تولید بردارهای آزمون به کمک ابزار حل‌کننده‌های صدق پذیر

پیمان بهنام

payman.behnam@ut.ac.ir.com

دانشگاه تهران

چکیده

آزمون یک مدار دیجیتال به منظور اطمینان از دست یابی به نتایج مورد انتظار از طرح اولیه و درستی عملکرد آن پس از ساخت ، صورت می گیرد. از این رو انجام عملیات آزمون ، یکی از فرایندهای حساس و اجتناب ناپذیر در روند طراحی تا ساخت مدارهای دیجیتال می باشد.که در مراحل مختلف این روند توسط افراد با تخصص های متفاوت ، انجام می پذیرد[1].در این گزارش با یک روش نام تولید بردارهای آزمون به کمک ابزار حل‌کننده‌های صدق پذیر آشنا می شویم. در این روش بردارهای آزمون از طریق نگاشت مساله صدق پذیری بر مداری که از ترکیب مدار سالم و خراب به دست می‌آید به دست می-آید[2].

کلمات کلیدی

بردار های آزمون، حل کننده‌های صدق پذیر، خرابی،پوشش خرابی

1. مقدمه

همواره مساله آزمون کیفیت و درستی عملکرد پس از تولید هر محصول،یکی از مهم ترین مراحل تولید محصول محسوب می شود.بمدارهای دیجیتال از این اصل مستثنی نیستند.به ظوری که 60 % قیمت تمام شده محصول در زمان حال صرف تست و بررسی درستی آن می شود[2]. تا کنون روش‌های بسیار زیادی برای تولید بردارهای آزمون برای مدارات ترکیبی و ترتیبی پیشنهاد شده‌است. تولید این مجموعه داده ها می تواند به روش های مختلفی انجام شود. اما اگر این مجموعه درست تولید نشود ، نمی‌توان به وسیله آن درباره درستی مدار مورد آزمون قضاوت تامی ارائه داد. از طرف دیگر معمولاً مجموعه داده-های آزمون بسیار زیاد است که در عمل استفاده از آن مشکل شود. بنابراین‌در فرآیند تولید مجموعه داده-های آزمون، باید به گزینش داده‌های آزمون از این مجموعه پرداخت به طوری که مجموعه گزینش شده کافی،کامل ودر عین حال دارای اندازه قابل اجرا باشد.در گام بعدی لیستی از خطا های مدار بر اساس مدل انتخاب شده تولید می گردد و سپس مرحله اصلی تولید الگو های آزمون برای کشف خطا های موجود در لست تهیه شده انجام می شود[1].دراین گزارش، ابتدا به بررسی مسئله تولید بردارهای آزمون می-پردازیم و روش‌های ارائه شده برای حل این مساله را بازگو خواهیم کرد.سپس به بررسی ابزار حل‌کننده-های صدق پذیرخواهیم پرداخت و نحوه استفاده از این ابزار برای حل مسئله تولید بردارهای آزمون را شرح می‌دهیم. در ادامه به بررسی ابزارهای موجود برای تبدیل مدار مورد نظر به فرمت ورودی این ابزار می‌پردازیم و در آخر نتایج به دست آمده را گزارش خواهیم نمود .

2. روش‌های تولید بردار آزمون

تولید مجموعه داده‌های بردار آزمون می تواند به روش های مختلفی انجام پذیرد که به طور کلی می توان آن‌ها را به دو دسته غیر قطعی و قطعی تقسیم نمود. در روش های غیر قطعی الگوهای آزمون به صورت تصادفی تولید می-شوند . بنابراین حتی اگر نتایج حاصل از اعمال این الگوها به مدار مطابق با نتایج مورد انتظار مدار باشد، باز هم نمی توان به صورت کامل از صحت عملکرد مدار اطمینان حاصل کرد. اما در روش های قطعی ، بردارهای آزمون با استفاده از الگوریتم های خاصی تولید می شوند.گام اول در تمامی روش های قطعی تولید الگوهای آزمون ، انتخاب مدل مناسب جهت نمایش خطا های که امکان رخ دادن آنها در آن سطح از طراحی وجو دارد می باشد.ازجمله روشهای قطعی می‌توان به الگوریتم های کلاسیک از قبیل PODEM، FAN، SOCRATES و ATOM اشاره کرد[1].

الگوریتم های متعددی برای تولید الگوهای آزمون مدارهای ترکیبی وجود دارد که با استفاده از آن‌ها در تولید کننده‌های آزمون خود کار می‌توان به پوشش خطای قابل قبولی برای مدار های ترکیبی دست یافت . اما برای مدار های ترتیبی بر خلاف مدارهای ترکیبی ، تولید مجموعه داده های آزمون از پیچیدگی خاصی بر خوردار است که ناشی از تفاوت های این دو نوع مدار است . تفاوت اصلی این دو نوع مدار ، عناصر حافظه و حلقه های پس خورد در مدار های ترکیبی می‌باشد[3].

به تازگی الگوریتم‌های مبتنی بر روشهای رسمی مورد استقبال قرار گرفته‌اند.رویکرد های رسمی از روش‌هایی است که دارای استحکام و مبنای ریاضی و متقن است و برای توصیف دقیق و کامل مدار استفاده می‌شود. با استفاده از یک روش رسمی ، می‌توان یک سیستم را به صورت انتزاعی توصیف کرد. از وارد شدن به جزئیات پرهیز نمود و در نتیجه به اهداف رویکرد آزمون جعبه سیاه کمک کردتا بتوان مجموعه نمونه‌های آزمون را از یک ضابطه انتزاعی تولید کرد[1].

3. حل‌کننده صدق‌پذیر-SAT (Solver)[2]

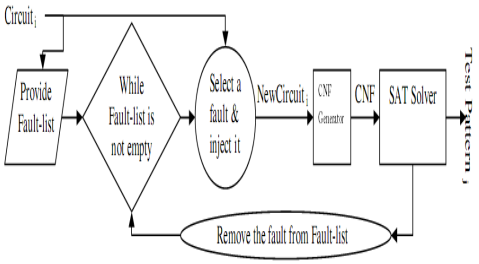
مسئله صدق پذیری بولی (**SAT**) کاربر دهای فراوانی در زمینه درستی یابی، عیب یابی و تولید بردارهای آزمون دارد.مسئله‌ی صدق پذیری روی یک فرمول گزاره‌ای، پیدا کردن یک مقدار دهی به متغیر هاست که منجر به درستی آن شود. چنین مقدار دهی (در صورت وجود) مقدار دهی صدق پذیر برای آن است و آن گزاره، یک گزاره صدق

پـذیر نامیـده مـی شـود. ایـن مسـئله اولـین مسـئله NP_Complete می‌باشد و پیشرفت های زیادی برای حـل آن ارائـه شـده اسـت . بـرای حـل سـایر مسـائل NP_Complete**موجود**، سعی می کنند آن را به نحوی بر مسئله SAT نگاشت کنند. سپس با استفاده از حل کننده صدق پذیر، به نحوی به حل آن بپردازند.

بسیاری از حل کننده های صدق پذیربرای نمایش فرم بولین از فرم نرمال عطفی (CNF) استفاده می کنند .در CNF ، فرمول به صورت ترکیب عطفی بندها نشان داده می‌شود که هر بند شامل ترکیب فصلی حروف است و هر حرف شامل یک متغیر یا نقیض آن می باشد.برای صدق پذیری یک فرمول ،در هر بند باید حداقل یکی از حروف، مقدار درستی به خود بگیرد

4. تولید بردارهای آزمون برای مدارات ترکیبی به کمک حل‌کننده‌های صدق-پذیر(SAT-Solver)

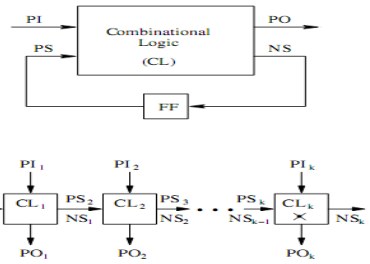
برای تولید بردار آزمون برای حل کننده صدق پذیر، ابتدا می‌بایست لیست خرابی‌ها برای مدار مورد نظرتهیه شوند.خرابی های مدار به صورت چسبیده به صفریا چسبیده به یک می باشند. سپس یکی از خرابی‌ها در نظر گرفته شود و فاز تزریق خرابی انجام شود. سپس خروجی های متناظر مدار سالم و خراب شده با هم **XOR** می شوند و در نهایت خروجی های **XOR** هابا هم **OR** می شوند. سپس **CNF** مدار حاصل استخراج می شود و به یک **SAT-Solver**داده می شود. در صورتی که خروجی حل‌کننده صدق پذیر، ارضا پذیر بود به این معنی است که حاصل حداقل یکی از **XOR** های اضافه شده یک می باشد و این یعنی خروجی مدار سالم و خراب شده متفاوت است و این یعنی خرابی تریق شده ، تشخیص داده شده است .این مراحل برای کلیه خرابی ها باید انجام شود . شکل 1 روند تولید الگوهای آزمون به کمک SAT-Solver را نشان می دهد.



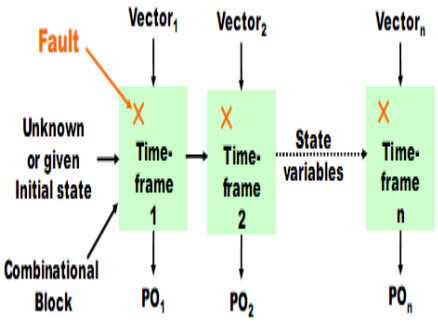
شکل 1نحوه تولید بردار آزمون به کمک SAT-Solver

6-تولید بردارهای‌آزمون برای‌مدارات‌ترتیبی به کمک حل‌کننده‌های صدق‌پذیر (SAT-Solver)

روند تولید بردارهای آزمون برای مدارات ترتیبی به کمک حل‌کننده‌های صدق‌پذیر(**SAT-Solver**) مشابه مدارات ترکیبی می باشد ، با این تفاوت که خرابی مورد نظر تریق می شود و سپس مدار مورد نظر، توسعه زمانی داده می-شود . برای این کار مطابق شکل 2 کلیه فلیپ فلاپ های مدار برداشته می‌شوند و خروجی فلیپ فلاپ ها به ورودی حالت گیت بعد می‌روند.تا مدار ترتیبی تبدیل به مدار ترکیبی شود. سپس قسمت ترکیبی حاصل، به تعداد لازم تکرار می شود.در نتیجه یک مدار ترکیبی حاصل با ورودی ها و خروجی های تکرار شونده داریم که سعی می کنیم با استفاده از روش‌های تولید بردارهای آزمون برای مدارات ترکیبی به کمک حل‌کننده‌های صدق‌پذیر-SAT (Solver) به حل مسئله بپردازیم. شکل 3 این موضوع را بهتر نشان می دهد[2].



شکل 2 توسعه زمانی یک مدار ترتیبی[2]



شکل 3 نحوه تزریق خرابی و توسعه زمانی برای تولید بردار آزمون[4]

7.ابزار Atalanta[5]

ابزارAtalantaیک ابزار برای مشخص کردن خرابی ها و هم چنین تولید بردارهای آزمون برای یک مدار ترکیبی می باشد. برای استفاده از این ابزار از دستوری همانند دستور 1 استفاده می کنیم.

atalanta-M -t c432.pat -P c432.rep -m c432.mask -W 1 –f fault.in c432.bench ***(1)***

این دستور**برای** فایل c432.bench کلیه خرابی های مدار مورد نظر را در فایل fault.in قرار می‌دهد. برای آن تست تولید می کند و آن را در فایل c432.pat قرار می-دهد . خرابی های غیر قابل تشخیص را در فایل *c432.mask* قرار می دهد و سپس نتایج حاصل مثل تعداد تست ها و خرابی های قابل تشخیص و غیر قابل تشخیص ومیزان پوشش خرابی را در قالب فایل گزارش در فایلc432.rep *قرار می دهد*

8.ابزار HOPE[6]

ابزار HOPE یک ابزار شبیه سازی موازی تولید تست برای مدارات ترتیبی سنکرون است این ابزار برای تولید

داده‌های آزمون با استفاده از روش های تصادفی و معین استفاده می کند[4]. همچنین می‌توان از این ابزار برای تولید لیست **خرابی های مدار استفاده نمود.** برای استفاده از این ابزار از دستوری همانند دستور 2 استفاده می‌کنیم.

Hope –l Circuit298.dict -D Circuit298.bench ***(2)***

این دستور**برای** فایل circuit298.bench کلیه خرابی های مدار مورد نظر و تست های تولیدی را در فایل Circuit298.in قرار می‌دهد. **و** سپس نتایج حاصل مثل تعداد تست ها و خرابی های قابل تشخیص و غیر قابل تشخیص ومیزان پوشش خرابی را در قالب فایل گزارش در فایل Circuit298.dict قرار می‌دهد این ابزارها کلیه عملیات fault dropping **و** fault collapsing را انجام می‌دهد.

9-نحوه تولید CNF مدار نهایی

ابزار CNF4ATPG Generator به زبان جاوا نوشته شده‌است. دلیل استفاده از این ابزار امکانات نسبتاًخوب این زبان برای عبارت‌های منظم و پارس‌کردن می‌باشد. اسم‌ها و متغیرهای زبان به‌گونه‌ای انتخاب شده‌اندکه می‌توان به عمل‌کرد توابع وکلاس های نوشته شده به‌راحتی پی برد. این ابزار به گونه‌ای نوشته شده‌است که کلیه عملیات لازم شامل تولید تولید فایل بنچ پس از تزریق خرابی، توسعه زمانی مدار ترتیبی سالم و مدار ترتیبی پس از تزریق خرابی و هم چنین تولید مدار نهایی شامل xor کردن خروجی‌های مورد نظر، or کردن xor های حاصل و تولید cnf را به صورت خودکار انجام می‌دهد. برای استفاده از این ابزار به طریق زیر عمل می‌کنیم:

1) فایل بنچ مدار مورد نظر را فراهم می‌کنیم و اسم آن را gate.in می‌گذاریم. برای این کار یا می توان کد Verilog مدار مورد نظر را به کمک ابزار NETLISTGEN به فرمت بنچ تبدیل کرد و یا به صورت مستقیم از فایل های بنچ موجود استفاده کرد.
2) برای مدارات ترکیبی به کمک ابزار Atalanta و برای مدارات ترتیبی به کمک ابزار HOPE لیست کلیه خرابی‌های مدار را در می آوریم و اسم‌آن را fault.in می‌گذاریم.

تذکر: در صورت استفاده از ابزار Hope قسمت گزارش به همراه عبارت list of undeteccted faults**و** potentially detected **و** شماره faultهای تشخیص داده نشده از فایل حاصل را پاک می‌کنیم.

در ابزار نوشته شده ، برای مدل کردن خطاهای چسبیده به یک از or کردن سیم خرابی با not همان سیم و برای مدل کردن خطاهای چسبیده به صفر از and کردن سیم خرابی با not همان سیم استفاده می‌کنیم و این مدار جدید را درمحل-ورودی یک گیت یا در ورودی‌ها و خروجی‌ها مدار که خطا بر روی آن اتفاق افتاده اضافه می-کنیم

3) مقدار توسعه زمانی را در قسمت main ودر تابع makeABCScript, makeABCScript2, makeABCScript3 مشخص می‌کنیم. در تابعmakeABCScript با استفاده از دستورات 3 ابتدا فایل بنچ خوانده می شود . سپس مدار به تعداد دلخواه توسعه زمانی داده می شود و در نهایت فایل بنچ نهایی دوباره نوشته می شود.

read_bench xxx.bench,frames –Fi num,write_bench xxx_unrolled.bench **(3)**

4) هم چنین با استفاده از makeABCScript2 می توان bench مناسب برای استفاده در دستور miter و تولید CNF مدار نهایی را به دست آورد. در این تابع با استفاده از دستورات 4 ابتدا فایل بنچ خوانده می شود . سپس مدار با دستور strash تبدیل به فرمت AIG(And Inverter Graph) می‌شود. پس از آن با دستور logic مدار به فرمت logic network تبدیل می شودتا فرمت مناسب برای استفاده ازدستور mitter به دست آید

read_bench xxx.bench,strash,logic,wsrite_bench xxx.bench**(4)**

5) هم چنین با استفاده از makeABCScript3 می توان CNF مدار نهایی را به دست آورد. در این تابع با استفاده از دستورات 5 ابتدا فایل بنچ خوانده می شود . سپس مدارخراب شده با دستور strash تبدیل به فرمت AIG(And Inverter Graph) می شود .پس از آن با دستور logic مدار به فرمت logic network تبدیل می شودتا فرمت مناسب برای استفاده از mitter به دست آید. سپس با دستور miter فرمت های سالم و خراب شده با هم ترکیب می‌شوند به طوری که ورودی‌های مشترک است و خروجی های متناظر با هم xorمی شوند . سپس خروجی های xor با هم or می شوند و در نهایت CNF نهایی به دست می‌آید

read_bench xxx.bench,strash,logic,miter file1, file2,write_cnf xxx.cnf **(5)**

6) ابزار **ABC** را در پوشه ابزار قرار می‌دهیم یا در صورت قرار دادن در پوشه دیگر مسیر آن را در ابزار ست می‌کنیم.
7) نرم افزار مورد نظر را اجرا می‌کنیم.

خروجی‌ها به قرار زیر است:
1)پوشه faults شامل کلیه بنچ‌های faulty مدار مورد نظر است .
2) پوشه unrolled faults کلیه بنچ‌های faulty مدار مورد نظر پس از توسعه زمانی است.
3) پوشه final_for_mitter شامل کلیه بنچ-های مدار نهایی حاصل برای استفاده در دستور mitter است.
4) پوشه CNF شامل کلیه cnf های مدار نهایی حاصل است.
5) فایل unrolledgate.bench شامل توسعه زمانی مدار سالم است.
6)unrolledgatein4miter.bench شامل فرمت مناسب مدار سالم برای استفاده در دستور miter است.
کد نوشته شده در پیوست موجود است.

10-ابزار **minisat**

این ابزار یک ابزار مناسب برای حل مسائل **sat** می باشد که کاربرد های فراوانی دارد و در بسیاری از ابزارهای دیگر هم می توان از آن بهره جست

Minisat[option] <read_file.cnf> <result_output_file> >>reprortfile.txt (6)

ورودی این ابزار یک فایل cnf و خروجی 2 فایل می باشند. Result_output_file شامل جواب مساله که آیا مساله ارضا پذیر است یا خیر و انتساب های لازم به متغیر ها در صورت ارضا پذیری مسئله می باشد .فایل reportfile.txt شامل گزارشی از نتایج حل مساله از قبیل تعداد متغیر ها ، تعداد عبارات ، تعداد restartها ، تعداد conflictها یا backtracking ها ، تعداد تصمیم گیری ها ، مقدار حافظه ی مصرفی و زمان صرف شده برای حل مساله می باشد . قسمت option شامل قسمت هایی از قبیل core option، main option و simp option می باشد که در آن می توان روش های هیوریستیک استفاده شده ، نوع حل مساله، حداکثر مقدار حافظه مصرفی ، حداکثر زمان که باید جواب در آن زمان به دست آید ، حداکثر تعداد جایگزینی ها و نظایر آن را به دست آورد.

11- نتایج به دست آمده

برای حل مساله، بنچ مارک های ITC'99 شامل b01,b03,b06 انتخاب شدند. ابزار CNF4ATPG را اجراو cnf های به دست آمده به ابزار minisat داده شد. در نهایت یک shell برای اجرای اتوماتیک و محاسبه پارامترهای مختلف نوشته شد که در پیوست 2 موجود است .نتایج حاصل شامل تعداد کل خرابی ها، تعداد خرابی های قابل آشکار سازی، و متوسط پارامتر های تعداد متغیر ها، تعداد عبارات،تعداد restart ، تعداد conflict(backtracking)، تعداد تصمیم گیری ها،مقدار حافظه‌ی مصرفی و زمان صورت گرفته برای تولید بردار های آزمون برای هر بنچ مارک و هر تعداد توسعه زمانی مطابق جدول 1 تا 9 می باشد.

جدول 1 نتایج به دست آمده برای **b01** با 2 بار توسعه زمانی

مقدار	پارامتر مورد نظر
118	تعداد کل خرابی‌ها
69	تعداد خرابی‌های غیر قابل آشکار سازی
11.68	متوسط تعداد متغیر ها
2.55	متوسط تعداد عبارات
0.41	متوسط تعداد **restart**
0	متوسط تعداد **(backtracking)conflict**
0.41	متوسط تعداد تصمیم گیری‌ها
7	متوسط مقدار حافظه‌ی مصرفی
0.00009	متوسط زمان صرف شده
41%	مقدار پوشش خطا

جدول 2نتایج به دست آمده برای **b01** با 4 بار توسعه زمانی

مقدار	پارامتر مورد نظر
118	تعداد کل خرابی‌ها
15	تعداد خرابی‌های غیر قابل آشکار سازی
54.84	متوسط تعداد متغیر ها
171.33	متوسط تعداد عبارات
1	متوسط تعداد **restart**
12.21	متوسط تعداد **(backtracking)conflict**
18.86	متوسط تعداد تصمیم گیری‌ها
7	متوسط مقدار حافظه‌ی مصرفی
0.00009	متوسط زمان صرف شده
87.28	مقدار پوشش خطا

جدول 3نتایج به دست آمده برای **b01** با 8 بار توسعه زمانی

مقدار	پارامتر مورد نظر
118	تعداد کل خرابی‌ها
0	تعداد خرابی‌های غیر قابل آشکار سازی
147.55	متوسط تعداد متغیر ها
538.27	متوسط تعداد عبارات
1.14	متوسط تعداد **restart**
28.89	متوسط تعداد **(backtracking)conflict**
66.72	متوسط تعداد تصمیم گیری‌ها
7	متوسط مقدار حافظه‌ی مصرفی
0.00008	متوسط زمان صرف شده
100	مقدار پوشش خطا

جدول 4نتایج به دست آمده برای **b03** با 4 بار توسعه زمانی

مقدار	پارامتر مورد نظر
394	تعداد کل خرابی‌ها
150	تعداد خرابی‌های غیر قابل آشکار سازی
75.10	متوسط تعداد متغیر ها
120.76	متوسط تعداد عبارات
0.63	متوسط تعداد **restart**
3.57	متوسط تعداد **(backtracking)conflict**
12.90	متوسط تعداد تصمیم گیری‌ها
7	متوسط مقدار حافظه‌ی مصرفی
0.001	متوسط زمان صرف شده
62	مقدار پوشش خطا

جدول 5نتایج به دست آمده برای **b03** با 8 بار توسعه زمانی

مقدار	پارامتر مورد نظر
394	تعداد کل خرابی‌ها
40	تعداد خرابی‌های غیر قابل آشکار سازی
367.30	متوسط تعداد متغیر ها
1279.54	متوسط تعداد عبارات
1.20	متوسط تعداد **restart**
47.12	متوسط تعداد **(backtracking)conflict**
120.41	متوسط تعداد تصمیم گیری‌ها
7	متوسط مقدار حافظه‌ی مصرفی
0.005	متوسط زمان صرف شده
90	مقدار پوشش خطا

جدول 6نتایج به دست آمده برای **b03** با 16 بار توسعه زمانی

مقدار	پارامتر مورد نظر
394	تعداد کل خرابی‌ها
16	تعداد خرابی‌های غیر قابل آشکار سازی
977.6	متوسط تعداد متغیر ها
3470.56	متوسط تعداد عبارات
2.12	متوسط تعداد **restart**
191.84	متوسط تعداد **(backtracking)conflict**
495.81	متوسط تعداد تصمیم گیری‌ها
7.19	متوسط مقدار حافظه‌ی مصرفی
0.009	متوسط زمان صرف شده
96	مقدار پوشش خطا

جدول 7نتایج به دست آمده برای**b06** با 2 بار توسعه زمانی

مقدار	پارامتر مورد نظر
140	تعداد کل خرابی‌ها
23	تعداد خرابی‌های غیر قابل آشکار سازی
17.5	متوسط تعداد متغیر ها
10.93	متوسط تعداد عبارات
83.57	متوسط تعداد **restart**
0	متوسط تعداد **(backtracking)conflict**
0.8357	متوسط تعداد تصمیم گیری‌ها
7	متوسط مقدار حافظه‌ی مصرفی
00.9	متوسط زمان صرف شده
84	مقدار پوشش خطا

جدول 8نتایج به دست آمده برای**b06** با 4 بار توسعه زمانی

مقدار	پارامتر مورد نظر
140	تعداد کل خرابی‌ها
0	تعداد خرابی‌های غیر قابل آشکار سازی
54.32	متوسط تعداد متغیر ها
184.29	متوسط تعداد عبارات
1	متوسط تعداد **restart**
2	متوسط تعداد **(backtracking)conflict**
8.23	متوسط تعداد تصمیم گیری‌ها
7	متوسط مقدار حافظه‌ی مصرفی
0.008	متوسط زمان صرف شده
100	مقدار پوشش خطا

جدول 9نتایج به دست آمده برای**b06** با 8 بار توسعه زمانی

مقدار	پارامتر مورد نظر
140	تعداد کل خرابی‌ها
0	تعداد خرابی‌های غیر قابل آشکار سازی
134.69	متوسط تعداد متغیر ها
552.35	متوسط تعداد عبارات
1	متوسط تعداد **restart**
3.89	متوسط تعداد **(backtracking)conflict**
16.82	متوسط تعداد تصمیم گیری‌ها
7	متوسط مقدار حافظه‌ی مصرفی
0.009	متوسط زمان صرف شده
100	مقدار پوشش خطا

11-نتیجه گیری

در این گزارش با روش های مختلف تولید بردار های آزمون آشنا شدیم . به خصوص جزئیات روش تولید بردارهای آزمون به کمک ابزار حل‌کننده‌های صدق پذیر(SAT_SOLVER)
مورد بررسی قرار گرفت . تعداد گسترش زمانی برای یک مدار ترتیبی برای چنین استفاده‌ای باید حداقل به گونه ای باشد که بتوان حالت‌های مختلف را چک کرد و به خروجی رسید تا بتوان به پوشش بالایی دست یافت . همان طور که نتایج نشان میدهد با افزایش توسعه زمانی پوشش خرابی بالاتر می رود اما روند تولید آزمون و دنباله تست تولید شده برای یک خطا بسیار پیچیده‌تر می‌شود.

منابع

[1]Michael L. Bushnell, Vishwani D. Agrawal, ESSENTIALS OF ELECTRONIC TESTING FORDIGITAL, MEMORY AND MIXED-SIGNAL VLSI CIRCUITS, kluwer academic publishrs, ISBN 0-306—47040-3

[2]N. Amla, X. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan, “An analysis of SAT-based model checking techniques in an industrial environment, in Correct Hardware Design and Verification Methods” (CHARME). Berlin/Heidelberg, Germany: Springer, pp. 254—268 , 2005

[3]Miczo, Alexander. Digital logic testing and simulation. 2nd ed.Published by John Wiley & Sons, Inc., Hoboken, New Jersey. ISBN 0-471-43995-9 (cloth).

[4] Alizadeh bijan, Assignment #2SAT_Based ATPG
, Tehran University, Department of Electrical and computer Engineering,spring 90

[5]Petr Fišer,Atalanta Manual, Virginia Polytechnic & State University, Copyright (C) 1991

[6]Petr Fišer,HOPE Manual, Virginia Polytechnic & State University, Copyright (C) 1991.

پیوست 1

```
package gateparser;

import java.io.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.*;

/**
 *
 * @author Onymous
 */
class Node{
    int type=-1;
    ArrayList<String> inputs=new ArrayList<String>();
    String output="";
    String typeName="";
}

class Fault{
    int type=-1;
    String wire1="";
    String wire2="";
    boolean value=false;
}

public class GateParser {
    static int OR=6,AND=4,XOR=9,XNOR=10,NOT=8,BUF=11,INPUT=3,OUTPUT=2,NAND=5,NOR=7,DFF=12,vdd=13,LUT=14,
gnd=15;
    static String[]
gateTypes={"","","OUTPUT","INPUT","AND","NAND","OR","NOR","NOT","XOR","XNOR","BUF","DFF","vdd","LUT","gnd"};

    public static void makeABCScript(String scriptName,int rollNum,String fin,String fout){
        try{
            Writer so=new OutputStreamWriter(new FileOutputStream(new File(scriptName)));
            so.write("read_bench "+fin+"\n");
            //so.write("fraig"+"\n");
            so.write("frames -Fi "+rollNum+"\n");
            so.write("write_bench "+fout+"\n");
            so.close();
        }catch(Exception e){System.out.println("error2");System.exit(1);}

    }
     public static void makeABCScript2(String scriptName,int rollNum,String fin,String fout){
        try{
            Writer so=new OutputStreamWriter(new FileOutputStream(new File(scriptName)));
            so.write("read_bench "+fin+"\n");
            so.write("strash"+"\n");
            so.write("logic "+"\n");
            so.write("write_bench "+fout+"\n");
            so.close();
        }catch(Exception e){System.out.println("error9");System.exit(1);}

    }
    public static void makeABCScript3(String scriptName,int rollNum,String fin,String finter,String fout){
        try{
            Writer so=new OutputStreamWriter(new FileOutputStream(new File(scriptName)));
            so.write("read_bench "+fin+"\n");
            so.write("strash"+"\n");
            //so.write("frames -Fi "+rollNum+"\n");
            so.write("logic "+"\n");
            so.write("miter "+fin+" "+ finter+"\n");
           // so.write("read_eqn "+fin+"\n");
            //so.write("fraig"+"\n");
            //so.write("strash"+"\n");
            so.write("write_cnf "+fout+"\n");
            so.close();
        }catch(Exception e){
```

```
            System.out.println(e.getMessage());System.exit(1);
        }

    }
    public static void main(String[] args) {
        ArrayList<Node> nodes=gatesParse("gate.in");
        ArrayList<Fault> faults=faultParse("faults.in");
        makeABCScript("abccmd.in",16,"gate.in","unrolledgate.in");
        String cmd="abc  -F abccmd.in";

        try{
            Process pr=Runtime.getRuntime().exec(cmd);
            pr.waitFor();
        }catch(Exception e){
            System.out.println("error3");System.exit(1);
        }

        makeABCScript2("abccmd2.in",16,"unrolledgate.in","unrolledgatein4miter.bench");
        String cmd2="abc  -F abccmd2.in";

        try{
            Process pr=Runtime.getRuntime().exec(cmd2);
            pr.waitFor();
        }catch(Exception e){
            System.out.println(e.getMessage());System.exit(1);
        }




        ArrayList<Node> unrolledNodes=gatesParse("unrolledgate.in");
        Iterator<Fault> it=faults.iterator();
        while(it.hasNext()){

            Fault fl=it.next();
//          if(fl.type==2)
//              continue;
            ArrayList<Node> faultyNodes=cloneNodes(nodes);
            ArrayList<Node> tmpNodes=new ArrayList<Node>();



            Iterator<Node> it2=faultyNodes.iterator();
            while(it2.hasNext()){
                Node n=it2.next();
                if(n.type==INPUT){//inserting buffres in inputs
                        Node n2=new Node();
                        n2.inputs.add(n.inputs.get(0));
                        n2.output=n.inputs.get(0)+"_WFCIRCUIT";
                        n2.type=BUF;
                        n2.typeName="BUF";
                        tmpNodes.add(n2);
                        continue;
                }
                n.output=n.output+"_WFCIRCUIT";//renaming faulty output
                for(int h=0;h<n.inputs.size();h++){
                    String s=n.inputs.get(h);
                    n.inputs.set(h,s+"_WFCIRCUIT");//renaming faulty input
                }
            }
            faultyNodes.addAll(tmpNodes);


            it2=faultyNodes.iterator();
            while(it2.hasNext()){
                Node n=it2.next();
                if(fl.type==2){//when we have xx->xx fault
                    if(n.output.compareToIgnoreCase(fl.wire2+"_WFCIRCUIT")!=0)
                        continue;
                }
                for(int h=0;h<n.inputs.size();h++){
                    String s=n.inputs.get(h);
                    if(s.compareToIgnoreCase(fl.wire1+"_WFCIRCUIT")==0){
```

```
                    n.inputs.set(h,fl.wire1 +"_FSA_WFCIRCUIT");
                }
            }
        }
        Node not=new Node();
        not.type=NOT;
        not.typeName="NOT";
        not.inputs.add(fl.wire1+"_WFCIRCUIT");
        not.output=fl.wire1+"_NOTF"+"_WFCIRCUIT";
        faultyNodes.add(not);
        if(fl.value==true){
            Node and=new Node();
            and.type=AND;
            and.typeName="AND";
            and.inputs.add(fl.wire1+"_WFCIRCUIT");
            and.inputs.add(fl.wire1+"_NOTF"+"_WFCIRCUIT");
            and.output=fl.wire1+"_FSA_WFCIRCUIT";
            faultyNodes.add(and);
        }else{
            Node or=new Node();
            or.type=OR;
            or.typeName="OR";
            or.inputs.add(fl.wire1+"_WFCIRCUIT");
            or.inputs.add(fl.wire1+"_NOTF"+"_WFCIRCUIT");
            or.output=fl.wire1+"_FSA_WFCIRCUIT";
            faultyNodes.add(or);
        }
        writeNodes(faultyNodes,"faults\\fault_"+fl.wire1+"_"+fl.wire2+"_"+fl.value+ ".out");
        makeABCScript("abccmd3.in",16,"faults\\fault_"+fl.wire1+"_"+fl.wire2+"_"+fl.value+
".out","unrolled_faults\\unrolledfault_"+fl.wire1+"_"+fl.wire2+"_"+fl.value+ ".out");
        cmd="abc -F abccmd3.in";
        try{
            Process pr=Runtime.getRuntime().exec(cmd);
            pr.waitFor();
            //Runtime.getRuntime()
        }catch(Exception e){
            System.out.println("e);System.exit(1)");
                }


//          writeNodes(faultyNodes,"unrolled_faults\\unrolledfault_"+fl.wire1+"_"+fl.wire2+"_"+fl.value+ ".out");//forgotten
//          if(fl.wire1.compareToIgnoreCase("G56")==0){
//              System.out.println("r1rr"+"unrolledfault_"+fl.wire1+"_"+fl.wire2+"_"+fl.value+ ".out");
//          }
        ArrayList<Node> unrolledFaultyNodes=gatesParse("unrolled_faults\\unrolledfault_"+fl.wire1+"_"+fl.wire2+"_"+fl.value+ ".out");

        //ArrayList<Node> faultyNodes2=cloneNodes(nodes);
        ArrayList<Node> tmpNodes2=new ArrayList<Node>();

        it2=unrolledFaultyNodes.iterator();
        Iterator<Node> it3 ;
        while(it2.hasNext()){
            Node n=it2.next();
            if(n.output.length()>0 && n.output.charAt(0)=='n')
            {
                n.output="WFCIRCUIT_" + n.output;
                it3 = unrolledFaultyNodes.iterator();
                while( it3.hasNext())
                {
                    Node n3 = it3.next() ;
                    for(int h=0;h< n3.inputs.size();h++)
                    {
                        String s= n3.inputs.get(h);
                        if(s.charAt(0)=='n')
                        n3.inputs.set(h,"WFCIRCUIT_" + s);
                    }
                }
            }
        }
        it2=unrolledFaultyNodes.iterator();
        while(it2.hasNext()){
            Node n=it2.next();
            if(n.type==INPUT && n.inputs.get(0).contains("_WFCIRCUIT")){//inserting buffres in inputs
                    Node n2=new Node();
                    n2.inputs.add(n.inputs.get(0).replace("_WFCIRCUIT", ""));
```

```
                n2.output=n.inputs.get(0);
                n2.type=BUF;
                n2.typeName="BUF";
                tmpNodes2.add(n2);
                n.inputs.set(0,  n.inputs.get(0).replace("_WFCIRCUIT", "") );
                continue;
            }
            if(n.type==OUTPUT && n.inputs.get(0).contains("_FSA_WFCIRCUIT")){//inserting buffres in inputs
                Node n2=new Node();
                n2.output = n.inputs.get(0).replace("_FSA_WFCIRCUIT" , "") ;
                n2.inputs.add( n.inputs.get(0)) ;
                n2.type=BUF;
                n2.typeName="BUF";
                tmpNodes2.add(n2);
                n.inputs.set(0, n.inputs.get(0).replace("_FSA_WFCIRCUIT", "")) ;
                continue;
            }
            //replaced output with inputs.get(0) for both n and n2
            if(n.type==OUTPUT && n.inputs.get(0).contains("_WFCIRCUIT")){//inserting buffres in inputs
                Node n2=new Node();
                n2.output = n.inputs.get(0).replace("_WFCIRCUIT" , "") ;
                n2.inputs.add( n.inputs.get(0)) ;
                n2.type=BUF;
                n2.typeName="BUF";
                tmpNodes2.add(n2);
                n.inputs.set(0, n.inputs.get(0).replace("_WFCIRCUIT", "")) ;
                continue;
            }

        }
            unrolledFaultyNodes.addAll(tmpNodes2);









        //writeNodes(unrolledFaultyNodes,"final_for_mitter\\final_for_mitter"+fl.wire1+"_"+fl.wire2+"_"+fl.value+ "beforeclone.out");
        ArrayList<Node> finalCircuit=cloneNodes(unrolledFaultyNodes);
        //appendNodes(finalCircuit,unrolledFaultyNodes);
        //writeNodes(finalCircuit,"final\\final_"+fl.wire1+"_"+fl.wire2+"_"+fl.value+ ".out");
       writeNodes(finalCircuit,"final_for_mitter\\final_for_mitter"+fl.wire1+"_"+fl.wire2+"_"+fl.value+ ".bench");
        makeABCScript3("abccmd4.in",16,"final_for_mitter\\final_for_mitter"+fl.wire1+"_"+fl.wire2+"_"+fl.value+
".bench","unrolledgatein4miter.bench","CNF\\cnf_"+fl.wire1+"_"+fl.wire2+"_"+fl.value+ ".out");
        String cmd3="abc -F abccmd4.in";
        try{
            Process pr=Runtime.getRuntime().exec(cmd3);
            pr.waitFor();
            Process pr2=Runtime.getRuntime().exec(cmd3);
            pr2.waitFor();
            System.out.println( Integer.toString(pr.exitValue())  +  pr.getErrorStream());
            int ret = pr.getErrorStream().read() ;
            int y = 11 ;
        }catch(Exception e){
            System.out.println(e.getMessage());
            System.exit(1);
        }



    }

}
```

```
    public static void appendNodes(ArrayList<Node> src,ArrayList<Node> app){
        Iterator<Node> it=app.iterator();



        while(it.hasNext()){
            Node n=it.next();
            if(n.type==INPUT || n.type==OUTPUT)
                continue;
            src.add(n);

        }

        /*Node or=new Node();
        or.type=OR;
        it=src.iterator();
        while(it.hasNext()){
            Node n=it.next();
            if(n.type==OUTPUT){
                n.type=XOR;
                n.typeName="XOR";
                n.output=n.inputs.get(0)+"_XOR";
                n.inputs.add(n.inputs.get(0)+"_WFCIRCUIT");
                or.inputs.add(n.output);
            }
        }

        or.output="EVENTUAL_OUT";
        or.typeName="OR";
        src.add(or);

        Node outfinal=new Node();
        outfinal.type=OUTPUT;
        outfinal.typeName="OUTPUT";
        outfinal.inputs.add("EVENTUAL_OUT");
        src.add(0,outfinal);
    */
    }



    public static ArrayList<Node> cloneNodes(ArrayList<Node> nn){
        Iterator<Node> it=nn.iterator();
        ArrayList<Node> cnn=new ArrayList<Node>();
        while(it.hasNext()){
            Node n=it.next();
            Node cn=new Node();
            cn.output=new String(n.output);
            cn.type=n.type;
            cn.typeName=new String(n.typeName);

            Iterator<String> it2=n.inputs.iterator();
            while(it2.hasNext()){
                cn.inputs.add(new String(it2.next()));
            }
            cnn.add(cn);
        }
        return cnn;
    }

public static void writeNodes(ArrayList<Node> nodes,String foutname){
    try{
        Writer fout=new OutputStreamWriter(new FileOutputStream(new File(foutname)));
        Iterator<Node> it4=nodes.iterator();
        while(it4.hasNext()){
            Node n=it4.next();
            String ss=new String("");
            if(n.type==INPUT || n.type==OUTPUT){
                //ss=gateTypes[n.type]+"("+n.inputs.get(0) +")";
                ss=n.typeName+"("+n.inputs.get(0) +")";
            }else{
                //ss=n.output+" = "+gateTypes[n.type]+"(";
                if(n.type==vdd ||n.type==gnd){
                    ss=n.output+" = "+n.typeName;
```

```
            }else{
                ss=n.output+" = "+n.typeName+"(";
                for(int e=0;e<n.inputs.size()-1;e++){
                    ss=ss+n.inputs.get(e)+", ";
                }
                if(n.inputs.size()>0)
                    ss=ss+n.inputs.get(n.inputs.size()-1)+")";
                else
                    ss=ss+")";
            }
            //System.out.println();
        }
        fout.write(ss+"\n");
    }
    fout.close();
  }catch(Exception e){System.out.println("error1"+e);System.exit(1);}

}
  public static ArrayList<Fault> faultParse(String fname){
    File file = new File(fname);
    int ch;
    StringBuffer strContent = new StringBuffer("");
    FileInputStream fin = null;
    try {
      fin = new FileInputStream(file);
      while ((ch = fin.read()) != -1){
        strContent.append(((char)ch));
      }
      fin.close();
    } catch (Exception e) {;}
    String str=strContent.toString();
    Pattern tag = Pattern.compile(".*?/\\d|test.*? faults detected");
    Matcher mtag = tag.matcher(str);
    int j=0;
    ArrayList<Fault> allFaults=new ArrayList<Fault>();
    while (mtag.find()){
      j=j+1;
      Fault n=new Fault();
      Pattern tagInner = Pattern.compile("[/\\-\\>]+");
      String tmpstr=mtag.group();
      String[] ttt=tagInner.split(tmpstr.trim());
      if(ttt.length<2)
        continue;
      n.wire1=ttt[0].trim();
      if(ttt.length>2){
        n.type=2;
        n.wire2=ttt[1].trim();
        int vv=new Integer(ttt[2].trim()).intValue();
        if(vv==1)
          n.value=true;
        else
          n.value=false;
      }else{
        n.type=1;
        int vv=new Integer(ttt[1].trim()).intValue();
        if(vv==1)
          n.value=true;
        else
          n.value=false;
      }
      allFaults.add(n);
    }
    return allFaults;
  }

  public static ArrayList<Node> gatesParse(String fname){
    File file = new File(fname);
    int ch;
    StringBuffer strContent = new StringBuffer("");
    FileInputStream fin = null;
    try {
      fin = new FileInputStream(file);
      while ((ch = fin.read()) != -1){
        strContent.append(((char)ch));
      }
```

```
            fin.close();
        } catch (Exception e) {;}
        String str=strContent.toString();
        Pattern tag = Pattern.compile("(#.*?)|(OUTPUT\\(.*?\\))|(INPUT\\(.*?\\))|(.*? = AND\\(.*?\\))|(.*? = NAND\\(.*?\\))|(.*? =
OR\\(.*?\\))|(.*? = NOR\\(.*?\\))|(.*? = NOT\\(.*?\\))|(.*? = XOR\\(.*?\\))|(.*? = XNOR\\(.*?\\))|(.*? = BUF\\(.*?\\))|(.*? = DFF\\(.*?\\))|(.*?
= LUT 0x.*?\\(.*?\\))|(.*? = vdd)|(.*? = gnd)");
        Matcher mtag = tag.matcher(str);
        int j=0;
        ArrayList<Node> allNodes=new ArrayList<Node>();
        while (mtag.find()){
            j=j+1;
            for(int i=2;i<=mtag.groupCount();i++){
                if(mtag.group(i)!=null){
                    //System.out.println(i+":"+mtag.group());
                    Node n=new Node();
                    Pattern tagInner = Pattern.compile("[,=\\(\\)]+");
                    String tmpstr=mtag.group();
                    String[] ttt=tagInner.split(tmpstr.trim());
                    if(i>=4){
                        n.output=ttt[0].trim();
                        n.type=getType(ttt[1].trim());
                        n.typeName=ttt[1].trim();
                        for(int k=2;k<ttt.length;k++){
                            n.inputs.add(ttt[k].trim());
                        }
                        allNodes.add(n);
                    }
                    if(i==INPUT || i==OUTPUT){
                        n.inputs.add(ttt[1].trim());
                        n.typeName=ttt[0].trim();
                        n.type=getType(ttt[0].trim());
                        allNodes.add(n);
                    }


                    break;
                }
            }
        }
        return allNodes;
    }


    public static int getType(String s){

        if(s.compareToIgnoreCase("vdd")==0){
            return vdd;
        }
        if(s.compareToIgnoreCase("gnd")==0){
            return gnd;
        }
        if(s.compareToIgnoreCase("XNOR")==0){
            return XNOR;
        }
        if(s.compareToIgnoreCase("DFF")==0){
            return DFF;
        }
        if(s.compareToIgnoreCase("INPUT")==0){
            return INPUT;
        }
        if(s.compareToIgnoreCase("OUTPUT")==0){
            return OUTPUT;
        }
        if(s.compareToIgnoreCase("AND")==0){
            return AND;
        }
        if(s.compareToIgnoreCase("NAND")==0){
            return NAND;
        }
        if(s.compareToIgnoreCase("NOR")==0){
            return NOR;
        }
        if(s.compareToIgnoreCase("OR")==0){
            return OR;
        }
```

```
        if(s.compareToIgnoreCase("XOR")==0){
           return XOR;
        }
        //if(s.compareToIgnoreCase("NOT")==0){
         //   return NOT;
        //}
        if(s.compareToIgnoreCase("BUF")==0){
           return BUF;
        }
        if(s.compareToIgnoreCase("NOT")==0){
           return NOT;
        }
        return -1;
    }
}
```

پیوست 2

```
uncov=0//initialization step
iter=0
t_vars=0
t_restarts=0
t_clauses=0
t_conflicts=0
t_memory=0
t_times=0
cd CNF
echo 4.0/10.0 | bc -l
files=$(ls *.out)
rm -r ./satorunsat
rm -r ./report_satorunsat
mkdir satorunsat
mkdir report_satorunsat
for file in $files
do
echo $file
../minisat $file  -rinc=1.5 -phase-saving=1 -rnd-freq=0.02 satorunsat/$file.cnf >> report_satorunsat/$file.cnf.txt
#sats=$(awk -F 'SAT' 'BEGIN{n=0} {n=1} {print n}' $file.cnf)//running _mnisat
vars=$(awk   '$4=="variables:" { print $5 }'  report_satorunsat/$file.cnf.txt  )//specifing variables
clauses=$(awk   '$4=="clauses:" { print $5 }'  report_satorunsat/$file.cnf.txt )
restarts=$(awk   '$1=="restarts" { print $3 }'  report_satorunsat/$file.cnf.txt )
conflicts=$(awk   '$1=="conflicts" { print $3 }'  report_satorunsat/$file.cnf.txt )
decisions=$(awk   '$1=="decisions" { print $3 }'  report_satorunsat/$file.cnf.txt )
memory=$(awk   '$1=="Memory" { print $4 }'  report_satorunsat/$file.cnf.txt )
times=$(awk   '$1=="CPU" { print $4 }'  report_satorunsat/$file.cnf.txt )
```

```
echo vars=$vars claues=$clauses restarts=$restarts conflicts=$conflicts decisions=$decisions memory=$memory time=$times
unsats=$(grep -o  UNSAT satorunsat/$file.cnf | wc -w)//for finding unsat in results
t_vars=$(expr $t_vars + $vars)// calculation for producing average
t_clauses=$(expr $t_clauses + $clauses)
t_restarts=$(expr $t_restarts + $restarts)
t_conflicts=$(expr $t_conflicts + $conflicts)
t_decisions=$(expr $t_decisions + $decisions)

t_memory=$(echo $t_memory+$memory | bc -l)
uncov=$(expr $uncov + $unsats)
iter=$(expr $iter + 1)
done
echo $uncov// printing results
echo $iter
echo $t_vars/$iter | bc -l
echo $t_clauses/$iter | bc -l
echo $t_restarts/$iter | bc -l
echo $t_conflicts/$iter | bc -l
echo $t_decisions/$iter | bc -l
echo $t_memory/$iter | bc -l
```