Measures of Syntactic Complexity for Modeling Behavioral VHDL

Neal S. Stollon, Ph.D., P.E.
DSC Communications Corporation
Dallas, Texas

John D. Provence, Ph.D., P.E.
Southern Methodist University, Electrical Engineering Dept.
Dallas, Texas

Abstract– Complexity measures are potentially useful in developing modeling and re–use strategies and are recognized as being useful indictors of development cost and lifecycle metrics for systems design. In this paper, a syntactic measure complexity model for VHDL descriptions is investigated. The approach leverages similarities between VHDL models and software algorithms, where syntactic modeling has been previously applied. Aspects of the measure, including observed and estimated model length, volume, syntactic information, and abstraction level are defined and discussed. As a principle result, syntactic information modeling is related to Kolmogorov intrinsic complexity as a minimum design size implementation. Experimental data on VHDL modeling and complexity measurement is presented, with potential model comprehensibility and resource estimation applications.

1. Introduction

This paper addresses a recent investigation into applying complexity measures to HDL model characterization as a means of quantizing the model quality [3]. The approach is inspired by observations that VHDL, in common with many other hardware description languages, share structural and process similarities to software high order languages (HOLs). In addition to hardware modeling features such as boolean and sequential functions, VHDL includes abstract HOL operations such as recursive, alternative, and user defined functions. Comparisons between types of HDL and HOL characteristics hold true only for abstracted behavioral descriptions.

Commonalities between VHDL model and software algorithm development are often noted, but seldom exploited. Our approach to leveraging this commonality is by evaluating ways of applying complexity measures previously used only in software design to VHDL modeling. Common measure metrics provide a means of comparing, estimating, and developing insights into the VHDL modeling process. In this context, complexity analysis supports potential applications in managing and reducing risk in the increasingly complex task of VHDL model generation and support.

Complexity measures for VLSI systems design can be based on graphical or extracted statistical representations of a design model [3]. For HDL/VLSI design, generating such a abstracted design representation requires intimate knowledge of the data and control flow structure of a model or the probability distributions associated with the design functions.

In this paper, an alternative information measure of design is presented which has significantly reduced dependance on availability and extraction of a model's structural and statistical functionality. The measure is based purely on the syntactic information available in textural description of a design. VHDL is used as the description language for this discussion, but concepts and applicability of the approach appear extensible to other design languages. A syntactic information theory of complexity of a VHDL model can be derived as a fusion of intrinsic complexity theories of Kolmogorov [1] and the software science approaches of Halstead [6]. This is the first work, to our knowledge, that attempts to relate these two disparate efforts in areas of modeling complexity.

2. Intrinsic Complexity

The concept of intrinsic or descriptive complexity was originally developed by A. Kolmogorov in the mid–1960s. Kolmogorov defines an algorithmic or descriptive complexity of a given function to be the length of the shortest model that describes the function after a finite amount of computation. This complexity definition is attractive for modeling at a behavioral level of abstraction, since it is independent of the both the structure and the probability distributions of the function. As such, Kolmogorov complexity is a largely computer and language independent measure [1], which allows a neutral reference point for other complexity measures. We interpret Kolmogorov complexity K(u) in a modeling context as follows:
Definition: A halting function U is described as one that generates a output vector in response to stimulus and which then halts or resets at some point during model function after the generating operation has been completed.
Definition: The Kolmogorov complexity K(u), of a function u where $u \in U$ are a set of functions that can be described in a modeling language (V), is defined as:

$$K(u) = \min_{p:\, V(p)=u} l(p) \qquad (1)$$

where l(u) is the minimum length model that that generates a desired output u and then halts. For a function with multiple orthogonal outputs s.t. for $[u_1,...,u_n] \in U$.

$$K(U) \simeq \sum^{n} K(u) \qquad (2)$$

It is often difficult in practice to define the shortest model of an arbitrary function. Alternately, minimal (while not

32nd ACM/IEEE Design Automation Conference ®

necessarily minimum) model implementations which meet the Kolmogorov criteria can be approximated for many functions.

3. Length and Volume Complexity Measures

Software complexity and its measurement has been an area of ongoing investigation since the mid–1970s [5–8]. Studies have shown that the proper selection and application of a complexity metric can be applied to estimations of interest software design, including; design creation and debug time, software error rates, and maintenance costs. Our primary proposition is that an equivalent VHDL model metric could potentially be effective in resource allocation during design synthesis and analysis; with a resulting better understanding of the life cycle issues associated with model reuse and redesign.

The measures utilized in this paper are based on a set of software measures originally developed by M. Halstead [6]. Halstead's measures are based on use of syntactic information, and specifically the distribution of operand and operator representations (which in high order languages follow a family of Zipf probability distributions [2], to describe the relative frequency occurrence of sets of words in language) as a measure of properties of software programs and algorithms. Since the measures also provide a means of expressing a minimum possible algorithm size, they also appear to be appropriate for comparison with Kolmogorov complexity.

Halstead measure defines four observable model statistics that form the basis for measures applied to model estimates:

n_1 = number of distinct (unique) operators
n_2 = number of distinct (unique) operands
N_1 = total number of operators
N_2 = total number of operands

These in turn allow us to define the observed length of a given program string that implements the model as:

$$N = N_1 + N_2 \qquad (3)$$

and a language vocabulary defining the model construction as:

$$n = n_1 + n_2 \qquad (4)$$

By applying a statistical upper limit on the assumed distribution of operators and operands in distinct vocabulary strings such that:

1. operator and operand instances occur within the model with a statistical regularity and
2. identical operator/operand combinations are not repeated

a calculated vocabulary algorithm length of a model can be estimated [6] as a power series formulation of:

$$2^{N} = n_1^{n1} \times n_2^{n2}. \qquad (5)$$

Solving for the length , the equation (5) becomes

$$\mathcal{N} = n_1 \log_2(n_1) + n_2 \log_2(n_2). \qquad (6)$$

where $\mathcal{N}$ is a calculated length value of the string N.

The program volume defines the size of the language dependent implementation of an algorithm, as a function of both length and vocabulary such that:

$$V = N(\log(n)). \qquad (7)$$

Kolmogorov complexity of a model can be inferred through these volume statistics of a potentially minimum implementation. As a most succinct form representation, operators (n_1) and operands (n2), define a potential minimum volume implementation as follows:

a) The minimum operator count of a model, n_1^* defines a minimum implementation constant. For VHDL function and procedure syntax, we assign can $n_1^* = 2$, as an implementation where the only required operators are

1. a defined functional procedure of the model operation and
2. a unique assignment of variables to that operation.

b) The potential minimum operand count, n2*, for a minimal model implementation are the operands that appear at the model periphery. Peripheral signals are assumed to approximate operands which define conceptually unique and independent input/output parameters.

For a potential minimal model representation, each operator and operand is referenced only once; therefore $N_1^*=n_1^*$, $N_2^*=n_2^*$, where N_1^* and N_2^* are minimal equivalents of N_1 and N_2 as previously defined. The potential minimum volume of a model is therefore defined as :

$$V^* = (N_1^* + N_2^*)\log(n_1^* + n_2^*) = (2 + n_2^*)\log(2 + n_2^*) \qquad (8)$$

The minimum potential volume shares characteristics with the Kolmogorov measure. Both minimum implementations are essentially language independent;. since defining unique operations minimize the operator count n_1^*. This is generally unrealizable for real high level languages since extending a language to efficiently support all operations is not viable. Despite this limitation, the minimum volume of a model is useful as a point of reference in language level measures.

Evaluated purely from a model complexity viewpoint, Kolmogorov and Halstead both introduce conceptually similar measure of minimum complexity as a function of model length. This leads us to a central proposition of the minimum volume measure as a structured method for implementing an estimate for a Kolmogorov minimum length model.

Lemma: For an abstract HDL description of a function (u), the potential minimum volume $V^*(u) = (k+n_2^*)\log(k+n_2^*)$ where (k is constant and n_2^* is the minimum operand count) is related to the Kolmogorov minimum length complexity K(u) s.t.

$$V^*(u) \eqsim K(u). \qquad (9)$$

Proof: Let the minimum model syntactic length

$$N^* = N_1^* + N_2^* = l(u).$$

Let V*(u) be a non–minimum length model implementation . Then this assumes a model implementation with either fewer operators (n1) or operands (n2) that forms the function l(u). Since V* defines a minimum operator count $n_1^*=2$ for an operation and n2* are the minimum required operands for the function, then any $n_1<n_1^*$ would not fully define the function. Similarly, any $n_2<n_2^*$ would not fully implement the function. By the definition of minimum implementation, $N_1^*=n_1^*$ and $N_2^*=n_2^*$. Since V* is a function of (n_1^*, n_2^*) of a model (u), then $V^*(u) \eqsim l(u)$. As l(u) is composed of n_1^*, n_2^*, it's length is minimum. By (1), $K(u) = \min l(u)$. Therefore V*(u) must be a minimum length description K(u) of a function u. ▪

4. Level and Information Measures

Using the previous observations, several synthetically derived parameters of an algorithmic model implementation can be defined. Of interest to this discussion are the Program Level ($\mathcal{P}_{\mathcal{L}}$); Syntactic Information ($\mathcal{I}$); and Language Level ($\mathcal{L}_{\mathcal{L}}$) of a model.

Program Level: $\mathcal{P}_{\mathcal{L}} = V^*/V \approx (n_1^* n_2)/(n_1 N_2)$ (10)

$\mathcal{P}_{\mathcal{L}}$ provides a volume measure (as a function of verboseness or comprehensibility) of a model's language implementation referenced against its potential minimum volume. High $\mathcal{P}_{\mathcal{L}}$ implies greater modeling efficiency, which is normally associated with higher levels of modeling abstraction.

Syntactic Information : $\mathcal{I} = V (n_1^* n_2)/(n_1 N2)$ (11)

$\mathcal{I}$ measures the level of information content communicated by implementation of the model. Halstead refers to syntactic information as 'intelligence' and notes [6] that for irredundant model implementations, $\mathcal{I}$ is largely independent of algorithm abstraction levels or implementation. This is supported by $\mathcal{I}$'s equivalence to theoretic minimum volume V* as seen in (10) .

Language Level : $\mathcal{L}_{\mathcal{L}} = \mathcal{P}_{\mathcal{L}} V^* = \mathcal{P}_{\mathcal{L}}^2 V$ (12)

$\mathcal{L}_{\mathcal{L}}$ provides a measure of the power of a language to contain information (intelligence) in a minimum volume and bounds achievable levels of modeling abstraction.

Syntactic predictions and measurements provide a means of evaluating model complexity for a given implementation style. Syntactic measures have been verified in several studies as being a highly correlated measure of prediction for software lifecycle metrics in diverse applications [6,7,8]. While in some applications (synthesis as an example), models deviate from these measures, in general application, significant variances from the syntactic norm are traceable to implementation "impurities" (or inefficiencies) in a model such as operand redundancy, misapplied variables, unfactored logical expressions and other practices that increase model ambiguity. This are discussed further in section 7.

5. VHDL application to the complexity measure

VHDL is a verbose modeling language; using many types of language constructs which are largely information neutral. Such constructs insure compliance to language typing, but may not provide valued information otherwise.

From a purely syntactic viewpoint,we can consider a VHDL model to consist of three types of language constructs:

1. operators; which include all language syntax (functions and procedures) which manipulates model data. These include defined function operators as well as VHDL keywords. In VHDL, procedural functions may be defined in externally referenced standard packages. Since these packages define functions externally to the model, the function can be treated as extending the operator set of the VHDL language. Some common VHDL language operators are listed in TABLE 1a.

2. operands; which include all signals, variables, and constants that are functionally transformed in some way during model execution. For a HDL–centric interpretation of the minimum operand count (n2*) of a model, conceptually unique and independent signals are referenced in the port section of the VHDL entity declaration. Signals and variables internal to the VHDL model are considered derivative (not being conceptually unique and independent), since they are formed from operations on input signals.

3. structural constructs; which include information neutral language syntax which define the structural infrastructure of the VHDL design, independently of model size, abstraction, or complexity level. Many are referenced only once (typically as a marker for parsing operations) in the model. Since structural constructs are largely independent of the underlying model. we ignore them in generation of our syntactic complexity measure. By eliminating structural constructs, the remaining operators and operands in the model can be mapped to the complexity measures in a straightforward manner. A listing of typical VHDL structural constructs is given in TABLE 1b.

TABLE 1: (a) VHDL operators (b) structural constructs

abs	guarded	rem	alias	label
access	if	report	architecture	library
after	'left	return	array	of
all	linkage	'right	attribute	out
and	loop	select	bit	package
assert	map	'stable	bit_vector	port
begin..end	mod	then	block	procedure
body	nand	to	bus	process
buffer	new	transport	component	range
case	next	wait for	configuration	record
constant	nor	wait on	entity	register
disconnect	not	wait until	file	severity
downto	null	when	function	signal
else	open	while	generic	subtype
elsif	for	xor	in	type
exit	others		inout	units
generate	'quiet		integer	use
<= => /= := =	&		is	variable

6. Experiments on Behavioral Level Benchmarks

In order to assess the applicability of complexity measures, a VHDL analyzer was developed to extract the statistics from several classes of VHDL test cases. The Model Analysis and Syntactic COmplexity uTility (MASCOT) utilizes syntactic rules previously discussed to extract operator and operand information from models and calculate measures.

Models were used from VHDL behavioral benchmark repositories created at University of California– Irvine [9] and University of Cincinnati [10]. These models have been widely used for high level synthesis and represent a good cross section of modeling styles and applications.

To evaluate contrasting lower abstraction (register level) design, finite state machine (FSM) benchmarks [11] from MCNC were also evaluated. Models in state table format were translated into equivalent VHDL using model generation utilities [12]. Resulting VHDL models had the dual properties of a consistent modeling style irregardless of model size and of conformance to common VHDL synthesis guidelines.

Tables 2 and 3 provide experimental results for behavioral and register level test cases respectively.

Two areas of interest for complexity modeling were

(a) whether and what types of VHDL models fit the software statistical profile assumed for syntactic measurement and

(b) were information levels associated with each model consistent with reasonable assumptions.

Correspondingly two measure relationships were monitored:

– the equivalence between N (observed program length) and $\mathcal{N}$, (calculated program length) for $N/\mathcal{N} \rightarrow 1$, calculates model vocabulary distribution. s.t. convergence between N and $\mathcal{N}$ indicate that the models fit the measure profile assumed.

– equivalence between $\mathcal{I}$ (syntactic information) and V* (potential minimum volume) s.t. $\mathcal{I}/V^* \rightarrow 1$. calculates model information transfer effectiveness. Since V* has minimal redundancy (operators and operands are uniquely referenced), convergence of $\mathcal{I}$ and V* indicates that the modeling style used is largely irredundant.

Fig. 1 shows the normalized correlation between the $N/\mathcal{N}$ (x axis) and $\mathcal{I}/V^*$ (y axis) measures for the cases. Optimal correspondence between extracted values and those predicted by the measures converge in a (1:1) relationship along both axes. Data from MASCOT for the behavioral test cases is given in Table 2. Analysis of these models shows a general relation between predicted and observed values, indicating that behavioral models conform to measure analysis. Fig. 1 shows that most models fell within a 1 sigma ($\pm 27\%$) range of confidence (area bounded by circle) window. with a statistically significant compliance (89% confidence level in Chi–Square fit testing). This indicated that for behavioral level models, syntactic measure analysis is a viable approach.

Convergence between model measurements and expected syntactic values was significantly contrasted by the test results developed from the FSM VHDL models. The FSM VHDL models were created using VHDL logic synthesis register transfer level (RTL) modeling templates which had a weak $N/\mathcal{N}$ relationship. Sample data for these test cases is given in Table 3a. Interestingly, the $\mathcal{I}/V^*$ relationship for the FSM models remained well correlated throughout the range of 20+ test cases used. While the models were verbose ($N >> \mathcal{N}$), their $\mathcal{I}/V^*$ ratio ($\mathcal{I} \cong V^*$) indicated they were largely irredundant in information content. It was assumed that a FSM case statement based structure should not invalidate the measures, The lack of convergence was hypothesized to indicate that the models' inefficiency exceeded the compliance bounds of the measure.

Subsequent FSM model evaluation indicated excessively large N values appeared attributable to the modeling style used in model creation. As auto–generated models, the FSM VHDL had a consistent modeling style. This caused uniformly high $N/\mathcal{N}$ ratios, since modeling templates for auto–generated models, while syntactically correct, are often inefficient (verbose), when compared to behavioral models of similar information complexity (Table 2).

model	N	$\mathcal{N}$	V	V*	$\mathcal{P}_\ell$	$\mathcal{I}$	$\mathcal{L}_\ell$
AM2910	577	515	3736	64	0.029	108	3.15
A8251	176	1456	24390	140	0.007	181	1.34
ARMCNT	229	234	1292	19	0.029	37	1.07
BLCKJAC	146	186	787	—	0.032	25	.81
DECODER	321	463	2046	24	0.032	66	2.13
DIFFEQ	149	187	803	38	0.041	33	1.36
ELEVATOR	86	81	389	19	0.047	18	.87
ELLIP	252	276	1482	38	0.075	111	8.43
ENCODER	177	271	1023	19	0.043	44	1.95
FIND	174	221	971	8	0.027	26	.71
FM8501	979	734	6750	28	0.013	93	1.30
FRISC	563	518	3672		0.014	52	.74

Table 2. Complexity Data for VHDL Synthesis benchmarks

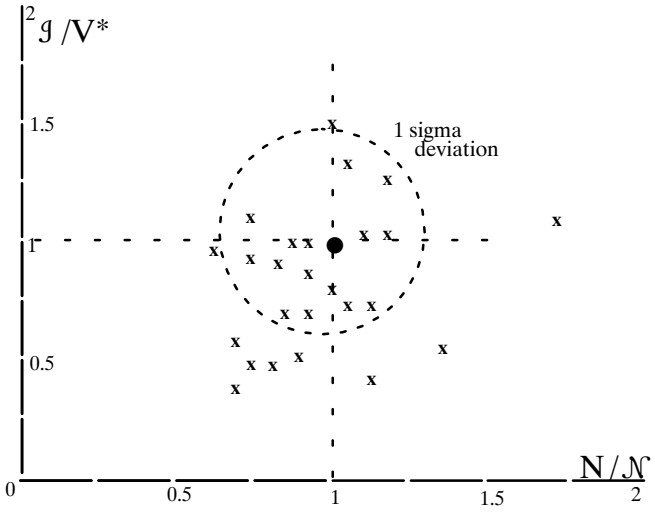

Fig. 1: Measures for Behavioral VHDL Cases

If model length inefficiencies could be corrected, then any resulting convergence between N and $\mathcal{N}$ would allow RTL abstraction (such as the FSM models) to be addressed by complexity analysis. Syntactic complexity measures of model length vs. information as a means of evaluating Incremental changes in VHDL modeling styles and related impact on syntactic complexity (model length, information) are investigated in the next section through a set of model "tuning" process experiments on the FSM models.

model	N	$\mathcal{N}$	V	V*	$\mathcal{P}_\ell$	$\mathcal{I}$	$\mathcal{L}_\ell$
BBARA	1200	135	6053	28	0.006	33	0.19
BBTAS	458	101	2177	19	0.010	22	0.23
CSE	2985	222	16580	69	0.003	56	0.19
KEYB	2873	209	15778	43	0.004	62	0.24
LION	181	86	829	15	0.022	17	0.38
S8	449	96	2110	24	0.011	23	0.27
SAND	7589	375	46357	104	0.002	100	0.22

Table 3a. Complexity Data for autogenerated VHDL models

model	N	$\mathcal{N}$	V	V*	$\mathcal{P}_\ell$	$\mathcal{I}$	$\mathcal{L}_\ell$
BBARA	689	140	3505	28	0.010	33	0.32
BBTAS	276	111	1340	19	0.014	19	0.28
CSE	1745	23	9797	69	0.005	53	0.29
KEYB	1496	219	8309	43	0.007	56	0.39
LION	177	101	841	15	0.027	22	0.62
S8	305	117	1496	24	0.018	26	0.47
SAND	3652	400	22605	104	0.004	91	0.37

Table 3b. Complexity Data for manually tuned VHDL models

7. Complexity Based Tuning of Models

"Syntactic tuning" defines a method of quantitatively reducing length and impurities in a model. Tuning is used in this context as a process of implementing model language modifications in structure, logic, or syntax changes to a model, which do not modify the model functionality.

For a hypothesis that significant inefficiencies in the FSM models resulted in excessive length relative to measure estimates, then syntactic tuning could provide a means of decreasing model size without adversely affecting other model parameters. Based on the VHDL model auto–generation templates, four tuning steps were identified as potential corrections to most identified inefficiencies. Tuning was manually applied to several FSM test cases (Fig. 3b). The impurity tuning steps consisted of:

1. reordering terms to combine into common CASE and IF..THEN statements.
2. minimization of AND/OR combinations of the function.
3. application of XOR operations to the logical function.
4. combining of redundant output terms and repositioning them outside of the sequential operations.

A pre and post–tuning VHDL fragment of a typical auto–generated FSM model is given in Listing 1(a) and (b). For this example, tuning reduced the model length by approximately 2x without changing functionality. Tuning decreased observed length (N) and increased calculated length ($\mathcal{N}$) converged estimated and measured values. Changes in syntactic information ($\mathcal{I}$) were minimal (<7% with respect to other parameters). Model conciseness after tuning improved by a 33% increase in language level.

```
CASE CURRENT_STATE is    – – baseline model of FSM
 WHEN st0 =>                    – – as auto–generated by tool
   if ( in1 and (not in2) ) ='1'
   then a <='0'; b <='1'; c <='0'; NEXT_STATE <= st1; end if;
   if ( (not in1) and in2 ) ='1'
   then a <='0'; b <='1'; c <='0'; NEXT_STATE <= st1; end if;
   if ( in1 and in3 ) ='1'
   then a <='0'; b <='0'; c <='1'; NEXT_STATE <= st0; end if;
   if ( in1 and in2 ) ='1'
   then a <='0'; b <='0'; c <='1'; NEXT_STATE <= st0; end if;
 WHEN st1 =>          – –. similar logic duplicated for s1 case
```

Listing 1(a): Fragment VHDL model of FSM (with impurities)
statistics: N= 182, $\mathcal{N}$ = 81, $\mathcal{I}$ = 17, V*=10, $\mathcal{L}_{\mathcal{L}}$ = .36

```
CASE CURRENT_STATE is    – – VHDL model after tuning
  WHEN st0 => a <='0';   – – to minimize impurities
   if ( in1 xor in2 ) ='1'
   then b <='1'; c <='0'; NEXT_STATE <= st1;    end if;
   if ( in1 and ( in2 or in3 ) ) ='1'
   then b <='0'; c <='1'; NEXT_STATE <= st0;    end if;
 WHEN st1 =>          – – similar logic duplicated for this case
```

Listing 1(b) : Fragment of VHDL model of FSM (after tuning)
statistics: N= 99, $\mathcal{N}$ = 86, $\mathcal{I}$= 16, V*=10, $\mathcal{L}_{\mathcal{L}}$ = .54

Table 3b presents complexity statistics for several FSM VHDL models re–analyzed after manual tuning. Fig. 2 shows the reductions in normalized length for different tuning steps and related changes in the information measures. Model size improved on average 2–3x using this informal procedure. While models remained verbose, they now begin to approximate N/$\mathcal{N}$ ratios achieved for the behavioral test cases. More formal and extensive application of a tuning process would be expected to improve the ratio still further.

Tuning operations implemented changes largely consistent with formal language modeling comprehensibility guidelines. Realistic reductions in model length from tuning appear to be limited in practice by considerations such as model synthesis (which limits language options in a model implementation upper bounding the language level measure $\mathcal{L}_{\mathcal{L}}$).

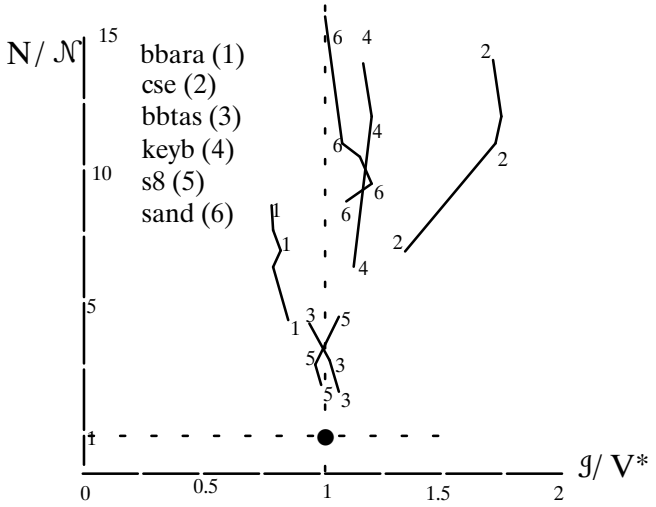

Fig. 2: Tuning Improvements in FSM VHDL models

8. VHDL behavior and structural complexity

In software applications, complexity measures in general, and variations of the syntactic measures presented have been used to model life cycle and process parameters such as sw construction, debug, and test times and program error rates. [7]

Since complexity measures can be used to analyze VHDL models in a similar mode as for software processes, it may be reasonable to propose complexity correlations in the VHDL life cycle and process to those for software development. Inadequate comprehensive VHDL documentation is available however to test this hypothesis.

For applying complexity measures to the design process, VHDL does offer a correlation opportunity not available for software applications. Given previous assumptions about minimum size in Kolmogorov models as being abstraction independent in the logical hardware description domain, then the physical realization, (as a VLSI hardware implementation) should have a corresponding minimum representation. In this context, comparison of pre–synthesis behavioral complexity measures to post–synthesis size measures (gates or chip area) is potentially appropriate. Correlation between VHDL abstract complexity measures and structural resources for hardware design can establish complexity measures as estimation tools in the design process. Such estimation could have potential use in the systems design tasks, including behavioral partitioning tradeoffs and higher abstraction resource allocation.

In logic synthesis operations, optimization is largely independent of syntactic features, which are typically stripped away during the synthesis process. The syntactic information measure $\mathcal{I}$, which has been previously discussed, is largely independent of a model's redundancy or impurity. Since $\mathcal{I}$ and V* converge, we can hypothesize extending our Lemma on

equivalence between intrinsic complexity and minimum volume to include $\mathcal{I}$ as a minimum complexity measure; s.t.

$$K(u) \approx V^*(u) \approx \mathcal{I}I(u) \quad (13)$$

A comparison of the relationship between syntactic information as a model complexity measure and literal count (a technology independent size measure used in synthesis evaluation) for 20 MCNC benchmarks [4] is presented in Fig. 3. The systemic correlation between complexity (as measured by information content) and structural realization size (as measured by literal count) is an experimental confirmation of syntactical complexity as a basis for logical hardware measure.

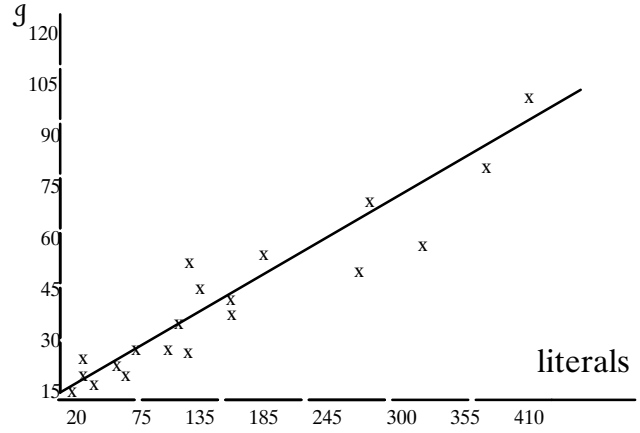

Fig. 3 : Literal Count vs. Model Syntactic Information ($\mathcal{I}$)

9. Conclusions

While the problem of managing the complexity of HDL based designs is widely recognized, the concept of applying software complexity measures to hardware design has received limited attention. In this paper, syntactic complexity measures are presented as a basis for VHDL model evaluation. This approach allows HDL models to be compared against other model abstractions [3]. In particular, model volume, length, information, and language level measures provide a means of comparing efficiency in model implementation alternatives. While syntactic measures of VHDL complexity are not complete model characterizations, they potentially provide a simplified means of capturing and representing information at initial modeling phases in the design process.

This paper overviews an approach and experimental data for VHDL model compliance to syntactic complexity measures. Establishing the relation between complexity and modeling issues can potentially mitigate resource related design process risks by giving estimates of model parameters early in the design flow. As an example, syntactic information is shown both experimentally and in theory to relate to literal counts of a design. Applications of this type indicate that potential application of complexity as a measure can transcend abstract VHDL modeling to provide predictors of parameters typically associated with lower levels of design abstraction.

The concept of complexity measures of a VHDL model can be approached from several aspects. Syntactic complexity's relation to formal information measures such as Kolmogorov intrinsic complexity indicates that, for efficient behavioral modeling styles, VHDL appears to comply with statistical complexity bounds. Syntactic measures can be applied in the design process though the tuning of a VHDL model for improved volume and complexity. This provides a basis for the somewhat ambiguous task of addressing and enhancing model comprehensibility. It is known through language studies that increased comprehension is a function of reduced model length and redundancy in a language model. Comprehension in the VHDL modeling task becomes increasingly relevant in context of graphical and model synthesis ESDA systems which auto–generate models.

A relation between complexity to comprehensibility is increasingly important, since hardware descriptions such as VHDL tend to discount modeling tradeoffs that address human comprehension considerations in favor of machine readability. This facet of design has not been addressed in large part due to lack of applicable quantitative measures. Syntactic measures, can potentially fulfill this need. Language abstraction and information/volume measures in particular address both characterization and comprehension issues of VHDL models.

[1] A. Kolmogorov, "Three Approaches to the Problem of Defining the Concept of "Quality of Information" *Problems in Information Transmission*, v.5 pp.1–7, 1969

[2] M. Shooman and A. Laemmel, "Statistical Theory of Computer Programs" *Proceedings of IEEE Computer Conference*, pp.511–517, Oct. 1977

[3] N. Stollon. *Information Content and Measurements for VLSI Functions.* Ph.D thesis, Electrical Engineering Dept., Southern Methodist University, Nov. 1994.

[4] S. Devadas, "Mustang: State Assignment of Finite State Machines Targeting Multi–Level Logic Implementation" *IEEE Transactions CAD*, pp.1290–1300, Dec. 1988

[5] E. Weyuker, "Evaluating Software Complexity Measures" I*EEE Transactions Software Eng.*, pp.1357–1365, Sept. 1988

[6] M. Halstead, *Elements of Software Science*. Elsevier North Holland, 1977

[7] J. Davis, R. LeBlanc, "A Study of the Applicability of Complexity Measures" *IEEE Transactions Software Eng*., pp. 1366–1371, Sept. 1988

[8] L. Baker, S. Zweben, "A Comparison of Measures of Control Flow Complexity" *IEEE Transactions Software Eng.*, pp.506–512, Nov. 1980

[9] N. Dutt, C. Ramachandran, "Benchmarks for the 1992 High Level Synthesis Workshop" University of California–Irvine Technical Report 92–107. 1992

[10] R. Vemuri, J. Roy, P. Mantor, N. Kumar, "Benchmarks for High Level Synthesis" University of Cincinnati Technical Memo ECE–DDE–91–11. 1991

[11] R. Lisanke, "Finite State Machine Benchmark Set"; Microelectronics Center of North Carolina (MCNC), Sept. 1987

[12] N. Stollon, "Automated VHDL Model Generation for Programmable Logic" *Proceedings of VHDL International Spring 1992 Conference*, pp.245–252, May 1992