Formal Verification by The Book: Error Detection and Correction Codes

K. Devarajegowda, V. Hiltl, T. Rabenalt, D. Stoffel, W. Kunz, W. Ecker
Outline

Error Correction Codes

Formal Verification of ECCs
- 1st try - Brute-force
- 2nd try - Divide & conquer
- Final try - Linearity

Results and Conclusion
Outline

Error Correction Codes

Formal Verification of ECCs
- 1st try - Brute-force
- 2nd try - Divide & conquer
- Final try - Linearity

Results and Conclusion
Error Detection and Correction Codes - ECCs

- 2 stages
  - encoding
  - decoding
ECC encoding

- **data bits** are encoded with additional **ECC bits**
- resulting **codeword** is written to the memory
Error Correction Codes

- Soft errors cause data corruption
  - $0 \rightarrow 1$ or $1 \rightarrow 0$
  - codeword becomes a non-codeword
Error Correction Codes

ECC decoding
- error flags are set
- data is corrected \textit{when \#bit-errors} $\leq$ \textit{correctable-range}
Outline

Error Correction Codes

Formal Verification of ECCs
- 1st try - Brute-force
- 2nd try - Divide & conquer
- Final try - Linearity

Results and Conclusion
Formal Verification of ECCs: Setup

- Program flash interface is irrelevant for proving the correctness of ECCs
Formal Verification of ECCs: Setup

- Program flash interface is irrelevant for proving the correctness of ECCs
Formal Verification of ECCs: Setup

- An RTL wrapper is written instantiating only the encoder and decoder
Outline

Error Correction Codes

Formal Verification of ECCs
- 1st try - Brute-force
- 2nd try - Divide & conquer
- Final try - Linearity

Results and Conclusion
Formal Verification of ECCs: Brute-force

property triple_bit_error_detect;
  $countones(wr_data ^ rd_data)==3 |
  ->
  err_3_out && err_det_out &&
  !err_1_out && !err_2_out;
endproperty
Formal Verification of ECCs: Brute-force

property triple_bit_error_detect;
  $countones(wr_data ^ rd_data)==3
  |->
  err_3_out && err_det_out && !err_1_out && !err_2_out;
endproperty

analysis space: $2^{256} \times \binom{278}{3}$
Formal Verification of ECCs: Brute-force

- Computation resources
  - memory
  - time
- Given up after 100 hours

analysis space: $2^{256} \times \binom{278}{3}$
Outline

Error Correction Codes

Formal Verification of ECCs
- 1st try - Brute-force
- 2nd try - Divide & conquer
- Final try - Linearity

Results and Conclusion
### Formal Verification of ECCs: Brute-force

<table>
<thead>
<tr>
<th>ecc_out</th>
<th>data_in</th>
</tr>
</thead>
<tbody>
<tr>
<td>21............0</td>
<td>255................0</td>
</tr>
</tbody>
</table>

**Analysis space:**

\[
2^{256} \times \binom{278}{3}
\]
Formal Verification of ECCs: Divide & conquer
property triple_bit_error_detect;
  (wr_data[255:16] == 239’h43865af3c5dfa) &&
  (wr_data[277:16] == rd_data[277:16]) &&
  $countones(wr_data[15:0] ^ rd_data[15:0]) == 3
  |->
  err_3_out && err_det_out &&
  !err_1_out && !err_2_out;
endproperty
Formal Verification of ECCs: Divide & conquer

property triple_bit_error_detect;
    (wr_data[255:16] == 239'h43865af3c5dfa) &&
    (wr_data[277:16] == rd_data[277:16]) &&
    $countones(wr_data[15:0] ^ rd_data[15:0])==3
|->
    err_3_out && err_det_out &&
    !err_1_out && !err_2_out;
endproperty
Formal Verification of ECCs: Divide & conquer

<table>
<thead>
<tr>
<th>ecc_out</th>
<th>fixed &amp; error free</th>
<th>Symbolic</th>
</tr>
</thead>
<tbody>
<tr>
<td>21......0</td>
<td>255..................0</td>
<td>15.........0</td>
</tr>
</tbody>
</table>

- Manageable complexity – full proof
- Large number of properties
- Not exhaustive
- Coverage?
- Verification gap?

analysis space: 

\[ 2^{16} \times \binom{16}{3} \]
Outline

Error Correction Codes

Formal Verification of ECCs
- 1st try - Brute-force
- 2nd try - Divide & conquer
- Final try - Linearity

Results and Conclusion
Formal Verification of ECCs: Linearity

Wrapper

ECC Encoder

Assume bit errors through formal assumptions

ECC Decoder

wr_data → ECC Encoder

rd_data → ECC Decoder

data_in (255:0)

data_out (255:0)

ecc_out (21:0)

err_2_out

err_3_out

er_1_out

err_det_out

data_out (255:0)

Assume bit errors through formal assumptions
Formal Verification of ECCs: Linearity

Assume bit errors through formal assumptions
Formal Verification of ECCs: Linearity

Assume bit errors through formal assumptions

ECC Encoder

wr_data

rd_data

Syndrome Generator

ECC Decoder

err_1_out
err_2_out
err_3_out
err_det_out

data_out (255:0)
ecc_out (21:0)
Formal Verification of ECCs: Linearity

- Key findings:
  1. \((wr\_data = rd\_data) \Rightarrow (syn\_out = 0)\)
Formal Verification of ECCs: Linearity

<table>
<thead>
<tr>
<th>data_in</th>
<th>insert bit-error at</th>
<th>syn_out</th>
</tr>
</thead>
<tbody>
<tr>
<td>256`h2874547623442</td>
<td>5th bit position</td>
<td>22`d0005</td>
</tr>
</tbody>
</table>
Formal Verification of ECCs: Linearity

<table>
<thead>
<tr>
<th>\textbf{data_in}</th>
<th>\textbf{insert bit-error at}</th>
<th>\textbf{syn_out}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{256`h2874547623442}</td>
<td>5th bit position</td>
<td>\texttt{22`d0005}</td>
</tr>
<tr>
<td>\texttt{256`h644abcf5472034}</td>
<td>5th bit position</td>
<td>\texttt{22`d0005}</td>
</tr>
</tbody>
</table>
Formal Verification of ECCs: Linearity

<table>
<thead>
<tr>
<th>data_in</th>
<th>insert bit-error at</th>
<th>syn_out</th>
</tr>
</thead>
<tbody>
<tr>
<td>256'h2874547623442</td>
<td>5th bit position</td>
<td>22'd0005</td>
</tr>
<tr>
<td>256'h644abcf5472034</td>
<td>5th bit position</td>
<td>22'd0005</td>
</tr>
<tr>
<td>256'h2874547623442</td>
<td>16th bit position</td>
<td>22'd0016</td>
</tr>
</tbody>
</table>
Formal Verification of ECCs: Linearity

<table>
<thead>
<tr>
<th>data_in</th>
<th>insert bit-error at</th>
<th>syn_out</th>
</tr>
</thead>
<tbody>
<tr>
<td>256`h2874547623442</td>
<td>5th bit position</td>
<td>22`d0005</td>
</tr>
<tr>
<td>256`h644abcf5472034</td>
<td>5th bit position</td>
<td>22`d0005</td>
</tr>
<tr>
<td>256`h2874547623442</td>
<td>16th bit position</td>
<td>22`d0016</td>
</tr>
<tr>
<td>256`h644abcf5472034</td>
<td>16th bit position</td>
<td>22`d0016</td>
</tr>
</tbody>
</table>
Formal Verification of ECCs: Linearity

- **Key findings:**
  1. \((wr\_data = rd\_data) \Rightarrow (syn\_out = 0)\)
  2. \(syn\_out\) is independent of \(data\_in\) and
  3. \(syn\_out\) depends **only** on the erroneous bit positions
Formal Verification of ECCs: Linearity

- **syn_out** is independent of **data_in**
  - Because, **Syndrome Generator** is a linear function
- **syn_out** is independent of **data_in**
  - because, **Syndrome Generator** is a linear function

- **Recall**: algebraic linear function
  - A function $f$ is linear if, $f(x) + f(y) = f(x+y)$
  - i.e., $f$ preserves the addition operation
**Formal Verification of ECCs: Linearity**

- **syn_out** is independent of **data_in**
  - *because, Syndrome Generator is a linear function*

- **syn(x ^ y) = syn(x) ^ syn(y)**
  - **syn(x)** – rtl function computing the syndrome
  - **x** and **y** are arbitrary vectors
  - `'^' (xor) is bitwise addition
Formal Verification of ECCs: Linearity

```
property syndrome_error_free_vector;
    wr_data == rd_data |-> syn_out == 0;
endproperty

property syndrome_linearity;
    syn(x) ^ syn(y) == syn(x ^ y);
endproperty
```
Formal Verification of ECCs: Linearity

property triple_bit_error_detect;
  data_in = 256'b76324abfdc23a8db &&
  $countones(wr_data ^ rd_data)==3
  |->
  err_3_out && err_det_out &&
  !err_1_out && !err_2_out;
endproperty

analysis space: $1 \times \binom{278}{3}$
Formal Verification of ECCs: Linearity

- The property converges in 01:45:54 (hh:mm:ss)

\[ 1 \times \left( \begin{array}{c} 278 \\ 3 \end{array} \right) \]
Outline

Error Correction Codes

Formal Verification of ECCs

- 1st try - Brute-force
- 2nd try - Divide & conquer
- Final try - Linearity

Results and Conclusion
### Experiments on 4 ECC Designs

<table>
<thead>
<tr>
<th>Design</th>
<th>Data bits</th>
<th>ECC bits</th>
<th>Detection</th>
<th>Correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC-1</td>
<td>256</td>
<td>22</td>
<td>1,2,3 bit errors</td>
<td>1,2 bit errors</td>
</tr>
<tr>
<td>SoC-1</td>
<td>64</td>
<td>22</td>
<td>1,2,3,4 bit errors</td>
<td>1,2,3 bit errors</td>
</tr>
<tr>
<td>SoC-2</td>
<td>26</td>
<td>6</td>
<td>1,2 bit errors</td>
<td>1 bit errors</td>
</tr>
<tr>
<td>SoC-3</td>
<td>16</td>
<td>6</td>
<td>1,2 bit errors</td>
<td>1 bit errors</td>
</tr>
</tbody>
</table>
### Results

<table>
<thead>
<tr>
<th>Design</th>
<th>Data bits</th>
<th>Detection</th>
<th>Correction</th>
<th>Brute-force</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC-1</td>
<td>256</td>
<td>1,2,3</td>
<td>1,2</td>
<td>given up (100hours)</td>
</tr>
<tr>
<td>SoC-1</td>
<td>64</td>
<td>1,2,3,4</td>
<td>1,2,3</td>
<td>given up (100hours)</td>
</tr>
<tr>
<td>SoC-2</td>
<td>26</td>
<td>1,2</td>
<td>1</td>
<td>1minute</td>
</tr>
<tr>
<td>SoC-3</td>
<td>16</td>
<td>1,2</td>
<td>1</td>
<td>4minutes</td>
</tr>
</tbody>
</table>

- **Formal tool:** OneSpin 360 DV
### Results

<table>
<thead>
<tr>
<th>Design</th>
<th>Data bits</th>
<th>Detection</th>
<th>Correction</th>
<th>Brute-force</th>
<th>Divide &amp; Conquer</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC-1</td>
<td>256</td>
<td>1,2,3</td>
<td>1,2</td>
<td>given up (100hrs)</td>
<td>40 hrs</td>
</tr>
<tr>
<td>SoC-1</td>
<td>64</td>
<td>1,2,3,4</td>
<td>1,2,3</td>
<td>given up (100hrs)</td>
<td>25 hrs</td>
</tr>
<tr>
<td>SoC-2</td>
<td>26</td>
<td>1,2</td>
<td>1</td>
<td>1min</td>
<td>---</td>
</tr>
<tr>
<td>SoC-3</td>
<td>16</td>
<td>1,2</td>
<td>1</td>
<td>4min</td>
<td>---</td>
</tr>
</tbody>
</table>

- **Formal tool:** OneSpin 360 DV
## Results

<table>
<thead>
<tr>
<th>Design</th>
<th>Data bits</th>
<th>Detection</th>
<th>Correction</th>
<th>Brute-force</th>
<th>Divide &amp; Conquer</th>
<th>Linearity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoC-1</td>
<td>256</td>
<td>1,2,3</td>
<td>1,2</td>
<td>given up (100hrs)</td>
<td>40 hrs</td>
<td>9hrs</td>
</tr>
<tr>
<td>SoC-1</td>
<td>64</td>
<td>1,2,3,4</td>
<td>1,2,3</td>
<td>given up (100hrs)</td>
<td>25 hrs</td>
<td>7hrs</td>
</tr>
<tr>
<td>SoC-2</td>
<td>26</td>
<td>1,2</td>
<td>1</td>
<td>1min</td>
<td>---</td>
<td>10sec</td>
</tr>
<tr>
<td>SoC-3</td>
<td>16</td>
<td>1,2</td>
<td>1</td>
<td>4min</td>
<td>---</td>
<td>1min</td>
</tr>
</tbody>
</table>

- **Formal tool**: OneSpin 360 DV
Formal verification of large ECCs

- **Prove the linearity of syndrome generator**
- **Prove the remaining properties with arbitrarily chosen, fixed data vector**
Summary

- Linearity approach
  - Significant reduction of property runtime
  - Previously *given-up* properties converge within 2hrs
  - Properties become simpler and are generated
  - 10x – 15x productivity gain

- Formal Verification - real benefit comes with the right approach
Thank You for listening!
Questions!