PHÁT HIỆN LỖI TRÀN SỐ TRÊN MÔ HÌNH HỆ THỐNG NHÚNG SỬ DỤNG PHƯƠNG PHÁP PHÂN TÍCH TĨNH

Đỗ Thị Bích Ngọc
Học Viện Công Nghệ Bưu Chính Viễn Thông

Tóm tắt: Ngày nay, các hệ thống nhúng đòi hỏi tính an toàn ngày cao, bên cạnh độ phức tạp cũng tăng theo. Vì vậy, phân tích, đánh giá ngay từ mô hình hệ thống nhúng đã và đang thu hút sự quan tâm của cả giới nghiên cứu và công nghiệp. Trong đó, việc phát hiện nguy cơ lỗi tràn số trong các hệ thống nhúng là quan trọng và cần thiết vì nó có thể dẫn tới sai lệch nghiêm trọng trong hệ thống. Lỗi tràn số có thể xảy ra do kiểu dữ liệu trong các hệ thống nhúng thường bị giới hạn số bít. Thêm vào đó, các tính toán bên trong hệ thống có thể dẫn tới các giá trị quá lớn so với phạm vi biểu diễn. Bài báo này đề xuất một phương pháp phát hiện lỗi tràn số cho các mô hình hệ thống nhúng dựa trên phương pháp phân tích tĩnh (static analysis) kết hợp với công cụ giải các ràng buộc SMT.

Từ khoá: ràng buộc SMT, mô hình hệ thống nhúng, lỗi tràn số, MATLAB/Simulink, phân tích tĩnh.

I. GIỚI THIỆU

Hệ thống nhúng

Hệ thống nhúng đang phát triển mạnh mẽ và ngày càng đóng vai trò quan trọng trong cuộc sống của con người. Lỗi của hệ thống nhúng có thể gây ra tai nạn khủng khiếp, đặc biệt là các hệ thống điều khiển máy bay, tên lửa, hệ thống điều khiển động cơ ô tô... Vì vậy, nhiều hệ thống nhúng có yêu cầu cao về chất lượng, tính ổn định và độ tin cậy. Bên cạnh đó, lỗi trên hệ thống nhúng có thể không sửa được, nếu sửa được thì chi phí cũng rất cao, phải thu hồi sản phẩm hoặc thiết kế lại toàn bộ. Đó là lý do các công cụ hỗ trợ thiết kế mô hình các hệ thống nhúng được áp dụng ngày càng nhiều. Việc thiết kế mô hình hệ thống nhúng trên các công cụ trước khi thiết kế mẫu thật là cần thiết để dễ dàng phát hiện, sửa lỗi cũng như chỉnh sửa thiết kế nhằm đảm bảo chất lượng. Có nhiều nghiên cứu, công cụ hỗ trợ việc đánh giá, kiểm tra các mô hình các hệ thống nhúng [7,8,9,10,13,14]. Trong các loại lỗi của hệ thống nhúng, lỗi tràn số được cần được kiểm tra, đánh giá thường xuyên.

Lỗi tràn số (number overflow)

Lỗi tràn số xảy ra khi giá trị số trong hệ thống vượt quá khả năng biểu diễn như trong thiết kế. Ví dụ số integer 8 bit chỉ có thể biểu diễn giá trị lớn nhất là 2^8 -1 = 255, trường hợp số lớn hơn sẽ vượt quá khả năng biểu diễn. Lỗi tràn số có dẫn tới hệ thống tính toán sai lệch nghiêm trọng. Một số ví dụ về lỗi tràn số sau đây cho thấy tầm nghiêm trọng của chúng:

- Vào tháng 8/2016, một máy casino ở Resorts World Casino đã in 1 giải thưởng trị giá $42,949,672.76 do lỗi tràn số. Phía Casino về sau đã xảy ra kiện tụng với khách hàng với giải thích là giải thưởng lớn nhất chỉ có $10.000, bất kì giải thưởng nào lớn hơn là do lỗi của hệ thống.[1]
- Vào tháng 6/1996, tên lửa Ariane 5 đã phát nổ sau khi được phóng 37 giây, gây thiệt hại 370 triệu $. Một lỗi tràn số đã xảy ra khi chuyển đổi số 64 bit sang số 16 bit dẫn tới điều khiển bị sai lệch. [2]

Vì vậy việc kiểm soát để hệ thống không bị tràn số là quan trọng, đặc biệt trong các hệ thống cần dùng các kiểu dữ liệu với khả năng biểu diễn giá trị nhỏ (8 bit, 16 bit).

Lỗi tràn số thường khó có thể phát hiện. Lý do là lỗi tràn số thường xảy ra sau một chuỗi các tính toán số học trong quá trình hệ thống thực hiện, chứ không phải chỉ xảy ra ngay từ input đầu vào. Việc kiểm thử để phát hiện lỗi tràn số cũng gặp thách thức lớn vì phải nghĩ ra các bộ dữ liệu để có thể dẫn tới các giá trị tràn số.

Một số phương pháp phổ biến được dùng để phát hiện lỗi tràn số:

Phân tích tĩnh (static analysis): đây là một phương pháp hiệu quả để phát hiện lỗi tràn số một cách tự động thông qua phân tích các luồng dữ liệu hoặc áp dụng các kĩ thuật dựa trên ràng buộc (constraint-based techinque) [3,4,6].

Kiểm thử: xây dựng các bộ dữ liệu kiểm thử lớn và thực hiện kiểm thử xem có lỗi hay không. Để xác định các ca kiểm thử có khả năng gây ra lỗi, một số nghiên cứu đã kết

Tác giả liên hệ: Đỗ Thị Bích Ngọc,
Email: dothibichngoc@gmail.com
Đến tòa soạn: 03/8/2021, chỉnh sửa: 24/9/2021, chấp nhận đăng: 12/10/2021.

[1] https://www.thesun.co.uk/news/2088642/woman-takes-slot-machine-selfie-of-43-million-casino-win-and-has-big-buck-payout-denied/

[2] http://sunnyday.mit.edu/nasa-class/Ariane5-report.html

hợp cả phân tích tĩnh, và một số cải tiến khác [7,8,9,13].

Rà soát thủ công: thực hiện phân tích và đánh giá xem có nguy cơ lỗi tràn số không. Phương pháp này sẽ phụ thuộc vào kinh nghiệm của người rà soát, trong nhiều trường hợp sẽ không đưa ra kết quả chính xác và tốn nhiều công sức.

Bài báo này đề xuất 1 phương pháp để phát hiện lỗi tràn số cho các mô hình hệ thống nhúng dựa trên phương pháp phân tích tĩnh áp dụng công cụ giải SMT (Satisfiability Modulo Theories). Các điểm có nguy cơ tràn số sẽ được kiểm tra bằng các công cụ giải ràng buộc SMT. Trường hợp có nguy cơ tràn số, một bộ dữ liệu kiểm thử (test data) tương ứng sẽ được xác định. Như vậy, thay vì phải xác định các bộ dữ liệu kiểm thử bằng tay, các công cụ SMT có thể được áp dụng để phát hiện các nguy cơ tràn số và các bộ dữ liệu tương ứng.

Các nghiên cứu liên quan

Trong nghiên cứu [4], tác giả đề xuất một phương pháp thiết kế phần cứng cho phép phát hiện các lỗi tràn số nguyên ở mức đặc tả. Cách tiếp cận thiết kế đã thiết lập sử dụng ngôn ngữ đặc tả SysML / OCL ở cấp đặc tả trong đó các kiểu số nguyên là vô hạn. Các kiểu này không chia sẻ hành vi ngữ nghĩa của các kiểu được triển khai trong ngôn ngữ lập trình SystemC vì trong SystemC, số nguyên là hữu hạn. Do vậy, vấn đề nảy sinh là khoảng cách ngữ nghĩa giữa hai cấp độ này. Để khắc phục điều này, tác giả đã sử dụng 1 công cụ trợ lý chứng minh Coq và sử dụng thư viện số nguyên CompCert mô tả các kiểu số nguyên hữu hạn thông qua các kiểu phụ thuộc. Bằng cách đó, các lỗi tràn số có thể được phát hiện và xác minh. Nghiên cứu đã hướng tới phát hiện lỗi tràn số, nhưng yêu cầu các đặc tả phải được biểu diễn bằng SysML/OCL.

Trong [6], tác giả đã trình bày một cách tiếp cận để kiểm tra bất biến của mô hình Simulink dựa trên các nguyên tắc của Bounded model checking (BMC) bằng cách sử dụng lý thuyết mô đun thỏa mãn (SMT) [2]. Hai đóng góp của tác giả: thứ nhất, tác giả cho thấy rằng có thể tự động tạo ra các đường thực thi hữu hạn trực tiếp dựa trên mô hình Simulink và thứ hai, tác giả cho thấy rằng có một số thiết kế Simulink nhất định mà thủ tục bất biến giới hạn đã hoàn tất. Để tự động hóa cách tiếp cận được đề xuất, tác giả giới thiệu những điều sau: i) một thủ tục tự động (được triển khai trong công cụ SyMC) để tạo đường dẫn thực thi có độ dài hữu hạn dựa trên mô hình Simulink và thứ tự thực thi của các khối, ii) mã hóa dựa trên khuôn mẫu của các đường dẫn thực thi sang định dạng SMT-LIB [2] phù hợp để phân tích bằng bộ giải Z3 SMT của Microsoft [1,12]. Nghiên cứu hướng tới kiểm chứng các tính chất bất biến cho mô hình Simulink, nhưng không đề cập tới bài toán kiểm chứng cho lỗi tràn số.

II. TỔNG QUAN MÔ HÌNH HỆ THỐNG NHÚNG VÀ PHƯƠNG PHÁP PHÂN TÍCH TĨNH

2.1 Các khái niệm cơ bản về mô hình hệ thống nhúng

Có nhiều công cụ dùng để thiết kế mô hình các hệ thống nhúng, trong đó MATLAB/Simulink là một công cụ được sử dụng nhiều trong cả nghiên cứu và thực tế. Bài báo sẽ thực hiện phân tích cho các mô hình nhúng biểu diễn bằng MATLAB/Simulink [11].

a. Biểu diễn mô hình hệ thống nhúng bằng Simulink

Simulink là một môi trường tích hợp cho phát triển dựa trên mô hình của các hệ thống nhúng. Nó cung cấp một giao diện đồ hoạ với các tính năng mô hình hoá, mô phỏng, tự động sinh mã nguồn… có khả năng sử dụng thực tế trong công nghiệp.

Một mô hình hình hệ thống nhúng trong Simulink được tạo bởi nhiều loại blocks (các khối mạch đơn được thiết kế sẵn) kết nối với nhau thông qua các tín hiệu dùng để mô hình hoá điều khiển và luồng dữ liệu bên trong mô hình. Các loại blocks bao gồm: inport/outport (vào/ra), mathematical operator (phép toán), logical/relational operator (phép logic/quan hệ), (multiport) switch, delay... Các blocks liên kết với nhau bằng lines, truyền dữ liệu Boolean, integer hoặc floating/fixed point… giữa chúng. Đặc biệt, một mô hình hệ thống nhúng cho phép nhận được một số tín hiệu (các giá trị liên tục theo thời gian) bằng cách sử dụng các block Inport và tạo ra một số tín hiệu đầu ra được đại diện bởi các block Outport.

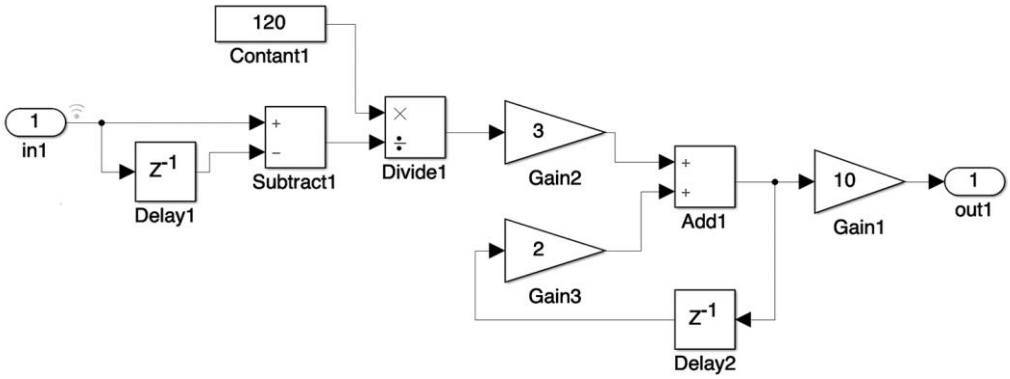
b. Phân loại blocks:

Block đơn: là các block đơn không có trạng thái con bên trong, thường thay đổi giá trị output khi input thay đổi. Ví dụ: relational (phép quan hệ), logic (phép logic), arithmetics (phép số học)…

Delay block: là các block làm chậm giá trị input 1 khoảng thời gian. Có nhiều block thuộc loại này, ví dụ: UnitDelay, RateTransition,…

SFunction: là các block được định nghĩa bởi người dùng thực hiện 1 tính năng/nhiệm vụ nhất định. Sfunction là một subsystem gồm nhiều block con hợp thành, và gồm các inputs và outputs tương ứng.

Cách kết hợp blocks: có 2 cách kết hợp các block chính: tuần tự và lặp. Với cách kết hợp tuần tự, dữ liệu đi từ block này sang block kia nhưng không tạo thành vòng lặp. Với cách kết hợp lặp, tồn tại 1 vòng lặp trong quá trình dữ liệu truyền trong mô hình. Ví dụ trong Hình 1, ta có vòng lặp: block Subtract2 về Delay2, từ Delay2 về Gain3, cuối cùng từ Gain3 quay về Subtract2.



Hình 1: Mô hình có vòng lặp biểu diễn trong Simulink

c. Lỗi tràn số trong mô hình hệ thống nhúng

Lỗi tràn số xảy ra khi giá trị số trong hệ thống vượt quá khả năng biểu diễn như trong thiết kế. Khi lỗi tràn số xảy ra, số đó có thể bị biến thành số rất nhỏ hoặc giá trị âm. Một số tình huống tràn số: (1) kết quả tính toán sai lệch; (2) chỉ số mảng nhầm hoặc trỏ vào vùng trái phép; (3) giá trị số điều khiển vòng lặp bị nhầm lẫn, hoặc dẫn tới lặp vô hạn.

Lỗi tràn số có thể xảy ra do giá trị đầu vào quá lớn. Tuy nhiên, đa phần là do các block tính toán số tăng/giảm giá trị, đặc biệt là trong các vòng lặp. Mặc dù giá trị output có thể nằm trong giới hạn biểu diễn, tuy nhiên, các giá trị trung gian trong quá trình tính toán có thể vượt ngưỡng cho phép.

2.2 Phân tích tĩnh cho mô hình hệ thống nhúng

Ý tưởng cơ bản của phân tích tĩnh sử dụng ràng buộc SMT là kiểm tra (giá trị phủ định của) một tính chất với độ sâu cho trước. Cho một hệ thống M, một tính chất φ và một biên k, phân tích tĩnh sẽ trải hệ thống ra k lần để thay thế cho vòng lặp và biến đổi nó thành một rang buộc SMT tương ứng. Bằng cách đó, điều kiện là thoả mãn nếu và chỉ nếu φ có 1 phản ví dụ (counterexample) với độ sâu nhỏ hơn hoặc bằng k. Vì vậy, các công cụ giải ràng buộc SMT có thể được áp dụng để kiểm tra điều kiện có thoả mãn hay không.

Giới thiệu về SMT [2]

Lý thuyết tính thỏa được (SMT - Satisfiability Modulo Theories) là kiểm tra sự thỏa mãn của một ràng buộc logic trong đó có sử dụng một hay nhiều kiểu dữ liệu. Các kiểu dữ liệu và phép toán tương ứng trên nó có thể là số thực, số nguyên và các cấu trúc dữ liệu như danh sách, mảng, bit vector.

Nhiệm vụ của SMT là kiểm tra xem một ràng buộc có thoả mãn hay không. Chúng ta gọi ràng buộc F là thỏa mãn (Satifiablity) nếu có một phép gán làm cho F đúng (True), ví dụ ràng buộc sau:

(c == a + b) && (a < 256) && (b < 256) && (c>255)

là thỏa mãn (True) khi a = 2, b = 255, c = 257

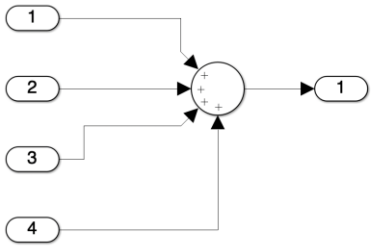
Có nhiều công cụ giải ràng buộc SMT, như Open-SMT, CVC3, Boolecto, Z3, Yices, MathSAT…. Trong những năm trở lại, nhờ sự cải tiến của các thuật toán, cấu trúc dữ liệu, các phương pháp heuristics, các công cụ SMT hiện tại có thể giải quyết các ràng buộc lớn với hàng chục nghìn biến, ràng buộc. Trong bài báo này, Bộ giải Z3 của Microsoft [1, 12] được áp dụng để phát hiện lỗi tràn số trong mô hình hệ thống nhúng.

Biến đổi các mô hình hệ thống nhúng thành ràng buộc SMT

Với mỗi block trong Simulink, và cho từng bước thời gian (time step), ta sẽ tạo ra một công thức ràng buộc SMT sao cho ngữ nghĩa vẫn được bảo toàn. Cụ thể là công thức ràng buộc SMT thể hiện được giá trị đầu vào (input), đầu ra (output) tương ứng với ngữ nghĩa của block. Bên cạnh đó, do trong SMT không có khái niệm vòng lặp, để thể hiện vòng lặp trong mô hình hệ thống nhúng, ta thực hiện trải mô hình hệ thống nhúng ra k lần bằng cách tạo k ràng buộc cho mỗi block, đánh số t. Phần dưới đây minh hoạ cách biến đổi một số block điển hình: block thực hiện phép toán, phép so sánh, rẽ nhánh, trễ thời gian. Với lưu ý j thể hiện ràng buộc ở lần lặp thứ j, k là số lần lặp.

Block Sum: thực hiện tính tổng các giá trị input, có 4 input (inSum_1,…inSum_4) và 1 output (outSum) như Hình 2. Ràng buộc tương ứng là:

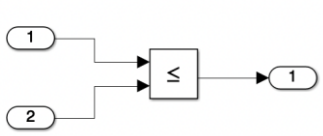
$$\wedge_{j=1}^{k} outSum_j == inSum\_1_j + inSum\_2_j + inSum\_3_j + inSum\_4_j$$



Hình 2: Ví dụ về block sum – tính tổng

Block relation less than: thực hiện phép so sánh nhỏ hơn giữa 2 inputs, có 2 input (inLT_1, inLT_2) và 1 output (outLT) như Hình 3. Ràng buộc tương ứng là:

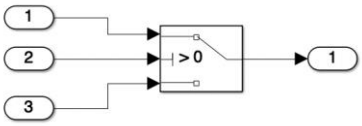
$$\wedge_{j=1}^{k} \left(outLT_j == inLT\_1_j \leq inLT\_2_j \right)$$



Hình 3: Ví dụ về block relation - so sánh

Block switch: thực hiện chọn input 1 hay input 3 dựa trên giá trị của tín hiệu điều khiển input 2, có 1 input điều khiển (inS_2), 2 input dữ liệu (inS_1, inS_3), và 1 output (outS) như Hình 4. Ràng buộc tương ứng là:

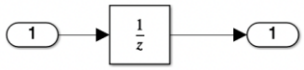
$$\Lambda_{j=0}^{k-1}\Big(\big((inS\_2_j > 0) \wedge (outS_j == inS\_1_j)\big) \vee \big(!(inS\_2_j > 0) \wedge (outS_j == inS\_3_j)\big)\Big)$$



Hình 4: Ví dụ về block switch - lựa chọn

Block UnitDelay: thực hiện truyền ra output trễ 1 đơn vị thời gian, với 1 input (inD) và 1 output (outD), giá trị mặc định output ở thời gian t= 0 là ic như Hình 5. Ràng buộc tương ứng là:

$$(out_0 = c) \wedge \Lambda_{j=1}^{k-1}(outD_j = inD_{j-1})$$



Hình 5: Ví dụ về block UnitDelay - làm trễ

III. ÁP DỤNG BỘ GIẢI SMT ĐỂ PHÁT HIỆN LỖI TRÀN SỐ TRONG MÔ HÌNH HỆ THỐNG NHÚNG

3.1. Các block có thể gây ra lỗi tràn số

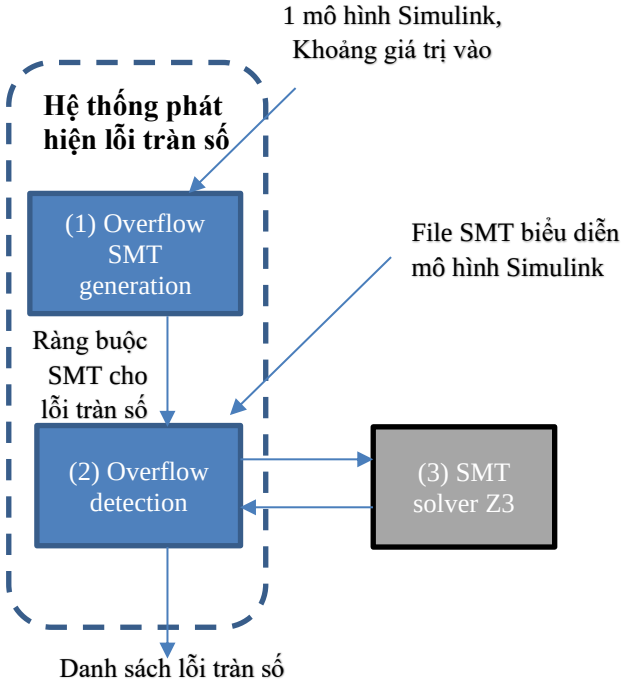
Trong Simulink [11], có nhiều loại block khác nhau. Tuy nhiên, các block thuộc loại tính toán (Math operations) có nguy cơ gây ra lỗi tràn số nhiều nhất. Bài báo này tập trung vào phân tích các block thuộc loại Math Operations. Đầu tiên, các block thuộc loại Math Operations sẽ được đánh giá để xem có nguy cơ gây ra tràn số hay không. Mặc dù các block này cho output dựa trên tính toán các input đầu vào, tuy nhiên, với các block không làm tăng giá trị của input sẽ không gây ra lỗi tràn số. Phụ lục 1 là kết quả phân tích chi tiết các block thuộc Math Operations có thể ảnh hưởng tới lỗi tràn số.

3.2. Giải pháp đề xuất

Bài báo đề xuất 1 giải pháp phát hiện lỗi tràn số minh hoạ như ở Hình 6 với 2 module chính (1), (2), và sử dụng thư viện sẵn có (3).

(1) Overflow SMT generation: với mô hình Simulink, thực hiện sinh các ràng buộc SMT để kiểm tra điều kiện vi phạm tràn số cho loại block Mathematic trong mô hình tuân thủ theo Phụ lục 1. Các ràng buộc được sinh theo nguyên tắc ở Bảng 1. Với MIN_NUM, MAX_NUM là giới hạn trên và giới hạn dưới của kiểu dữ liệu số tương ứng). Bên cạnh đó, để tối ưu quá trình phát hiện lỗi tràn số, các block được sắp xếp theo thứ tự block gần inport block trước, xa inport block sau.

(2) Overflow detection: đây là module chính thực hiện việc phát hiện lỗi tràn số. Với từng block có nguy cơ tràn số,



Hình 6: Giải pháp đề xuất

module này sẽ nhận file smt tương ứng với mô hình Simulink, và ràng buộc tràn số SMT của block tương ứng. Từ đó gọi (3) SMT solver Z3 để giải ràng buộc và kết luận có nguy cơ tràn số hay không, cùng với bộ giá trị cho block input tương ứng với nguy cơ tràn số. Chi tiết module 3 thể hiện ở Thuật toán 1.

(3) SMT solver Z3: là công cụ thực hiện giải ràng buộc và trả về nghiệm cho trường hợp ràng buộc thoả mãn (có tràn số). (4) được gọi bởi (3) để phát hiện lỗi tràn số.

Bảng 1: Ràng buộc SMT để phát hiện lỗi tràn số

Loại Block	Ràng buộc số	Ràng buộc SMT
Inport block	Khoảng giá trị cho trước của inport inp là [a,b], mô hình được trải k lần. ∀i ∈ [1,k], inp_i>= a && inp_i	assert(and (inp_1>=a) (inp_1<=b)) …

	<=b	assert(and (inp_k>=a) (inp_k<=b))
Block không có nguy cơ tràn số	Không cần ràng buộc	
Block có nguy cơ tràn số	Với block b, nếu mô hình được trải k lần, ràng buộc để phát hiện lỗi tràn số: $\forall i \in [1,k]$, !(b_i>= MIN_NUM && b_i <=MAX_NUM)	assert(not(and (b_1>=MIN_NUM) (b_1<=MAX_NUM))) … assert(not(and (b_k>=MIN_NUM) (b_k<=MAX_NUM)))

Thuật toán 1: Phát hiện lỗi tràn số
input:

- R: ràng buộc SMT của mô hình nhúng

- B: Danh sách cặp (block, constraint) gồm block cần kiểm tra lỗi tràn số và ràng buộc SMT tương ứng.

Output:

- T: Danh sách cặp (block, testdata) gồm block có nguy cơ lỗi tràn số và dữ liệu kiểm thử tương ứng

```
1. T = ∅;
2. while B ≠ ∅
3.    rb = B[0];// lấy đầu danh sách
4.    constraint =  (R+ rb.constraint) ;
5.    result = callZ3(constraint); //gọi Z3
6.    if (result.SAT == true) //là lỗi tràn số
7.       t.block = rb.block;
8.       t.testdata = extractInput(result.data);
9.       T = add(T, t);//thêm t vào T
10.      tConstraint = makeConstraint(t.testdata) ;
11.      tR = R + tContraint;
12.      for b ∈ B
13.         tResult = callZ3(tR + b.constraint);
14.         if (tResult.SAT == true)
15.            B= remove (B,b);//xoá b khỏi B
16.            tb.block = b.block; tb.testdata = t.testdata;
17.            T = add(T, tb);//thêm tb vào T
18.         end if
19.      end for
20.   end if //(result.SAT==true)
21. end while
22. return T;
23. end.
```

Thuật toán 1 nhận đầu vào là ràng buộc SMT của 1 mô hình nhúng, danh sách cặp (block, constraint) gồm thông tin block và ràng buộc cho lỗi tràn số tương ứng của các block có nguy cơ tràn số trong mô hình nhúng (sinh ra bởi (2) Overflow SMT generation trong Hình 6); trả về danh sách (block, testdata) là các block bị lỗi tràn số và bộ dữ liệu cho các inport block (testdata) dẫn tới lỗi tràn số. Thuật toán duyệt từng phần tử rb của B để kiểm tra ràng buộc tràn số tương ứng có thoả mãn hay không bằng cách gọi tới Z3 (dòng 5). Trường hợp Z3 trả về kết quả là thoả mãn (dòng 6) nghĩa là rb bị tràn số, khi đó ta sẽ thêm thông tin block tương ứng (dòng 7) và dữ liệu cho các inport đầu vào gây ra tràn số (dòng 8) vào danh sách T (dòng 9).

Bên cạnh đó, dựa trên nhận xét khi 1 block bị tràn số, các block tiếp theo nó cũng có nguy cơ bị tràn số. Để phát hiện lỗi tràn số nhanh hơn, tiết kiệm thời gian chạy Z3, thuật toán thực hiện bổ xung 1 ràng buộc tương ứng với bộ dữ liệu cho các inport đầu vào gây tràn số (dòng 10) và với từng phần tử của b, kiểm tra lỗi tràn số bằng cách gọi Z3 với ràng buộc mới bổ xung này (dòng 13). Trường hợp phát hiện thêm lỗi tràn số (dòng 14), thông tin block sẽ được thêm vào T (dòng 16,17).

IV. THỰC NGHIỆM VÀ ĐÁNH GIÁ

Bảng 2 trình bày kết quả phân tích lỗi tràn số theo phương pháp đã đề xuất. Cột đầu tiên là tên các mô hình, với M1 là mô hình ở Hình 7, M2 là mô hình ở Hình 6, và M3 một mô hình với một số tính toán khác ở Hình 8. Cột 2 là số lượng block toán học có nguy cơ tràn số trong mô hình, cột 3 là thông tin về việc có tồn tại vòng lặp trong mô hình hay không, cột 4 là khoảng giá trị của các input block trong mô hình, cột 5 là kiểu dữ liệu lựa chọn cho mô hình, cột cuối là số lượng block bị phát hiện lỗi tràn số.

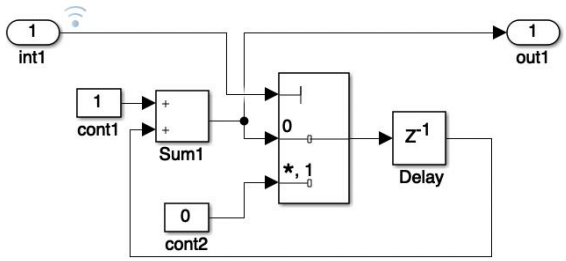
Bảng 2: Kết quả thử nghiệm

Mô hình	**Số Math blocks**	**Có vòng lặp?**	**Khoảng giá trị của input block**	**Kiểu dữ liệu**	**Số block bị tràn số**
M1	1	Yes	[-100,100]	Int 8	0
				Int 16	0
M2	6	Yes	[-100,100]	Int 8	4
				Int 16	3

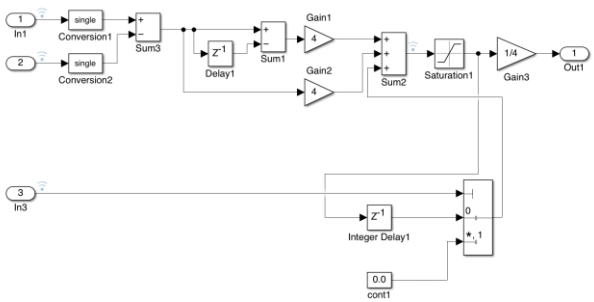
M3	9	No	[-2000, 2000]	Fixed point (1,16,0)	1
				Single	0

Kết quả thử nghiệm cho thấy phương pháp đề xuất đã phát hiện thành công lỗi tràn số cho các mô hình, kể cả có vòng lặp. Khi kiểu dữ liệu lớn hơn, số lượng lỗi tràn số sẽ giảm đi.

Tuy nhiên, phương pháp đề xuất không phát hiện ra lỗi tràn số của mô hình M1 (Hình 7). Lý do là do mô hình này mô phỏng bộ đếm số, với mỗi vòng lặp (sau 1ms) chỉ tăng 1 đơn vị, lỗi tràn số chỉ xảy ra khi khi trải hệ thống MAX lần lặp. Tuy nhiên, phương pháp đề xuất không thể phát hiện lỗi xuất hiện với số lần lặp vượt quá k lần (là số lần trải hệ thống như trình bày ở Mục 2.2), giá trị này thường không được thiết lập quá lớn để đảm bảo hiệu năng của phương pháp.



Hình 7: Mô hình Simulink bộ đếm thời gian mili giây



Hình 8: Mô hình Simulink với nhiều tính toán số

Ngoài ra, bài báo mới phân tích mới chỉ tập trung vào các block có nguy cơ gây lỗi tràn số nhiều nhất (Math Operations), chưa xử lý được với các mô hình dùng nhiều block phức tạp, có block subsystem (hệ thống con) . Tuy nhiên, chúng ta có thể khắc phục bằng cách thực hiện phân tích từng sub system con.

V.KẾT LUẬN

Bài báo đề xuất một phương pháp để phát hiện lỗi tràn số cho các mô hình hệ thống nhúng, từ đó giúp người thiết kế có thể cải tiến hệ thống tốt hơn. Phương pháp đề xuất áp dụng một bộ giải SMT (Z3) để tìm ra nghiệm của các ràng buộc tương ứng với mô hình hệ thống nhúng và có nguy cơ gây ra lỗi tràn số. Từ đó, đưa ra danh sách các block có lỗi tràn số và bộ dữ liệu của inport block tương ứng. Phương pháp đề xuất đã tự động phát hiện ra các block có nguy cơ gây ra lỗi tràn số, cũng như tự động tính toán ra bộ dữ liệu cho các inport block tương ứng.

Hướng phát triển tiếp theo của bài báo là mở rộng phát hiện lỗi cho các mô hình lớn hơn, cũng như nghiên cứu các kĩ thuật để tìm các ràng buộc đặc trưng của vòng lặp (loop invariant) nhằm khắc phục hạn chế đã nêu ở Mục 4.

PHỤ LỤC 1: CÁC BLOCK TÍNH TOÁN TRONG SIMULINK CÓ ẢNH HƯỞNG TỚI LỖI TRÀN SỐ

Block	Mô tả	Phân tích	Nguy cơ tràn số
Abs	Trả về giá trị tuyệt đối của input	Chỉ lấy giá trị tuyệt đối	Không
Add	Cộng/trừ các inputs	Có thể tăng/giảm giá trị	Có
Algebraic Constraint	Ràng buộc cho các tín hiệu input	Có thể tăng/giảm giá trị	Có
Assignment	Gán giá trị	Không làm tăng/giảm giá trị	Không
Bias	Thêm độ lệch (bias) vào input	Có làm tăng giá trị	Có
Complex to Magnitude-Angle	Tính toán độ lớn và / hoặc góc pha của tín hiệu phức	Không tạo ra giá trị lớn hơn hiện tại	Không
Complex to Real-Imag	Tính phần thực/phần ảo của 1 tín hiệu số phức	Không tạo ra giá trị lớn hơn hiện tại	Không
Divide	Chia 1 input cho 1 input khác	Có thể tăng/giảm giá trị	Có
Dot Product	Tính dot product của 2 vectors	Có thể tăng/giảm giá trị	Có
Find Nonzero	Tìm phần tử khác 0 trong mảng	Không thay đổi giá trị	Không

Elements			
Gain	Nhân input với 1 hằng số	Có thể tăng/giảm giá trị	Có
Magnitude-Angle to Complex	Biến đổi độ lớn và / hoặc góc pha thành tín hiệu phức	Không tạo ra giá trị lớn hơn hiện tại	Không
Math Function	Thực hiện các hàm toán học	Có thể tăng/giảm giá trị	Có
MinMax	Trả về giá trị lớn nhất/nhỏ nhất của 1 input	Không thay đổi giá trị	Không
MinMax Running Resettable	Xác định giá trị lớn nhất/nhỏ nhất của 1 tín hiệu theo thời gian	Không thay đổi giá trị	Không
Permute Dimensions	Biến đổi lại số chiều của mảng nhiều chiều.	Không thay đổi giá trị	Không
Polynomial	Đánh giá các hệ số của đa thức theo giá trị input	Có thể tăng/giảm giá trị	Có
Product	Nhân và chia các ma trận vô hướng và không vô hướng hoặc nhân và nghịch đảo ma trận.	Có thể tăng/giảm giá trị	Có
Product of Elements	Sao chép hoặc đảo ngược một đầu vào vô hướng hoặc thu gọn một đầu vào không vô hướng	Có thể tăng/giảm giá trị	Có
Real-Imag to Complex	Chuyển đổi các đầu vào thực và / hoặc ảo thành tín hiệu số phức.	Không tạo ra giá trị lớn hơn hiện tại	Không
Reshape	Thay đổi chiều của tín hiệu	Không thay đổi giá trị	Không
Rounding Function	Áp dụng chức năng làm tròn	Không tạo ra giá trị lớn hơn hiện tại	Không
Sign	Xác định dấu của input	Không thay đổi giá trị	Không
Sine Wave Function	Tạo sóng sin, sử dụng tín hiệu bên ngoài	Có thể tăng/giảm giá trị	Có
Slider Gain	Tích với 1 số biến đổi sử dụng slider	Có thể tăng/giảm giá trị	Có
Sqrt	Tính căn bậc hai, căn bậc hai có dấu hoặc nghịch đảo của căn bậc hai	Không tạo ra giá trị lớn hơn hiện tại	Không
Squeeze	Loại bỏ các chiều 1 giá trị (singleton) khỏi tín hiệu đa chiều	Không thay đổi giá trị	Không
Trigonometric Function	Hàm lượng giác với input	Không tạo ra giá trị lớn hơn hiện tại	Không
Unary Minus	Nghịch đảo input	Không tạo ra giá trị lớn hơn hiện tại	Không
Vector Concatenate, Matrix Concatenate	Kết hợp các tín hiệu input của cùng một kiểu dữ liệu để tạo ra tín hiệu output liên tục.	Không thay đổi giá trị	Không
Weighted Sample Time Math	Hỗ trợ tính toán liên quan đến thời gian lấy mẫu	Không thay đổi giá trị	Không

TÀI LIỆU THAM KHẢO

[1] Andreas, F.; Armin, B.; Christoph, M. W.; Youssef, H. Stochastic Local Search for Satisfiability Modulo Theories. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence 2015, 1136-1143

[2] Barrett, C.; Stump, A.; and Tinelli, C. 2010. The SMT-LIB standard: Version 2.0. In SMT'10.

[3] Bessa, I.; Abreu, R.; Filho, J. E.; and Cordeiro, L., SMT-based bounded model checking of fixed-point digital controllers. IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society, 2014, pp. 295-301, doi: 10.1109/IECON.2014.7048514.

[4] Bornebusch, F.; Lüth, C.; Wille, R. and Drechsler, R., Integer Overflow Detection in Hardware Designs at the Specification Level. In Proceedings of the 8th International Conference on Model-Driven Engineering and Software Development - MODELSWARD, 2020, pp. 41-48, ISBN 978-989-758-400-8 ISSN 2184-4348, pages 41-48. DOI: 10.5220/0008960200410048.

[5] Eldib, H. & Wang, C., An SMT Based Method for Optimizing Arithmetic Computations in Embedded Software Code, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, pp. 129-136, doi: 10.1109/FMCAD.2013.6679401.

[6] Filipovikj, P.; Rodriguez-Navas, G. and Seceleanu, C., Bounded invariance checking of simulink models. SAC '19: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 2168-2177. 10.1145/3297280.3297493.

[7] Holling, D., Pretschner, A. and Gemmar, M., September. 8cage: lightweight fault-based test generation for simulink. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 2014, pp. 859-862, ACM.

[8] Lennon, C.; Iury, B.; Lucas, C.; Daniel, K. and Eddie, L. Verifying digital systems with MATLAB. In Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017). Association for Computing Machinery, New York, NY, USA, 2017, pp 388–391. DOI:https://doi.org/10.1145/3092703.3098228.

[9] Matinnejad, R.; Nejati, S.; Briand, L. C. and Bruckmann, T., Test Generation and Test Prioritization for Simulink Models with Dynamic Behavior, in IEEE Transactions on Software Engineering, vol. 45, no. 9, pp. 919-944, 1 Sept. 2019, doi: 10.1109/TSE.2018.2811489.

[10] Matinnejad, R.; Nejati, S.; Briand, L. C. & Bruckmann, T. (2016, May). SimCoTest: A test data generation tool for Simulink/Stateflow controllers. In Proceedings of the 38th International Conference on Software Engineering Companion (pp. 585-588). ACM.

[11]. MathWorks. Simulink. https://www.mathworks.com/products/simulink.html. Ngày truy cập: 1/8/2021

[12] Microsoft Research. *The Z3 Theorem Prover*: https://github.com/Z3Prover/z3. ngày truy cập 01/08/2021.

[13] Ren, H.; Bhatt, D. & Hvozdovic, J. Improving an Industrial Test Generation Tool Using SMT Solver. In Procedings of NASA Formal Methods Symposium. 2016, 100-106. 10.1007/978-3-319-40648-0_8.

[14] Shiva, N.; Khouloud, G.; Claudio, M.; Lionel. C. B.; Stephen, F.; & David, W. Evaluating model testing and model checking for finding requirements violations in Simulink models. In Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019). Association for Computing Machinery, New York, NY, USA, 2019, 1015–1025. DOI:https://doi.org/10.1145/3338906.3340444.

DETECTING NUMBER OVERFLOW ERROR IN EMBEDDED MODEL USING STATIC ANALYSIS TECHNIQUE.

Abstract: Nowadays, embedded systems require high safety. Besides, these systems are more and more complex. Thus, anayzling, testing the embedded system model has been attracted many attention and investment of both academic research and industry communities. In embedded systems, oveflow error may occurdue to limited data type (e.g. 8 bits, 16 bits). This error may cause wrong computation in the system. Thus, detecting overflow error in embdeded model is important and neccesary. This paper proposed a method to detect overflow error in embedded model based on static analysis and SMT solver.

Keywords: embedded model, MATLAB/Simulink, SMT solver, overrflow error, static analysis.

Đỗ Thị Bích Ngọc sinh ngày 08/03/1981 tại Hà Nội. Năm 2004, bà tốt nghiệp Kỹ sư tài năng Công nghệ thông tin tại Trường Đại học Bách khoa Hà Nội. Năm 2007 bà nhận bằng Thạc sĩ khoa học tại Trường Đại học Sư Phạm Hà Nội. Năm 2010, bà nhận học vị Tiến sĩ chuyên ngành Khoa học thông tin tại Viện Khoa học và Công nghệ Tiên tiến Nhật Bản (JAIST). Từ năm 2010-2012, bà làm sau tiến sĩ tại viện AIST – Nhật Bản. Từ 2013- nay, bà là Giảng viên tại Học viện Công nghệ Bưu chính Viễn thông.

Lĩnh vực nghiên cứu: Phân tích mã nguồn, Kiểm thử phần mềm, Kiểm chứng phần mềm, Công nghệ phần mềm.