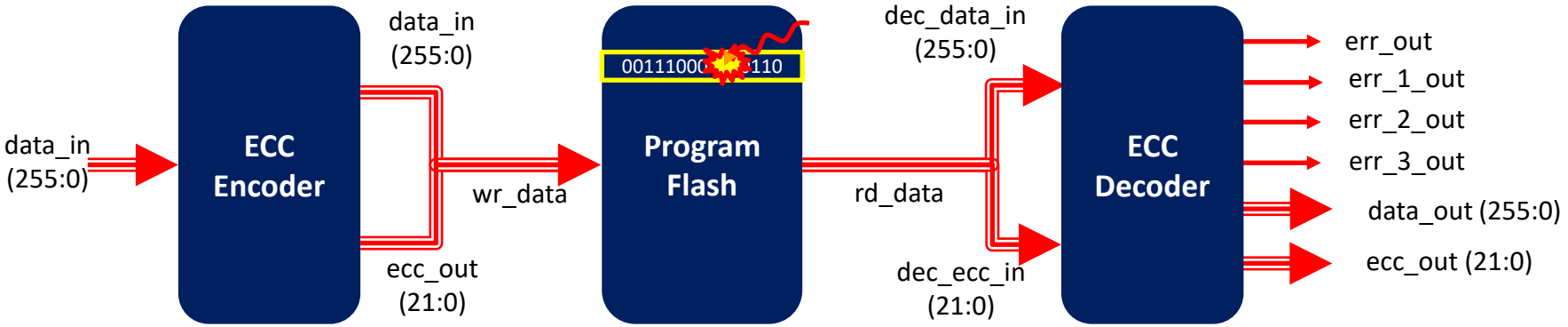
Formal Verification of Safety Mechanisms: Error Correction Codes

Keerthi Devarajegowda
08.02.2022

Functional Safety in Automotive Designs

- Safety functions are added to
  - Detect faults, Trigger alarm signals
  - Ensure correct operation even during unexpected scenarios
- Various safety mechanisms exist: DMR, TMR, ***ECC***, CRC, lock-step,...
  - A suitable mechanism selected based on
    - Criticality of the design block or ASIL rating required for the design
      - (Diagnostic coverage of safety mechanisms, area/performance/power overhead)

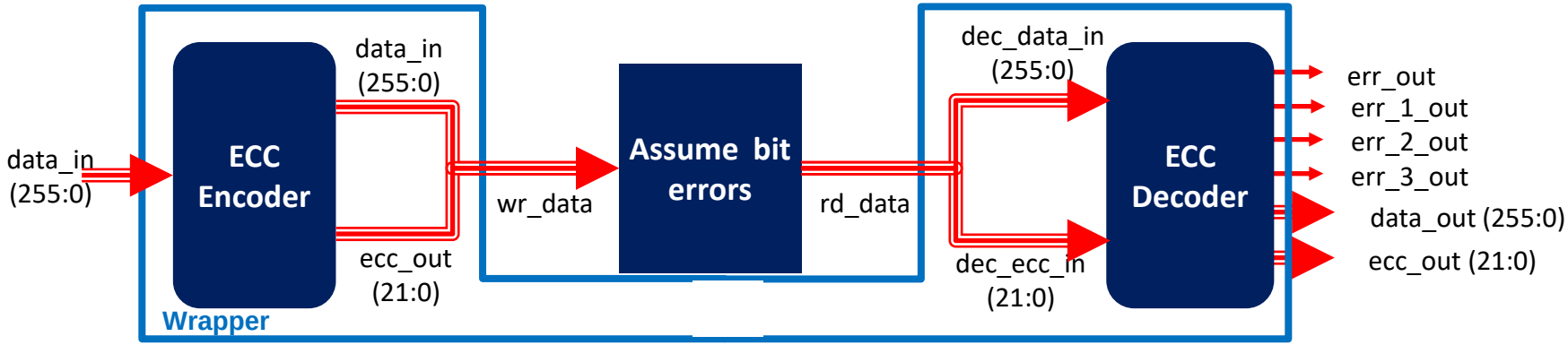
Error Correction Codes



Double Error Correction and Triple Error Detection (DECTED)

- ECC encoding
  - data bits are encoded with additional ECC bits
  - resulting codeword is written to the memory
- ECC decoding
  - error flags are set
  - data is corrected *when #bit-errors ≤ correctable-range*

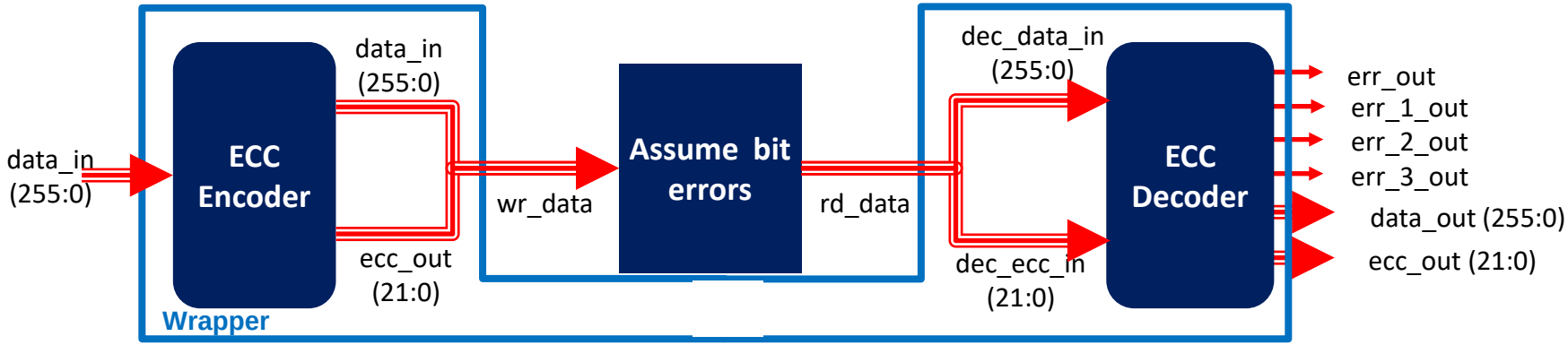
Formal Verification Setup



Formal verification setup for ECCs in Pflash memory interface

- For verifying the correct ECC design, memory (Pflash) interface is irrelevant
- Therefore, implement an RTL wrapper instantiating only the encoder and decoder
- Assume bit flips through formal assumptions

Formal Verification: Brute-force



ecc_out
21.....0

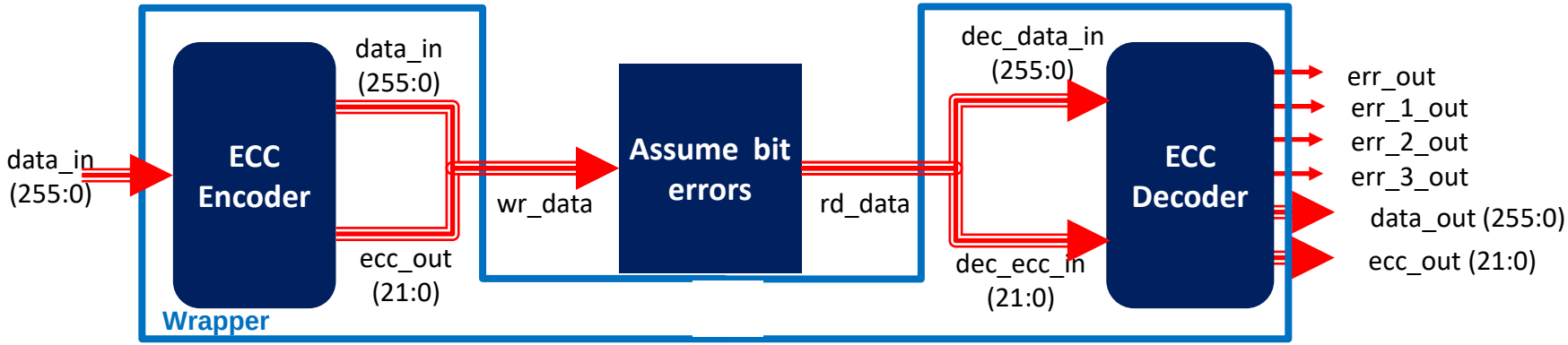
data_in
255.....0

```
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: Brute-force



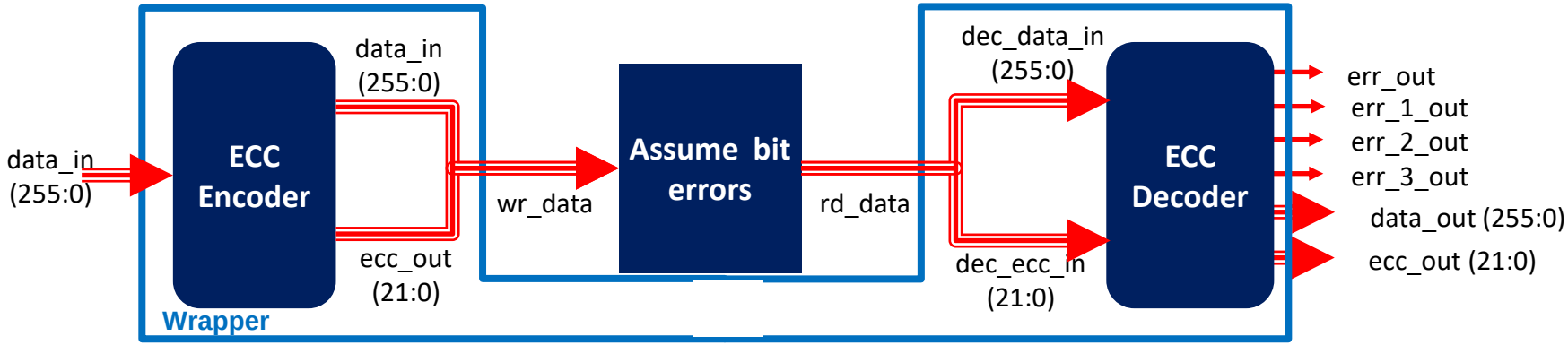
ecc_out	data_in
21.....0	255.....0

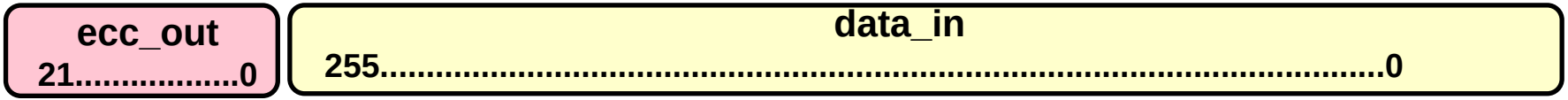
```
property triple_bit_error_detect;
 $countones(wr_data ^ rd_data)==3
 |->
 err_3_out && err_det_out && !err_1_out && !err_2_out;
endproperty
```

Analysis Space

$$2^{256} \times \binom{278}{3}$$

Formal Verification: Brute-force



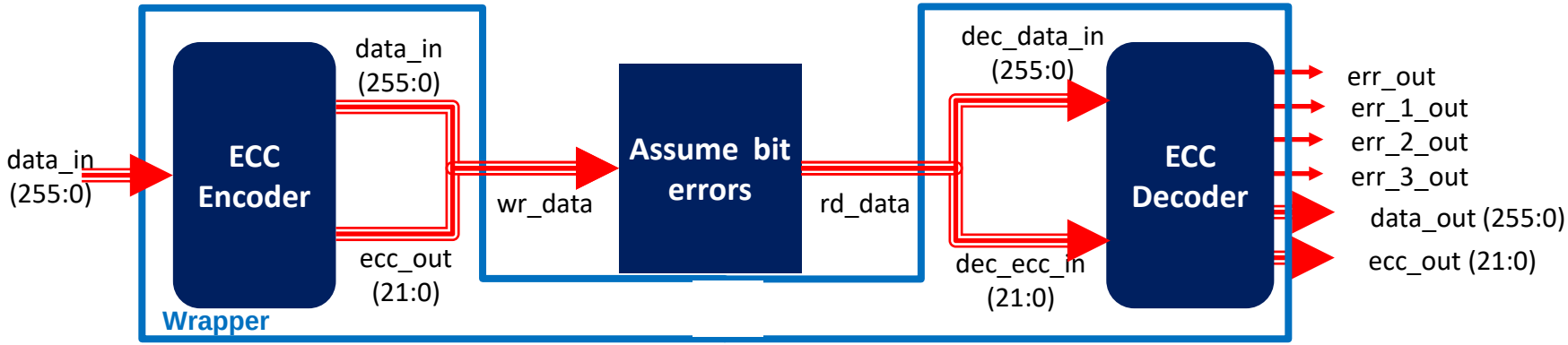


The tool gives up after running the property for 120hours

Analysis Space

$$2^{256} \times \binom{278}{3}$$

Formal Verification: Divide and Conquer



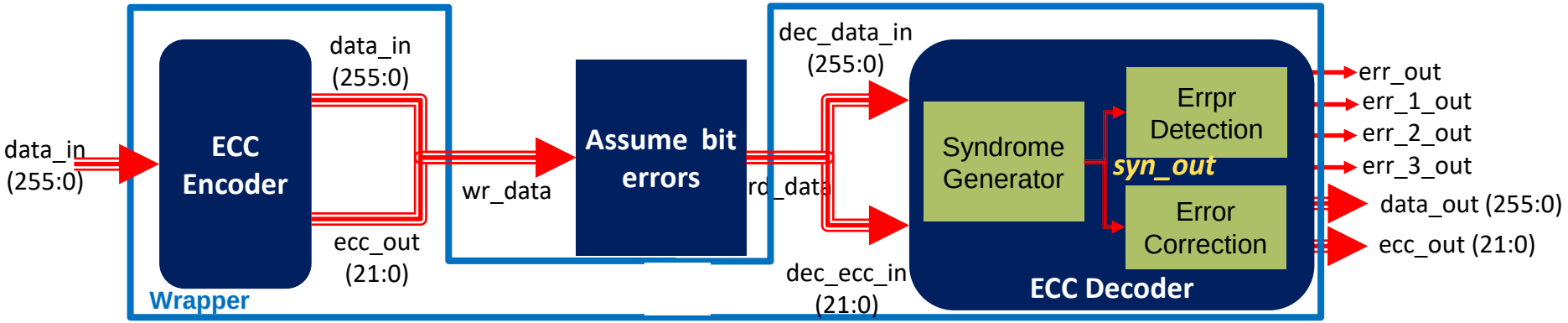
ecc_out	fixed and error-free	symbolic
21.....0	255.....16	15.....0

```
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
```

Analysis Space

$$2^{16} \times \binom{16}{3}$$

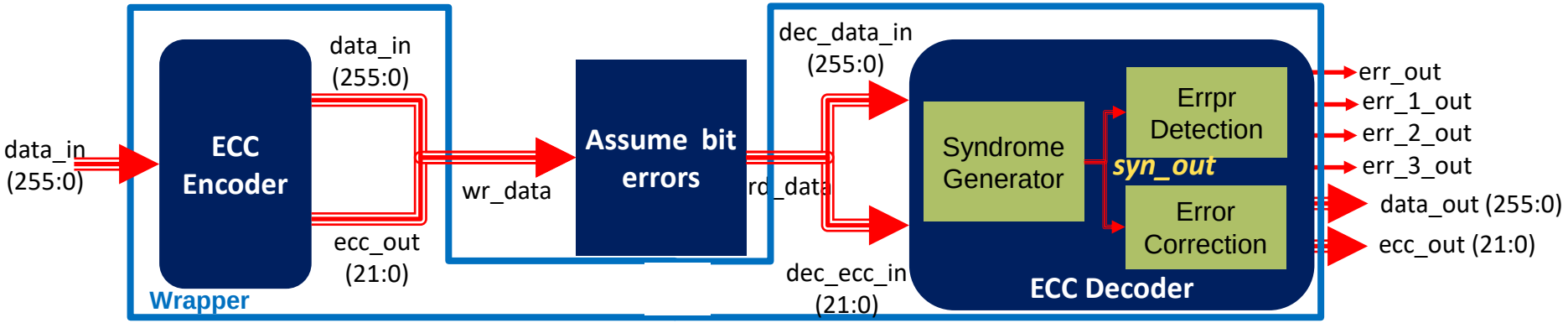
Formal Verification: Linearity



- The decoder generates the syndrome of the read data
- Characteristics of ***Syndrome Generator***
  - Let ***syn()*** be the operation implemented by the Syndrome Generator and ***syn_out*** be the output
  - Syndrome generator output is zero when there are no bit errors

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

Formal Verification: Linearity

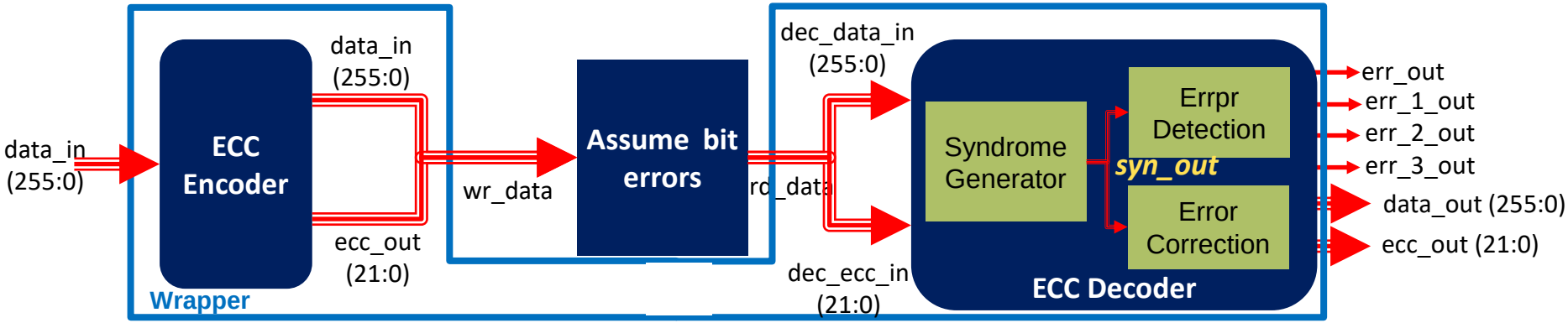


- Syndrome generation is a linear operation

```
property syndrome_linearity;
  syn(x) ^ syn(y) == syn(x ^ y);
endproperty
```

- **Syndrome generator output is independent of the input data and depends only on the erraneous bit positions**

Formal Verification: Linearity



- Once the ***syndrome_linearity*** property is proven, the rest of the properties can be proven with a fixed arbitrarily chosen input data vector

```
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

$$1x \binom{278}{3}$$

Results

Design	Data bits	Detection	Correction	Brute-force	Divide & Conquer	Linearity
Aurix	256	1,2,3	1,2	given up	40 hrs	9hrs
Aurix	64	1,2,3,4	1,2,3	given up	25 hrs	7hrs
RiVal	26	1,2	1	1min	---	10sec
Lidar	16	1,2	1	4min	---	1min

- Formall verified using: onespin
- ***Exploiting linearity of syndrome generators leads to property convegence on large ECC designs and reduces the proof runtime by more than 50X***

THANK YOU!

Part of your life. Part of tomorrow.