# Formal Verification of Automatic Circuit Transformations for Fault-Tolerance

Dmitry Burlyaev
Pascal Fradet

## Outline

For a given (fault-tolerance) transformation $\mathcal{T}$, we want to prove a property of the form

$$\forall C : circuit,\ \forall i : inputs,\ \forall o : outputs,$$
$$C\ i \longrightarrow o \ \ \Rightarrow\ \ \mathcal{T}[\![C]\!]\ \overline{i} \xrightarrow{\text{faulty}} \overline{o}$$

## Outline

For a given (fault-tolerance) transformation $\mathcal{T}$, we want to prove a property of the form

$$\forall C : circuit,\ \forall i : inputs,\ \forall o : outputs,$$
$$C\ i \longrightarrow o \;\Rightarrow\; \mathcal{T}[\![C]\!]\ \bar{i} \xrightarrow{\text{faulty}} \bar{o}$$

- Syntax of circuits

## Outline

For a given (fault-tolerance) transformation $\mathcal{T}$, we want to prove a property of the form

$$\forall C : \mathit{circuit},\ \forall i : \mathit{inputs},\ \forall o : \mathit{outputs},$$
$$C\ i \longrightarrow o \quad \Rightarrow \quad \mathcal{T}[\![C]\!]\ \bar{i} \xrightarrow{\text{faulty}} \bar{o}$$

- Syntax of circuits
- Circuit transformations on syntax

## Outline

For a given (fault-tolerance) transformation $\mathcal{T}$, we want to prove a property of the form

$$\forall C : circuit,\ \forall i : inputs,\ \forall o : outputs,$$
$$C\ i \longrightarrow o \quad \Rightarrow \quad \mathcal{T}[\![C]\!]\ \bar{i} \xrightarrow{\text{faulty}} \bar{o}$$

- Syntax of circuits
- Circuit transformations on syntax
- Semantics of circuits

## Outline

For a given (fault-tolerance) transformation $\mathcal{T}$, we want to prove a property of the form

$$\begin{aligned} &\forall C : \mathit{circuit},\ \forall i : \mathit{inputs},\ \forall o : \mathit{outputs}, \\ &\qquad C\ i \longrightarrow o \quad \Rightarrow \quad \mathcal{T}[\![C]\!]\ \bar{i} \xrightarrow{\text{faulty}} \bar{o} \end{aligned}$$

- Syntax of circuits
- Circuit transformations on syntax
- Semantics of circuits
- Fault-models described in semantics:
  bit-flip (SEU), glitch (SET), ...

## Outline

For a given (fault-tolerance) transformation $\mathcal{T}$, we want to prove a property of the form

$$\forall C : circuit,\ \forall i : inputs,\ \forall o : outputs,$$
$$C\ i \longrightarrow o \quad \Rightarrow \quad \mathcal{T}[\![C]\!]\ \bar{i} \xrightarrow{\text{faulty}} \bar{o}$$

- Syntax of circuits
- Circuit transformations on syntax
- Semantics of circuits
- Fault-models described in semantics:
  bit-flip (SEU), glitch (SET), ...
- Case study: our fault-tolerance solution
  required full confidence

## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

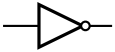
$$\begin{array}{lcll} \mathsf{Gate} & ::= & \textsc{not} \mid \textsc{and} \mid \textsc{or} & \textit{logic} \\ & & \textsc{id} \mid \textsc{swap} \mid \textsc{fork} \mid \textsc{rsh} \mid \textsc{lsh} & \textit{wiring} \end{array}$$

## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, …)

| | | | |
|---|---|---|---|
| Gate | ::= | NOT \| AND \| OR | *logic* |
| | | ID \| SWAP \| FORK \| RSH \| LSH | *wiring* |

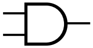


## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, …)

| | | | |
|---|---|---|---|
| Gate | ::= | NOT \| AND \| OR | *logic* |
| | | ID \| SWAP \| FORK \| RSH \| LSH | *wiring* |

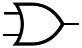


## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

| | | | |
|---|---|---|---|
| Gate | ::= | NOT \| AND \| OR | *logic* |
| | | ID \| SWAP \| FORK \| RSH \| LSH | *wiring* |

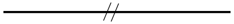


# LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, . . . )

| | | | |
|---|---|---|---|
| Gate | ::= | NOT \| AND \| OR | *logic* |
| | | ID \| SWAP \| FORK \| RSH \| LSH | *wiring* |

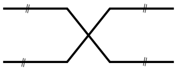


## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

$$\begin{array}{lllr} \textsf{Gate} & ::= & \textsc{not} \mid \textsc{and} \mid \textsc{or} & \textit{logic} \\ & & \textsc{id} \mid \textsc{swap} \mid \textsc{fork} \mid \textsc{rsh} \mid \textsc{lsh} & \textit{wiring} \end{array}$$

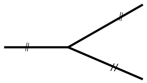


## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

$$\mathsf{Gate} \quad ::= \quad \text{NOT} \mid \text{AND} \mid \text{OR} \qquad \textit{logic}$$
$$\qquad\qquad \text{ID} \mid \text{SWAP} \mid \text{FORK} \mid \text{RSH} \mid \text{LSH} \qquad \textit{wiring}$$

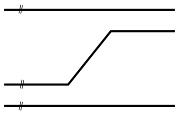


## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

| | | | |
|---|---|---|---|
| Gate | ::= | NOT \| AND \| OR | *logic* |
| | | ID \| SWAP \| FORK \| RSH \| LSH | *wiring* |

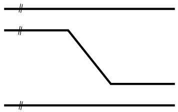


## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

| | | | |
|---|---|---|---|
| Gate | ::= | NOT \| AND \| OR | *logic* |
| | | ID \| SWAP \| FORK \| RSH \| LSH | *wiring* |



## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

$$\begin{array}{lcll} \mathsf{Gate} & ::= & \textsc{not} \mid \textsc{and} \mid \textsc{or} & \textit{logic} \\ & & \textsc{id} \mid \textsc{swap} \mid \textsc{fork} \mid \textsc{rsh} \mid \textsc{lsh} & \textit{wiring} \end{array}$$

$$C ::= \mathsf{Gate} \mid C1 \multimap C2 \mid [\![C1, C2]\!] \mid \boxed{b}\!\!-\!C$$

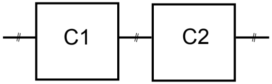
## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, …)

$$
\begin{array}{lcll}
\mathsf{Gate} & ::= & \text{NOT} \mid \text{AND} \mid \text{OR} & \textit{logic} \\
 & & \text{ID} \mid \text{SWAP} \mid \text{FORK} \mid \text{RSH} \mid \text{LSH} & \textit{wiring}
\end{array}
$$

$$C ::= \mathsf{Gate} \quad \mid \quad C1 \multimap C2 \quad \mid \quad [\![ C1, C2 ]\!] \quad \mid \quad \boxed{b}\!-\!C$$



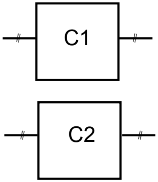
## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

$$\begin{array}{lllr} \mathsf{Gate} & ::= & \text{NOT} \mid \text{AND} \mid \text{OR} & \textit{logic} \\ & & \text{ID} \mid \text{SWAP} \mid \text{FORK} \mid \text{RSH} \mid \text{LSH} & \textit{wiring} \end{array}$$

$$C ::= \mathsf{Gate} \quad \mid \quad C1 \multimap C2 \quad \mid \quad [\![ C1, C2 ]\!] \quad \mid \quad \boxed{b}\!-\!C$$



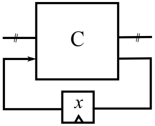
## LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

$$\begin{array}{lll} \mathsf{Gate} ::= & \text{NOT} \mid \text{AND} \mid \text{OR} & \textit{logic} \\ & \text{ID} \mid \text{SWAP} \mid \text{FORK} \mid \text{RSH} \mid \text{LSH} & \textit{wiring} \end{array}$$

$$C ::= \mathsf{Gate} \mid C1 \multimap C2 \mid [\![C1, C2]\!] \mid \boxed{b}\!-\!C$$



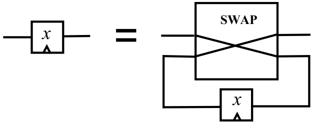
# LDDL- language to describe circuits

- Gate level HDL
- as simple as possible

A combinator language (inspired from Sheeran's $\mu$FP, Ruby, ...)

$$\begin{array}{lcll} \mathsf{Gate} & ::= & \text{NOT} \mid \text{AND} \mid \text{OR} & \textit{logic} \\ & & \text{ID} \mid \text{SWAP} \mid \text{FORK} \mid \text{RSH} \mid \text{LSH} & \textit{wiring} \end{array}$$

$$C ::= \mathsf{Gate} \quad \mid \quad C1 \multimap C2 \quad \mid \quad [\![C1, C2]\!] \quad \mid \quad \boxed{b}\!-\!C$$



$\boxed{x}\!-\!(\text{SWAP})$

## LDDL types

**Bus**

$$B := \omega \mid (B_1 * B_2)$$

**Gates**

$$\text{NOT} \;:\; \mathsf{Gate}\ \omega\ \omega \qquad\qquad \text{AND, OR} \;:\; \mathsf{Gate}\ (\omega * \omega)\ \omega$$

**Plugs**

$$\begin{array}{lll} \dots & & \\ \text{SWAP} & : & \forall \alpha\ \beta, \mathsf{Plug}\ (\alpha * \beta)\ (\beta * \alpha) \\ \dots & & \end{array}$$

## LDDL types

**Circuits**

$$
\begin{array}{llcl}
C ::= & & & \\
 & \dots & & \\
 & \mid\ C_1 \multimap C_2 & : & \forall \alpha\ \beta\ \gamma, \mathsf{Circ}\ \alpha\ \beta \to \mathsf{Circ}\ \beta\ \gamma \\
 & & & \quad \to \mathsf{Circ}\ \alpha\ \gamma \\
 & \dots & & \\
 & \mid\ [\![C_1, C_2]\!] & : & \forall \alpha\ \beta\ \gamma\ \delta, \mathsf{Circ}\ \alpha\ \gamma \to \mathsf{Circ}\ \beta\ \delta \\
 & & & \quad \to \mathsf{Circ}\ (\alpha * \beta)\ (\gamma * \delta) \\
 & \dots & &
\end{array}
$$

## Language feature summary

- Correct circuits by construction
  - correctly connected (typing)
  - all loops contain a cell (Loop operator)

## Language feature summary

- Correct circuits by construction
  - correctly connected (typing)
  - all loops contain a cell (Loop operator)
- No variables
  - Simpler semantics (no environment)

## Language feature summary

- Correct circuits by construction
  - correctly connected (typing)
  - all loops contain a cell (Loop operator)
- No variables
  - Simpler semantics (no environment)
- We represent the state (FF values) by circuit itself
  - $e.g.,\ (\boxed{false}\!-\!\textsc{swap})\ true\ \rightarrow\ (\boxed{true}\!-\!\textsc{swap})$

## LDDL semantics of a clock cycle w/o fault

A predicate: $\mathsf{step}\ C\ a\ b\ C'$

$C$ - an original circuit; $a$ - an input
$b$ - an output; $C'$ - resulting state after a cycle

$$\text{Gates \& Plugs}\ \frac{[\![G]\!]a = b}{\mathsf{step}\ G\ a\ b\ G}$$

$$\text{Seq}\ \frac{\mathsf{step}\ C_1\ a\ b\ C_1' \qquad \mathsf{step}\ C_2\ b\ c\ C_2'}{\mathsf{step}\ (C_1 \multimap C_2)\ a\ c\ (C_1' \multimap C_2')}$$

$$\text{Par}\ \frac{\mathsf{step}\ C_1\ a\ c\ C_1' \qquad \mathsf{step}\ C_2\ b\ d\ C_2'}{\mathsf{step}\ [\![C_1, C_2]\!]\ (a,b)\ (c,d)\ [\![C_1', C_2']\!]}$$

$$\text{Loop}\ \frac{\mathsf{step}\ C\ (a, \mathsf{b2s}\ x)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{step}\ \boxed{x}\!\!-\!C\ a\ b\ \boxed{y}\!\!-\!C'}$$

## LDDL semantics of a clock cycle w/o fault

A predicate: $\mathsf{step}\ C\ a\ b\ C'$

$C$ - an original circuit; $a$ - an input
$b$ - an output; $C'$ - resulting state after a cycle

$$\text{Gates \& Plugs}\ \frac{[\![G]\!]a = b}{\mathsf{step}\ G\ a\ b\ G}$$

$$\text{Seq}\ \frac{\mathsf{step}\ C_1\ a\ b\ C_1' \qquad \mathsf{step}\ C_2\ b\ c\ C_2'}{\mathsf{step}\ (C_1 \mathbin{-\!\circ\!-} C_2)\ a\ c\ (C_1' \mathbin{-\!\circ\!-} C_2')}$$

$$\text{Par}\ \frac{\mathsf{step}\ C_1\ a\ c\ C_1' \qquad \mathsf{step}\ C_2\ b\ d\ C_2'}{\mathsf{step}\ [\![C_1, C_2]\!]\ (a, b)\ (c, d)\ [\![C_1', C_2']\!]}$$

$$\text{Loop}\ \frac{\mathsf{step}\ C\ (a, \mathsf{b2s}\ x)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{step}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

## LDDL semantics of a clock cycle w/o fault

A predicate: $\mathsf{step}\ C\ a\ b\ C'$

$C$ - an original circuit; $a$ - an input
$b$ - an output; $C'$ - resulting state after a cycle

$$\text{Gates \& Plugs}\ \frac{[\![G]\!]a = b}{\mathsf{step}\ G\ a\ b\ G}$$

$$\text{Seq}\ \frac{\mathsf{step}\ C_1\ a\ b\ C_1' \qquad \mathsf{step}\ C_2\ b\ c\ C_2'}{\mathsf{step}\ (C_1 \multimap C_2)\ a\ c\ (C_1' \multimap C_2')}$$

$$\text{Par}\ \frac{\mathsf{step}\ C_1\ a\ c\ C_1' \qquad \mathsf{step}\ C_2\ b\ d\ C_2'}{\mathsf{step}\ [\![C_1, C_2]\!]\ (a,b)\ (c,d)\ [\![C_1', C_2']\!]}$$

$$\text{Loop}\ \frac{\mathsf{step}\ C\ (a, \mathsf{b2s}\ x)\ (b,s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{step}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

## Evaluation of a circuit w/o faults

As a predicate from Stream to Stream
$\mathsf{eval} : \mathsf{Circ}\ \alpha\ \beta \to \mathsf{Stream}\ \alpha \to \mathsf{Stream}\ \beta$

$$\text{Eval}\ \frac{\mathsf{step}\ C\ i\ o\ C' \qquad \mathsf{eval}\ C'\ is\ os}{\mathsf{eval}\ C\ (i : is)\ (o : os)}$$

If $C$ applied to input $i \to$ output $o$ and $C'$
and if $C'$ applied to infinite stream $is \to$ stream $os$
$\Rightarrow$ evaluation of $C$ with stream $(i : is) \to$ stream $(o : os)$.

## LDDL semantics of a cycle with a fault

$SET(1, K)$::"at most 1 glitch within K clock cycles"

$$Signal := 0 \mid 1 \mid ϟ$$

- Evaluation with glitches is non deterministic
  - not deterministically latched (as *true* or *false*) by cells
  - can be be logically masked (*e.g.*, $\text{AND}(0, ϟ) = 0, \ldots$)

A predicate: stepg $C\ a\ b\ C'$

$C$ - an original circuit; $a$ - an input
$b$ - an output; $C'$ - possibly corrupted state after a cycle with a glitch at any wire

# LDDL semantics of a cycle with a fault

$$\text{Gates}\ \frac{}{\mathsf{stepg}\ G\ a\ \lightning\ G}$$

$$\text{SeqL}\ \frac{\mathsf{stepg}\ C_1\ a\ b\ C_1' \qquad \mathsf{step}\ C_2\ b\ c\ C_2'}{\mathsf{stepg}\ (C_1 \multimap C_2)\ a\ c\ (C_1' \multimap C_2')}$$

$$\text{SeqR}\ \frac{\mathsf{step}\ C_1\ a\ b\ C_1' \qquad \mathsf{stepg}\ C_2\ b\ c\ C_2'}{\mathsf{stepg}\ (C_1 \multimap C_2)\ a\ c\ (C_1' \multimap C_2')}$$

$$\text{LoopC}\ \frac{\mathsf{stepg}\ C\ (a, b2s\ x)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{stepg}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

$$\text{LoopM}\ \frac{\mathsf{step}\ C\ (a, \lightning)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{stepg}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

# LDDL semantics of a cycle with a fault

$$\text{Gates}\ \frac{}{\mathsf{stepg}\ G\ a\ \lightning\ G}$$

$$\text{SeqL}\ \frac{\mathsf{stepg}\ C_1\ a\ b\ C_1' \qquad \mathsf{step}\ C_2\ b\ c\ C_2'}{\mathsf{stepg}\ (C_1 \multimap C_2)\ a\ c\ (C_1' \multimap C_2')}$$

$$\text{SeqR}\ \frac{\mathsf{step}\ C_1\ a\ b\ C_1' \qquad \mathsf{stepg}\ C_2\ b\ c\ C_2'}{\mathsf{stepg}\ (C_1 \multimap C_2)\ a\ c\ (C_1' \multimap C_2')}$$

$$\text{LoopC}\ \frac{\mathsf{stepg}\ C\ (a, b2s\ x)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{stepg}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

$$\text{LoopM}\ \frac{\mathsf{step}\ C\ (a, \lightning)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{stepg}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

# LDDL semantics of a cycle with a fault

$$\text{Gates}\ \frac{}{\mathsf{stepg}\ G\ a\ \lightning\ G}$$

$$\text{SeqL}\ \frac{\mathsf{stepg}\ C_1\ a\ b\ C_1' \qquad \mathsf{step}\ C_2\ b\ c\ C_2'}{\mathsf{stepg}\ (C_1 \mathbin{-\!\circ\!-} C_2)\ a\ c\ (C_1' \mathbin{-\!\circ\!-} C_2')}$$

$$\text{SeqR}\ \frac{\mathsf{step}\ C_1\ a\ b\ C_1' \qquad \mathsf{stepg}\ C_2\ b\ c\ C_2'}{\mathsf{stepg}\ (C_1 \mathbin{-\!\circ\!-} C_2)\ a\ c\ (C_1' \mathbin{-\!\circ\!-} C_2')}$$

$$\text{LoopC}\ \frac{\mathsf{stepg}\ C\ (a, b2s\ x)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{stepg}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

$$\text{LoopM}\ \frac{\mathsf{step}\ C\ (a, \lightning)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{stepg}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

## LDDL semantics of a cycle with a fault

$$\text{Gates}\;\frac{}{\mathsf{stepg}\ G\ a\ \lightning\ G}$$

$$\text{SeqL}\;\frac{\mathsf{stepg}\ C_1\ a\ b\ C_1' \qquad \mathsf{step}\ C_2\ b\ c\ C_2'}{\mathsf{stepg}\ (C_1 \mathbin{-\!\circ\!-} C_2)\ a\ c\ (C_1' \mathbin{-\!\circ\!-} C_2')}$$

$$\text{SeqR}\;\frac{\mathsf{step}\ C_1\ a\ b\ C_1' \qquad \mathsf{stepg}\ C_2\ b\ c\ C_2'}{\mathsf{stepg}\ (C_1 \mathbin{-\!\circ\!-} C_2)\ a\ c\ (C_1' \mathbin{-\!\circ\!-} C_2')}$$

$$\text{LoopC}\;\frac{\mathsf{stepg}\ C\ (a, b2s\ x)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{stepg}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

$$\text{LoopM}\;\frac{\mathsf{step}\ C\ (a, \lightning)\ (b, s)\ C' \qquad \mathsf{s2b}\ s\ y}{\mathsf{stepg}\ \boxed{x}\!-\!C\ a\ b\ \boxed{y}\!-\!C'}$$

## Evaluation along the $SET(1, K)$ fault model

$SET(1, K)$::"at most 1 glitch within K clock cycles"

As a predicate from Stream to Stream with a counter

$$\text{SetG}\ \frac{\mathsf{stepg}\ C\ i\ o\ C' \qquad \mathsf{setk\_eval}\ (K-1)\ C'\ is\ os}{\mathsf{setk\_eval}\ 0\ C\ (i : is)\ (o : os)}$$

$$\text{SetN}\ \frac{\mathsf{step}\ C\ i\ o\ C' \qquad \mathsf{setk\_eval}\ (n-1)\ C'\ is\ os}{\mathsf{setk\_eval}\ n\ C\ (i : is)\ (o : os)}$$

# Evaluation along the $SET(1, K)$ fault model

$SET(1, K)$::"at most 1 glitch within K clock cycles"

As a predicate from Stream to Stream with a counter

$$\text{SetG}\ \frac{\textsf{stepg}\ C\ i\ o\ C' \qquad \textsf{setk\_eval}\ (K-1)\ C'\ is\ os}{\textsf{setk\_eval}\ 0\ C\ (i : is)\ (o : os)}$$

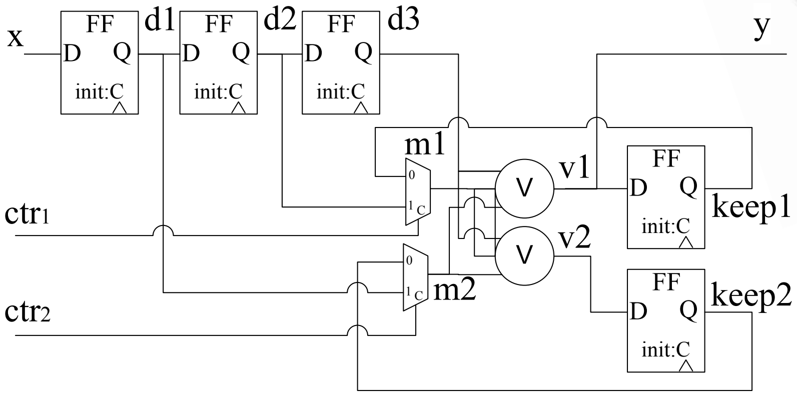
$$\text{SetN}\ \frac{\textsf{step}\ C\ i\ o\ C' \qquad \textsf{setk\_eval}\ (n-1)\ C'\ is\ os}{\textsf{setk\_eval}\ n\ C\ (i : is)\ (o : os)}$$

## Applying the framework to

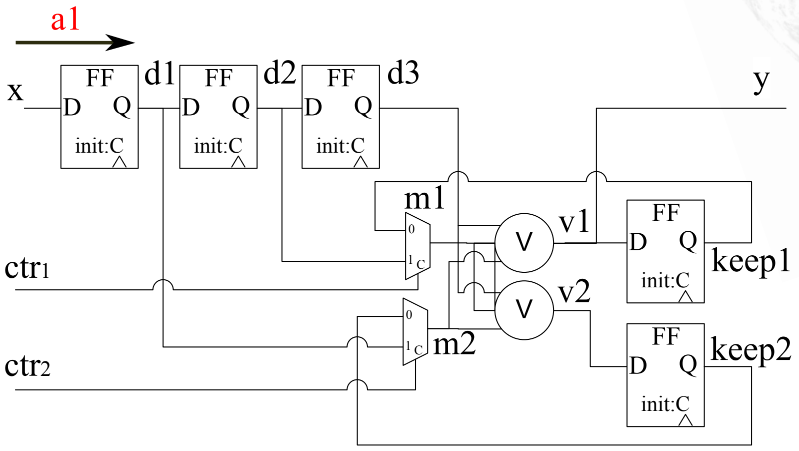
## Double Time Redundancy (DTR) Circuit Transformation*

*in FPGA'15

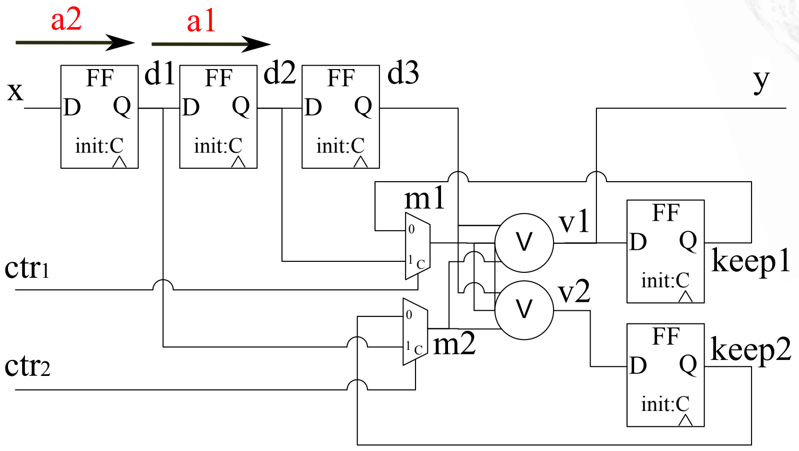
## Triple-Time Redundancy



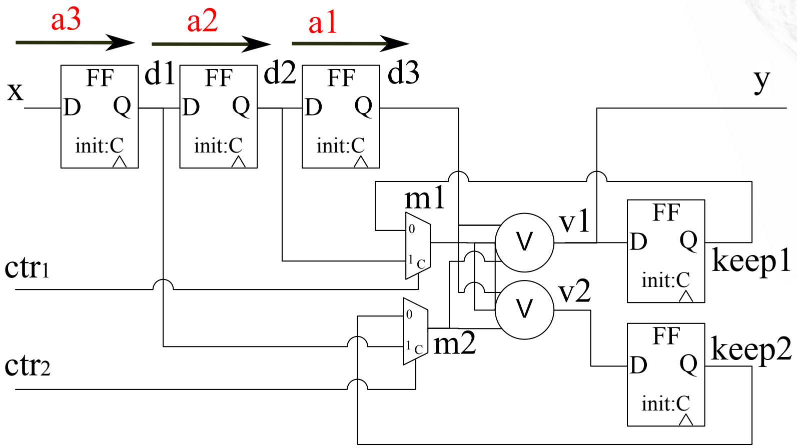
# Triple-Time Redundancy



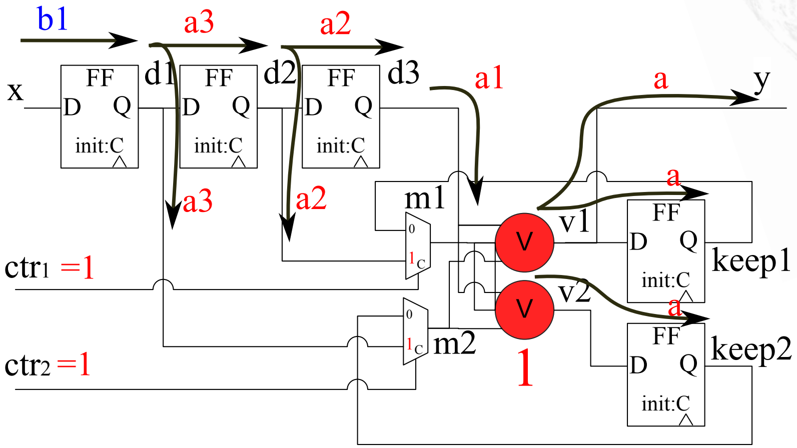
# Triple-Time Redundancy



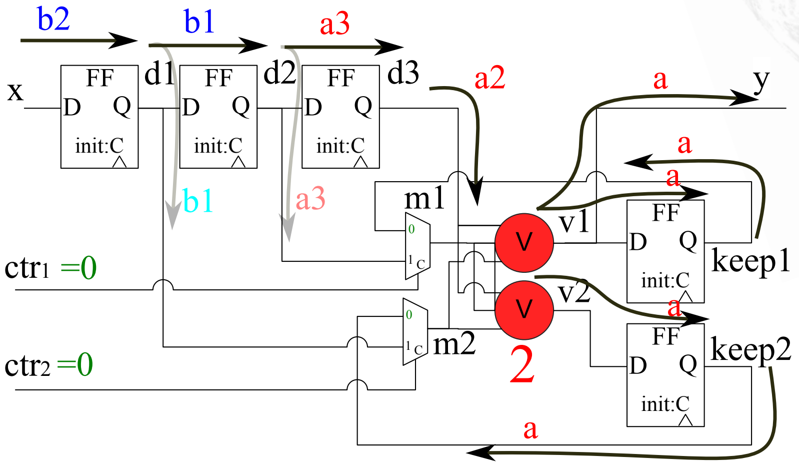
# Triple-Time Redundancy



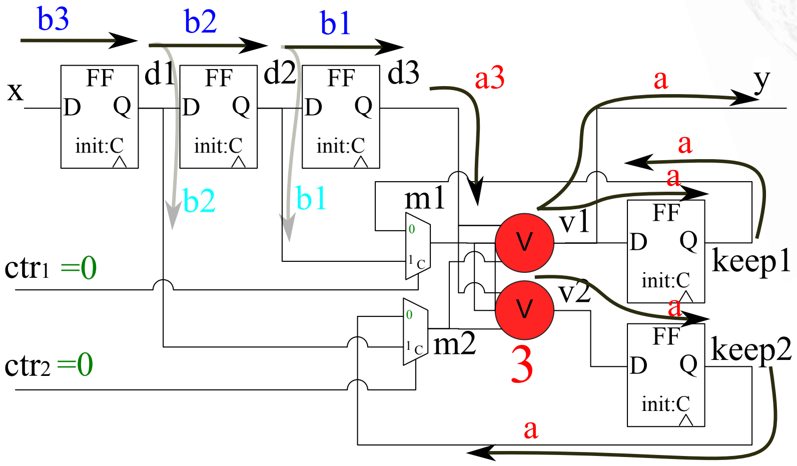
# Triple-Time Redundancy



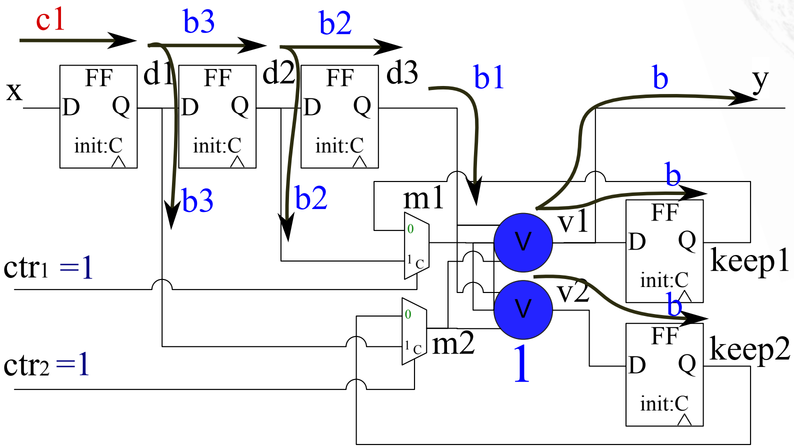
# Triple-Time Redundancy



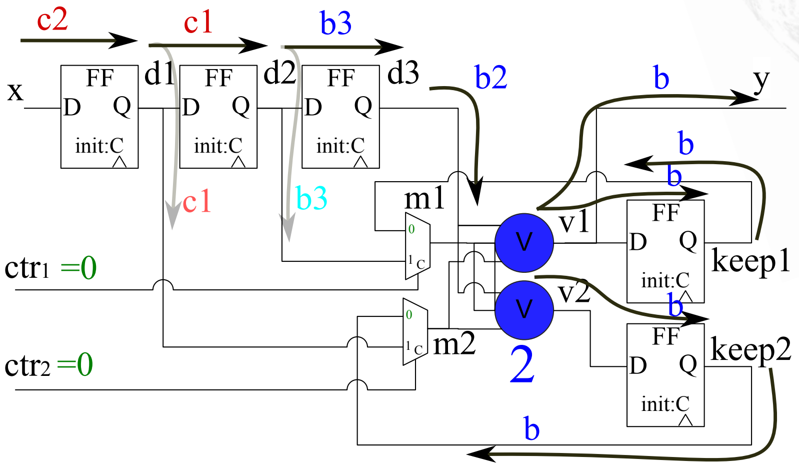
# Triple-Time Redundancy



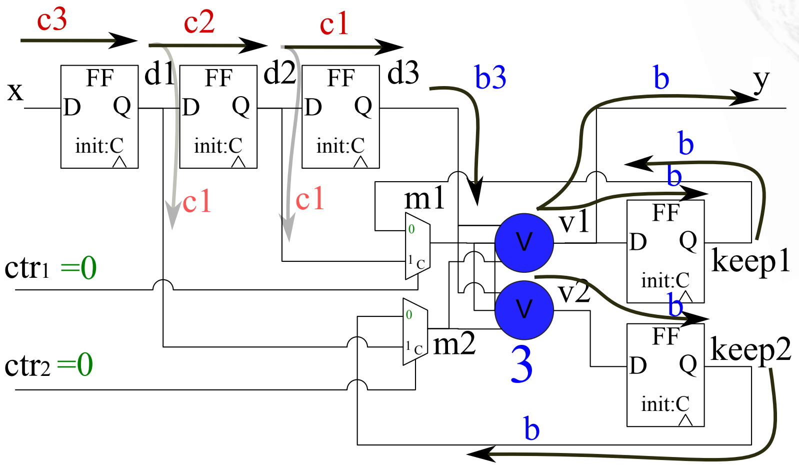
## Triple-Time Redundancy



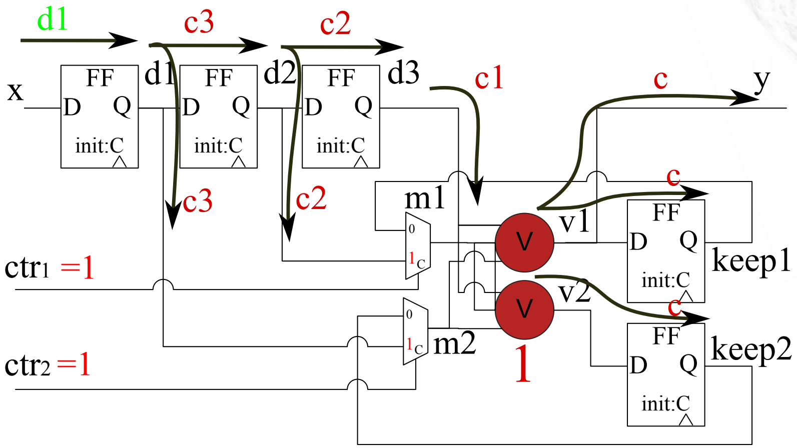
# Triple-Time Redundancy



# Triple-Time Redundancy



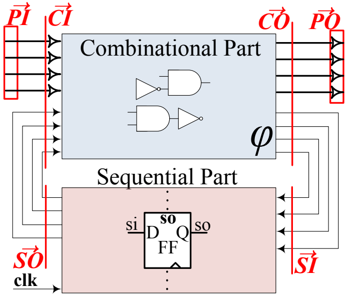
# Triple-Time Redundancy



# Double Time Redundancy Transformation

- only double-time redundancy for error detection
- micro checkpointing-rollback
- speed-up mode (switching-off time-redundancy)
- input/output buffers (input/output transparency)
- tolerance to **at most one SET in 10 clock cycles**
- **1.9-2.5** smaller than TMR
  (with double throughput loss)

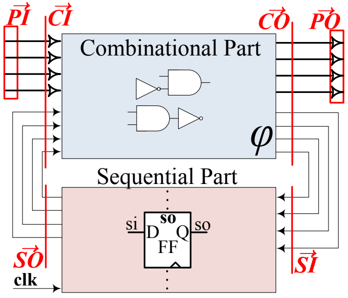
## Transformation DTR



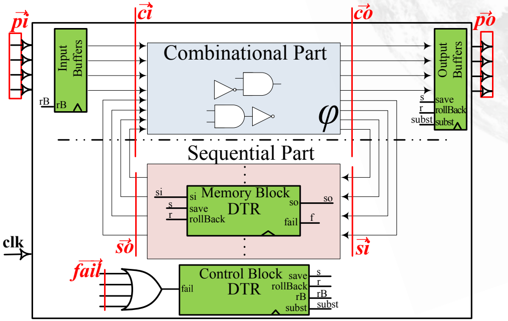
**Original circuit**

1) Memory Cell ← Memory Block
2) Control Block Introduction
3) Input stream upsampling **x2**
4) Input/Output Buffers Insertion

## Transformation DTR



Original circuit

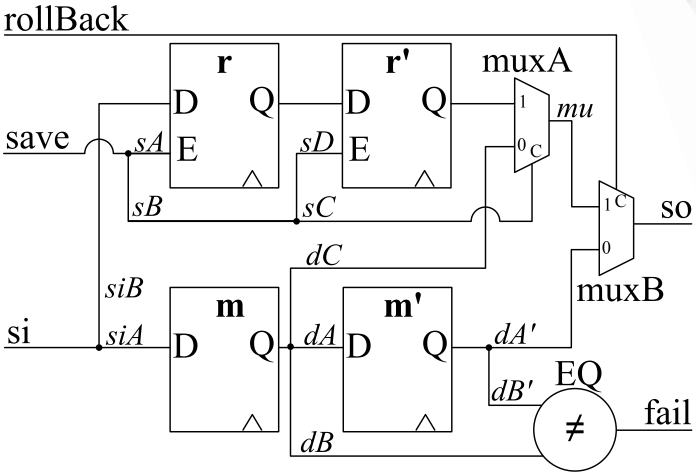


Transformed DTR circuit

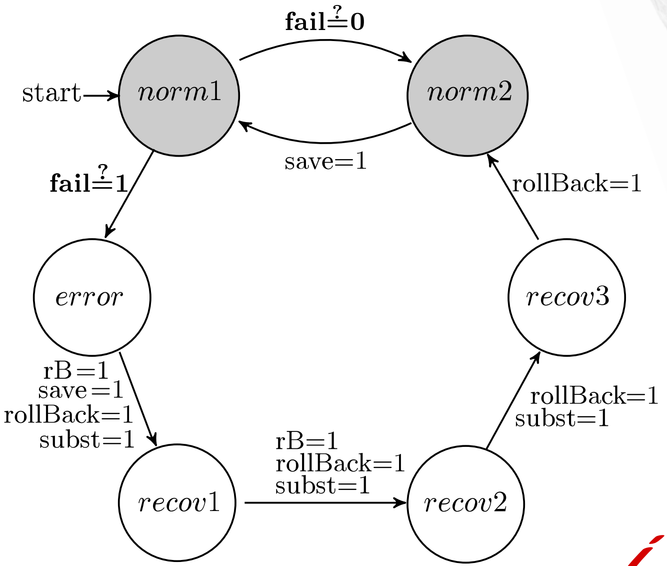
1) Memory Cell ← Memory Block
2) Control Block Introduction
3) Input stream upsampling **x2**
4) Input/Output Buffers Insertion

[TO PROVE]: output correctness with $SET(1, 10)$

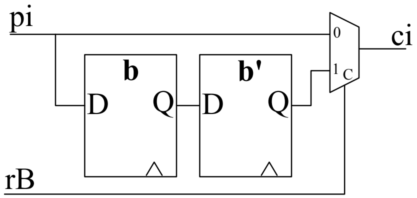
# Memory Block: Working Cycle



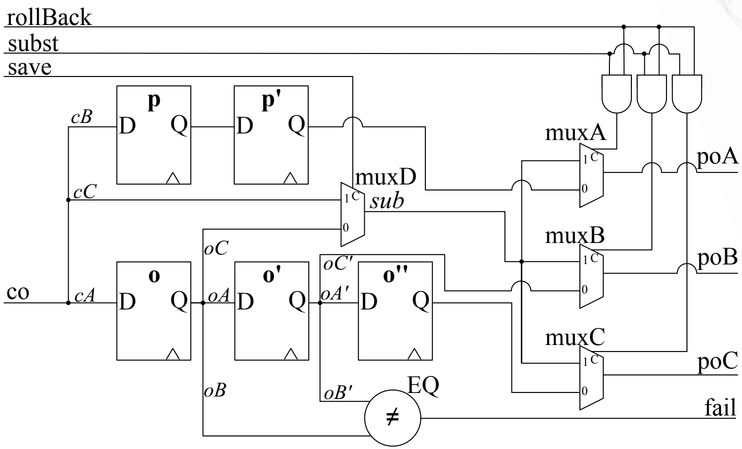
## Control Block protected by TMR



## Input Buffer



# Output Buffer



## Main theorem for DTR

DTR transformation is expressed
on LDDL syntax as $\textsc{dtr}(C)$

For any glitch at any wire,
the I/O behavior stays the same & correct

$$\begin{array}{ll} & \mathsf{eval}\ C_0\ i\ o\ \wedge\ \mathsf{set10\_eval}\ \textsc{dtr}(C_0)\ (\mathsf{upsampl}\ i)\ oo \\ \Rightarrow & \mathsf{outDTR}\ o\ oo \end{array}$$

- upsampl:: DTR input stream is the original stream $i$ with twice repeated bits
- outDTR:: correctness property of DTR outputs

## General Proof Strategy - w/o faults

$$\begin{tikzcd}
 & \textcircled{1} & & \textcircled{2} & & \textcircled{3} & & \textcircled{4} & & \cdots & \\
\cdots & C_0 \arrow[rr, "a"] & & C_1 \arrow[rr, "b"] & & C_2 \arrow[rr, "c"] & & C_3 \arrow[rr, "d"] & & C_4 \arrow[r] & \cdots \\
\cdots & C_0^T \arrow[r, "a"] & C_0'^T \arrow[r, "a"] & C_1^T \arrow[r, "b"] & C_1'^T \arrow[r, "b"] & C_2^T \arrow[r, "c"] & C_2'^T \arrow[r, "c"] & C_3^T \arrow[r, "d"] & C_3'^T \arrow[r, "d"] & C_4^T \arrow[r] & \cdots
\end{tikzcd}$$

Dtrs0 $(ibs0\ a)\ (obs0\ o\ o')\ C_0\ C_1\ C_1^T$

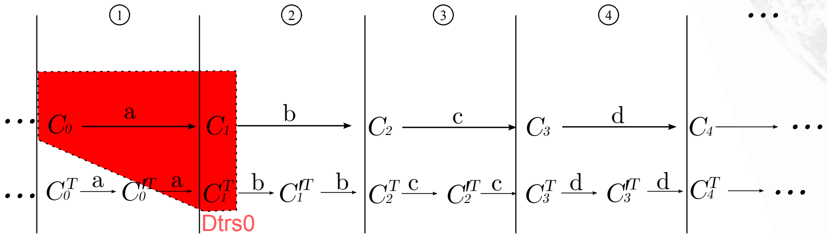
$\Rightarrow$ step $C_1\ b\ t_1\ C_2$

$\Rightarrow$ step $C_1^T\ b\ t_1'\ C_1'^T$

$\Rightarrow t_1' = (o, o, o') \wedge$

Dtrs1 $(ibs1\ b\ a)\ (obs1\ t_1\ o)\ C_0\ C_1\ C_2\ C_1'^T$

## General Proof Strategy - w/o faults



Dtrs0 $(ibs0\ a)\ (obs0\ o\ o')\ C_0\ C_1\ C_1^T$

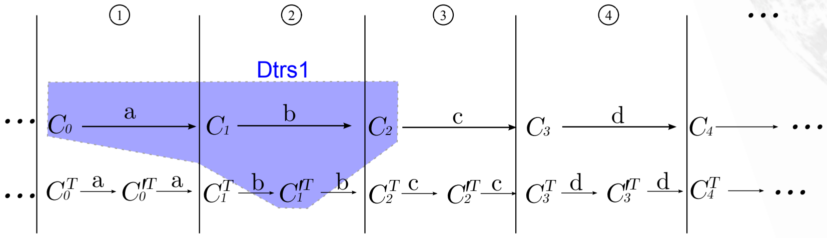
$\Rightarrow$ step $C_1\ b\ t_1\ C_2$

$\Rightarrow$ step $C_1^T\ b\ t_1'\ C_1'^T$

$\Rightarrow t_1' = (o, o, o') \wedge$

Dtrs1 $(ibs1\ b\ a)\ (obs1\ t_1\ o)\ C_0\ C_1\ C_2\ C_1'^T$

## General Proof Strategy - w/o faults


1
2
3
4
...
Dtrs1
... $C_0$ a $C_1$ b $C_2$ c $C_3$ d $C_4$ ...
... $C_0^T$ a $C_0'^T$ a $C_1^T$ b $C_1'^T$ b $C_2^T$ c $C_2'^T$ c $C_3^T$ d $C_3'^T$ d $C_4^T$ ...

Dtrs0 $(ibs0\ a)\ (obs0\ o\ o')\ C_0\ C_1\ C_1^T$

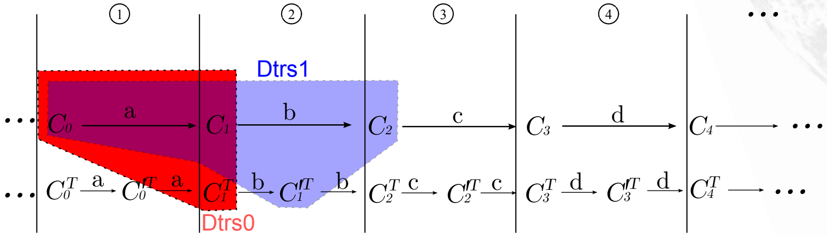
$\Rightarrow$ step $C_1\ b\ t_1\ C_2$

$\Rightarrow$ step $C_1^T\ b\ t_1'\ C_1'^T$

$\Rightarrow t_1' = (o, o, o') \wedge$

Dtrs1 $(ibs1\ b\ a)\ (obs1\ t_1\ o)\ C_0\ C_1\ C_2\ C_1'^T$

## General Proof Strategy - w/o faults

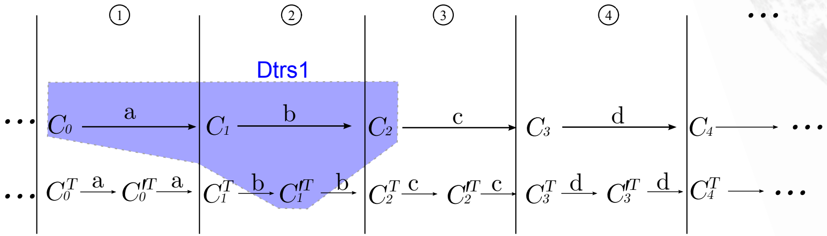


1
2
3
4
...
Dtrs1
... $C_0$ a $C_1$ b $C_2$ c $C_3$ d $C_4$ ...
... $C_0^T$ a $C_0'^T$ a $C_1^T$ b $C_1'^T$ b $C_2^T$ c $C_2'^T$ c $C_3^T$ d $C_3'^T$ d $C_4^T$ ...
Dtrs0

Lemma:

$$\begin{aligned}
&\mathsf{Dtrs0}\ (ibs0\ a)\ (obs0\ o\ o')\ C_0\ C_1\ C_1^T \\
&\quad \Rightarrow \mathsf{step}\ C_1\ b\ t_1\ C_2 \\
&\quad \Rightarrow \mathsf{step}\ C_1^T\ b\ t_1'\ C_1'^T \\
&\quad \Rightarrow t_1' = (o, o, o') \wedge \\
&\qquad\quad \mathsf{Dtrs1}\ (ibs1\ b\ a)\ (obs1\ t_1\ o)\ C_0\ C_1\ C_2\ C_1'^T
\end{aligned}$$

# General Proof Strategy - w/o faults



①
②
③
④
...
Dtrs1
$C_0 \xrightarrow{a} C_1 \xrightarrow{b} C_2 \xrightarrow{c} C_3 \xrightarrow{d} C_4 \longrightarrow \cdots$
$C_0^T \xrightarrow{a} C_0'^T \xrightarrow{a} C_1^T \xrightarrow{b} C_1'^T \xrightarrow{b} C_2^T \xrightarrow{c} C_2'^T \xrightarrow{c} C_3^T \xrightarrow{d} C_3'^T \xrightarrow{d} C_4^T \longrightarrow \cdots$

Dtrs1$(ibs1\ b\ a)\ (obs1\ t_1\ o)\ C_0\ C_1\ C_2\ C_1'^T$

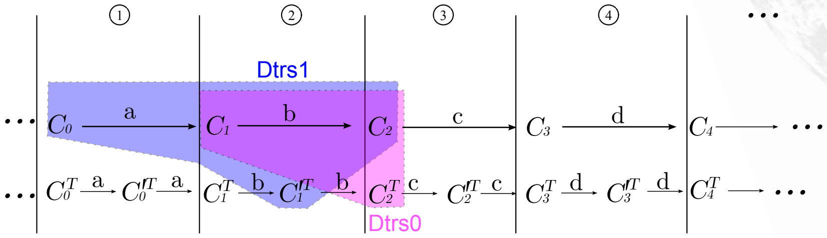
$\Rightarrow$ step $C_1\ b\ t_1\ C_2$

$\Rightarrow$ step $C_1'^T\ b\ t_1''\ C_2^T$

$\Rightarrow t_1'' = (o, o, o) \wedge$

Dtrs0 $(ibs0\ b)\ (obs0\ t_1\ o)\ C_1\ C_2\ C_2^T$

# General Proof Strategy - w/o faults



1
2
3
4
...
Dtrs1
... C0 a C1 b C2 c C3 d C4 ...
... C0^T a C0'^T a C1^T b C1'^T b C2^T c C2'^T c C3^T d C3'^T d C4^T ...
Dtrs0

Dtrs1$(ibs1\ b\ a)\ (obs1\ t_1\ o)\ C_0\ C_1\ C_2\ C_1'^T$

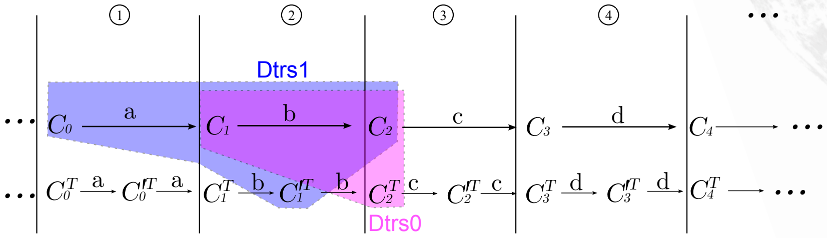
$\Rightarrow$ step $C_1\ b\ t_1\ C_2$

$\Rightarrow$ step $C_1'^T\ b\ t_1''\ C_2^T$

$\Rightarrow t_1'' = (o, o, o) \wedge$

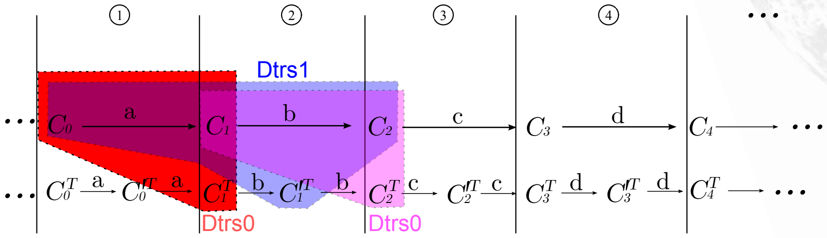
Dtrs0 $(ibs0\ b)\ (obs0\ t_1\ o)\ C_1\ C_2\ C_2^T$

# General Proof Strategy - w/o faults



$$
\begin{aligned}
&\mathsf{Dtrs1}(ibs1\ b\ a)\ (obs1\ t_1\ o)\ C_0\ C_1\ C_2\ C_1'^T\\
&\qquad\Rightarrow \mathsf{step}\ C_1\ b\ t_1\ C_2\\
&\qquad\Rightarrow \mathsf{step}\ C_1'^T\ b\ t_1''\ C_2^T\\
&\qquad\Rightarrow t_1'' = (o, o, o) \wedge\\
&\qquad\qquad \mathsf{Dtrs0}\ (ibs0\ b)\ (obs0\ t_1\ o)\ C_1\ C_2\ C_2^T
\end{aligned}
$$

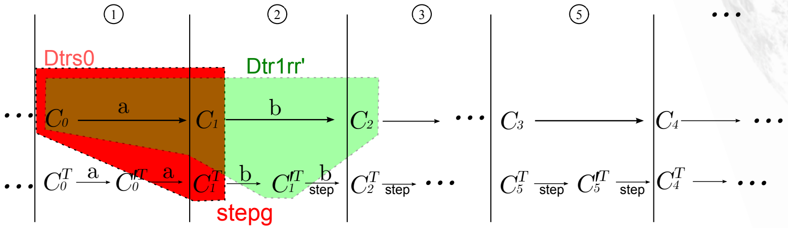
# General Proof Strategy - w/o faults



①
②
③
④
...
Dtrs1
$C_0$ a $C_1$ b $C_2$ c $C_3$ d $C_4$
$C_0^T$ a $C_0'^T$ a $C_1^T$ b $C_1'^T$ b $C_2^T$ c $C_2'^T$ c $C_3^T$ d $C_3'^T$ d $C_4^T$
Dtrs0
Dtrs0

$$\begin{aligned}
&\mathsf{Dtrs1}(ibs1\ b\ a)\ (obs1\ t_1\ o)\ C_0\ C_1\ C_2\ C_1'^T\\
&\quad\Rightarrow \mathsf{step}\ C_1\ b\ t_1\ C_2\\
&\quad\Rightarrow \mathsf{step}\ C_1'^T\ b\ t_1''\ C_2^T\\
&\quad\Rightarrow t_1'' = (o,o,o)\wedge\\
&\qquad\qquad \mathsf{Dtrs0}\ (ibs0\ b)\ (obs0\ t_1\ o)\ C_1\ C_2\ C_2^T
\end{aligned}$$

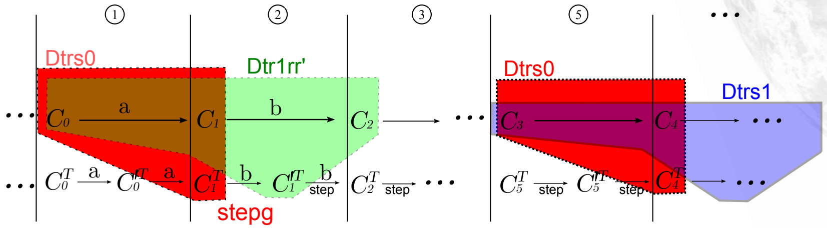
# General Proof Strategy - with a glitch



- 15 different corruption cases
- Dtr1rr′ describes one of the corruption cases
- Within 10 cycles returns to a correct state:

Dtrs0 → Dtr1rr′ → Dtr0r′ → Dtr1r′ → Dtrs0

## General Proof Strategy - with a glitch

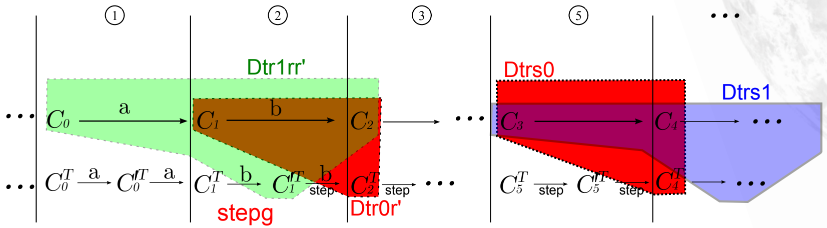


1
2
3
5
...
Dtrs0
Dtr1rr'
Dtrs0
Dtrs1
$C_0 \xrightarrow{a} C_1 \xrightarrow{b} C_2 \rightarrow \cdots C_3 \rightarrow C_4 \rightarrow \cdots$
$C_0^T \xrightarrow{a} C_0'^T \xrightarrow{a} C_1^T \xrightarrow{b} C_1'^T \xrightarrow[step]{b} C_2^T \xrightarrow[step]{} \cdots C_5^T \xrightarrow[step]{} C_5'^T \xrightarrow[step]{} C_4^T \rightarrow \cdots$
stepg

- 15 different corruption cases
- Dtr1rr′ describes one of the corruption cases
- Within 10 cycles returns to a correct state:

Dtrs0 → Dtr1rr′ → Dtr0r′ → Dtr1r′ → Dtrs0

# General Proof Strategy - with a glitch



①
②
③
⑤
...
Dtr1rr'
Dtrs0
Dtrs1
$C_0$ a $C_1$ b $C_2$
$C_3$ $C_4$
$C_0^T$ a $C_0'^T$ a $C_1^T$ b $C_1'^T$ b $C_2^T$ step
step
$C_5^T$ step $C_5'^T$ step $C_4^T$
stepg
Dtr0r'

Dtrs0 → Dtr1rr′ → Dtr0r′ → Dtr1r′ → Dtrs0

$$
\begin{aligned}
&\mathsf{Dtr1rr'}(ibs1\ b\ a)\ (obs1\ t_1\ o)\ C_0\ C_1\ C_2\ C_1'^T\\
&\quad\Rightarrow \mathsf{step}\ C_1\ b\ t_2\ C_2\\
&\quad\Rightarrow \mathsf{step}\ C_1'^T\ b\ t_2''\ C_2^T\\
&\quad\Rightarrow t_2'' = (o, o, o) \wedge\\
&\qquad\qquad \mathsf{Dtr0r'}\ (ibs0\ b)\ (obs0\ t_2\ t_1)\ C_1\ C_2\ C_2^T
\end{aligned}
$$

## Summary

## Summary of case study

- Automatic DTR transformation:

## Summary of case study

- Automatic DTR transformation:
  - formalized on the syntax of LDDL

## Summary of case study

- Automatic DTR transformation:
  - formalized on the syntax of LDDL
  - formally proven in Coq proof assistant (7000 LOCs- 5 man-months)

## Summary of case study

- Automatic DTR transformation:
  - formalized on the syntax of LDDL
  - formally proven in Coq proof assistant (7000 LOCs- 5 man-months)
  - by simple inductions:
    - on syntax
    - on types
    - on streams (co-induction)

## Conclusion

- LDDL language: syntax, semantics

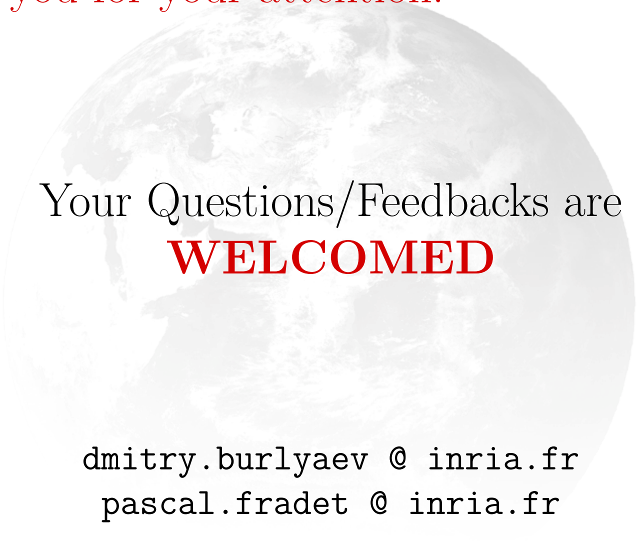
## Conclusion

- LDDL language: syntax, semantics
- Coq benefits:
  - dependent types → circuits well-formedness
  - reflection replaces some proofs with computation

## Conclusion

- LDDL language: syntax, semantics
- Coq benefits:
  - dependent types $\rightarrow$ circuits well-formedness
  - reflection replaces some proofs with computation
- Future work:
  - good to have better automation with tactics
  - proof of other fault-tolerance techniques

## Thank you for your attention!



Your Questions/Feedbacks are
**WELCOMED**

`dmitry.burlyaev @ inria.fr`
`pascal.fradet @ inria.fr`