QBF-Based Synthesis of Optimal Word-Splitting in Approximate Multi-Level Storage Cells

Daniel E. Holcomb
University of Michigan
danholcomb@umich.edu

Kevin Fu
University of Michigan
kevinfu@umich.edu

Abstract

In applications such as multimedia that tolerate imprecise results, approximate computing techniques can sacrifice precision to save power or time. One aspect of approximate computing is imprecise storage in multi-level cells (MLCs). Computer words that are too large for a single MLC must be distributed across multiple approximate MLCs. The word-level imprecision depends on how the words are split across the MLCs. This work gives an automated synthesis approach for splitting words across MLCs. Given bounds on the imprecision of individual MLCs, the technique synthesizes solutions for splitting words across cells to minimize the worst-case imprecision at the word level. The technique is based on quantified Boolean formula solving, within an overall optimization loop. Worst-case word-level error is shown to vary by over an order of magnitude across otherwise comparable word-splitting alternatives.

Keywords Approximate Computing, Multi-Level Cells, QBF Solving, Synthesis

1. Introduction

Approximate computing techniques trade away precision to save power, energy, or time. The tradeoff is appealing in applications such as multimedia that are inherently tolerant of imprecision. Recent work demonstrates that approximate storage of data in non-volatile multi-level cells (MLCs) can speed up write operations by 1.7x in exchange for a quality loss of under 10% [10]. The data words on which computers operate are too large for storing in a single MLC, and must therefore be distributed across multiple imprecise MLCs. The loss of precision at the word level depends on how the words are distributed across the approximate MLCs.

This work presents a synthesis approach for optimally storing data words across multiple approximate MLCs. The inputs to the synthesis are bounds on the worst-case error of the MLCs, and the output is a way to distribute data words across the MLCs such that the worst-case word-level error is minimized. The approach frames the synthesis problem as a quantified Boolean formula (QBF), within an overall optimization loop.

The specific contributions of this paper are as follows:

- We demonstrate that a previously proposed word-splitting has a word-level error that is good in the average case, but pathologically bad in the worst case.
- We present an automated QBF-based technique for formal synthesis of an optimal word-splitting solution, and validate the solution using random simulation.
- We use the synthesis technique to find a counter-intuitive approximate storage solution where optimal word-level precision is achieved by deliberately introducing small errors to written data.

2. Related Work

As technology scaling nears physical limits, maintaining a perfect error-free digital abstraction becomes ever more expensive in power, time, and silicon area. Workloads such as multimedia do not require perfect computation, and this has given rise to techniques for approximate computation, where an acceptable amount of precision is traded away to save costs. EnerJ is a Java extension that adds approximate data types, and its use can cut energy by up to 50% versus fully precise computation [9]. Microarchitectural support for approximate computing is provided in Truffle, where power is saved by processing approximate and precise data at different supply voltages [1].

In addition to imprecise computation, imprecise data storage can also save cost. Increasing DRAM refresh interval beyond point of first failure saves 20-25% of memory power in error-tolerant applications [4]. The use of error-inducing low voltages in flash memory is shown to decrease write power by 34% [8]. Similar techniques can be applied to multi-level cells (MLCs) in non-volatile memories. MLCs encode multiple bits of information per cell by allowing a wider variety of stored levels, instead of just the traditional "0" and "1" levels. For example, X4 memory stores 4 bits per cell by storing one of 16 discrete levels [12]. An error occurs in an MLC when the level read from the cell is not the exact level that was written. Due to the ordering of the stored levels, MLC errors may still produce data that is nearly correct, and this presents an opportunity for approximate computing. Approximate storage in phase-change MLCs is shown to offer 1.7x speedup in write operations [10]. In flash memory, an alternative to demarcating MLC levels with thresholds is the use of rank modulation [2] to encode data in the relative ordering of charge levels across cells.

2.1 QBF Solving

The quantified Boolean formula problem (QBF) is a Boolean satisfiability problem (SAT) with both universally ($\forall$) and existentially ($\exists$) quantified variables. SAT can therefore be viewed as a special case of QBF in which all variables are existentially quantified. A QBF problem is typically described in prenex normal form, where all variables are quantified at the start of the problem, and the formula follows in conjunctive normal form. Among the many publicly available QBF solvers, the experiments in this work are performed using the solver DepQBF [7][6]. DepQBF is a search-based QBF solver, and its speed comes largely from its use of restarts and efficient techniques for uncovering and storing variable ordering [5].

3. Formulation

Figure 1 illustrates the problem of splitting 8-bit data word D over two 4-bit MLCs C_1 and C_0. The 4-bit value written to each cell C_i is denoted C_i^W, and the 4-bit value being read is C_i^R. The data word being written is D^W, and the data word being read is D^R. When writing data word D^W, a mapping denoted S_W maps bit positions of D^W to the bit positions of C_1^W and C_0^W, and these values are written to the MLCs. When reading a data word, C_1^R and C_0^R are read from the cells, and mapping S_R reconstructs data word D^R.

Mapping S_R inverts S_W, such that $S_R(S_W(D^W)) = D^W$. If the cells are fully precise (i.e. $(C_1^W, C_0^W) = (C_i^R, C_0^R)$), then any S_R and S_W that are inverses will result in fully precise data words (i.e. $D^W = D^R$). To see this, first let $D^R := S_R(C_1^R, C_0^R))$ which becomes on account of fully-precise cells $D^R := S_R(C_1^W, C_0^W))$. Next, rewriting (C_1^W, C_0^W) as $S_W(D^W)$ yields $D^R := S_R(S_W(D^W))$ which reduces to $D^R := D^W$ whenever S_R inverts S_W.

The mappings S_W and S_R are represented using a shared set S of Boolean variables. Let k be the number of MLCs, and n be the number of bits stored in each MLC; the width of the data word D is therefore kn. The mapping between the data word and cells is represented by kn^2 Boolean variables $s_{i,j}$, where $0 \leq i < kn$ and $0 \leq j < kn$. If $s_{i,j}$ is true, then S_W maps the i^{th} bit of D^W to the j^{th} among all MLC bits, and S_R maps the j^{th} MLC bit to the i^{th} bit of D^R. Eq. 1 ensures that each mapping is between exactly one bit of D and one bit of an MLC.

$$\bigwedge_{j\in[0,kn-1]} \left(\left(\sum_{i=0\ldots kn-1} s_{i,j} \right) = 1 \right) \bigwedge_{i\in[0,kn-1]} \left(\left(\sum_{j=0\ldots kn-1} s_{i,j} \right) = 1 \right) \quad (1)$$

Under the assumption of approximate storage, it is only claimed that for each MLC the read value (C_i^R) is similar within E_C levels to the written value (C_i^W). Assuming that the difference in magnitude between the written and read value of each MLC is bounded by E_C, the goal of synthesis is to minimize the worst-case difference in magnitude between D^W and D^R, as this difference represents the word-level error. The task of synthesis is therefore to find an assignment to S that defines a correspondence between data word D and MLCs C_i that minimizes the worst case word-level error.

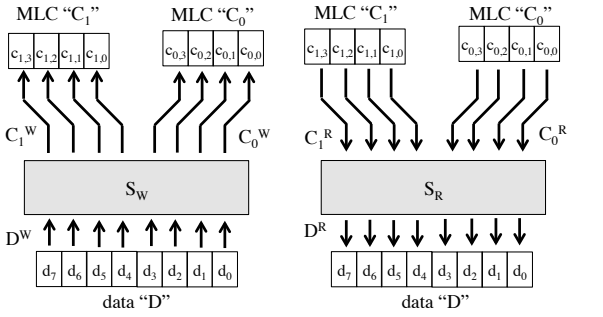

Figure 1. Mapping for write and read operations. At left is a write operation, where a data word D^W is distributed across two MLCs according to mapping S_W. At right is a read operation where, according to mapping S_R, data word D^R is reconstructed from the values read from the two MLCs.

3.1 Synthesis using QBF Solving

The QBF problem in Eq. 2 checks whether there exists a mapping that guarantees a worst-case word-level error of E_D, given a worst-case cell-level error of E_C.

The independent variables in Eq. 2 are (1) S: the set of Boolean variables that determine the mappings (S_R and S_W) between bits of the data word and bits of MLCs; (2) D^W: the data word being written; and (3) C_1^R and C_0^R: the values read from the MLCs.

Computed as a deterministic function of the independent variables are (1) $(C_1^W, C_0^W) := S_W(D^W)$: the values written to MLCs; and (2) $D^R := S_R(C_1^R, C_0^R)$: the data word reconstructed from the values read from MLCs.

Eq. 2 then asks whether there exists mapping S, such that if the written data word D^W maps to written MLC levels C_1^W, C_0^W, any cell levels C_1^R, C_0^R that are within E_C of the written levels will map back to a word D^R that is within E_D of the written word D^W.

$$\begin{aligned} &\exists S \forall D^W \forall C_1^R \forall C_0^R \\ &\big(\big(|C_1^W - C_1^R| \leq E_C\big) \wedge \big(|C_0^W - C_0^R| \leq E_C\big)\big) \\ &\implies \big(|D^W - D^R| \leq E_D\big) \end{aligned} \quad (2)$$

An equivalent formulation to Eq. 2 is given by Eq. 3. This formulation replaces independent variable D^W with variables C_1^W and C_0^W, and computes as a dependent variable $D^W := S_R(C_1^W, C_0^W)$, which holds because $(C_1^W, C_0^W) := S_W(D^W)$ and S_R inverts S_W. While the formulations are equivalent, and shown to produce identical results, the formulation of Eq. 3 is found to be easier for the solver than is Eq. 2 (see Appendix A).

$$\exists S \forall C_1^W \forall C_0^W \forall C_1^R \forall C_0^R$$
$$\big((|C_1^W - C_1^R| \leq E_C) \wedge (|C_0^W - C_0^R| \leq E_C)\big) \quad (3)$$
$$\implies \big(|S_R(C_1^W, C_0^W) - S_R(C_1^R, C_0^R)| \leq E_D\big)$$

3.2 Optimization

A binary search optimization loop performs iterated calls to the QBF solver to find a mapping S that minimizes the worst-case error E_D. Each QBF call solves Eq. 3 with a particular constant for E_D, and either synthesizes a value of S that guarantees the bound E_D, or else determines that no such S exists. The total range searched for the minimal E_D is 1 to $2^{kn} - 1$, and therefore kn QBF calls are needed to complete the binary search.

4. Methodology

Each QBF problem is formulated by first constructing a combinational circuit in structural Verilog to implement the necessary logic for evaluating the property in Eq. 3. The inputs to the circuit are the Boolean variables comprising S and $C_1^W, C_0^W, C_1^R, C_0^R$, and the constants to set E_C and E_D. The output of the circuit is a single Boolean signal, denoted ϕ, representing the condition that synthesis must satisfy. This signal ϕ is false when the inputs applied to the circuit satisfy $|C_1^W - C_1^R| \leq E_C$ and $|C_0^W - C_0^R| \leq E_C$, but not $|D^W - D^R| \leq E_D$.

The mappings for S_R and S_W are defined according to the values assigned to the $s_{i,j}$ variables in S. The mappings are implemented in the circuit by multiplexers with $s_{i,j}$ variables as one-hot encoded control inputs. In the implementation of S_W, the value of the j^{th} total bit to the MLCs is set by a multiplexer with select signals $s_{0,j}, s_{1,j}, \ldots, s_{kn-1,j}$ and inputs $d_0, d_1, \ldots, d_{kn-1}$. Likewise, in the implementation of S_R, each bit d_i is assigned a bit from an MLC, with choice depending on select signals $s_{i,0}, s_{i,1}, \ldots, s_{i,kn-1}$. Therefore, if $s_{i,j}$ is true, then S_W maps the i^{th} bit of the data word to the j^{th} bit of cells, and S_R maps the j^{th} bit of cells back to the i^{th} bit of the data word.

The combinational circuit is encoded into the CNF clauses of the QBF problem by translating each gate to clauses using the approach of Larrabee [3]. The CNF formula created in this way is satisfied by any assignment of values to variables that is consistent with the logic functions of the gates. Because the goal of synthesis is to ensure that property ϕ holds, ϕ is added to the CNF formula as a unit clause; now any variable assignment that satisfies the CNF formula is an assignment of values to variables that is consistent with the circuit logic and causes ϕ to be true. Finding such an assignment is exactly the problem of SAT-based test generation [3], where the goal is to find any input pattern that satisfies a particular fault sensitization condition.

The inputs to the Verilog circuit are the variables that are either existentially or universally quantified in the QBF. Specifically, in the formulation of Eq. 3 these variables are the $(kn)^2$ bits of S, and for each cell C_i ($0 \leq i < k$) the n bits for C_i^W and the n bits for C_i^R. The total number of Boolean input variables is therefore $(kn)^2 + 2kn$. As shown in Eq. 3, the $(kn)^2$ Boolean variables of S are existentially quantified, and the $2kn$ Boolean variables for the C_i^R and C_i^W are universally quantified. A solution to the QBF problem gives an assignment to the S variables, such that the property ϕ is guaranteed to hold for all possible assignments to the C_i^R and C_i^W variables.

Note that when a QBF problem is described in prenex normal form, all variables must have quantifiers. To accomplish this, any variable not explicitly quantified in a stated QBF problem is implicitly existentially quantified at the end of the explicitly quantified variables. This implicit existential quantifier indicates that these variables may take any value in a solution to the QBF problem.

5. Evaluation

Two variants of QBF-based synthesis are performed, and compared against a baseline approach of interleaved word-splitting. Interleaved word-splitting, shown in Fig. 2, is proposed by Sampson *et al.* [10] for minimizing the word-level impact of errors in MLCs.

In interleaved word-splitting, the more significant bits of a word (e.g. d_7 and d_6) are mapped to the most significant bits of each MLC (e.g. $c_{1,3}$ and $c_{0,3}$). The intuition behind interleaved word-splitting is that errors in imprecise MLCs are more likely to flip to the less significant bits of the cell than the more significant bits. For example, in a 4-bit MLC, all single-level errors (e.g. 0000 to 0001; 0001 to 0010; etc) will flip the cell's LSB, while the only single-level errors that flip the cell's MSB are transitions between the levels 0111 and 1000. Therefore, interleaved word-splitting minimizes the average word-level impact of the common case of single-level errors, by mapping the less significant bits of data words to the MLC bits that are most likely to flip.

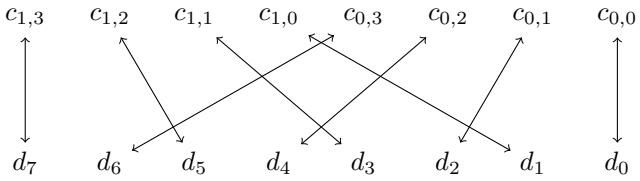

Figure 2. Interleaved word-splitting.

Interleaved word-splitting is proposed with average-case word-level error in mind, but in the worst case even single-level MLC errors can lead to large word-level errors. Consider the written value $D^W := 00111111$, which according to Fig. 2 is stored in the two cells as the levels $C_1^W := 0111$ and $C_0^W := 0111$. If single-level errors cause the cells to be read out as $C_1^R := 1000$ and $C_0^R := 1000$, then the reconstructed data word is $D^R := 11000000$. Despite only having a single level of imprecision in each 16 level cell, with interleaving the word-level error can be over 200%, as in this example the written word (D^W) is 63, and the read word (D^R) is 192.

5.1 Synthesis of Optimal Word-Splitting

Motivated by the pathologically bad worst-case word error with interleaving, the first problem addressed is synthesizing a word-splitting that minimizes worst-case error. Under the assumption that MLC error is no more than one-level per cell (i.e. $E_C := 1$), using the QBF formulation of Eq. 3, the binary search yields an optimal assignment to S that guarantees a word-level error magnitude of no more than 17 (i.e. $E_D := 17$). The runtimes for each QBF call in the binary search are shown in Table 1, and the optimal synthesized mapping is shown in Fig. 3.

The synthesized solution splits the data word into contiguous upper and lower blocks that are written to cells C_1 and C_0, respectively. In this solution, any single-level error in C_1 equates to a word error of 16, and any single-level error in C_0 equates to a word error of 1. This synthesized mapping is equivalent to an approach denoted "distributed analog" by Sarpeshkar [11] in his work on analog computation. Beyond reproducing Sarpeshkar's result, a contribution of this work is that it produces the result through automated synthesis, and shows the result to be optimal for worst-case word error by giving a formal guarantee of optimality.

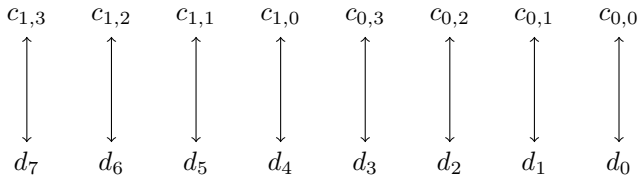

Figure 3. Synthesized optimal word-splitting according to Eq. 3.

E_D	solution exists	runtime [seconds]
16	no	12.05
17	**yes**	**16.81**
18	yes	15.28
20	yes	11.58
24	yes	11.13
32	yes	10.94
64	yes	13.16
128	yes	10.19

Table 1. Results of QBF solving Eq. 3 at each step of the binary search to minimize word error E_D. E_D is the bound on $|D^W - D^R|$ in each call to the QBF solver. Whenever a solution exists for the checked value of E_D, the solution is an assignment to the $s_{i,j}$ variables that defines the mappings between data and cells. Each call to the QBF solver has 64 existentially quantified variables, 16 universally quantified variables, 1951 total variables, and 5182 CNF clauses. The synthesized optimal result is the word-splitting shown in Fig. 3.

5.2 Synthesis of Optimal Word-Splitting with Remapping

The next synthesis formulation finds that, counter-intuitively, a smaller word-level error bound is possible by making intentional errors in writing to MLCs. This result is achieved by first synthesizing the erroneous writes at the MLC level, and then back-propagating them to the word-level, where they are ultimately implemented.

To allow synthesis the freedom to work around any pathological corner cases, among the sixteen 4-bit levels that can be written to C_1, any two are allowed to be remapped. The two levels that are remapped can be any of the sixteen possible levels, and the target level that each is remapped to can also be any of the sixteen possible levels. This remapping at the cell level is denoted M_C, and M_C is implemented using 16 Boolean variables $m_0, \ldots, m_{15}$. Each of the remapped levels uses eight of the m_i variables: four to select the level to remap, and four to choose what level it remaps to. The m_i variables that define M_C are added to the existentially quantified variables in the QBF problem, so that synthesis will assign them values that guarantee the worst-case word-level bound E_D. In the formulation of Eq. 4, the level that would be written to C_1 in absence of remapping is denoted $\widetilde{C_1^W}$, and the potentially erroneous level that is written after the remapping is $M_C(\widetilde{C_1^W})$.

$$\begin{aligned} &\exists S \exists M_C \forall \widetilde{C_1^W} \forall C_0^W \forall C_1^R \forall C_0^R \\ &\left(\left(|M_C(\widetilde{C_1^W}) - C_1^R| \le E_C\right) \wedge \left(|C_0^W - C_0^R| \le E_C\right)\right) \\ &\implies \left(|S_R(\widetilde{C_1^W}, C_0^W) - S_R(C_1^R, C_0^R)| \le E_D\right) \end{aligned} \tag{4}$$

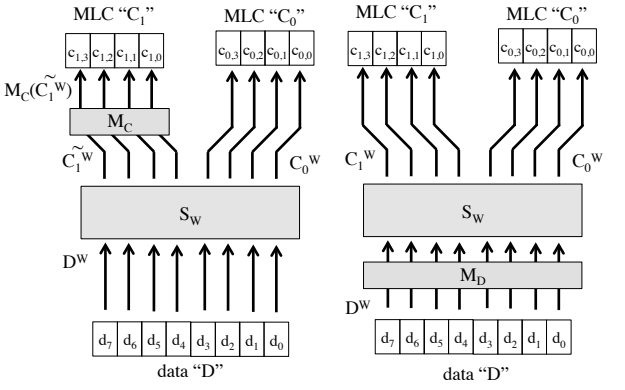

Figure 4. Synthesized write operations with remapping. At left is the synthesized version where the remapping is applied on the cell inputs, and at right is the final version where remapping is applied to the data word before splitting.

Table 2 shows the QBF solver runtimes when using the QBF formulation of Eq. 4 inside of a binary search to minimize E_D. The optimization loop terminates with a synthesized result that guarantees a worst-case word-level error bound of 10. The synthesized result assigns values to the

80 existentially quantified Boolean variables comprising S and M_C. The remapping (M_C) in the synthesized solution (Eq. 5) prevents levels 1000 and 0111 from ever being written to the cell, as they are always replaced by erroneous levels 1001 and 0110, respectively. This allows an additional error of one level (since Eq. 5 shows that $\widetilde{C_1^W}$ and $M_C(\widetilde{C_1^W})$ may differ by at most one), and ultimately allows an overall two level error because the value read from the cell (C_1^R) may further deviate from the written value ($M_C(\widetilde{C_1^W})$) by one level on account of $E_C := 1$.

While the synthesis problem of Eq. 4 solves for S and M_C, ultimately the remapping should be applied at the word level instead of the cell level. Given that S is known from synthesis, M_C is back-propagated to the word level using knowledge of which bits of the data word will be mapping to cell C_1. This corresponds to transitioning from the schematic at left in Fig. 4 to the schematic at right in Fig. 4. The mapping M_C (Eq. 5) when applied at the word level becomes M_D (Eq. 6).

Overall, the synthesized word-splitting with remapping is applied as follows. Given an 8-bit data word D^W to be written in two 4-bit MLCs, first apply function M_D; this introduces error if the value of D^W is modified by M_D. The output of M_D is written across the cells using the mapping shown in Fig. 5. When reading the word from the cells, the level read out of each cell can differ by one level from the value that was written. Finally, mapping S_R is applied to reconstruct D^R. The difference between D^R and D^W, including both the error due to remapping by M_D and the error due to imprecise cells, can never exceed 10. Since the error bound using word-splitting alone was 17, the worst-case bound is improved by the addition of intentional errors on the written values.

Note that in this synthesis result, even if the value read from each cell matches exactly to the value written, D^R may still not match D^W due to the error introduced by the remapping by M_D during the write process. Furthermore, note that a bit-flip in position 0 of cell C_1 (i.e. $c_{1,0}$ in Fig. 5) equates to word error of 2 because it maps to d_1, while a bit-flip in position 1 of the same cell (i.e. $c_{1,1}$) will cause a word error of only 1 because it maps to the LSB d_0. Since single-level errors are more likely to flip bit 0 (i.e. $c_{1,0}$) of the cell, a better average-case word error may be obtained by having $c_{1,1}$ correspond to d_1, and $c_{1,0}$ correspond to d_0. However, a synthesis that is guided by worst-case has no reason to prefer this alternative solution over the one in Fig. 5.

$$M_C(\widetilde{C_1^W}) = \begin{cases} 1001 & \text{if } \widetilde{C_1^W} = 1000 \\ 0110 & \text{if } \widetilde{C_1^W} = 0111 \\ \widetilde{C_1^W} & \text{otherwise} \end{cases} \quad (5)$$

$$M_D(D^W) := \begin{cases} 1[d_{6\ldots3}]010 & \text{if } D^W = 1[d_{6\ldots3}]000 \\ 0[d_{6\ldots3}]101 & \text{if } D^W = 0[d_{6\ldots3}]111 \\ D^W & \text{otherwise} \end{cases} \quad (6)$$

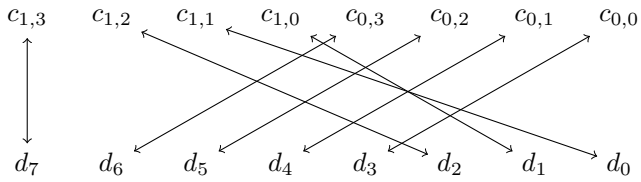

Figure 5. Synthesized word-splitting that is optimal when used in conjunction with the remapping M_D (Eq. 6) applied to D^W.

E_D	solution exists	runtime [seconds]
8	no	298.20
9	no	477.90
10	**yes**	**417.94**
12	yes	384.47
16	yes	315.57
32	yes	49.28
64	yes	34.49
128	yes	21.22

Table 2. Results of binary search to generate the mapping in the case with remapping of input levels allowed. The QBF problem at each loop iteration has 80 existentially quantified variables, 16 universally quantified variables, 2037 total variables, and 5376 CNF clauses.

5.3 Comparison of Results

The interleaved word-splitting is compared to the two optimal word-splittings synthesized using QBF solving in this paper. The first of the synthesized results uses only word-splitting, and the second using word-splitting and level remapping. The synthesis technique optimizes for worst-case word error, but it is instructive to also consider the distribution of word errors and average word error. Note that while the worst-case bounds are an absolute result that holds for any probability of single-level errors, comparisons of average case depend on the probabilities of single level MLC errors.

Fig. 6 shows the distribution of word-level error if each cell is assumed to have a very high 50% probability of a single-level error, and the error is equally likely to be an increase or decrease in level. On 25% of trials, the two cells are both error free.

- The upper bound on word error in the interleaved word splitting is 129, by far the highest among the three results. However, the interleaved approach has the lowest probability of a word-level error exceeding 5.
- In the synthesized word-splitting (Sec. 5.1), the worst-case word error is 17. The word-level error contribution of cell C_1 can only be either -16,0, or 16, and the word-level contribution of single-level errors for cell C_0 is -1,0, or 1, and therefore the only possible magnitudes of word errors are 0,15,16, and 17.
- In the synthesized solution for word-splitting with remapping (Sec. 5.1), the worst-case word error is 10. Note that

the probability of having a word error of 0 is less than 25%, because the remapping M introduces a systematic error even when the cells are error free.

Figure 7 shows the average word error as a function of the probability of a single-level error in each MLC. When the cell error probability is 0, the only approach to have non-zero word-level error is the synthesized word-splitting with remapping. This error is on account of the remapping, which introduces a systematic error for the remapped levels. As the probability of cell error increases, the synthesized result with remapping has the best average-case error, in addition to the best worst-case error of 10.

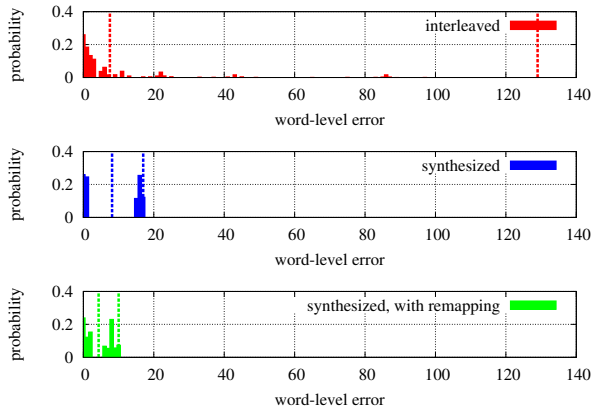

Figure 6. Probability distribution of word-level error ($|D^R - D^W|$) using the three approaches discussed in the paper. The distribution is obtained from 500k random trials using Verilog simulation. The results are for the the interleaved word-splitting and the two synthesized word-splitting approaches. The probability of a single-level error in each cell is 50%, and the errors are independent for each cell. The vertical lines are the average word-level error and the largest error observed in the random trials.

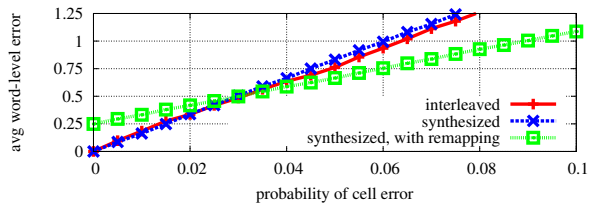

Figure 7. Observed average word-level error ($|D^R - D^W|$) as a function of the probability of a single-level error in each MLC.

6. Future Work and Conclusions

This work proposed the use of automated synthesis techniques for splitting data words across approximate MLC storage, such that the worst-case error is minimized at the word-level. The synthesis technique is based on QBF-solving, and the results are validated through random simulation.

While MLC errors will generally occur as transitions between neighboring levels as assumed in this work, future work will explore an error model that is more closely matched to the particular errors of a specific MLC technology. Other QBF solving techniques such as quantifier instantiation can be applied. To address the limitation of having no preference among solutions with equivalent worst-case word-level error bounds (Sec. 5.3), the synthesis technique can be extended to consider all solutions that guarantee the optimal worst-case bound, and to choose among them based on a metric such as average-case word-level error.

Acknowledgement: This work was supported in part by C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA, and NSF CNS-0845874. Any opinions, findings, conclusions, and recommendations expressed in these materials are those of the authors and do not necessarily reflect the views of the sponsors

References

[1] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *ASPLOS XVI: Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, Apr. 2012.

[2] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck. Rank modulation for flash memories. *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 1731–1735, 2008.

[3] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, 1992.

[4] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flikker: saving DRAM refresh-power through critical data partitioning. In *ASPLOS XVI: Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, June 2012.

[5] F. Lonsing and A. Biere. DepQBF: A Dependency-Aware QBF Solver. *JSAT*, 7(2-3):71–76, 2010.

[6] F. Lonsing, U. Egly, and A. Van Gelder. Efficient Clause Learning for Quantified Boolean Formulas via QBF Pseudo Unit Propagation. In *SAT 2013*, pages 100–115, 2013.

[7] A. Niemetz, M. Preiner, F. Lonsing, M. Seidl, and A. Biere. Resolution-Based Certificate Extraction for QBF - (Tool Presentation). In *SAT*, pages 430–435, 2012.

[8] M. Salajegheh, Y. Wang, A. A. Jiang, E. Learned-Miller, and K. Fu. Half-Wits: Software Techniques for Low-Voltage Probabilistic Storage on Microcontrollers with NOR Flash Memory. *ACM Transactions on Embedded Computing Systems*, 12 (2s), May 2013.

[9] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: approximate data types for safe and general low-power computation. In *PLDI '11: Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, June 2011.

[10] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate Storage in Solid-State Memories. *IEEE Micro*, 2013.

[11] R. Sarpeshkar. Analog Versus Digital: Extrapolating from Electronics to Neurobiology. *Neural Computation*, 10(7): 1601–1638, Oct. 1998.

[12] C. Trinh, N. Shibata, T. Nakano, M. Ogawa, J. Sato, Y. Takeyama, K. Isobe, B. Le, F. Moogat, N. Mokhlesi, K. Kozakai, P. Hong *et al.* A 5.6MB/s 64Gb 4b/Cell NAND Flash memory in 43nm CMOS. In *Solid-State Circuits Conference - Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pages 246–247, 2009.

A. QBF Formulation

Fig. 8 shows the runtime for solving the word-splitting synthesis problem as in Sec. 5.1, except using linear search instead of binary search. The formulation using Eq. 3 is the same experiment as Table 1, except with different values of E_D. The two formulations of the problem are equivalent, and both formulations produce the same optimal result and the same optimal bound of $E_D := 17$. Future work will investigate why the formulation of Eq. 3 is solved considerably faster than that of Eq. 2 for bounds less than 17 (i.e. where the problem is unsatisfiable.)

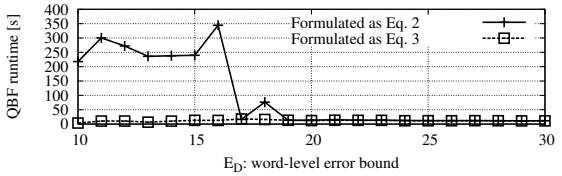

Figure 8. Runtime of QBF solver when checking various bounds using two equivalent formulations of the same QBF problem (Eq. 2 and Eq. 3). The optimal synthesis result in both cases has a worst-case bound of $E_D = 17$. All synthesis attempts where E_D is less than 17 fail, and in these cases, the encoding of Eq. 3 is significantly faster. All synthesis attempts when E_D is greater than or equal to 17 are successful, and in these cases, the choice of encoding has a smaller impact on runtime.