VHDL 모델의 효율적인 검증 방법

김강철

여수대학교

Efficient Strategies to Verify VHDL Model

Kangchul Kim

Yosu National University

E-mail : kkc@yosu.ac.kr

요 약

논 논문에서는 VHDL 모델을 검증하기 위하여 정지법을 사용할 때 클럭 사이클을 줄일 수 있는 2가지 방법을 제안한다. 첫 번째 방법은 세미랜덤변수를 정의하고, 정지법이 동작 중에 세미랜덤변수의 영역에 존재하는 데이터를 생략하여 정지점(stopping point)을 줄이고, 두 번째 방법은 정지법의 페이즈가 변화시에 베이지안 파라미터의 기존 값을 그대로 유지하여 클럭 사이클을 줄이는 방법이다. 제안된 방법의 효율성을 입증하기 위하여 12개의 VHDL 모델에 대하여 분기검출률에 관한 모의실험을 하였다.

ABSTRACT

This paper presents two strategies to reduce clock cycles when using stopping rule in VHDL model verification. The first method is that a semi-random variable is defined and the data that stay in the range of semi-random variable are skipped when stopping rule is running. The second one is to keep the old values of parameters when phases are changed. 12 VHDL models are examined to observe the effectiveness of strategies.

키워드

Verification, stopping rule, branch coverage, stopping point, semi-random variable

Ⅰ. 서 론

*VLSI*의 공정기술이 급격하게 발전함에 따라 CPU, 메모리, 제어로직, 버스제어회로 등을 하나의 칩 속에 구현하는 것이 가능하게 되었다. 그러나 설계기술과 복잡한 칩의 검증은 공정기술에 따라가지 못하고 있으며, 칩이 커짐에 따라 VHDL에 대한 설계검증은 칩 설계에 있어서 점점 더 어려워지고 시간을 소모하는 과정이 되고 있다. SoC(systems on chips), IP(intellectual property), ASIC 등의 설계시간에 대한 70%는 검증과정에서 소모되고 있다.[1]

1990년 이후부터 하드웨어 표현 언어를 이용한 설계기술이 급격하게 증가하였으며, 대부분의 복잡한 칩은 VHDL에 의해서 설계되고 있다. VHDL은 개념적으로 동시에 수행되는 많은 과정을 가지고 있는 전형적인 모델로 이루어진 복잡한 병렬처리 언어이다. 따라서 VHDL 프로그램의 오류 검증시에 소프트웨어 영역에서 정의된 검출률 측정법(coverage metrics)을 빌려와 하드웨어 설계의 검증에 사용하고 있다.

모든 테스트벤치(testbench)가 VHDL 코드를 성공적으로 검증하였어도 검증된 VHDL 코드가 기능적으로 100% 정확하다는 것을 증명하는 것은 불가능하기 때문에 코드 검출률(code coverage)이 사용된다. 이러한 코드 검출률을 검사하는 방법에는 문장(statement), 분기(branch), 표현(expression), 경로(path) 검출률 등 여러 종류의 방법들이 사용되고 있다.[2-4] 복잡한 VHDL 코드에 검출률 측정법을 사용하는 데는 일반적으로 임의 테스트패턴(random test pattern)를 사용하고 있으므로 많은 클럭 사이클(clock cycle)이 소모되어야 한다. 즉 [5], [6]에서는 확실하게 고장이 없는 칩을 설계하기 위해서는 50억 명령어 클럭 사이클을 모의실험에 사용해야 한다는 것을 보여준다.

그러므로 VHDL 코드의 검증과정에서 검증 시간과 비용을 줄이기 위하여 좋은 stopping rule과 전략(strategy)을 사용하는 것은 매우 중요하다.

본 논문에서는 VHDL 코드의 검증과정에서 클럭 사이클을 줄이기 위한 2 가지 방법을 제안한다. 첫 번째 방법으로 세미임의변수(semi-random variable)가 정의되고, stopping rule이 동작할 때 세미임의변수가 존재하는 영역에 있는 데이터를 건너뛰는 방법이다. 두 번째 방법은 stopping rule의 상태(phase)가 바뀔 때 이전의 파라미터를 사용하는 방법이다. 제안된 방법을 사용하여 VHDL 코드를 검증한 결과 분기 검출률의 손실이 아주 작으면서 25% 이상의 클럭 사이클이 감소한 것을 확인하였다.

본 논문의 2장에서는 VHDL 코드 검증시간을 줄이기 위한 기존의 방법들을 설명하고, 3장에서 본 논문에서 제안한 2 가지 방법에 대하여 기술한다. 그리고 4장에서 제안된 방법에 대한 모의실험 결과를 보여주고, 기존의 방법들과 비교한다. 5장에서 결론을 기술한다.

II. 기존의 방법

소프트웨어 프로그램을 검증하기 위하여 많은 방법들이 개발되어 왔으며[7-10], 최근에는 포아손 분포(Poisson distribution)와 베이지안 예측(Bayesian estimation)을 사용하는 방법들이 소개되고 있다.[7][11]

[7]에서는 경험적 베이지안 원리(empirical Bayesian principles)을 사용하는 합성계산과정(compounded counting process)이 제안되었고, VHDL 프로그램의 검증에 stopping rule이 소개되었다. 주된 아이디어는 두 확률분포 즉 인터럽션(interruptin)의 수와 크기를 조합하는 것이다. 확률분포의 파라미터는 유명한 베이지안 예측을 사용하여 임의변수(random variable)로 가정된다.

[11]에서는 테스팅의 매 클럭 사이클에서 분기검출률의 결과를 관찰하고 시간 t에서 검출된 분기의 수를 예측하였다. 그리고 검출된 수는 분기검출 임의프로세스 X_t에 의해서 표현되었다. 분기검출 임의프로세스는 두 개의 확률로 나누어진다. 즉 시간 t에서 하나 또는 그 이상의 새로운 분기가 검출되는 인터럽션 N_t와 인터럽션의 크기 이다. 그리고 인터럽션 발생(interruption occurrences)의 조건적 분포함수 D_t가 정의되었다. 그리고 PMF(probability mass function) 추출실험을 통하여 매 이산시간 t에서 가장 적합한 분포함수를 예측하고, 포아손 확률분포 함수가 선택되었다. t>T인 미래의 어느 시간에 검출될 분기의 전체 수는 식 (1)과 같이 예측될 수 있다.

$$E\{X_t \mid \vec{x}\} = x_T + \sum_{j=T+1}^{t} \left(1 + \frac{r + x_T - n_T}{\gamma + G(t)}\, g(j)\right) gf(j) \qquad (1)$$

대부분의 연구에서는 모의실험의 매 클럭 사이클에서 새로운 테스트 패턴이 생성되어 입력으로 사용되고 있으나[12][13], [11]에서는 새로운 방법을 제안하고 있다. 즉 일정한 클럭 시간 동안 하나의 입력 패턴이 사용된다. 4개의 상태(phase state)를 설정하였으며, 각각의 상태는 1, 2, 4, 6 개의 클럭 사이클 동안 같은 입력패턴이 유지된다. 임의테스트 패턴을 사용할 경우 이 방법은 분기검출률을 증가시켰다.[14]

III. 제안하는 방법

식 (1)에서 보여준 것과 같이 X_t의 전체 예측값은 인터럽트의 예측 값의 합으로 주어진다. 여기서 X_t와 W_t는 모든 클럭에 대하여 임의변수로 가정된다. 식 (1)에서 x_T가 크다면 X_t의 예측 값도 크게 된다. 즉 시간 T 전에 모든 검출률의 합이 크면 클수록 정지점(stopping point)이 길어지게 된다. 그러므로 시간 축에서 정지점을 짧게 하기 위해서는 시간 t까지의 검출률의 합은 작아야 한다. 여기서 임의변수의 값들은 확실하게 알 수 없다. 즉 가능한 시간의 집합과 임의변수의 확률만을 알 수 있다. 이 논문에서는 새로운 세미임의변수를 정의한다. 이 변수는 값은 확실하게 알려지지 않지만, 변수의 모든 값이 어느 기준 값보다 큰 값을 갖는 변수로 정의된다.

표 1은 12개의 VHDL 모델에 대하여 시작점으로부터 10개의 모의실험 클럭 사이클 동안의 전체 분기검출률을 보여주고, 처음부터 몇 클럭 사이클 동안에 분기검출률이 매우 크다는 것을 알 수 있다.

예를 들어 S1의 경우에 분기검출이 6 이상이 되는 클럭 사이클은 t=1, 2인 경우에 분기검출 수는 각각 35, 18이며, S5의 경우에는 t가 5보다 작으면 모든 경우에 분기검출 수가 6보다 큰 것을 알 수 있다. 표 2에서 보는 것과 같이 의 값이 정확히 예측할 수는 없다 하더라도 시작점부터 처음 몇 사이클은 의 값이 비교적 크다는 것을 알 수 있다.

Table 1. Total branch coverage vs. CC

t \ S#	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12
1	35	35	58	47	47	69	58	47	35	58	49	93
2	53	92	222	94	94	153	111	94	92	111	92	166
3	54	95	123	102	102	156	123	102	95	123	127	218
4	58	111	123	133	133	178	123	133	111	123	129	228
5	68	140	126	145	145	210	126	145	140	126	132	231
6	68	143	139	146	146	214	139	146	143	139	136	238
7	68	148	142	146	146	216	142	146	148	142	139	239
8	68	151	170	146	146	222	170	146	151	170	139	239
9	68	151	172	154	154	222	172	154	151	172	144	244
10	68	152	182	159	159	222	182	159	152	182	148	249

그러므로 임의변수를 식 (2)와 같이 두 영역으로 구분할 수 있다. 즉 는 분기검출 수가 시작점부터 모두 어느 기준 값보다 큰 값만을 가지고 있는 영역의 변수로서 시작점부터 정지점 사이에 있는 임의변수이므로 세미임의변수로 정의된다. 그리고 는 기존의 임의변수와 같다.

$$W_t = W_{ts} + W_{tr} \quad (2)$$

시간 축의 앞 부분에 존재하는 W_{ts}는 매우 커서 W_t가 존재할 확률이 1이므로 포아손의 분포함수에 의하여 예측되지 않는다고 고려할 수 있다. 그러므로 stopping rule이 동작할 때 세미임의변수 영역에 존재하는 값들을 임의변수의 값들에서 제외시킬 수 있다. 이 세미임의변수의 영역이 매우 적다 하더라도 이 영역에서 검출되는 분기 검출 수는 거의 전체 검출수의 반 정도를 차지하므로 중요한 부분이 될 수 있다. 따라서 기준값이 어떻게 결정되느냐에 따라 정지점이 결정되므로 적절한 기준 값을 결정하는 것이 아주 중요하다.

그림 1에서 실선은 한 VHDL 프로그램에 대한 W_t의 확률을 나타낸 것이다.[11] 분기검출 수가 6보다 큰 경우에는 크다면 W_t의 확률은 매우 낮아지게 된다. 즉 세미임의변수 영역에서는 W_t가 6보다 큰 값을 가질 확률은 언제나 1이지만, 세미임의변수 영역을 벗어나면 그 확률은 아주 작아지게 된다. 따라서 분기검출 수가 처음으로 6 보다 작아지는 클럭 사이클을 세미임의변수와 임의변수를 분할하는 기준으로 선택한다. VHDL 모델의 종류와 테스트 패턴에 따라 분기검출 수가 달라지므로 기준점은 어느 특정 클럭으로 고정될 수 없다. 따라서 실험 과정에서 처음으로 수가 6보다 작은 분기검출 수가 나타나는 곳을 동적으로 결정하게 된다.

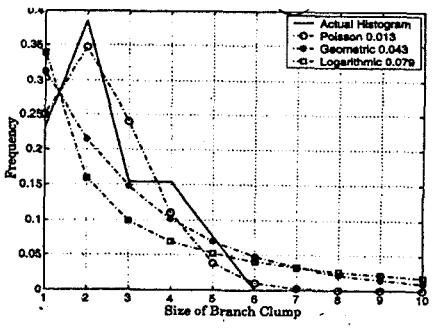

Fig. 1 Histogram fitting example of Wt for Sample VHDL code

[11]에서는 식 (1)에서 시간 T까지 전상태에서 얻어진 파라미터의 값들은 사용되지 않고, 새로운 상태가 시작될 때마다 모두 초기화되었다. 그러나 베이지안 모델은 앞에서 얻어진 분기검출률을 사용하여 미래의 분기검출률을 예측한다. 즉 시간 T까지 검증과정이 주어지면 t>T인 어떤 시간에 X_t의 전체 기대값은 시간 T 이후에 발생한 인터럽션의 모든 전체 크기의 합으로 주어진다. 그러므로 본 논문에서는 전상태에서 얻어진 데이터를 다음 상태의 초기값으로 사용한다.

IV. 모의실험 결과 및 논의

본 논문에서 제안한 방법들의 효율성을 확인하기 위하여 표 2에 있는 12개의 VHDL 프로그램을 사용하여 모의실험을 하였다

Table 2. Sample programs

Model	Description	# of code line	#of branch
S1	16 megabit byte-wide top boot 150ns	1880	373
S2	CMOS syncBIFIFO 256x36x2	4657	251
S3	SyncFIFO with bus-matching 1024x36	5015	302
S4	SyncFIFO 2048x36	4667	225
S5	SyncFIFO 2048x36	4710	225
S6	CMOS syncBIFIFO 1024x36x2	4949	296
S7	SyncFIFO with bus-matching 1024x36	4963	302
S8	SyncFIFO 2048x36	4777	225
S9	CMOS syncBIFIFO 512x36x2	4752	251
S10	SyncFIFO with bus-matching 512x36	4973	302
S11	SyncBIFIFO with bus-matching 512x36x2	5498	399
S12	SyncBIFIFO with bus-matching 1024x36x2	5770	470

표 2는 3 종류의 stopping rule에 대하여 Mentor QuickVHDL을 사용하여 각 클럭 사이클에 대한 분기검출에 대한 모의실험을 수행한 결과를 나타내고 있다. SB는 [11]에서 제안한 정적 Bayesian estimator를 사용한 결과이다. SBF는 본 논문에서 제안한 방법으로 SB와 같은 조건하에서 시작점부터 분기검출수가 6보다 큰 것은 제외하고, 식 (1)에서 상태가 바뀔 때마다 전에 얻어진 파라미터의 값을 사용한 것이다. SB30은 SBF에 연속된 30개의 테스트패턴에 대하여 분기검출이 0인 제한조건을 첨가한 것이다. SB와 비교하여 SBF는 24.6%의 클럭 사이클이 줄어든 반면에 0.6%의 분기검출이 줄어들었고, SB30은 59.1%의 클럭 사이클과 1.5%의 분기 검출이 줄어든 것을 보여준다. 대부분의 분기는 상태 1에서 검출되고 클럭 사이클의 수가 증가할수록 검출률은 떨어진다. 따라서 포화점 이후에 모의실험을 멈추게 하는 제한조건을 삽입하면 분기 검출수의 감소를 최소로 하면서 많은 클럭 사이클을 줄일 수 있다는 것을 알 수 있다.

Table 3. Simulation results of SB, SB30 and SBF

	SB		SB30		SBF	
	CC	BC	CC	BC	CC	BC
S1	1854	128	530	128	1460	128
S2	631	199	493	204	1461	206
S3	808	232	495	231	485	231
S4	673	192	485	192	482	192
S5	789	195	485	195	482	195
S6	2249	273	490	247	489	249
S7	809	232	495	231	496	231
S8	2181	208	649	203	1456	208
S9	631	199	493	204	1461	206
S10	808	232	495	231	496	231
S11	1982	245	596	245	1460	247
S12	2080	364	628	348	1462	360
Sum	***15,495***	***2,699***	***6,334***	***2,659***	***11,690***	***2,684***

V. 결 론

본 논문은 VHDL 프로그램의 검증 시간을 단축시키는 효율적인 방법을 제안하였다. 이 방법의 효율성을 증명하기 위하여 12 개의 예제 프로그램을 사용하여 모의실험을 하였으며, 실험 결과는 분기검출률이 조금 줄어들었지만 많은 클럭 사이클 수를 줄여 모의실험 시간을 단축할 수 있다는 것을 보여 주었다.

참고문헌

[1] Janick Bergerson, *Writing testbenches : functional verification of HDL model*, Kluwer Academic, 2000.

[2] Jjm Lipman, Covering your HDL chip-design bets, EDN, Oct. 22, 1998, pp 65-70.

[3] Brian Barrera, Code coverage analysis-essential to a safe design, Electronic Engineering, pp 41-43, Nov. 1998.

[4] Kevin Skahill, A designer's guide to VHDL design and verification, Engineering Software, pp 149-158, Feb. 1996.

[5] S. Surya, P. Bose, and J. Abraham, Architectural Performance Verification: PowerPC Processors, IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp344-347, Oct. 1994.

[6] J. Gately, Verifying a million gate processor, Integrated System Design, pp 19-24, 1997.

[7] M. Sahinoglu, A. Von Mayrhauser, A. Hajjar, T. Chen, C. Anderson, On the efficiency of a compound Poisson stopping rule for mixed strategy testing, IEEE Aerospace conference, Track 7, Mar. 1997.

[8] A. Goel, Software reliability models: assumptions, limitations, and application, IEEE transactions on software engineering, vSE-11,n 12, pp 1411-1423, Dec. 1985.

[9] W. Howden, Confidence-based reliability and statistical coverage estimation, Proceedings on the international symposium on software reliability engineering, pp 283-291, Nov. 1997.

[10] S. Chen, S. Mills, A binary markov process model for random testing, IEEE transactions on software engineering, v22, n3, pp 218-223, Mar. 1996.

[11] Amjad Fuad A. Hajjar, Bayesian Based Stoping Rules for Behavioral VHDL Verification, Ph. D Dissertation, Fall 2000.

[12] R. Ramchandani, D. Thomas, Behavioral test generation using mixed integer nonlinear programming, International test conference, pp 958-967, 1994.

[13] G. Hayek, C. Robach, From specification validation to hardware testing: a unified method, International Test conference, pp 885-893, 1996

[14] T. Chen, M. Sahinoglu, A. Mayrhauser, A. Hajjar, A. Anderson, Achieving the quality for behavioral models with minimum effort, 1st international symposium on quality in electronic design, pp, Mar. 2000.