How To Verify Encoder And Decoder Designs Using Formal Verification

Jin Hou

Mentor-A Siemens Business

Mentor®
A Siemens Business

accellera
SYSTEMS INITIATIVE

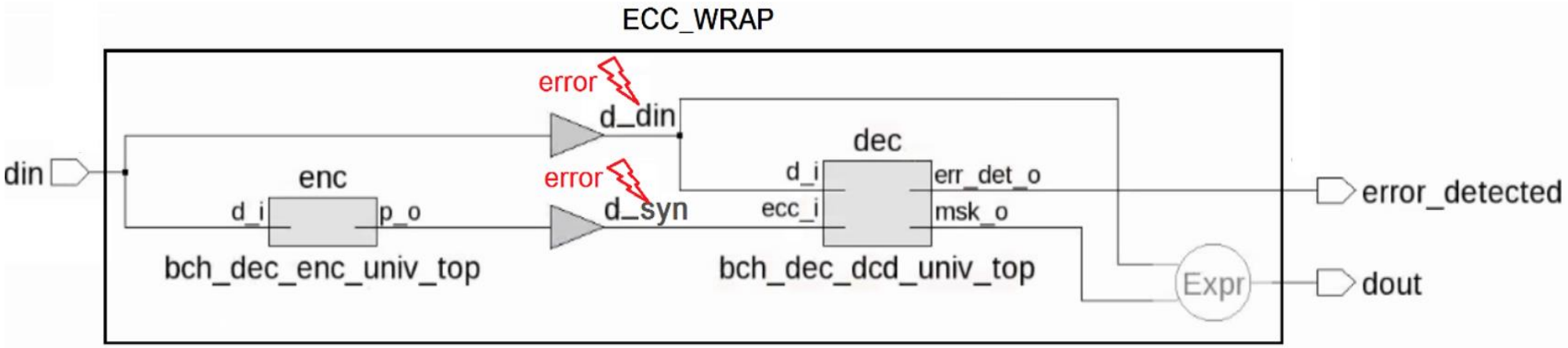
2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Introduction
- How to verify BCH encoder and decoder using property checking
- How to verify BCH encoder and decoder using SLEC (Sequential Logic Equivalence Checking)
- Conclusions

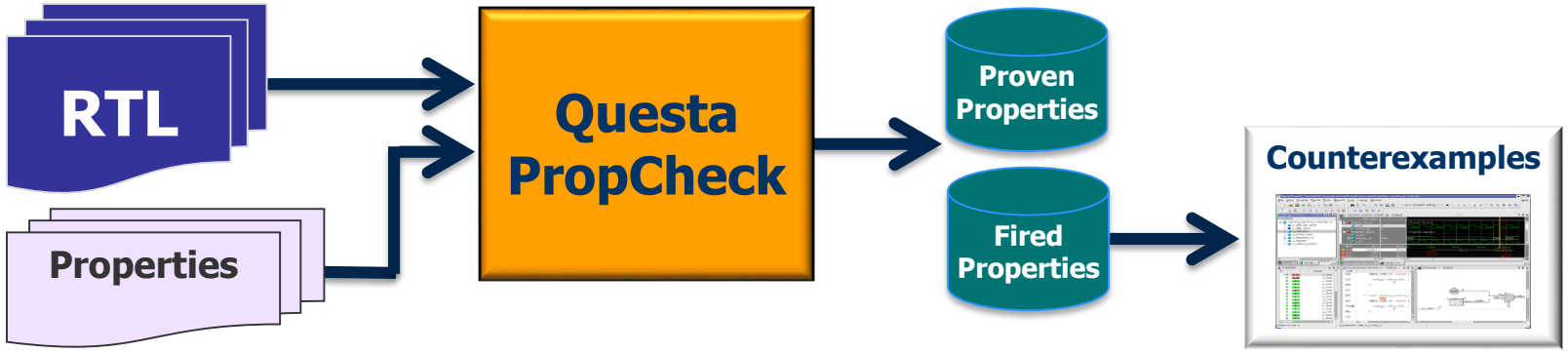
Introduction

- The challenges for fully verifying encoder and decoder designs:
  - How to cover all possible data coming into the encoder
  - How to cover all possible random faults if the design should correct them?
- A case study: BCH encoder and decoder design from Opencore website.
  - The function of BCH: Fix up to 2-bit random faults injected to the transmission lines from the encoder to the decoder.



Verifying BCH Design Using Property Checking

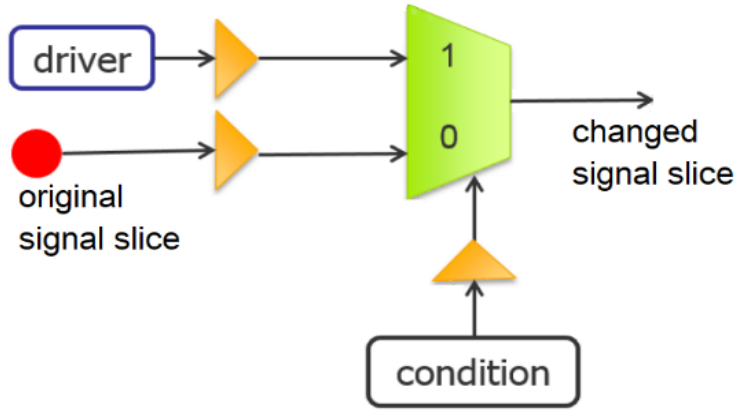
- Property checking flow:



- The functions of BCH
  1. When there is no error, *dout* should be the same as *din* and *error_detected* should be 0.
  2. When 1-bit or 2-bit random errors are happening on the lines to DEC inputs, the errors can be detected, and *error_detected* should be 1.
  3. When 1-bit or 2-bit random errors are happening on the lines to DEC inputs, the errors can be corrected, and *dout* should be the same as *din*.

Inserting Random Faults

- Define an undriven wire *foo*. When a bit of *foo* is 1, Questa PropCheck directive *netlist cutpoint* inserts an error to one data transmission line.
- Formal verification automatically considers all possible values for the undriven wire *foo*.



insert_errors.do

```
set DATA_WIDTH 16
set ECC_WIDTH 10
set WIDTH [expr $DATA_WIDTH + $ECC_WIDTH]
for {set i 0} {$i < $WIDTH } {incr i} {
  if {$i < $DATA_WIDTH} {
    netlist cutpoint d_din\[$i\] -cond (foo\[$i\]) -driver ~din\[$i\] }
  else {
    netlist cutpoint d_syn\[[expr {$i-$DATA_WIDTH}]\] -cond (foo\[$i\])\
      -driver ~e_syn\[[expr {$i-$DATA_WIDTH}]\] }}
```

Inject error

Inject error

Writing Properties In SVA

- Translate the properties to executable SVA assertions.
  - *foo* controls the number of faults injected to the transmission lines between the encoder and the decoder

```
check_no_error: assert property (@($global_clock)
      foo==0 |-> ~error_detected);
check_error_detection: assert property (@($global_clock)
      ($countones(foo)==1 || $countones(foo)==2) |-> error_detected);
check_error_correction: assert property (@($global_clock)
      $countones(foo)<=2 |-> dout==din);
```

Verifying BCH Design With Questa PropCheck

- The script to run Questa PropCheck is as follows. The *insert_errors.do* file inserts random errors.

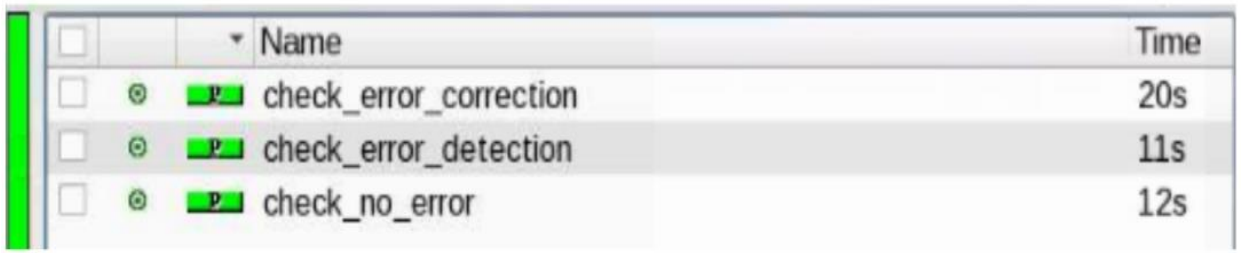
```
vlog -sv -f qft_files/flist.vl
qverify -c -od log -do " \
do insert_errors.do; \
formal compile -d ecc_wrap;\
formal verify; \
exit"
```

Compile RTL source files.

Read in the tcl file for inserting errors

Build formal model, and run formal analysis

- The verification result:



Sanity Waveforms

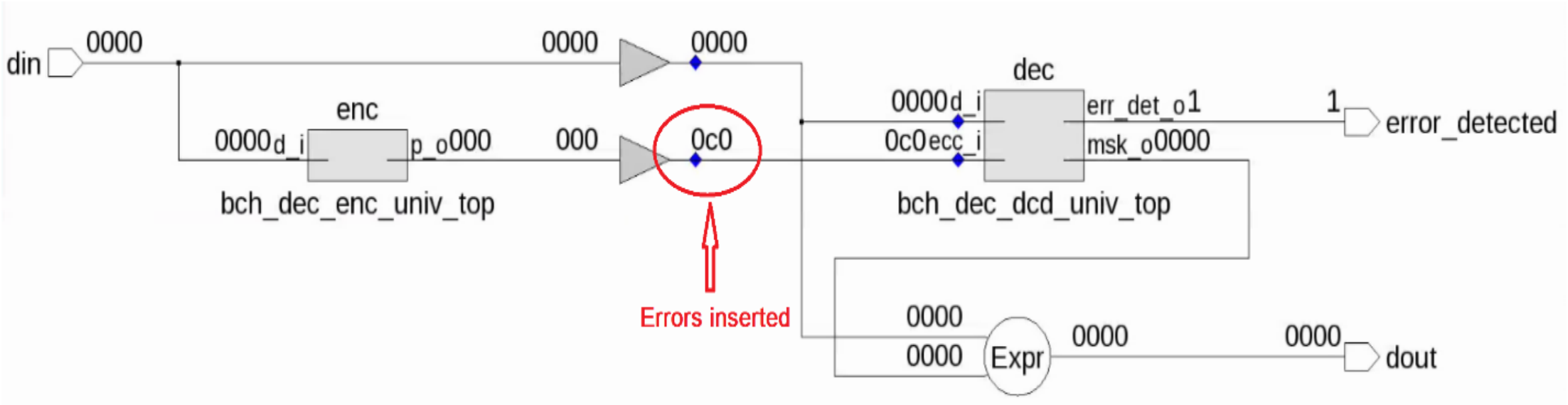
- The sanity waveforms of the property *check_error_detection*

ecc_wrap_sva.v

```
11
12      assign d_din = din;
                0       0
13      assign d_syn = e_syn;
                c0      0
14
15      bch_dec_enc_univ_top #(.P_D_WIDTH(DATA_WIDTH))
16      enc
17      (
18      .d_i(din),
              0
19      .p_o(e_syn) // width for 16 is 10
              0
20      );
21
22
23      bch_dec_dcd_univ_top #(.P_D_WIDTH(DATA_WIDTH))
24      dec
25      (
26      .d_i(d_din),
              0
27      .ecc_i(d_syn),
                c0
28      .msk_o(mask),
                0
29      .err_det_o(error_detected)
                   1
30      );
31
32
33      assign  dout = d_din ^ mask;
                0       0       0
34
```

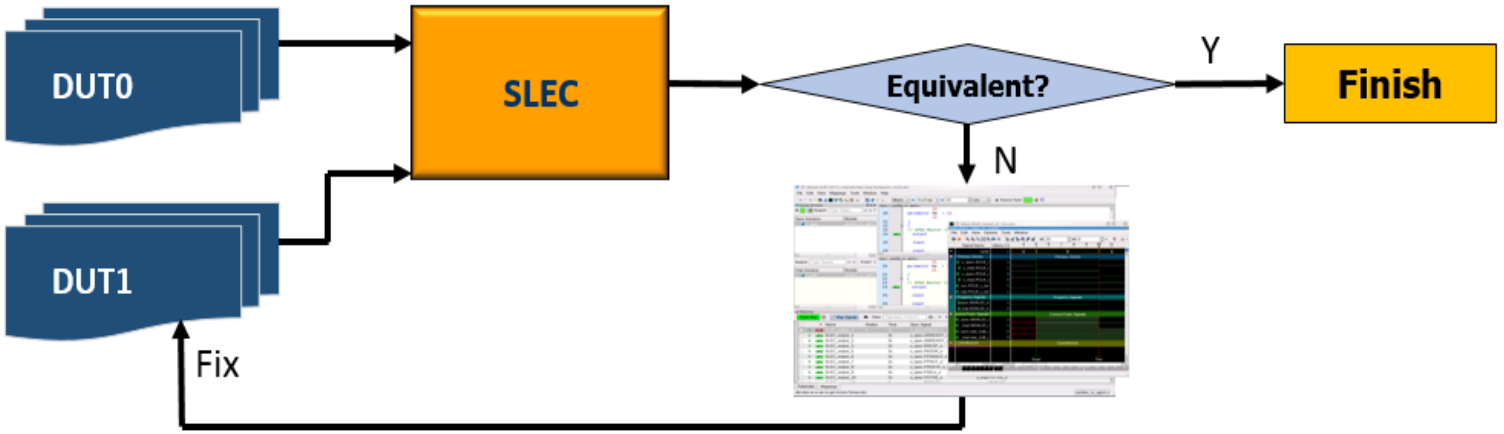
The Schematic View

- The Schematic View shows the inserted faults.



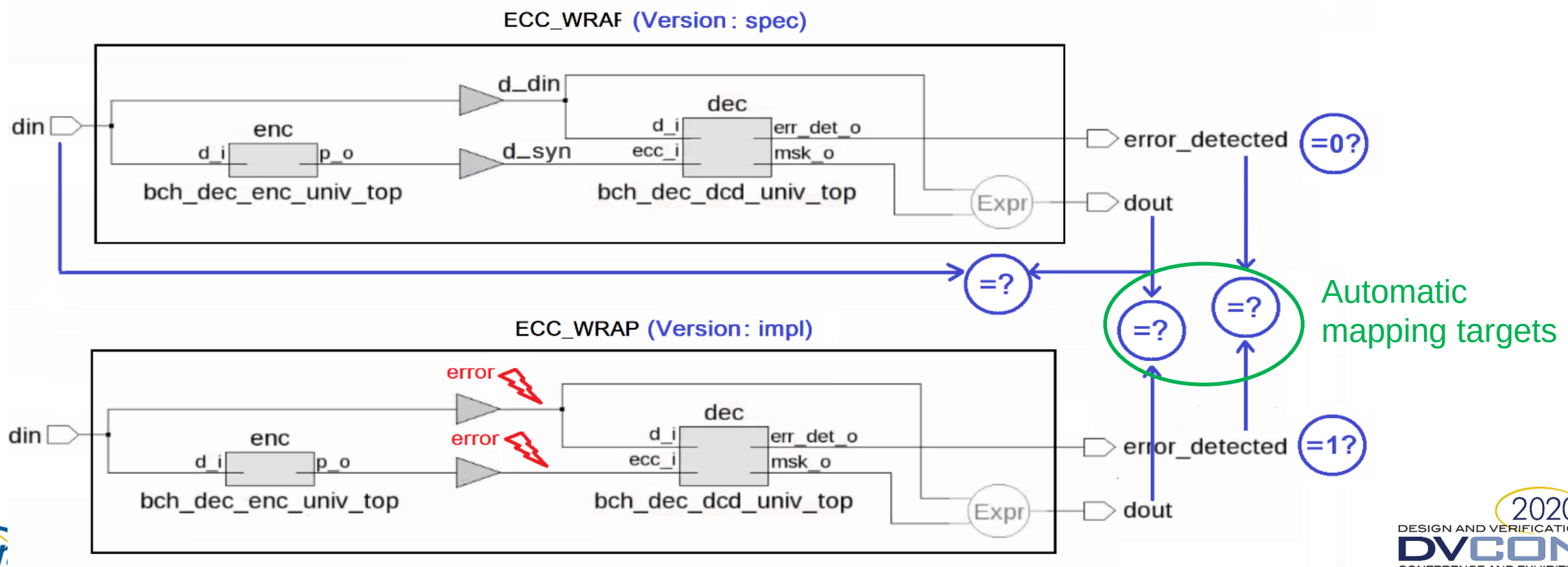
The Flow Of SLEC

- The flow of SLEC (Sequential Logic Equivalent Checking)
  - The SLEC tool automatically maps the inputs of DUT0 and DUT1 and ties them together
  - The SLEC tool automatically maps the outputs of DUT0 and DUT1 and verifies their equivalency.



Verifying BCH Design Using SLEC

- The design version *spec* is the BCH design without faults
- The design version *impl* is the BCH design with random faults



Verifying BCH Design With Questa SLEC

- The script to run Questa SLEC to verify BCH design:

```
vlog -sv -f qft_files/flist.vl
qverify -c -od log_slec -do "\
slec configure -spec -d ecc_wrap;\
slec configure -impl -d ecc_wrap;\
netlist wire const_0 -width 1 -module ecc_wrap -driver 1'b0;\
netlist wire const_1 -width 1 -module ecc_wrap -driver 1'b1;\
do qft_files/inject-errors.do;\
slec map {spec.din impl.dout} -target;\
slec map {spec.const_1 impl.error_detected} -target;\
slec map {spec.error_detected impl.const_0} -target;\
slec compile;\
slec verify;\
exit"
```

Compile RTL source files

Configure spec and impl versions

Define constant signals

Read in the tcl file for inserting errors

Define equivalence targets

Build formal model, create SLEC targets, and run formal analysis

Inserting Random Faults

- Inject-errors.do file:

```
set DATA_WIDTH 16
set ECC_WIDTH 10
set WIDTH [expr $DATA_WIDTH + $ECC_WIDTH]

## Define undriven wire foo
netlist wire foo -width $WIDTH

## Insert random errors to {d_syn, d_din}
for {set i 0} {$i < $WIDTH} {incr i} {
  if {$i < $DATA_WIDTH} {
    netlist cutpoint impl.d_din\[$i\] -cond (foo\[$i\]) -driver ~spec.d_din\[$i\]
  } else {
    netlist cutpoint impl.d_syn\[[expr {$i-$DATA_WIDTH}]\] -cond (foo\[$i\]) \
             -driver ~spec.d_syn\[[expr {$i-$DATA_WIDTH}]\]
  }
}

## Constrain foo: inject 1 or 2 bit errors
netlist property -name assume_foo -assume {$countones(foo)==1 || $countones(foo)==2}
```

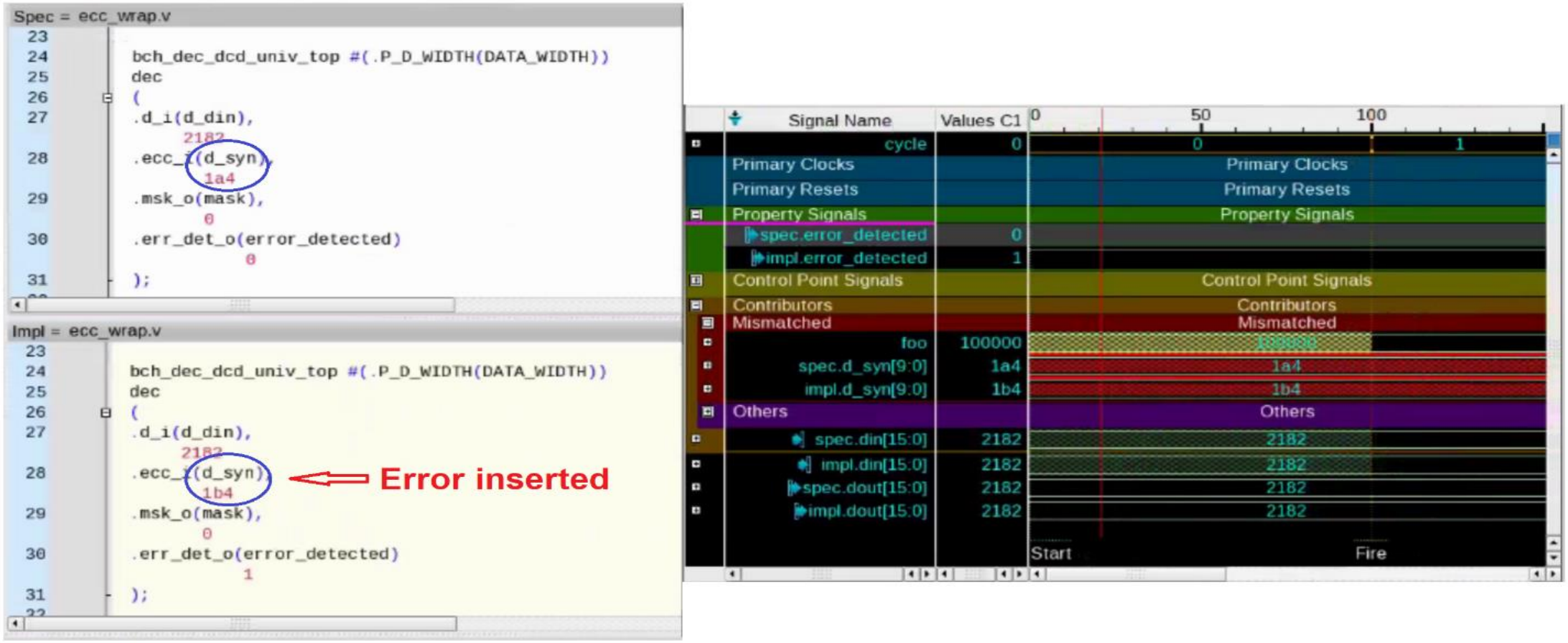
The Verification Results With Questa SLEC

- The verification results:

	Name	Radiu	Time		Spec Signal	Impl Signal
	SLEC_output_2	1	0s		spec.error_detected	impl.error_detected
	SLEC_output_1		7s		spec.dout	impl.dout
	SLEC_target_1		7s		spec.din	impl.dout
	SLEC_target_2		6s		spec.const_1	impl.error_detected
	SLEC_target_3		1s		spec.error_detected	impl.const_0
✓	SLEC_input_1				spec.clk	impl.clk
✓	SLEC_input_2				spec.din	impl.din
✓	SLEC_input_3				spec.rstn	impl.rstn

Counterexample Of Non-equivalence

- The counterexample of the target {*spec.error_detected impl.error_detected*}



Conclusions

- Both property checking and SLEC can exhaustively verify BCH encoder and decoder design
  - Property checking verifies assertions against the design.
  - SLEC verifies the equivalency between the versions with/without injected faults.
  - Formal algorithms consider all possible inputs and random faults.
    - Simulation is lacking of this capability, and cannot exhaustively prove design functions.
- Setup is easy and fast for both formal methods
  - No lengthy simulation testbenches.
- Formal verification methods property checking and SLEC can be applied to other encoder and decoder designs.

Questions