یک مدل سطح بالا برای وارسی خواص CTL در طرح توصیف شده توسط VHDL

بیژن علیزاده و زین‌العابدین نوابی

چکیده: در این مقاله قصد داریم مدل سطح بالایی بر پایهٔ معادلات چندجمله‌ای با متغیرهای صحیح ارائه دهیم که مناسب برای وارسی[1] خواص[2] بر پایهٔ CTL (Computational Temporal Logic) می‌باشد. اکثر ابزارهای وارسی از ساختمان داده‌های سطح پایینی مانند BDD استفاده می‌کنند و این ساختمان داده‌ها به علت نیاز به حافظه زیاد، قابل اعمال به بخش مسیر داده[3] از یک طرح نمی‌باشند، در حالی که مدل سطح بالای پیشنهادی در این مقاله قادر است بخش‌های مسیر داده و کنترلر را با هم مورد ارزیابی قرار دهد. ضمن اینکه روش پیشنهادی به گونه‌ای است که نیاز به حل صریح معادلات نمی‌باشد و این کار توسط عملیات جایگزینی و ساده‌سازی انجام می‌گیرد. در انتها نتایج کارمان با ابزار VIS، بعنوان یک ابزار وارسی بر پایهٔ BDD، مقایسه می‌گردند.

کلید واژه: **توابع انتقال حالت، حل‌کننده معادلات صحیح، معادلات چندجمله‌ای با متغیرهای صحیح، BDD و Boolean Satisfiability.**

۱- مقدمه

همزمان با افزایش پیچیدگی مدارهای دیجیتالی از لحاظ عملکردی[4] و افزایش هزینه ساخت این قطعات، وارسی مدارهای دیجیتالی در فازهای اولیهٔ طراحی نیز مهم شده است تا مشکلات طراحی هرچه زودتر مشخص گردند [۱]. شبیه‌سازی، بخاطر انعطاف‌پذیری و مقیاس‌پذیری، از روشهای مناسب برای وارسی مدارهای سنکرون دیجیتالی بوده است. اما کسر کوچکی از یک طرح بزرگ توسط شبیه‌سازی قابل کاوش می‌باشد. برای حل این مشکل روشهای وارسی بر پایهٔ اثبات ریاضیاتی[5] مانند روش چک کردن مدل بصورت سمبلی[6] مورد توجه قرار گرفتند [۲] و [۳]. این روشها سعی دارند که طرح موردنظر را بدون اعمال مقادیری به ورودی‌ها، مورد ارزیابی عملکردی قرار دهند [۴] تا [۶]. بدین ترتیب، طراح قادر خواهد بود که کاوش کلی بر حسب تمام مقادیر متغیرها انجام دهد.

در روشهای بر پایه اثبات ریاضیاتی، هر سیستم دیجیتال بصورت یک ماشین حالت محدود[7] نشان داده می‌شود که آنالیزهای گوناگونی روی این مدل پیاده‌سازی شده است و قابل استفاده می‌باشند [۷] تا [۹]. اکثر روشهای وارسی مانند روش چک کردن مدل بصورت سمبلی، از FSM برای مدل‌کردن طرح سود می‌جویند [۱۰] تا [۱۲]. همچنین اکثر این روشها از BDD برای نمایش توابع انتقال حالت[8] استفاده می‌کنند. ساختار BDD مشکل استفاده زیاد از حافظه را دارد، مخصوصاً وقتی که کاربرد مسیر داده مد نظر باشد [۱۳] و [۱۴]. بییر و همکاران [۱۵] سعی کردند از روش Boolean Satisfiability بعنوان جایگزینی برای BDD استفاده کنند. اما این روش نیز از ساختار سطح پایینی برخوردار بود و هنوز با کاربردهای مسیرداده مساله داشت. از طرف دیگر، ساختارهای سطح بالایی برای بررسی برابری دو طرح پیشنهاد گردید، اما قابل استفاده برای وارسی خواص نبودند [۱۶].

اخیراً استفاده از نامعادلات با متغیرهای صحیح در تولید بردار آزمون[9] مورد استفاده قرار گرفته است. در این راستا چنگ و همکاران [۱۷] روشی بر پایهٔ حل نامعادلات برای تولید اتوماتیک بردار آزمون پیشنهاد کردند. همچنین فلاح نیز روشی بر پایهٔ Satisfiability پیشنهاد کرد که با حل نامعادلات خطی، تولید بردار آزمون سطح بالا را انجام می‌دهد [۱۸] و [۱۹]. برنکمن و همکاران از روش فلاح استفاده کرده و روشی را برای وارسی مسیر داده‌ها پیشنهاد کردند. این روش اولاً نیاز به حل نامعادلات خطی دارد و ثانیا محدود به کاربردهای مسیر داده می‌باشد و قابل اعمال به بخش کنترلر نیست [۲۰].

هدف ما ارائهٔ مدل سطح بالایی برای وارسی خواص بر پایهٔ CTL [۲۱] و [۲۲] است که کاربردهای مسیر داده و کنترلر با هم را شامل گردد و نیاز به حل نامعادلات نباشد. زبان توصیف خصوصیت بر مبنای منطقهای زمانی[10] استوار می‌باشد، که می‌تواند رفتار یک سیستم را در زمانهای مشخص بیان کند. منطق زمانی CTL یک منطق شاخه‌ای می‌باشد که توسط عملگرهایی مانند EX، EG و EF، قادر به توصیف شاخه‌های مختلف یک FSM است. عملگر EX(p) نشان می‌دهد که آیا مسیری در FSM می‌توان یافت که در حالت بعد شرط p برقرار باشد. عملگر EG(p) نشان می‌دهد که آیا مسیری در FSM می‌توان یافت که در آن شرط p همیشه برقرار باشد. عملگر EF(p) نشان می‌دهد که آیا مسیری در FSM می‌توان یافت که در آن سرانجام شرط p برقرار گردد.

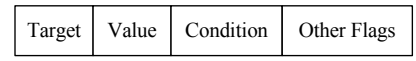
بنابراین از VHDL برای توصیف طرح و از CTL برای بیان خواص استفاده کرده‌ایم. هر خاصیت دارای فرم کلی $P \Rightarrow Q$ بوده که شامل دو بخش فرضیات[11] (P) و نتایج[12] (Q) می‌باشد [۲۳]. مراحل انجام کار به این صورت خواهد بود که ابتدا از طرح توصیف شده به زبان VHDL ساختار CDFG[13] مربوط به توابع حالتهای بعدی[14] و خروجی‌ها[15] استخراج می‌شوند و سپس CDFG به معادلات چند جمله‌ای با متغیرهای صحیح تبدیل می‌گردد و در انتها خاصیت موردنظر روی این معادلات

این مقاله در تاریخ ۴ اردیبهشت ۱۳۸۲ دریافت و در تاریخ ۱۶ اسفند ۱۳۸۲ بازنگری شد.

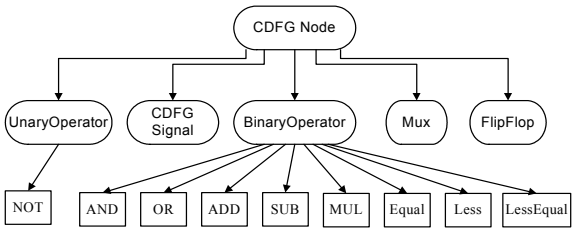
بیژن علیزاده، دانشجوی دکترای کامپیوتر, دانشکده فنی، دانشگاه تهران، انتهای خیابان کارگر شمالی، پردیس شماره ۲ (email: bijan@cad.ece.ut.ac.ir).

زین‌العابدین نوابی، گروه برق و کامپیوتر، دانشکده فنی، دانشگاه تهران، انتهای خیابان کارگر شمالی، پردیس شماره ۲ (email: navabi@ece.neu.edu).

1. Verification
2. Properties
3. Datapath
4. Functionality
5. Formal verification
6. Symbolic model checking
7. FSM
8. State transition functions
9. Test vector generation
10. Temporal logic
11. Assumptions
12. Commitments
13. Control data flow graph
14. Next state functions
15. Output functions



شکل ۱: ساختمان داده اصلی.



شکل ۲: ساختار نود CDFG.

بررسی می‌شود. مزایای این روش در زیر آورده شده است.

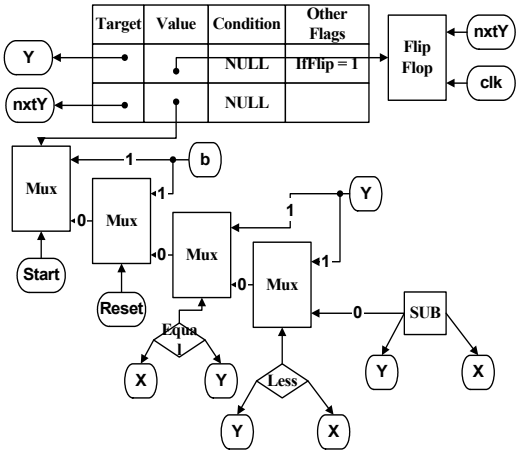
مدل پیشنهادی قابل استفاده در وارسی طرحهای در سطح سیستم می‌باشد، چون از ساختار سطح بالای معادلات صحیح سود می‌جوید. این روش محدود به کاربردهای مسیرداده یا کنترلر به تنهایی نیست و نیازی به جداسازی بخشهای مسیرداده و کنترلر نیز ندارد. مخصوصاً وقتی که مسیرداده از بردارهای با تعداد بیتهای زیاد استفاده می‌کند، روشهای بر پایه BDD قادر به ساختن BDD مربوطه نخواهند بود. اما روش پیشنهادی در این مقاله، به دلیل استفاده از ساختارهای سطح بالا، با کمترین مصرف حافظه این بردارها را تشکیل می‌دهد. بنابراین، این روش در مقایسه با روشهای بر پایهٔ BDD از لحاظ زمان اجرا و حافظهٔ استفاده شده، به مراتب بهتر عمل می‌کند. این روش بر خلاف روشهای مطرح شده در منابع [۱۵] و [۱۷] تا [۲۰]، نیاز به حل معادلات خطی یا انجام Satisfiability ندارد.

این مقاله کارمان را در شش بخش نشان خواهد داد. در بخش ۲ نشان خواهیم داد که چگونه CDFG استخراج می‌گردد و در بخش ۳ نحوهٔ تبدیل CDFG به معادلات چندجمله‌ای با متغیرهای صحیح را خواهیم دید. بخش ۴ به الگوریتم‌های وارسی خواص CTL می‌پردازد و در بخش ۵ نتایج بدست آمده مورد بررسی قرار خواهد گرفت. در بخش آخر جمع‌بندی از کل کار آورده شده است.

۲- استخراج CDFG

همانطورکه در مقدمه اشاره شد اولین فاز استخراج CDFG است. در اکثر گرافهای ارائه شده برای نمایش طرح توصیفی، اطلاعات مدار به صورت دو بخش جدا از هم نگهداری می‌شوند. این دو بخش شامل کنترلر و مسیر داده می‌باشند که در حقیقت این گرافها از دو بخش متفاوت بنامهای CFG (برای بخش کنترلی طرح) و DFG (برای بخش مسیر داده طرح) تشکیل شده‌اند. مشکل اصلی استفاده از دو گراف مجزا برای نشان دادن عملکرد مدار، در الگوریتم‌های وارسی و سنتز بخوبی مشخص می‌شود. چون در این نوع نمایش مجبور هستیم که قسمتهای کنترلر و مسیرداده را بطور جداگانه مورد وارسی قرار دهیم و بدین ترتیب نمی‌توانیم درصد زیادی از کل مدار را ارزیابی کنیم. بنابراین گراف جدیدی از ترکیب دو گراف CFG و DFG بنام CDFG در این راه استفاده شده است. یکی از ساده‌ترین روش‌های بسط گراف جریان داده از طریق اضافه کردن گره‌هایی برای بررسی شرط و انجام پرش می‌باشد.

هر طرح بصورت آرایه‌ای از عناصر شکل ۱ در نظر گرفته می‌شود که



شکل ۳: ساختار CDFG مربوط به سیگنالهای Y و $nxtY$.

سه میدان[1] نخست آن اشاره‌گرهایی به گراف شکل ۲ می‌باشند. بنابراین این ساختارها لیستی از گرافها را نشان می‌دهند که طرح توصیف شده در VHDL را مدل می‌کنند. بعبارت دیگر، یک طرح توصیف شده در VHDL با مدل معروف هافمن در نظر گرفته می‌شود و توابع مربوط به حالت بعدی و خروجی توسط نودهای CDFG شکل ۲ ساخته می‌شوند.

بعنوان مثال بزرگترین مقسوم علیه مشترک دو عدد را در نظر بگیرید که کد VHDL آن در ادامه آورده شده است:

```
ARCHITECTURE dataflow OF gcd IS
 SIGNAL nxtX، nxtY، X، Y : INTEGER;
 SIGNAL nxtReset، Reset: std_logic;
BEGIN
 PROCESS (clk)  BEGIN
  IF (clk='1' AND clk'EVENT) THEN
    X <= nxtX;  Y <= nxtY; Reset <= nxtReset;
  END IF;
 END PROCESS;
 PROCESS (start، a، b، X، Y، Reset) BEGIN
   o <= 0;
   nxtX <= X;  nxtY <= Y; nxtReset <= '0';
   IF (start='1') THEN nxtX <= a;  nxtY <= b;
   ELSE
      IF (Reset = '1') THEN
         nxtX <= a;  nxtY <= b; o <= X;
      ELSE
         IF (X = Y) THEN nxtReset <= '1';
         ELSIF (X > Y) THEN nxtX <= X - Y;
         ELSIF (Y > X) THEN nxtY <= Y - X;
         END IF;
      END IF;
   END IF;
 END PROCESS;
END dataflow;
```

CDFG استخراج شده، برای سیگنالهای Y و $nxtY$ در شکل ۳ نشان داده شده است. همانطورکه در این شکل مشخص است، میدان value مربوط به سیگنال $nxtY$ بر پایهٔ مالتی‌پلکسر ساخته می‌شود و سیگنال Y فلیپ فلاپی را نشان می‌دهد که $nxtY$ بعنوان ورودی آن می‌باشد.

۳- معادلات چندجمله ای صحیح

بعد از استخراج CDFG نوبت به تبدیل آن به معادلات چندجمله‌ای با متغیرهای صحیح می‌رسد. در روش پیشنهادی هر طرحی توسط سیستم چندتایی (PF، O، NS، PS، D=I) نشان داده می‌شود که در آن I مجموعه ورودی‌ها، $PS = (v_1, v_2, ..., v_n)$ مجموعه حالتهای

1. Field

$$a\ AND\ b;\ a,b: Boolean \rightarrow a \times b;\ a,b: Integer$$
$$a\ OR\ b;\ a,b: Boolean \rightarrow a + b - a \times b;\ a,b: Integer$$
$$NOT\ a;\ \ a: Boolean \rightarrow ۱ - a;\ a: Integer$$

شکل ۴: نحوهٔ تبدیل عملگرهای بولی به عملگرهای صحیح.

جاری[1]، $NS = (v'_۱, v'_۲, ..., v'_n)$ مجموعه حالتهای بعدی[2]، $O = (o_۱, o_۲, ..., o_n)$ مجموعه خروجی‌ها و PF مجموعه‌ای از توابع چندجمله‌ای است که مربوط به توابع حالت بعدی و خروجی، نشان داده شده در رابطه (۱)، می‌باشند.

$$\begin{aligned} v'_۱ &= f_{ns۱}(PS, I) \qquad & o_۱ &= f_{o۱}(PS, I) \\ v'_۲ &= f_{ns۲}(PS, I) \qquad & o_۲ &= f_{o۲}(PS, I) \\ &... & &... \\ v'_n &= f_{nsn}(PS, I) \qquad & o_m &= f_{om}(PS, I) \end{aligned} \tag{۱}$$

همچنین بخش P از خاصیت نیز به معادلات صحیح تبدیل شده و در مرحله وارسی خاصیت به صورت محدودیت[3] به معادلات استخراج شده از طرح اعمال خواهد شد تا آن معادلات به فرم ساده‌تری تبدیل شوند.

۳-۱ نحوه استخراج معادلات صحیح

عملگرهای بولی مانند *And*، *Or* و *Not* بر طبق شکل ۴ به معادلات چندجمله‌ای با متغیرهای صحیح تبدیل می‌گردند و سایر عمگرها، مانند *Xor* و غیره، نیز براساس این عملگرها به معادلات صحیح تبدیل می‌شوند [۱۶] و [۱۸].

از طرف دیگر، عملگرهای ریاضیاتی مانند جمع و ضرب بدون تغییر باقی می‌مانند، اما عملگرهای مقایسه‌ای فقط به عملگرهای کوچکتر و مساوی محدود می‌شوند و سایر عملگرهای مقایسه‌ای به این دو عملگر تبدیل می‌شوند.

نود مالتی‌پلکسر نیز بر طبق رابطه (۲) به معادلات چندجمله‌ای قابل تبدیل خواهد بود.

$$MUX(In۰, In۱, Sel) \Rightarrow Sel * In۱ + (۱ - Sel) * In۰ \tag{۲}$$

شکل ۵ معادلات چندجمله‌ای با متغیرهای صحیح مربوط به سیگنال *nxtY* از مثال GCD را نشان می‌دهد. پس از جایگزینی سیگنالهای میانی، به عبارت چندجمله‌ایی برای *nxtY* خواهیم رسید که فقط بر حسب ورودی‌ها و حالتهای جاری می‌باشند.

۳-۲ ساختار واحد معادلات صحیح

بطور کلی یک چند جمله‌ای صحیح که از یک متغیر تشکیل شده باشد، به فرم رابطه (۳) نشان داده می‌شود.

$$f(x) = \sum a_p . x^p \mid a_p \in \Re, p = ۰,۱,,... \tag{۳}$$

همچنین یک چند جمله‌ای شامل چند متغیر در رابطه (۴) آورده شده است.

$$\begin{aligned} &f(x_۱, x_۲, ..., x_n) = \sum a_{p۱..pn} . x_۱^{p۱} ... x_n^{pn} \mid \\ &a_{p۱..pn} \in \Re\ ,\ p_i = ۰,۱,۲,...\ \ for\ i = ۰,۱,...,n \end{aligned} \tag{۴}$$

$$T\_۲۹ <= (X == Y)$$
$$T\_۳۵ <= (Y < X)$$
$$T\_۴۳ <= (X < Y)$$
$$T\_۴۷ <= (Y - X)$$
$$T\_۴۸ <= T\_۴۳ \times T\_۴۷ + (۱ - T\_۴۳) \times Y$$
$$T\_۴۹ <= T\_۳۵ \times Y + (۱ - T\_۳۵) \times T\_۴۸$$
$$T\_۵۲ <= T\_۲۹ \times Y + (۱ - T\_۲۹) \times T\_۴۹$$
$$T\_۲۵ <= \mathrm{Re}set \times b + (۱ - \mathrm{Re}set) \times T\_۵۲$$
$$NxtY <= Start \times b + (۱ - Start) \times T\_۲۵$$

شکل ۵: نمونه‌ای از معادلات صحیح مربوط به سیگنال *nxtY*

حاصلضرب توانهای مختلف چند متغیر را Monomial یا بطور ساده MON می‌گوییم (رابطه (۵) را ببینید).

$$x_۱^{p۱} x_۲^{p۲} ... x_n^{pn} \qquad where\ p_i \in \{۰,۱,۲,..\} \quad for\ i = ۱,۲,...,n \tag{۵}$$

از مجموع چند Monomial یک چندجمله‌ای تشکیل می‌شود. برای نشان دادن یک چندجمله‌ای، ترتیبهای مختلفی برای Monomialها وجود دارد که در آنها ترتیب‌های مختلفی نیز برای متغیرها موجود است. از این رو ساختار واحدی باید برای آنها درنظر گرفت.

برای این منظور، اگر فرض کنیم که عملگر $\succ$ مشخص‌کننده الویت یک Monomial در یک چندجمله‌ای باشد، آنگاه روابط (۶) و (۷) برقرار می‌باشند.

$$if\ M_۲ \succ M_۱ \Rightarrow \forall MON: M_۳ \rightarrow M_۲ M_۳ \succ M_۱ M_۳ \tag{۶}$$

$$\forall\ M_۱, M_۲\ , M_۲ \neq ۱ \Rightarrow M_۱ M_۲ \succ M_۱ \tag{۷}$$

برای مثال، اگر ترتیب متغیرها بصورت $x_۱ > x_۲ > x_۳ > x_۴$ بوده و $M_۱ = x_۱ x_۳ x_۴$، $M_۲ = x_۱^۲ x_۳ x_۴$ و $M_۳ = x_۲$ در نظر گرفته شوند، رابطه (۶) نشان می‌دهد که اولویت $M_۲ M_۳ = x_۱^۲ x_۲ x_۳ x_۴$ از $M_۱ M_۳ = x_۱ x_۲ x_۳ x_۴$ بیشتر است. همچنین رابطه (۷) نشان می‌دهد که اولویت $M_۱ M_۳ = x_۱ x_۲ x_۳ x_۴$ از $M_۱ = x_۱ x_۳ x_۴$ بیشتر است.

بدین ترتیب، اگر برای متغیرها ترتیب $x_۱ > x_۲ > ... > x_n$ را در نظر بگیریم رابطه (۸) را خواهیم داشت.

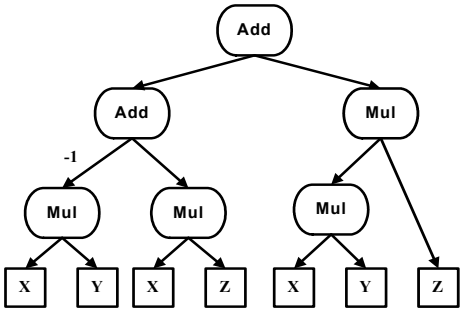
$$x_۱^{p۱} x_۲^{p۲} .... x_n^{pn} > x_۱^{q۱} x_۲^{q۲} .... x_n^{qn} \Leftrightarrow (p_۱ > q_۱) \vee ((p_i = q_i, i = ۱,۲,...,s) \wedge (p_{s+۱} > q_{s+۱}))\ where\ \ s < n \tag{۸}$$

به عنوان مثال، اگر اولویت متغیرها بصورت $x_۱ > x_۲ > x_۳$ لحاظ گردد، آنگاه اولویت $x_۱^۲ x_۲^۲$ از $x_۱^۲ x_۳^۲$ و همچنین اولویت $x_۱^۵$ از $x_۱^۲ x_۲^۸$ بیشتر خواهند شد.

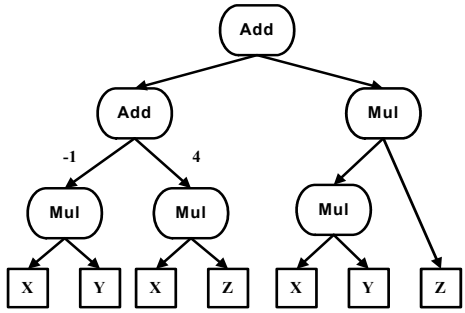
با مقدمه مطرح شده، هر معادله چندجمله‌ای با متغیرهای صحیح بصورت مجموع حاصلضربهایی نشان داده می‌شود که شامل نودهای جمع، تفریق و ضرب می‌باشد. این ساختار درختی قابل تبدیل به مجموع حاصلضرب‌هایی است که فقط نودهای جمع و ضرب را شامل می‌شوند، وقتی نودهای تفریق به جمع تبدیل شوند.

اما برای تغییر در معادلات و مقایسهٔ دو معادله، نیاز است که ساختار واحدی برای معادلات در نظر گرفته شود. این کار در دو فاز کلی انجام می‌شود. ابتدا نودهای ضرب به سطوح پایین‌تری از نودهای جمع و تفریق آورده می‌شوند و سپس نودهای تفریق به جمع تبدیل می‌گردند. ضمناً نودهای جمع به شکل واحدی تبدیل می‌گردند که فقط نود جمع دیگری

1. Present states
2. Next states
3. Constraint



شکل ۶: نمونه‌ای از نمایش مجموع حاصلضربها در ساختار واحد.



شکل ۸: ساختار واحد تابع Z*Y*X + 4*Z*X - Y*X.

را در سمت چپ خود می‌بینند و در سمت راست خود فقط نود ضرب را خواهند دید. همچنین متغیرها در نود ضرب و جمع بر اساس ترتیب از پیش تعیین شده‌ای از راست به چپ مرتب می‌شوند.

برای مثال شکل ۶ ساختار درختی تابع $F = Z \times Y \times X + Z \times X - Y \times X$ را نشان می‌دهد که ترتیب متغیرها Z، Y و X می‌باشد. برای نمایش این تابع، ابتدا به سراغ حاصل‌ضربهایی می‌رویم که شامل متغیر با بالاترین اولویت، یعنی Z، باشند که دو حاصل ضرب $Z \times Y \times X$ و $Z \times X$ یافت می‌شوند. سپس به سراغ دومین متغیر با بالاترین اولویت، یعنی Y، می‌رویم و ترم $Z \times Y \times X$ انتخاب شده و در سمت راست اولین نود جمع قرار می‌گیرد (شکل۶ را ببینید).

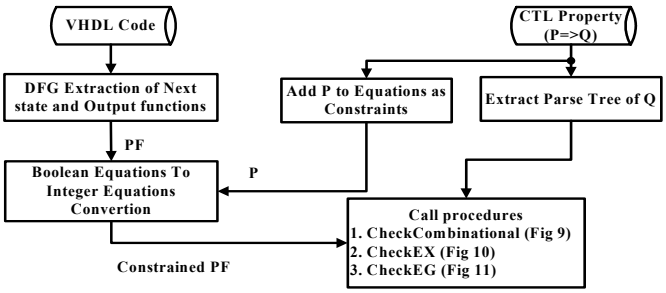
سپس ترم $Z \times X$ و در انتها ترم $Y \times X$، که به جمع تبدیل شده است، ساخته می‌شوند. لازم به ذکر است که در نود حاصلضرب، متغیرها براساس اولویت‌شان از راست به چپ قرار داده می‌شوند.

۴- بررسی خواص CTL در طرح

همانطورکه در مقدمه نیز اشاره شد، ساختار عمومی $P \Rightarrow Q$ برای خواص درنظر گرفته شده است که شامل ساختارهای زیر می‌باشند:

$$P ::= (P) \mid P \wedge P \mid \neg P \mid P = P \mid Variable \mid Integer$$
$$Q ::= P \mid EX(Q) \mid EG(Q)$$

شکل ۷ ساختار کلی کار را نشان می‌دهد. ابتدا معادلات صحیح، متناظر با حالت بعدی و خروجی، از طرح سنتز شده استخراج می‌گردد (PF) و سپس بخش P از خاصیت بصورت محدودیتی به این معادلات اضافه می‌شود. از طرف دیگر، ساختار درختی از بخش Q استخراج می‌شود تا مشخص گردد چه رویه‌ای در هر سطح از درخت باید صدا زده شود. سه رویهٔ ChechCombinational، CheckEX و CheckEG کار وارسی فلوچارت شکل ۷ را انجام می‌دهند.



شکل ۷: فلوچارت کل کار.

```
CheckCombinational (Q)
  1. Convert Q to Integer Equations
  2. Add them to other Equations, i.e. PF
  3. Do Replacement and Simplification,
       i.e. canonical form convertion
  4. If there is an invalid Equation
       Verification Fail!
  5. Else      Verification O.K.
```

شکل ۹: رویهٔ بخش ترکیبی

۴-۱ جایگزینی و ساده‌سازی به جای حل معادلات

از آنجایی که حل معادلات چندجمله‌ای با متغیرهای صحیح جزء مسایل سخت[1] می‌باشد و راه حل دقیقی برای آن وجود ندارد، به سراغ روشی رفته‌ایم که نیاز به حل معادلات نباشد. در نتیجه، یکی از مزایای کار ما نسبت به روش برینکمن [۲۰] عدم نیاز به حل معادلات می‌باشد. ضمن اینکه کار ما محدود به کاربردهای مسیرداده نمی‌شود و قابل اعمال به هر دو بخش مسیرداده و کنترلر می‌باشد.

به عنوان مثال، معادلات مربوط به سیگنال $nxtY$ را مجدداً در نظر بگیرید (شکل۵ را ببینید). فرضیات $Start = ۰$ و $Reset = ۰$ منجر می‌گردند تا $nxtY = T\_۲۵$ و $T\_۲۵ = T\_۵۲$ شوند. بنابراین هرعملیاتی که باید بر روی $nxtY$ انجام شود، بر روی $T\_۵۲$ انجام خواهد شد. مثالهای تکمیلی در بخش۴-۲ آورده شده است.

توجه شود که رویهٔ ساده‌سازی[2] در طول فاز تبدیل به ساختار واحد صورت می‌گیرد، چون جابجایی نودهای جمع از سمت راست به سمت چپ و ساختن نودهای ضرب براساس ترتیب متغیرها، باعث می‌گردند تا جمله‌های یکسان در یک موقعیت قرار گیرند و عمل ساده‌سازی روی آنها صورت پذیرد (بخش ۳-۲ را ببینید). بعنوان مثال اگر $F۱ = ۳ \times Z \times X$، $F۲ = Z \times Y \times X + Z \times X - Y \times X$ و ترتیب متغیرها Z ، Y و X درنظر گرفته شوند و بخواهیم $F۱ + F۲$ را محاسبه کنیم، خواهیم داشت: ساختار واحد تابع $F۲$ در شکل۶ نشان داده شده است که وقتی با $F۱$ جمع شود، عبارت $۳ \times Z \times X$ در $F۱$ با عبارت $Z \times X$ در $F۲$ از لحاظ مکانی یکسان خواهند بود. بنابراین ضرایب ثابت آنها با هم جمع شده و عبارت $۴ \times Z \times X$ را خواهیم داشت. شکل ۸ ساختار واحد $F۱ + F۲$ را نشان می‌دهد.

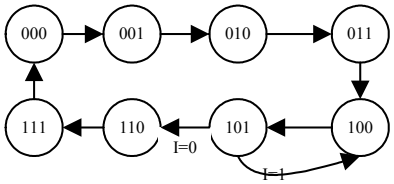
1. NP-Complete
2. Simplification procedure

```
CheckEX(Q)
  1. Convert state variables to next
       state variables in Q
  2. Convert them to Integer Equations
  3. Add them to other Equations, i.e. PF
```

شکل ۱۰: رویهٔ عملگر حالت بعد.

```
CheckEG (Q, Limitation)
 Z1 = Q (i = 1)
For i = 1 to i < Limitation
        1. exZi = EX( Zi )
        2. Zi+1 = exZi * Q
        3. IF Zi+1 == Zi      RETURN Zi+1
```

شکل ۱۱: رویهٔ عملگر تمام حالتها.



شکل ۱۲: مثال شمارندهٔ خاص.

۴-۲ الگوریتم‌های بررسی خواص CTL

شکل ۹ رویهٔ CheckCombinational را نشان می‌دهد، وقتی که بخش Q بصورت ترکیبی است و شامل عملگرهای حالت[1] نمی‌باشد. بخش Q نشان می‌دهد چه نودهای خروجی نیاز به بررسی دارند. این نودها از روی مجموعه معادلات، یعنی رابطه (۱)، بدست آمده و معادلات صحیح آنها با محدودیت P، در نظر گرفته می‌شوند. سپس سیگنالهای میانی موجود در Q جایگزین می‌شوند تا در نهایت به توابعی برای خروجی‌های استفاده شده در Q برسیم که فقط بر حسب ورودی‌ها و حالتهای جاری هستند. در انتها به معادلاتی می‌رسیم که نشان‌دهندهٔ شرایط ارضاء خاصیت موردنظر می‌باشند.

شکل ۱۰ رویهٔ CheckEX را نشان می‌دهد که بخش Q تنها شامل عملگر حالت بعدی، یعنی X، می‌باشد. درستی خاصیت در سه قدم زیر مورد بررسی قرار می‌گیرد:

۱- حالتهای جاری موجود در Q به حالتهای بعدی تبدیل می‌شوند.

۲- حالتهای بعدی استفاده شده در قدم ۱ با معادلات مربوطه از مجموعه معادلات استخراج شده از بخش طرح، یعنی PF، جایگزین می‌شوند.

۳- در فاز تبدیل به ساختار واحد، عملیات ساده‌سازی معادلات قدم ۲ صورت می‌گیرد.

برای مثال خاصیت $Start = ۰ \,\&\, Reset = ۰ \,\&\, X = ۳ \,\&\, Y = ۶ \Rightarrow EX(Y = ۳)$ را در نظر بگیرید. ابتدا بخش Q این خاصیت به $nxtY = ۳$ تبدیل شده و سپس معادله چندجمله‌ای مربوط به $nxtY$، از معادلهٔ (۱)، جایگزین $nxtY$ در معادلهٔ $nxtY = ۳$ می‌شود. برای روشن‌تر شدن موضوع، مجدداً به شکل ۵ مراجعه شود که معادلات مربوط به سیگنال $nxtY$ در آن آورده شده است. فرض $Start = ۰$ منجر به $nxtY = T\_۲۵$ و فرض $Reset = ۰$ منجر به $T\_۲۵ = T\_۵۲$ می‌شوند. از طرف دیگر، شرطهای $X = ۳$ و $Y = ۶$ باعث می‌گردند که مقادیر ۰، ۰، ۱ و ۳ به ترتیب به سیگنالهای $T\_۲۹$، $T\_۳۵$، $T\_۴۳$ و $T\_۴۷$ داده شوند. بنابراین $T\_۴۹ = T\_۵۲$، $T\_۴۹ = T\_۴۸$ و $T\_۴۸ = T\_۴۷$ خواهند شد. در انتها معادله $nxtY = ۳$ تبدیل به معادله $T\_۴۷ = ۳$ می‌شود. معادله نهایی یک عبارت همیشه درست است، چون مقدار سیگنال $T\_۴۷$ برابر با ۳، در مرحله قبل، محاسبه شده است. بنابراین خاصیت بیان شده درست است.

شکل ۱۱ رویهٔ CheckEG را نشان می‌دهد که بخش Q شامل عملگر تمام حالتها[2]، یعنی G، می‌باشد و رابطه (۹) در طول این رویه بصورت تکراری محاسبه می‌گردد.

$$Z_۱ = Q; \qquad Z_{i+۱} = Q \wedge EX(Z_i) \qquad (۹)$$

در هر تکرار، $EX(Z_i)$ در سه مرحله محاسبه می‌شود که در رویهٔ CheckEX توضیح داده شد. این الگوریتم زمانی به پایان می‌رسد که $Z_i = Z_{i+۱}$ شود یا تعداد تکرار از پارامتر Limitation، که تعداد حالتهای طرح را مشخص می‌کند، تجاوز کند. توجه شود که این برابری به سادگی صورت می‌پذیرد چراکه ساختار واحد بدست آمده از آن دو این امکان را فراهم می‌کند تا مقایسه نود به نود صورت پذیرد.

پارامتر Limitation بیشتر با هدف محدود کردن تعداد تکرارها بیان شده است و براساس تعداد حالتهای هر مدار قابل محاسبه است. نحوه محاسبه این تعداد، توسط الگوریتم‌های مختلف بر پایه روش چک کردن مدل ارائه شده است که قابل استفاده در اینجا نیز می‌باشند [۲] و [۳]. برای سادگی کار می‌توان مقدار ثابتی برای این پارامتر در نظر گرفت تا نگران ارائه روش دقیقی برای محاسبه این پارامتر نباشیم.

بعنوان مثال خاصیت $Start = ۰ \,\&\, Reset = ۰ \,\&\, X = ۱۸ \,\&\, Y = ۳ \Rightarrow EG(Y = ۳)$ را درنظر بگیرید. معادلهٔ (۱۰) تکرارهای اول و دوم این مثال را نشان می‌دهد. در اولین تکرار $Y = ۳$ درنظر گرفته می‌شود. در تکرار دوم $Y = ۳ \,\&\, EX(Y = ۳)$ باید محاسبه شود. برای محاسبه این معادله، ابتدا $EX(Y = ۳)$ بر طبق فرضیات محاسبه می‌شود (شکل ۵ معادلات مربوط به سیگنال $nxtY$ را نشان می‌دهد). این محاسبه مانند مثال الگوریتم CheckEX انجام شده و منجر به $Y = ۳$ می‌گردد. چون $Z_۱ = Z_۲$ است، انتهای الگوریتم می‌باشد و نشان می‌دهد که خاصیت درست است.

$$\begin{aligned} &Z_۱ = (Y = ۳); \\ &Z_۲ = (Y = ۳) \wedge EX(Y - ۳ = ۰); \end{aligned} \qquad (۱۰)$$

1. State operators

2. All states

جدول ۱: مقایسه روش پیشنهادی با روش VIS.

مدار نمونه		TLC	GCD	SC	EL	2CA
P۱	PIE	۳۱	۱	۰/۴۴	۲۶	۱۹
	VIS	۹۵	۶	۱/۱	۷۲	۵۸
P۲	PIE	۳۴	۱/۵	۰/۳۵	۲۸	۱۵
	VIS	۱۴۲	۱۱	۰/۹	۸۴	۴۹
NN	PIE	۲۴۹۵	۳۴۵	۶۶	۱۸۴	۸۶
	VIS	۲۵۲۱۴۶	۱۰۵۷۶۴	۱۱۳	۲۰۳۳۷	۶۹۲
MU	PIE	۳/۲	۰/۵۲۵	۰/۲۲	۰/۳۱	۰/۲۵
	VIS	۱۰/۱	۷/۵	۴/۷	۵/۲	۴/۸

P۱: زمان اجرای خاصیت ۱ بر حسب ثانیه
P۲: زمان اجرای خاصیت ۲ بر حسب ثانیه
NN : تعداد نودها بر حسب BDD یا DFG
MU : حافظه مصرف شده بر حسب مگابایت
PIE (Polynomial Integer Equation) : روش پیشنهادی ما

۵- نتایج

برای اینکه ارائهٔ مناسبی از زمان اجرا و میزان حافظهٔ استفاده شده توسط روش پیشنهادی داشته باشیم، پنج مثال متنوع کنترل‌کننده چراغ راهنما[1]، بزرگترین مقسوم علیه مشترک[2]، کنترل‌کننده آسانسور[3]، آربیتر با دو مشتری[4] و شمارندهٔ با حالت خاص[5] شکل ۱۲ در نظر گرفته شده است [۲۴]. این مثالها شامل ساختارهای مسیرداده و کنترلر بوده و برای اثبات ادعای ما مناسب می‌باشند.

همچنین روش پیشنهادی با روش برینکمن [۲۰]، در مثال‌های مسیرداده‌ای، مقایسه شده است که از لحاظ حافظه مصرفی و زمان اجرا وضعیت بهتری را نشان می‌دهد. این امر به دلیل استفاده روش برینکمن از موتور حل معادلات است که باعث می‌گردد حافظه مصرفی و زمان اجرا افزایش یابند. ضمن اینکه روش برینکمن در مثالهای کنترلی کارایی لازم را ندارد.

دو خاصیت زیر برای مثال کنترل‌کننده چراغ راهنما در نظر گرفته شده است:

۱- $yellow\_expire = ۰ \;\&\; farm\_light = YELLOW \;\&\; hwy\_light = RED \Rightarrow EX(hwy\_light = GREEN)$

این خاصیت نشان می‌دهد که اگر چراغ بزرگراه قرمز و چراغ مسیر دیگر زرد باشند و زمان زرد بودن به اتمام رسیده باشد، باید در حالت بعدی چراغ بزرگراه سبز شود.

۲- $frame\_light = RED \;\&\; hwy\_light = GREEN \;\&\; car\_present = ۰ \Rightarrow EG(hwy\_light = GREEN)$

این خاصیت بدین معنی است که اگر چراغ بزرگراه سبز و چراغ مسیر دیگر قرمز باشد ولی ماشینی در مسیر دیگر وجود نداشته باشد، باید چراغ بزرگراه سبز بماند.

دو خاصیت زیر برای مثال بزرگترین مقسوم علیه مشترک مورد ارزیابی قرار می‌گیرد:

۱- $Reset = ۰ \;\&\; Start = ۰ \;\&\; X = Y \Rightarrow EX(Reset = ۱)$ این خاصیت می‌گوید که اگر X با Y برابر باشد، در حالت بعد باید $Reset$ فعال شود.

۲- $Reset = ۰ \;\&\; Start = ۰ \;\&\; X = ۱۵ \;\&\; Y = ۵ \Rightarrow EX^{۲}(X = Y)$ این خاصیت نشان می‌دهد که اگر $X = ۳Y$ باشد، باید در دو حالت بعد X با Y برابر گردد.

دو خاصیت زیر برای مثال شمارنده شکل۱۲ لحاظ شده است:

۱- $I = ۱ \;\&\; Count = ۵ \Rightarrow EX^{۲}(Count = ۵)$ این خاصیت بیان می‌کند که اگر $Count$ برابر با ۵ باشد، دو حالت بعد نیز $Count$ می‌تواند ۵ باقی بماند.

۲- $I = ۱ \;\&\; Count = ۵ \Rightarrow EX(Count = ۴)$ این خاصیت نشان می‌دهد که اگر $Count$ برابر با ۵ باشد، حالت بعد $Count$ می‌تواند ۴ شود.

دو خاصیت زیر برای مثال آسانسور مورد بررسی قرار می‌گیرد:

۱- $Start = ۰ \;\&\; door = CLOSED \Rightarrow EG(door = CLOSED)$ این خاصیت می‌گوید که اگر در بسته باشد، می‌توان مسیری را یافت که در آن مسیر در همیشه بسته می‌ماند.

۲- $Start = ۰ \;\&\; door = CLOSED \Rightarrow EG(movement = MOVING)$ این خاصیت بیان می‌کند که اگر در بسته باشد، می‌توان مسیری را یافت که در آن مسیر آسانسور همیشه در حال حرکت باشد.

دو خاصیت زیر برای مثال آربیتر با دو مشتری در نظر گرفته شده است:

۱- $Start = ۰ \;\&\; (req۱ \; or \; req۲ = ۱) \Rightarrow EG(clin = A)$ این خاصیت نشان می‌دهد که اگر مشتری اول یا دوم درخواستی داشته باشند، می‌توان حالتهایی را یافت که همیشه مشتری اول انتخاب گردد. این خاصیت درست نمی‌باشد.

۲- $Start = ۰ \;\&\; req۱ = ۱ \Rightarrow EG(\; pass\_token۱ = ۰)$ این خاصیت می‌گوید که اگر مشتری اول درخواستی داشته باشد، می‌توان مسیری در طرح یافت که همیشه مشتری دوم فعال شود. این خاصیت نیز نادرست است.

جدول ۱ زمان اجرا و میزان حافظه مصرف شده در این مثالها را نشان می‌دهد که با VIS مقایسه شده است [۲۴]. لازم به ذکر است که VIS ابزار وارسی بر پایه BDD می‌باشد و در دانشگاه برکلی توسعه یافته است. همانطورکه این جدول نیز نشان می‌دهد، زمان اجرا و حافظه استفاده شده در روش پیشنهادی ما بهتر از VIS می‌باشد. در مثال کنترل‌کننده چراغ راهنما، خاصیت اول حدود ۳۱ ثانیه بر روی یک سیستم پنتیوم III با ۲۵۶ مگابایت RAM، زمان نیاز داشته است، در حالی که زمان اجرای VIS حدود ۹۵ ثانیه بوده است. همچنین حافظه استفاده شده برای روش ما حدود ۳/۲ مگابایت و برای VIS حدود ۱۰/۱ مگابایت بوده است.

مثال بزرگترین مقسوم علیه مشترک دو عدد ۸ بیتی، نتایجی را نشان می‌دهد که گواهی بر ادعای خوب بودن روش ما برای کاربردهای مسیر داده و کنترلر می‌باشد. این سطر از جدول ۱ نشان می‌دهد که VIS حدود ۷/۵ مگابایت حافظه مصرف کرده است و ۱۰۵۷۶۴ نود BDD ساخته شده است. در حالی که روش ما ۰/۵۲۵ مگابایت مصرف حافظه داشته و ۳۴۵ نود CDFG تولید کرده است. دلیل این امر به سطح بالا بودن نحوه نمایش طرح GCD برمی‌گردد. در روش پیشنهادی ساختمان داده سطح بالایی بر پایه معادلات صحیح ارائه شده است که منجر به استفاده کمتر

1. Traffic light controller (TLC)
2. Greatest common divisor (GCD)
3. Elevator controller (EL)
4. 2 Client arbiter (2CA)
5. Special counter (SC)

از حافظه برای نمایش طرحهای در سطح انتقال ثبات[1] می‌شود. اما VIS مجبور است که طرح مربوطه را به سطح بیت تبدیل کرده و توسط BDD آنرا نشان دهد و همین موضوع باعث افزایش قابل توجه مصرف حافظه شده است.

لازم به ذکر است که ما از نسخه تحت ویندوز VIS استفاده کرده‌ایم و زمان اجرای ذکر شده در جدول ۱ تنها مربوط به بخشی از VIS است که توابع EX یا EG صدا زده می‌شوند. برای این منظور، توابع محاسبه زمان اجرا به ابتدا و انتهای آن توابع اضافه شده تا مقایسه‌ها واقعی شوند.

۶- جمع بندی

به منظور غلبه بر مشکلات مربوط به ساختمان داده‌های سطح پایین مانند BDDها، مدل سطح بالایی بر پایه معادلات چندجمله‌ای با متغیرهای صحیح پیشنهاد گردید. این مدل ما را قادر می‌سازد تا توصیف‌های سطح بالا در ساختار سطح بالایی نگه داشته شده و الگوریتم‌های بررسی خواص در سطح پایین، با کمی تغییرات، قابل اعمال به این مدل باشند. مزایای این روش عبارتند از، اولاً قابل استفاده بودن در وارسی طرحهای در سطح سیستم، چون از ساختار سطح بالای معادلات صحیح سود می‌جوید. دوم این که این روش محدود به کاربردهای مسیر داده یا کنترلر به تنهایی نیست و نیازی به جداسازی بخشهای مسیر داده و کنترلر ندارد. سوم این که این روش در مقایسه با روشهای بر پایهٔ BDD از لحاظ زمان اجرا و حافظهٔ استفاده شده، به مراتب بهتر عمل می‌کند. چهارم این که این روش، برخلاف روشهای دیگر [۱۷]، [۱۹]، [۲۰] و [۲۵]، نیاز به حل معادلات خطی یا انجام Satisfiability ندارد و مراحل جایگزینی و ساده‌سازی را انجام می‌دهد. بعبارت بهتر، یک نوع شبیه‌سازی نمادین[2] انجام می‌گیرد.

مراجع

[1] R. P. Kurshan, "Formal verification in a commercial setting," in *Proc. Design Automation Conf.*, pp. 258-262, Jun. 1997.
[2] S. Devadas, H. T. Ma, and A. R. Newton, "On the verification of sequential machines at differing levels of abstraction," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 6, pp. 713-722, Jun. 1988.
[3] M. Yoeli, *Formal Verification of Hardware Design*, IEEE Computer Society Press, Los Alamos, NM, 1990.
[4] P. Camurati and P. Prinetto, "Formal verification of hardware correctness," *IEEE Computer*, vol. 21, no. 7, pp. 8-19, Jul. 1988.
[5] A. Gupta, S. Malik, and P. Ashar, "Toward formalizing a validation methodology using simulation coverage," in *Proc. IEEE Design Automation Conf.,* pp. 740-745, Jun. 1997.
[6] R. Eastham and K. Thirunarayan, "Proof strategies for hardware verification," in *Proc. of National Aerospace and Electronics Conf. (NAECON),* vol. 2, pp. 451-458, May 1996.
[7] G. Cabodi, P. Camurati, and F. Corno, "Sequential circuit diagnosis based on formal verification techniques," in *Proc. IEEE Int. Test Conf.*, pp. 187-196, Sep. 1992.
[8] K. McMillan, *Symbolic Model Checking,* Kluwer Academic Publishers, Boston, 1993.
[9] J. Burch, E. Clarke, K. McMillan, and D. Dill, "Symbolic model checking: 1020 states and beyond," in *Proc. of the Fifth Annual IEEE Symp. on Logic in Computer Science,* pp. 428-439, Jun. 1990.
[10] M. C. McFarland, "Formal verification of sequential hardware," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 5, pp. 633-645, May 1993.
[11] C. Kern and M. R. Greenstreet, "Formal verification in hardware design," *ACM Trans. on Design Automation of Electronic Systems,* vol. 4, no. 2, pp. 123-193, Apr. 1999.
[12] J. Burch, E. Clarke, D. Long, K. McMillan, and D. Dill, "Symbolic model checking for sequential circuit verification," *IEEE Trans. Computer Aided Design*, vol. 13, no. 4, pp. 401-424, Apr. 1994.
[13] J. R. Burch, E. M. Clarke, and K. L. McMillan, "Sequential circuit verification using symbolic model checking," in *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 46-51, 1990.
[14] H. Touati, H. Savoj, and B. Lin, "Implicit state enumeration of finite state machines using BDD's," in *Proc. of the 2th Int. Conf. on Computer Aided Design*, pp. 130-133, Nov. 1990.
[15] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. of Design Automation Conf.*, pp. 317-320, Jun. 1999.
[16] R. Drechsler, *Formal Verification of Circuits*, Kluwer Academic Publishers, 2000.
[17] C. Y. Huang and K. T. Cheng, "Assertion checking by combined word-level ATPG and modular arithmetic constraint-solving techniques," in *Proc. of Design Automation Conf.*, pp. 118-123, Jun. 2000.
[18] F. Fallah, S. Devadas, and K. Keutzer, "Functional vector generation for HDL models using linear programming and 3-satisfiability," in *Proc. of 35th DAC*, pp. 528-533, Jun. 1998.
[19] F. Fallah, Coverage Directed Validation of Hardware Models, Ph.D. Thesis, MIT, 1999.
[20] R. Brinkmann and R. Drechsler, "RTL-datapath verification using integer linear programming, " in *Proc. of IEEE VLSI Design'01 & Asia and South Pacific Design Automation Conf.,* pp. 741-746, Bangalore, Jan. 2002.
[21] E. Clarke and E. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," *in Logic of Programs Workshop*, pp. 52-71, 1981.
[22] E. Clarke, R. Enders, and T. Filkorn, "Exploiting symmetry in temporal logic model checking," *Formal Methods in System Design,* vol. 9, no. 2, pp. 77-104, 1996.
[23] I. Beer, S. Ben-David, C. Eisner, and A. Landver, "RuleBase: an industry-oriented formal verification tool," in *Proc. of the 33rd Design Automation Conf.*, pp. 655-660, Las Vegas, 1996.
[24] R. K. Brayton, A. Sangiovanni, A. Aziz, et al., "VIS: A system for verification and synthesis," in *Proc. of the 8th Int. Conf. on Computer Aided Verification*, pp. 428-432, 1996.
[25] J. C. Corbett and George S. Avrunin, "Using integer programming to verify general safety and liveness properties," in *Journal of Formal Methods in System Design*, vol. 6, no. 1, pp. 97-123, Jan. 1995.

بیژن علیزاده تحصیلات خود را در مقاطع کارشناسی کامپیوتر (سخت افزار)، و کارشناسی ارشد معماری کامپیوتر به ترتیب در سالهای ۱۳۷۴ و ۱۳۷۷ از دانشگاه تهران به پایان رسانده است و هم اکنون دانشجوی دکترای کامپیوتر در دانشگاه تهران می‌باشد. نامبرده مدت ۴ سال در شرکت نیمه هادی عماد بعنوان مدیر بخش طراحی فیزیکی و همچنین طراح مدارهای دیجیتالی در سطح تراشه مشغول بوده است. همچنین به مدت ۳ سال به توسعه الگوریتم‌های شبیه‌سازی و ارزیابی پرداخته است. زمینه‌های تحقیقاتی مورد علاقه ایشان عبارتند از: الگوریتمهای وارسی، طراحی مدارهای سخت افزاری در سطح تراشه، الگوریتمهای طراحی فیزیکی، آزمون‌پذیری در سطح بالا، سنتز مدارهای دیجیتالی، ساختمان داده و الگوریتمهای هوش مصنوعی

زین‌العابدین نوابی در سال ۱۳۵۴ مدرک کارشناسی الکترونیک خود را از دانشگاه تگزاس و در سال ۱۳۵۷ مدرک کارشناسی ارشد الکترونیک خود را از دانشگاه آریزونا کشور آمریکا و در سال ۱۳۶۰ مدرک دکترای الکترونیک خود را از همان دانشگاه دریافت نمود و هم اکنون دانشیار دانشگاه تهران و نورس ایسترن آمریکا می‌باشد. نامبرده قبل از پیوستنش به دانشگاه تهران در سالهای ۱۳۶۱ الی ۱۳۶۲ و ۱۳۶۵ الی ۱۳۶۶ استادیار دانشگاه آریزونا در ایالت متحده آمریکا و در سالهای ۱۳۶۲ الی ۱۳۶۵ استادیار دانشگاه شریف بوده است. سپس در بین سالهای ۱۳۶۶ الی ۱۳۷۰ و ۱۳۷۰ الی ۱۳۷۱ به ترتیب استادیار و دانشیار دانشگاه نورس ایسترن آمریکا گردید. ایشان در سال ۱۳۷۰ به دانشگاه تهران پیوست. زمینه‌های تحقیقاتی مورد علاقه ایشان متنوع بوده و شامل موضوعاتی مانند وارسی مدارهای دیجیتالی، طراحی مدارهای سخت افزاری، آزمون پذیری، طراحی SOC و توسعه ابزار CAD می‌باشد.

1. Register transfer level
2. Symbolic simulation