggunakan Bahasa SystemVerilog dan disimulasikan menggunakan Questasim 2021.1. Peneliti berharap supaya penelitian ini dapat diakui sebagai hak cipta sebagai bagian dari Hak Kekayaan Intelektual (HKI).

Malang, Oktober 2023
Peneliti

Widianto, ST., MT

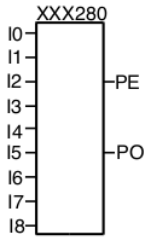
Daftar Isi

Daftar Gambar

Daftar Tabel

Rangkaian Terpadu Pengecek Paritas

Rangkaian terpadu pengecek paritas dapat ditemukan dalam SCSI (*Small Computer System Interface*) dan PCI (*Peripheral Component Interconnect*) [2]. Simbol logika rangkaian pengecek paritas bisa dilihat dalam Gambar 1. Deskripsi pin dari rangkaian pengecek paritas bisa dilihat dalam Tabel 1.



Gambar 1 Simbol logika rangkaian pengecek paritas

Tabel 1 Deskripsi Pin rangkaian pengecek paritas

Simbol	Deskripsi
I0 – I8	Input
PE	Output ganjil
PO	Output genap

Rangkaian pengecek paritas dapat digunakan mengecak paritas di input. Untuk mengecek paritas, rangkaian pengecek paritas bekerja berdasarkan deskripsi fungsi yang ditunjukkan dalam Tabel 2.

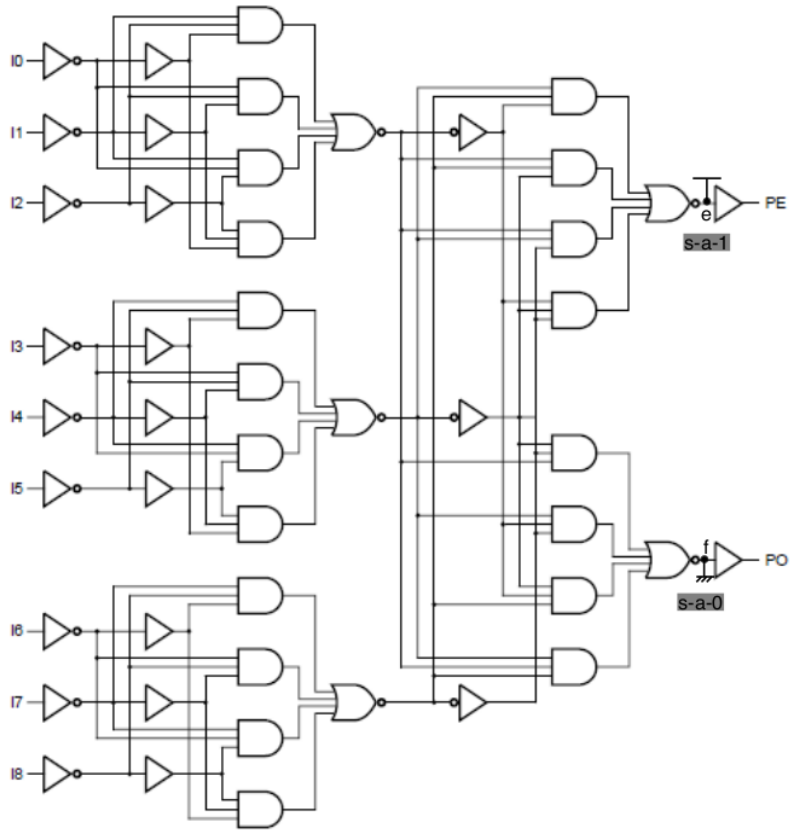
Tabel 2 Tabel Fungsi Rangkaian pengecek paritas

Input (I0-I8)	Output	
Jumlah bit 1	**PE**	**PO**
Ganjil	1	0
Genap	0	1

Kerusakan Hubung Singkat ke Suplai Tegangan dan ke Ground

Kerusakan hubung singkat ke suplai tegangan atau stuck-at-1 (s-a-1) menyebabkan nilai logika pada input atau output rangkaian bernilai tinggi atau 1[1]. Contoh hubung singkat s-a-1 yang terjadi pada titik *e* dalam rangkaian penyusun rangkaian terpadu pengecek paritas yang ditunjukkan dalam Gambar 2. Rangkaian penyusun dari rangkaian terpadu pengecek paritas tediri 12 gerbang inverter, 11 gerbang buffer, 20 gerbang AND, dan 5 gerbang NOR. Sebagaimana ditunjukkan dalam Gambar 2, nilai logika PE (output ganjil) pada output logika akan tetap bernilai 1 apapun nilai logika dari inputnya.

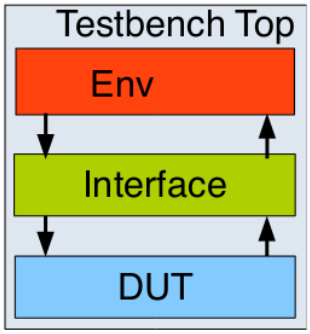
Sedangkan nilai logika input atau output rangkaian akan bernilai rendah atau 0 ketika terjadi hubung singkat ke ground atau stuck-at-0 (s-a-0) pada input atau output rangkaian[1]. Contoh hubung singkat s-a-0 yang terjadi pada titik f dalam rangkaian penyusun rangkaian terpadu pengecek paritas bisa ditunjukkan dalam Gambar 2. Dalam Gambar 2, tanpa memperdulikan nilai logika dari inputan, maka nilai logika pada output logika PO (output genap) akan tetap bernilai 0.



Gambar 2 Contoh hubung singkat s-a-0 dan s-a-1

UVM

UVM (*Universal Verification Methodology*) diajukan untuk memverifikasi rangkaian terpadu pengecek paritas dari kerusakan hubung singkat ke suplai tegangan dan ke *groun*d yang terjadi pada input atau output rangkaian penyusun di dalam rangkaian terpadu tersebut. UVM terdiri dari *testbench* dan DUT. Desain UVM bisa dilihat seperti dalam Gambar 3.



Gambar 3 Desain UVM

Testbench tersusun dari beberapa komponen, yaitu: *interface*, *environment*, dan *testbench top*. Sedangkan DUT atau desain yang akan diuji yaitu rangkaian terpadu pengecek paritas. *Environment* berisi berbagai sinyal stimulus input untuk digerakkan ke DUT. *Environment* juga berfungsi untuk memeriksa apakah sinyal output yang dihasilkan oleh DUT yang tertangkap sudah sesuai dengan spesifikasinya. *Interface* berfungsi untuk menggerakkan sinyal dari *environment* yang masuk ke DUT atau untuk mengamati sinyal yang keluar dari DUT. *Testbench top* berisi *interface*, *environment*, dan juga DUT

Testbench yang diajukan dapat mendeteksi kerusakan hubung singkat yang terjadi pada input dan output dari rangkaian penyusun pengecek paritas. Sedangkan DUT adalah rangkaian terpadu pengecek biner. *Testbench* dan DUT dirancang menggunakan SystemVerilog dan disimulasikan menggunakan Questasim 2021.1.

SystemVerilog

SystemVerilog merupakan Bahasa deskripsi perangkat keras atau *Hardware Description Languange* (HDL) yang dapat digunakan untuk memprogram desain blok digital yang terdiri dari rangkaian kombinasional dan sekuensial yang bisa disimulasikan dan bisa disintesis ke dalam hardware, misalnya adalah FPGA (*Field Programmable Gate Array*) dan ASIC (*Application Specific Integrated Circuit*)[16],[17]. SystemVerilog juga dapat memverifikasi desain blok digital tersebut apakah dapat bekerja sesuai dengan spesifikasi yang diharapkan. SystemVerilog telah didukung fitur pemrograman yang berorientasi pada objek atau OOP (*Object Oriented Programming*), sehingga memudahkan dalam memverifkasi desain blok digital yang kompleks.

Untuk mendesain testbench dan rangkaian terpadu menggunakan Bahasa SystemVerilog, ada beberapa fitur yang perlu diperhatikan yaitu tipe data, arah pengaturan,

proses, komunikasi, antarmuka, kelas, constraint, jenis-jenis konstruksi, cakupan fungsi, dan assertion.

SystemVerilog memiliki tujuh tipe data, yaitu: logic, bit, byte, shortint, int, longint, shortreal. Tipe data logic memiliki empat keadaan data yaitu 0, 1, x (tidak diketahui), dan z (impedansi tinggi). Tipe data logic dapat digunakan dalam perintah blok procedural yaitu always and initial dan juga perintah assign.

Tipe data bit memiliki dua keadaan data yaitu 0 dan 1. Perintah tipe data bit banyak digunakan di testbench untuk merepresantikan bit tunggal dan untuk menyimpan beberapa bit dari MSB (most significant bit) ke LSB (least significant bit).

Fitur arah pengaturan memiliki beberapa konstruksi, yaitu: forever, repeat, while, for, do while, for each, break, continue, if-else-if, case, events, function, tasks, blocking dan nonblocking. Sedangkan fitur proses memiliki konstruksi, yaitu fork join, fork join any, fork join-none.

Fitur komunikasi memiliki konstruksi, yaitu: event, semaphores, dan mailbox. Untuk fitur antarmuka memiliki konstruksi, yaitu: interface, clocking block dan modport. Fitur kelas memiliki konstruksi, yaitu: class, this, super, typedef, extern, rand, randomize, virtual. Fitur constraint memiliki konstruksi, yaitu: inside, foreach, solve before. Sedangkan fitur jenis-jenis konstruksi memiliki konstruksi, yaitu: program, cast, package. Untuk fitur cakupan fungsi memiliki konstruksi, yaitu: mode, coverpoint, covergroup, bins, iff, start, stop. Fitur assertion memiliki konstruksi, yaitu: assert, assume, cover, restrict, sequence, property.

DUT

Rangakaian terpadu pengecek paritas sebagai DUT sebagiamana ditunjukkan dalam Gambar 2 bisa dituliskan dalam bentuk kode SystemVerilog. Gambar 4 merupakan hasil penulisan kode DUT dalam Bahasa SystemVerilog.

```
module and_gate (A,B,C,Y);
input A,B,C;
output Y;
assign Y = A&B&C;
endmodule

module not_gate (A,Y);
input A;
output Y;
assign Y = ~A;
endmodule

module nor_gate (A,B,C,D,Y);
input A,B,C,D;
output Y;
```

```
assign Y = ~(A|B|C|D);
endmodule

module buffer_gate (A,Y);
output Y;
input A;
assign Y = A;
endmodule

module pg280 (I0,I1,I2,I3,I4,I5,I6,I7,I8,PE,PO);
input I0,I1,I2,I3,I4,I5,I6,I7,I8;
output PE,PO;
logic [8:0] out0;
logic [8:0] out1;
logic [11:0] out2;
logic [2:0] out3;
logic [2:0] out4;
logic [7:0] out5;
logic [1:0] out6;
logic [1:0] out7;

not_gate not0 (.A(I0), .Y(out0[0]));
not_gate not1 (.A(I1), .Y(out0[1]));
not_gate not2 (.A(I2), .Y(out0[2]));
not_gate not3 (.A(I3), .Y(out0[3]));
not_gate not4 (.A(I4), .Y(out0[4]));
not_gate not5 (.A(I5), .Y(out0[5]));
not_gate not6 (.A(I6), .Y(out0[6]));
not_gate not7 (.A(I7), .Y(out0[7]));
not_gate not8 (.A(I8), .Y(out0[8]));

buffer_gate buf0 (.A(out0[0]), .Y(out1[0]));
buffer_gate buf1 (.A(out0[1]), .Y(out1[1]));
buffer_gate buf2 (.A(out0[2]), .Y(out1[2]));
buffer_gate buf3 (.A(out0[3]), .Y(out1[3]));
```

```
buffer_gate buf4 (.A(out0[4]), .Y(out1[4]));
buffer_gate buf5 (.A(out0[5]), .Y(out1[5]));
buffer_gate buf6 (.A(out0[6]), .Y(out1[6]));
buffer_gate buf7 (.A(out0[7]), .Y(out1[7]));
buffer_gate buf8 (.A(out0[8]), .Y(out1[8]));

and_gate and0 (.A(out0[1]), .B(out0[2]), .C(out1[0]), .Y(out2[0]));
and_gate and1 (.A(out0[0]), .B(out0[2]), .C(out1[1]), .Y(out2[1]));
and_gate and2 (.A(out0[1]), .B(out0[0]), .C(out1[2]), .Y(out2[2]));
and_gate and3 (.A(out1[2]), .B(out1[1]), .C(out1[0]), .Y(out2[3]));

and_gate and4 (.A(out0[4]), .B(out0[5]), .C(out1[3]), .Y(out2[4]));//
and_gate and5 (.A(out0[3]), .B(out0[5]), .C(out1[4]), .Y(out2[5]));//
and_gate and6 (.A(out0[4]), .B(out0[3]), .C(out1[5]), .Y(out2[6]));//
and_gate and7 (.A(out1[5]), .B(out1[4]), .C(out1[3]), .Y(out2[7]));//
and_gate and8 (.A(out0[7]), .B(out0[8]), .C(out1[6]), .Y(out2[8]));//
and_gate and9 (.A(out0[6]), .B(out0[8]), .C(out1[7]), .Y(out2[9]));//
and_gate and10 (.A(out0[7]), .B(out0[6]), .C(out1[8]), .Y(out2[10]));//
and_gate and11 (.A(out1[8]), .B(out1[7]), .C(out1[6]), .Y(out2[11]));//

nor_gate nor0 (.A(out2[0]), .B(out2[1]), .C(out2[2]), .D(out2[3]), .Y(out3[0]));//
nor_gate nor1 (.A(out2[4]), .B(out2[5]), .C(out2[6]), .D(out2[7]), .Y(out3[1]));//
nor_gate nor2 (.A(out2[8]), .B(out2[9]), .C(out2[10]), .D(out2[11]), .Y(out3[2]));//

not_gate not9 (.A(out3[0]), .Y(out4[0]));//
not_gate not10 (.A(out3[1]), .Y(out4[1]));//
not_gate not11 (.A(out3[2]), .Y(out4[2]));//

and_gate and12 (.A(out3[1]), .B(out3[2]), .C(out4[0]), .Y(out5[0]));//
and_gate and13 (.A(out3[0]), .B(out3[2]), .C(out4[1]), .Y(out5[1]));//
and_gate and14 (.A(out3[0]), .B(out3[1]), .C(out4[2]), .Y(out5[2]));//
and_gate and15 (.A(out4[0]), .B(out4[1]), .C(out4[2]), .Y(out5[3]));//

and_gate and16 (.A(out4[1]), .B(out4[2]), .C(out3[0]), .Y(out5[4]));//
and_gate and17 (.A(out3[1]), .B(out4[0]), .C(out4[1]), .Y(out5[5]));//
```

```
and_gate and18 (.A(out4[1]), .B(out4[0]), .C(out3[2]), .Y(out5[6]));//
and_gate and19 (.A(out3[1]), .B(out3[0]), .C(out3[2]), .Y(out5[7]));//

nor_gate nor3 (.A(out5[0]), .B(out5[1]), .C(out5[2]), .D(out5[3]), .Y(out6[0]));//
nor_gate nor4 (.A(out5[4]), .B(out5[5]), .C(out5[6]), .D(out5[7]), .Y(out6[1]));//

buffer_gate buf9 (.A(out6[0]), .Y(PE));
buffer_gate buf10 (.A(out6[1]), .Y(PO));

endmodule
```

Gambar 4 Kode DUT

UVM Package

UVM *package* berisi sekumpulan deklarasi yang terkait dengan penyertaan file. Selain itu *package* dapat terdiri dari parameter, task, function, tipe data yang dapat ditentukan pengguna seperti enum, struct, union. Ini juga dapat terdiri dari urutan dan deklarasi properti. Kode UVM package bisa dilihat dalam Gambar 5.

```
import uvm_pkg::*;
`include "uvm_macros.svh"
```

Gambar 5 Kode UVM *Package*

Environment

Environment berisi berbagai sinyal stimulus input untuk digerakkan ke DUT. *Environment* juga berfungsi untuk memeriksa apakah sinyal output yang dihasilkan oleh DUT yang tertangkap sudah sesuai dengan spesifikasinya. Kode environment bisa dilihat seperti dalam Gambar 6. *Interface* tersusun dari beberapa konstruksi, yaitu: *virtual*, *function, assert*.

```
Class env extends uvm_env;
  virtual pg280_if      m_pg280_if;
 virtual clk_if m_clk_if;

 7
 function new(string name, uvm_component parent = null);
   super.new(name, parent);
 endfunction
```

```
  function void connect_phase(uvm_phase phase);
    `uvm_info("LABEL", "Started connect phase.", UVM_HIGH);
      assert(uvm_resource_db#(virtual pg280_if)::read_by_name(
      get_full_name(), "pg280_if", m_pg280_if));
    assert(uvm_resource_db#(virtual clk_if)::read_by_name(
      get_full_name(), "clk_if", m_clk_if));
    `uvm_info("LABEL", "Finished connect phase.", UVM_HIGH);
  endfunction: connect_phase

  task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    `uvm_info("LABEL", "Started run phase.", UVM_HIGH);
    begin

      integer I0 = 1'b1,I1 = 1'b0,I2=1'b1,I3=1'b0,I4=1'b1,I5=1'b0,I6=1'b1,I7=1'b0,I8=1'b1;


      m_pg280_if.I0 <= I0;
      m_pg280_if.I1 <= I1;
      m_pg280_if.I2 <= I2;
      m_pg280_if.I3 <= I3;
      m_pg280_if.I4 <= I4;
      m_pg280_if.I5 <= I5;
      m_pg280_if.I6 <= I6;
      m_pg280_if.I7 <= I7;
      m_pg280_if.I8 <= I8;
      `uvm_info("RESULT", $sformatf("%0d+ %0b+ %0d+ %0d+ %0d+ %0d+ %0d+ %0d+
%0d = %0d, %0d",
        I0,I1,I2,I3,I4,I5,I6,I7,I8,m_pg280_if.PE,m_pg280_if.PO), UVM_LOW);
    end
    `uvm_info("LABEL", "Finished run phase.", UVM_HIGH);
    phase.drop_objection(this);
  endtask: run_phase
endclass
```

Gambar 6 Kode *Environment*

Interface

Interface berfungsi untuk menggerakkan sinyal dari *environment* yang masuk ke DUT atau untuk mengamati sinyal yang keluar dari DUT. Kode *interface* bisa dilihat dalam Gambar 7. *Interface* tersusun dari beberapa konstruksi, yaitu: *interface*, *logic*.

```
interface pg280_if();
  logic I0,I1,I2,I3,I4,I5,I6,I7,I8;
  logic PE,PO;

endinterface

interface clk_if();
  logic tb_clk;

  initial tb_clk <= 0;

  always #1 tb_clk = ~tb_clk;
endinterface
```

Gambar 7 Kode *Interface*

Testbench top

Testbench top digunakan untuk menguji DUT. Kode *testbench* bisa dilihat dalam Gambar 8. *Testbench top* tersusun dari beberapa konstruksi, yaitu: *module, bit, test*.

```
module pg280_uvm_tba2;
  reg tb_clk;
  env environment;

  always #1 tb_clk = ~tb_clk;

  clk_if m_clk_if();
  pg280_if m_pg280_if();
   pg280
u0(.I0(m_pg280_if.I0),.I1(m_pg280_if.I1),.I2(m_pg280_if.I2),.I3(m_pg280_if.I3),.I4(m_pg2
80_if.I4),.I5(m_pg280_if.I5),
.I6(m_pg280_if.I6),.I7(m_pg280_if.I7),.I8(m_pg280_if.I8),.PE(m_pg280_if.PE),.PO(m_pg2
80_if.PO));
```

```
initial begin
environment = new("env");
tb_clk<= 0;
    uvm_resource_db#(virtual pg280_if)::set("env",
        "pg280_if",m_pg280_if);
    run_test();
end
initial begin
 $dumpvars(0, pg280_uvm_tba2);
end
endmodule
```

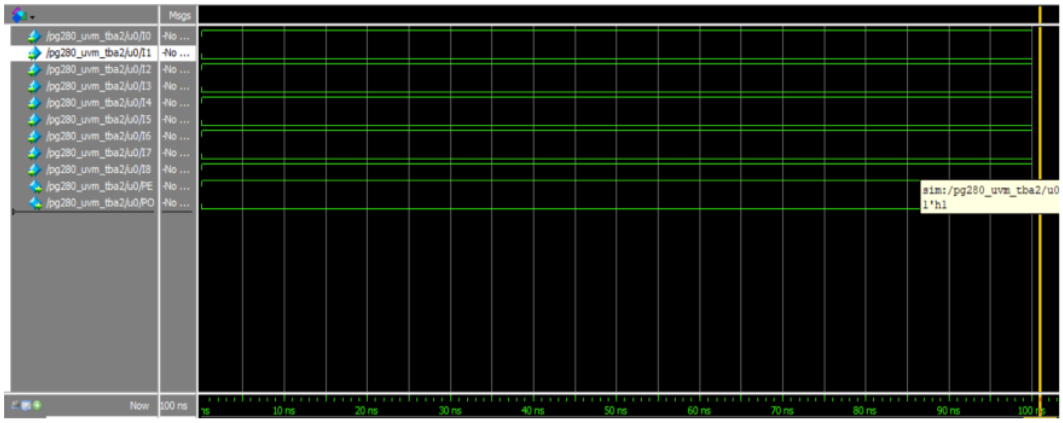
Gambar 8 Kode *testbench top*

Questasim 2021.1

Questasim 2021.1 merupakan *software* simulasi dan debugging yang dapat digunakan untuk memverifikasi desain rangkaian terpadu yang dirancang menggunakan Bahasa SystemVerilog. *Software* ini memberikan fasilitas antar muka grafis yang menarik atau GUI (*Graphical User Interface*).

Hasil Simulasi dan Pembahasan

Hasil simulasi DUT tanpa kerusakan hubung singkat bisa ditunjukkan dalam Gambar 9. Hasil simulasi menujukkan kesesuaian dengan fungsi rangkaian DUT dalam Tabel 2. Hasil *transcript* dari Questasim juga memberikan keterangan *UVM_ERROR*: 0 sebagaimana ditunjukkan dalam Gambar 10.



Gambar 17 Hasil simulasi DUT tanpa kerusakan hubung singkat

```
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    5
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [Questa UVM]     2
# [RESULT]     1
# [RNTST]     1
# [TEST_DONE]     1
```

Gambar 10 Hasil *transcipt* DUT tanpa kerusakan hubung singkat

Kesimpulan

UVM (*Universal Verification Methology*) yang diajukan untuk memverifikasi kerusakan hubung singkat ke suplai tegangan dan ke *ground* yang terjadi di dalam rangkaian terpadu pengecek paritas. UVM terdiri dari *testbench* dan DUT (*Design Under Tes*t). *Testbench* dan DUT dirancang menggunakan Bahasa SystemVerilog dan disimulasikan menggunakan Questasim 2021.1. Hasil simulasi DUT tanpa kerusakan memberikan *transcript* di Questasim dengan keterangan *UVM_ERROR*: 0.

Daftar Pustaka

1. K. Smirnov, A. Nazarov and V. Blinov, "Methods of Automated Test Solutions Design for VLSI Testing," 2020 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), 2020, pp. 1-6, doi: 10.1109/ICIEAM48468.2020.9111875.
2. Christian Bakhau, et al. Logic Application Handbook, Product Feature and Application Insights, Design Engineer Guide. Nexperia. 2020
3. D. Wang, J. Yan and Y. Qiao, "Research on Chip Verification Technology Based on UVM," 2021 6th International Symposium on Computer and Information Processing Technology (ISCIPT), 2021, pp. 117-120, doi: 10.1109/ISCIPT53667.2021.00030.
4. T. M. Pavithran and R. Bhakthavatchalu, "UVM based testbench architecture for logic sub-system verification," 2017 International Conference on Technological Advancements in Power and Energy (TAP Energy), 2017, pp. 1-5, doi: 10.1109/TAPENERGY.2017.8397323.
5. S. El-Ashry and A. Adel, "Efficient Methodology of Sampling UVM RAL During Simulation for SoC Functional Coverage," 2018 19th International Workshop on Microprocessor and SOC Test and Verification (MTV), 2018, pp. 61-66, doi: 10.1109/MTV.2018.00022.
6. P. Wang, A. M. Gharehbaghi and M. Fujita, "Automatic Test Pattern Generation for Double Stuck-at Faults Based on Test Patterns of Single Faults," 20th International Symposium on Quality Electronic Design (ISQED), 2019, pp. 284-290, doi: 10.1109/ISQED.2019.8697831.
7. P. Wang, A. M. Gharehbaghi and M. Fujita, "An Automatic Test Pattern Generation Method for Multiple Stuck-At Faults by Incrementally Extending the Test Patterns," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 10, pp. 2990-2999, Oct. 2020, doi: 10.1109/TCAD.2019.2957364.
8. F. Wang and S. K. Gupta, "An Effective and Efficient Automatic Test Pattern Generation (ATPG) Paradigm for Certifying Performance of RSFQ Circuits," in IEEE Transactions on Applied Superconductivity, vol. 30, no. 5, pp. 1-11, Aug. 2020, Art no. 1300711, doi: 10.1109/TASC.2020.2965933.
9. Y. -C. Kung, K. -J. Lee and S. M. Reddy, "Generating Single- and Double-Pattern Tests for Multiple CMOS Fault Models in One ATPG Run," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 6, pp. 1340-1345, June 2020, doi: 10.1109/TCAD.2019.2921345.
10. M. Chern et al., "Diagnosis of Intermittent Scan Chain Faults Through a Multistage Neural Network Reasoning Process," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 10, pp. 3044-3055, Oct. 2020, doi: 10.1109/TCAD.2019.2957356.
11. S. A. Farman, C. Duggineni, K. N. V. Khasim and H. B. Valiveti, "Optimization of Energy and Area of a Randshift: Fault-Tolerant Technique using FPGA design flow," 2021 6th International Conference on Inventive Computation Technologies (ICICT), 2021, pp. 404-409, doi: 10.1109/ICICT50816.2021.9358555.
12. T. Akada and K. Fujimori, "Designing Microwave Circuits Using Genetic Algorithms Accelerated by Convolutional Neural Networks," 2020 50th European Microwave Conference (EuMC), 2021, pp. 61-64, doi: 10.23919/EuMC48046.2021.9337992.
13. M. Handique, J. K. Deka and S. Biswas, "Detection of Stuck-at and Bridging Fault in Reversible Circuits using an Augmented Circuit," 2021 IEEE 30th Asian Test Symposium (ATS), 2021, pp. 55-60, doi: 10.1109/ATS52891.2021.00022.

Verifikasi Rangkaian Terpadu Pengecek Paritas Menggunakan UVM

ORIGINALITY REPORT

9%	9%	3%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Wright State University Student Paper	2%
2	www.chipverify.com Internet Source	1%
3	svn.common-lisp.net Internet Source	1%
4	Submitted to Universitat Politècnica de València Student Paper	1%
5	www.waynewolf.us Internet Source	1%
6	Submitted to University of Newcastle Student Paper	1%
7	Submitted to Indira Gandhi Delhi Technical University for Women Student Paper	1%
8	www.doulos.com Internet Source	1%

9	es.scribd.com Internet Source	1 %

Exclude quotes	On	Exclude matches	< 1%
Exclude bibliography	On		

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Artikel 1
Assignment title:	Widianto
Submission title:	Verifikasi Rangkaian Terpadu Pengecek Paritas Menggunaka...
File name:	WIDIANTO_T._ELKTRONIKA_UMM_2023.docx
File size:	176.27K
Page count:	18
Word count:	2,498
Character count:	17,135
Submission date:	20-Oct-2023 03:56PM (UTC+0700)
Submission ID:	2201669982

Verifikasi Rangkaian Terpadu Pengecek Paritas Menggunakan UVM

PENGUSUL

Widianto, ST., MT

D3 TEKNOLOGI ELEKTRONIKA

DIREKTORAT PENDIDIKAN VOKASI

UNIVERSITAS MUHAMMADIYAH MALANG

2023