A Novel Approach for Digital Waveform Compression*

Edwin Naroska
Computer Engineering Institute
University of Dortmund, Germany
edwin@ds.e-technik.uni-dortmund.de

Shanq-Jang Ruan
Department of Electrical Engineering
National Taiwan University, Taiwan
stj@orchid.ee.ntu.edu.tw

Chia-Lin Ho
Department of Computer Science
and Information Engineering
National Taiwan University, Taiwan
hcl@orchid.ee.ntu.edu.tw

Said Mchaalia
Computer Engineering Institute
University of Dortmund, Germany
said@ds.e-technik.uni-dortmund.de

Feipei Lai
Department of Computer Science
and Information Engineering
and Department of Electrical Engineering
National Taiwan University, Taiwan
flai@cc.ee.ntu.edu.tw

Uwe Schwiegelshohn
Computer Engineering Institute
University of Dortmund, Germany
uwe.schwiegelshohn@udo.edu

The waveform data, which is gathered during digital simulation of large and complex systems easily fill up huge amounts of disk space on the simulation computer. This is specially true for many deep-submicron designs. Hence, we developed a set of algorithms and techniques which can be used to efficiently compress digital waveforms without losing any significant information from the original data. Experimental results show that compression ratios of up to 300 compared to a original VCD formatted file can be achieved.

I. Introduction

Verification has become one of the most important and time consuming part in the design cycle of digital systems. The most common functional verification method is simulation. During simulation, a set of test vectors are applied to the circuit and huge waveform files are generated. Hence, fast compression techniques to reduce the file size of the waveform database without sacrificing simulation speed are indispensable.

Despite waveform compression being a challenging problem, only few work is known to the authors. In [1], the authors remove those signals from the watch list that can be easily rebuilt. However, this approach is mainly dedicated to gate level models. In contrast to [1], our techniques can be applied on each kind of model. Moreover, our approach can be applied on top of the technique from [1] to further improve compression ratio. There are commercial tools that support dumping compressed waveform files [2]. However, their algorithms are not disclosed. Alternatively, common compression algorithms may be used to decrease the file size of a waveform database [3, 8, 4]. Although these algorithms are well known and may be optimized for waveform compression, they usually consume a significant amount of computation power and do not provide sufficient compression ratios.

Our approach exploits typical waveform properties to achieve good compression ratio with less computation power. First, we separate the waveform stream into two sub-streams *signal-id stream* and *value stream*. The *signal-id stream* stores for each (simulated) time instance the signals that are having a transition. *Value stream* contains the actual signal values after a corresponding transition. To compress the *signal-id stream*, we make use of a caching like technique that identifies similar signal activity behavior in an efficient manner. For *value stream* compression, we apply a set of techniques, which predict a new signal value based on previously recored history information. Afterwards, the compressed streams are encoded and merged to form a combined waveform stream again. For further size reduction, the combined output streams may be additionally processed using common compression programs.

*This work has been sponsored by Novas Software, Inc. Patent pending.

II. VCD file format

VCD is a widely used ASCII based file format developed by Cadence to store signal waveforms [9]. In a VCD file transitions are grouped into (so called) *transition blocks*. Each transition block starts with a new (simulated) time value followed by a list of signal/value pairs defining the transitions at that specific time instance.

Definition 1 *Let $S = \{s_0, s_1, \cdots, s_{n-1}\}$ be the set of signals. Then, a transition is a tuple $tr = (id, v)$. id is an integer number which uniquely identifies the signal s the transition belongs to. v is the value of signal s after the transition took place. Function $id(tr)$ returns the signal identifier of tr.*

Definition 2 *A transition block is a tuple $TRB^{(c)} = (t, T)$, where $t = time(TRB^{(c)})$ is the transition time, $T = T(TRB^{(c)}) = \{tr_0, tr_1, \ldots\}$ is a set of transitions. c is a unique integer number denoted as simulation cycle. It is obtained by enumerating the transition blocks. A signal s must not be associated with more than a single transition of T, i.e. $\forall tr_a, tr_b \in T, a \neq b : id(tr_a) \neq id(tr_b)$.*

Definition 3 *Let $TRB^{(c)}$ be a transition block at simulation cycle c. Then, the transition signal block $S^{(c)}$ is the set of signal identifiers that are associated with a transition in $T(TRB^{(c)})$. I.e., $S^{(c)} = \{id(tr) : tr \in T(TRB^{(c)})\}$.*

III. Signal ID compression

Experiments on RTL level designs as well as gate (timing) level designs showed that in many cases for a particular transition signal block $S^{(c)}$ an *identical* block $S^{(j)}$ with $j < c$ can be found. Note that this only means that the *same signals* are having a transition at both cycles while the new *values* of the corresponding signals will usually differ. Moreover, it turned out that a small group of the signal sets occur frequently while some other appear only once. To exploit this property, we do signal ID compression using a so called *transition block cache* TBC. This cache stores all different kind of transition signal blocks that are detected during compression (i.e., there are no identical blocks in the cache). In order to refer to a cache entry, each entry is associated with a unique *cache entry identifier*[1].

As a result, a transition block at cycle i is processed as follows: First, $S^{(i)}$ is determined and looked up in the transition block cache TBC. If $S^{(i)}$ is already in the cache (namely, cache hit) then the unique cache entry identifier associated with $S^{(i)}$ is determined. Otherwise (namely, cache miss), a new entry $S^{(i)}$ is added to TBC and associates with a new unique identifier. Next, the cache entry identifier is stored to the output stream. Further, In case of a cache miss, the signal set is also written to the output stream (see Section A).

We applied this technique on 5 RTL level models and 3 gate (timing) level models. The two worst hit ratios (= how many of the transition blocks could be matched against an appropriate block in the *transition block cache*) were 84% and 89%, while all other ratios were above 94%.

A. Signal set encoding

In case of a TBC miss, the transition signal block $S^{(i)}$ must be encoded and written to the output stream along with the associated cache entry identifier. As the identifier occupies only a couple of bytes it is not further encoded. The signal identifiers of $S^{(i)}$ are dumped in increasing identifier number order. However, only the first identifier is directly saved while for the following identifier the increment to the corresponding preceding identifier are dumped. To save space, we applied the rather simple and fast *variable length encoding* technique from [10] instead of using complex algorithms like Huffman encoding [5], arithmetic encoding [6] or dictionary based (Lempel-Ziv) techniques [3, 4]. *Variable length encoding* translates a 32 bit integer value into a sequence of 1 to 5 bytes. Each output byte contains a payload of 7 bits and a stop bit that determines whether an additional byte is needed to encode the number. A major advantage of this approach is its low runtime overhead. Further, because the output stream remains byte oriented, dictionary based techniques can be subsequently applied.

[1] E.g., the simulation cycle at which the corresponding entry was added to the cache may be used as cache entry identifier

IV. Value compression

Similar to signal set encoding, the signal values are encoded for each transition block separately. First, the bits defined in the VCD file must be mapped to binary values. The VCD format defines bit values 0, 1, x and z which are mapped to binary bit patterns 00, 01, 10 and 11.

Dumping the signal values of a transition block $TRB^{(c)}$ is done as follows: The transition set $T(TRB^{(c)})$ is sorted in increasing id-value order of the transitions. Next, all signal values of the transition set are written to a temporary stream according to the sorted order. To save space, the value bits are converted to corresponding binary bits and seamlessly packed together. Finally, the temporary stream is split up into sections of 32 bits which are *variable length encoded* (see Section III) and afterwards written to the output.

Efficient *variable length encoding* requires that the upper bits of the numbers to be encoded are mostly 0. However, this pre-condition is often not matched by the packed value stream. This issue is addressed in the following.

A. Shuffling

Usually, the signal values are 0 or 1 during simulation most of the time while values x or z seldom appear. As a result, the odd bits in the packed value stream are usually 0. To exploit this property, we split the stream into words (sections) of 32 bits and re-arrange the bits within each word so that all even bits go into the lower 16 bit part and the odd bits are moved to the upper part of the word. Hence, the upper part of the shuffled words will be 0 most of the time enabling efficient *variable length encoding* compression.

We also extended this approach to handle 64 bit wide words by grouping the value stream into sections of 64 bits, separating even and odd bits within each 64 bit word and applying 64 bit wise *variable length encoding*.

B. Value prediction

Value prediction has been successfully applied on image compression [7] and is also used in processor research [11]. Our approach is similar as it tries to determine signal values based on previously observed transitions. Instead of storing the actual signal values, the difference (xor) between predicted values and the actual signal values of a transition block are dumped. Thus, the number of 0 bits in the dumped stream are increased, which in turn can be exploited by further encoding techniques (e.g., by *variable length encoding*).

B.1 Signal history based prediction

Often, only a few bits of a vector signal change at a time. Hence, instead of storing the new value the difference (xor) of the current and the previous value may be dumped. This

of course requires that for each signal its previous value is recorded in a so called *history buffer*.

Although this feature can reduce the number of bits set in the value stream, there are some downsides: It does not work well for single (VCD) bit signals transitions because for each such transition at least one of the corresponding binary bits changes. Further, a significant amount of memory is occupied by the history buffer. Finally, storing the current values into the history buffer pollutes the data cache of the processor due to the random like access pattern.

B.2 Extended signal history based prediction

Fortunately, based on the previous signal value the next value of a single (VCD) bit signal can be usually predicted very easily. Usually, x and z seldom appear on any signals. Hence, if the previous value of a (single bit) signal is 0 then the next value will most probably become 1 and vice versa. For previous values x or z we arbitrarily chose 0 as prediction.

For vector signals we selected a simple schema which assumes that the least significant bit of a vector is flipping while all other bits remain stable. However, if a previous vector bit is x or z then we predict the next value to be 0.

B.3 Signal set history based prediction

Although extended signal history based prediction is capable of predicting signal values more accurately than the approach described in Section B.1, it also suffers from the additional memory overhead and data cache pollution. To overcome this drawback, we developed an approach which stores history data into the transition block cache TBC introduced in Section III.

Definition 4 *A transition data block is a triple* $TRDB^{(c)} = (t, T, V)$*, where* c*,* t *and* T *are defined as in Definition 2 and* V *is a packed data array section consisting of all signal values listed in* T*.*

The modified transition block cache TBC' consists of $TRDB^{(c)}$ entries. The signals values of a transition block at cycle i are now processed as follows: The signal value stream is xor-ed with the data section stored in the corresponding *transition data block* and the number of bits set to 1 in the result are counted (population count). If the population count is above a specific threshold then the original value section is *variable length encoded* and written to the value output stream. The value data is prepended by a special token to indicate that original values are written. Otherwise, another token is written to the output stream followed by the *variable length encoded* xor result. Finally, the value data is written back to the corresponding cache entry of the TBC'.

Contrary to the approach from Section B.2, there may be more than one "previous" signal value stored for a signal (stored in different TBC' entries). However, signal values are stored in packed array format reducing memory consumption significantly. Further, processor data cache read/write operations with respect to the signal values are more regular because the TBC' value arrays are always read and written sequentially.

B.4 Hybrid history based prediction

The *hybrid history based prediction* approach uses a *history table* to hold the previous value for each signal as well as corresponding value arrays for each TBC' cache entry. However, the value arrays in the TBC' cache are now used to hold the difference between the *predicted* and the *real* values for a corresponding transition block.

In detail, signals values of a transition block at cycle i are processed as follows:

- The previous values obtained from the history table are used to determine a set of *predicted values* for the current cycle. The predicted values are packed together to build a *predicted value stream*, which is xor-ed with the real signal values to form a *predicted value difference stream*.
- The *predicted value difference stream* is xor-ed with the value data section of the corresponding TBC' cache entry. Depending on which population count is lower, either this result or the *predicted value difference stream* is dumped to the output stream. In each case the data is prepended with a special token to indicate which of the two data sets were written.
- The *predicted value difference stream* is stored back to the corresponding TBC' cache entry and the current signal values are stored back into the history table.

V. Experimental results

We implemented 10 compression programs that take a VCD file and output a compressed file. The programs are named: plain, sh, lsh, lsh_hist, lsh_hist_pred, xor, xor_sh, xor_lsh, xor_lsh_hist and xor_lsh_hist_pred. All versions applied signal id compression (Section III). Program plain did not use any other additional technique presented in this paper while the features of the other programs are encoded in their names: "sh" = 32 bit shuffling (Section A), "lsh" = 64 bit shuffling (Section A), "hist" = signal based history (Section B.1), "pred" = value prediction (Section B.2) and "xor" = signal set based history (Section B.3). Note that xor_lsh_hist and xor_lsh_hist_pred are hybrid history based prediction approaches (Section B.4): xor_lsh_hist uses signal prediction based on Section B.1 while xor_lsh_hist_pred uses the approach from Section B.2.

In order to test the compression programs, we applied them on a set of VCD files. Table I shows the compression ratio with respect to the original VCD file size. In order to enhance size reduction, we also applied common compression programs UNIX *compress* (column "compr.") as well as *bzip2* with option "-9" (highest compression) on the output files. The overall compression results are shown in the corresponding columns. Obviously, using xor_lsh_hist_pred gives best compression results for RTL models and lsh_hist_pred is best suited to gate level models.

TABLE I
COMPRESSION RESULTS FOR VARIOUS MODELS AND PROGRAM CONFIGURATIONS

program	model											
	rtl (RTL level model)			sxp2 (RTL level model)			gate (gate level model)			sota (gate level model)		
	no LZ	compr.	bzip2	no LZ	compr.	bzip2	no LZ	compr.	bzip2	no LZ	compr.	bzip2
plain	7.1	14.8	84.4	4.1	18.6	56.1	12.4	20.2	46.9	7.0	19.8	67.1
sh	11.3	21.0	145.2	5.2	20.9	55.1	17.5	21.9	60.6	7.1	19.8	67.3
lsh	12.0	21.8	157.0	5.6	22.1	55.7	18.6	21.5	62.1	7.1	19.8	66.7
lsh_hist	11.7	20.6	150.4	4.9	23.1	81.1	18.1	58.7	106.2	6.8	19.5	67.7
lsh_hist_pred	12.0	22.0	154.5	6.1	28.0	86.9	34.5	69.3	116.6	7.6	23.0	83.3
xor	19.1	106.3	308.2	7.3	51.1	193.3	25.3	54.7	97.5	7.1	18.0	60.8
xor_sh	20.6	113.6	315.6	7.8	54.3	189.3	26.0	54.4	99.2	7.1	18.0	60.7
xor_lsh	29.8	114.2	311.3	11.1	61.4	197.3	30.9	55.0	99.0	7.2	18.2	60.9
xor_lsh_hist	29.8	113.8	311.7	11.3	64.1	207.9	30.9	59.2	111.2	7.1	18.1	61.6
xor_lsh_hist_pred	29.9	115.6	317.9	11.7	68.1	215.4	31.9	60.2	112.6	7.2	19.8	71.6

TABLE II
COMPRESSION RATIO (RELATED TO ORIGINAL FILE SIZE) AND SPEEDUP (COMPARED TO RUNTIME OF "BZIP2")

model	bzip2		compress		xor_lsh_hist_pred		xor_lsh_hist_pred + bzip2		xor_lsh_hist_pred + compress	
	ratio	speedup	ratio	speedup	ratio	speedup	ratio	speedup	ratio	speedup
rtl	5.0	1	3.5	5.4	29.9	12.6	317.9	9.5	115.6	11.9
sxp2	27.1	1	6.3	5.9	11.7	9.0	215.4	5.8	68.1	8.3
leon	22.3	1	6.8	5.3	9.5	10.7	26.6	4.6	18.2	8.7
gate	2.3	1	1.8	5.0	31.9	9.9	112.6	2.9	60.2	9.5
sota	4.0	1	2.2	4.9	7.2	5.7	71.6	4.5	19.9	4.9

We also compared the performance of our algorithm xor_lsh_hist_pred with UNIX programs "compress" and "bzip2". Table II shows the compression ratio as well as the speedup compared to the runtime of "bzip2". Note that the last two columns show the results obtained from applying "compress" and "bzip2" on the output of xor_lsh_hist_pred. The table clearly shows the excellent performance of our algorithms.

VI. CONCLUSION

In this paper, we developed a set of algorithms to compress digital waveforms. In order to efficiently process the data, we separate the waveform stream into signal ID stream and value stream and apply appropriate algorithms to both stream types. For the signal IDs, we developed a technique which is based on identifying identical signal sets. The value stream was compressed by predicting signal values based on previously gathered transition information.

Our experimental results show that compression ratios of up to 300 can be achieved. Compared to using "bzip2" only, our approach combined with "bzip2" achieves better compression ratios (up to 63 times) while consuming less runtime (up to 9 times faster).

REFERENCES

[1] J. Marantz, "Enhanced Visibility and Performance in Functional Verification by Reconstruction", in Proceedings DAC-98, June 1998

[2] Novas Software Inc., "Debussy: Total Debug System", http://www.novas.com.tw/products/index.html, 2002/04/11

[3] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Transaction on Information Theory, Vol. IT-23, No. 3, May 1977

[4] Dzung T. Hoang, Philip M. Long and Jeffrey Scott Vitter, "Dictionary Selection Using Partial Matching", Information Sciences, Vol. 119, No. 1–2, 57–72, 1999

[5] J.S. Vitter, "Design and Analysis of Dynamic Huffman Codes", Journal of ACM, 34(4):825–845, October 1987

[6] J.J. Rissanen and G.G. Langdon, "Arithmetic Coding", IBM Journal of Research and Development, 23(2): 149–162, March 1979

[7] X. Wu and N.D. Memon, "CALIC – A Context Based Adaptive Lossless Image Coding Scheme", IEEE Transaction on Communications, 45:437–444, May 1996

[8] G. Mandyam, N. Ahmed and N. Magotra, "A DCT-Based Scheme for Lossless Image Compression", IS&T/SPIE Electronic Imaging Conference, February 1995

[9] Cadence Design Systems Inc., "Verilog-XL Reference Manual", Version 2.3, 1995

[10] Wireless Application Protocol Forum Ltd., "Wireless Application Protocol, Wireless Session Protocol Specification", WAP WSP Standard, Version 5-November-1999

[11] Bohuslav Rychlik, John Faistl, Bryon Krug and John P. Shen, "Efficancy and Performance Impact of Value Prediction", in Proceedings PACT-98, October 1998

[12] Marius Evers, Po-Yung Chang and Yale N. Patt, "Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches", in Proceedings ISCA, 1996