# An Automated Formal Verification Flow for Safety Registers

Holger Busch

Infineon Technologies


accellera SYSTEMS INITIATIVE

2015 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

# Contents

- Motivation
- Register Specification
- Register Safety Requirements
- Register Safeguarding
- Formal Register Properties
- Automated Register Verification Flow
- Experience and Results
- Conclusions

# Motivation

- Automotive safety applications
  - ADAS/Autonomous driving, breaking systems,…
- Microcontrollers need
  - Hardware and software safety mechanisms
    - Redundancy
    - Robustness
  - ISO 26262 compliant development processes
    - Modular hierarchical design
    - Evidence for effectiveness of installed safety mechanisms

# Register Specification

SFRX000 $200_H$ Reset Value:22C0 $0000_H$

| 31 | 30 | 29 | 28 | 27 … 26 | 25 | 24 | 23 | 22 … 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| LCK | 0 | EN | ERR | DIV | BTE | BYP | STAT | CLR | MODE |
| rwh | r | rw | rwh | rw | rwh | rw | rh | w | rw |

- Short register name
- Offset address
- Reset value
- Reset class
- Access modes
- Bit-fields
  - Short name
  - Access type
  - Index range

```
<SFR>
 <SFRNAME>SFRX00</SFRNAME>
 <OADDR>200</OADDR>
 <RVAL>22C00000> <RCLASS>PORST<RCLASS>
 <WACC>SV</WACC> <RACC>U,SV</RACC>
 <BF>
  <BFNAME>DIV</BFNAME>
  <BFTYPE>rw</BFTYPE>
  <BFIH>27</BFIH> <BFIL<26</BFIL>
 </BF>
```

# White-Box Specification

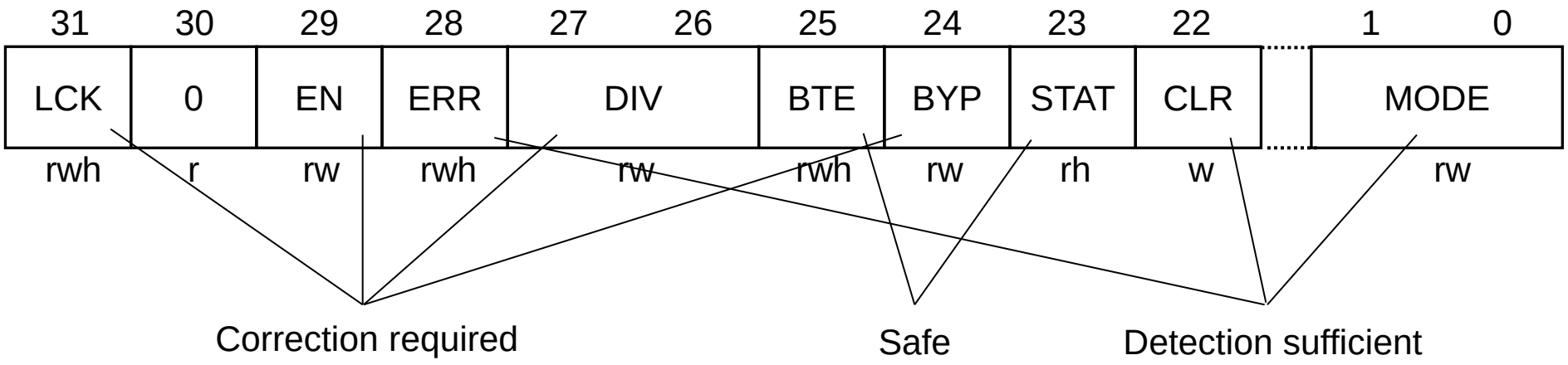
SFRX000 $200_H$ Reset Value:22C0 $0000_H$

| 31 | 30 | 29 | 28 | 27 … 26 | 25 | 24 | 23 | 22 … 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| LCK | 0 | EN | ERR | DIV | BTE | BYP | STAT | CLR | MODE |
| rwh | r | rw | rwh | rw | rwh | rw | rh | w | rw |

- RTL signal name
- Bit-fields
  - SW write protection
  - HW write condition
  - HW write value
  - Default value
  - Write latenc

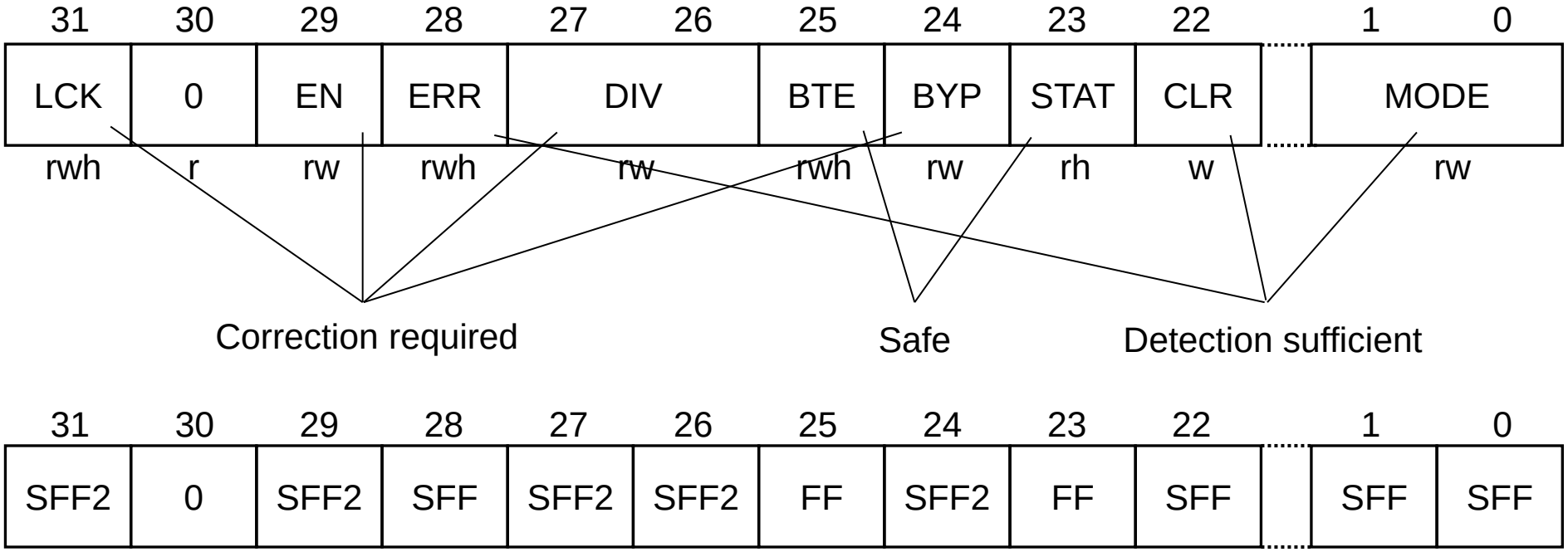
```
<SFR>
 <SFRNAME>SFRX000</SFRNAME>
 <RTLNAME>inst0/inst02/inst021/sfrx00</RTLNAME>
<BF>
  <BFNAME>BTE</BFNAME>
   <PR>SFRX00.LCK = 1</PR>
   <HCOND>MODE = TEST</HCOND>
   <HVAL>1</HVAL>
   <DVAL>BF</DVAL>
   <LAT>1</LAT>>
</BF>
```

# Safety Requirements



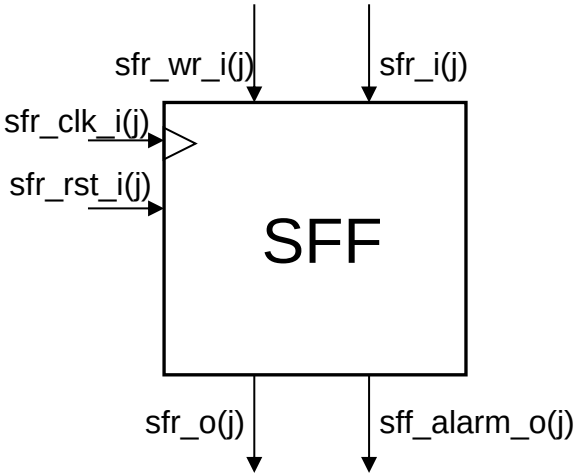

```
<SFR>
  <SFRNAME>SFRX00</SFRNAME>
  <BF> <BFNAME>LCK</BFNAME>  <SFTY>CORR</PR> </BF>
  <BF> <BFNAME>ERR</BFNAME>  <SFTY>DET</PR> </BF>
  <BF> <BFNAME>STAT</BFNAME>  <SFTY>None</PR> </BF>
```

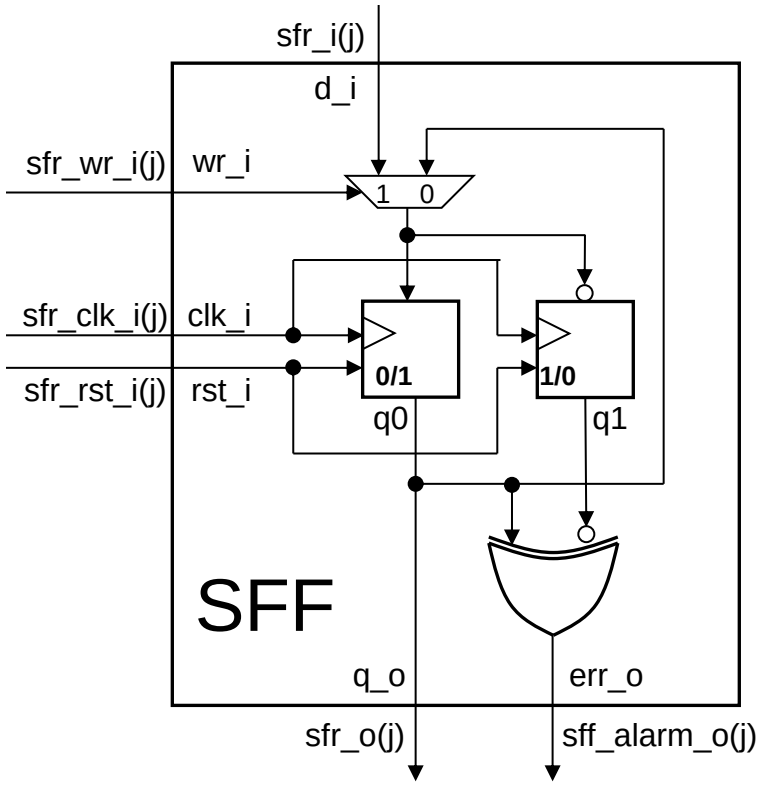
# Register Safeguarding




- FF : simple Flip-Flop
- SFF : 2 FF-cells ~> alarm
- SFF2: 3 FF-cells + majority decision ~> correction + alarm

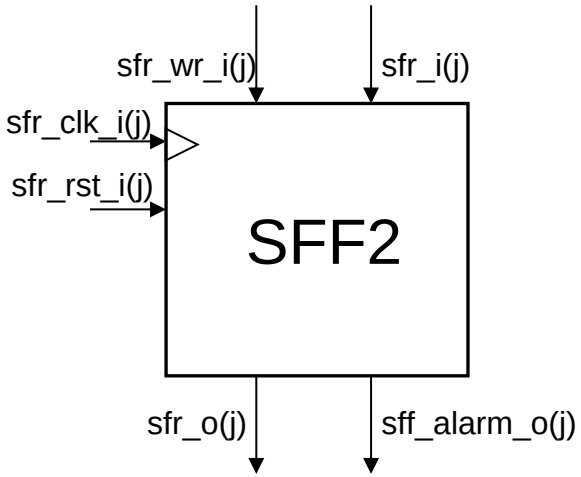
# Detection Only

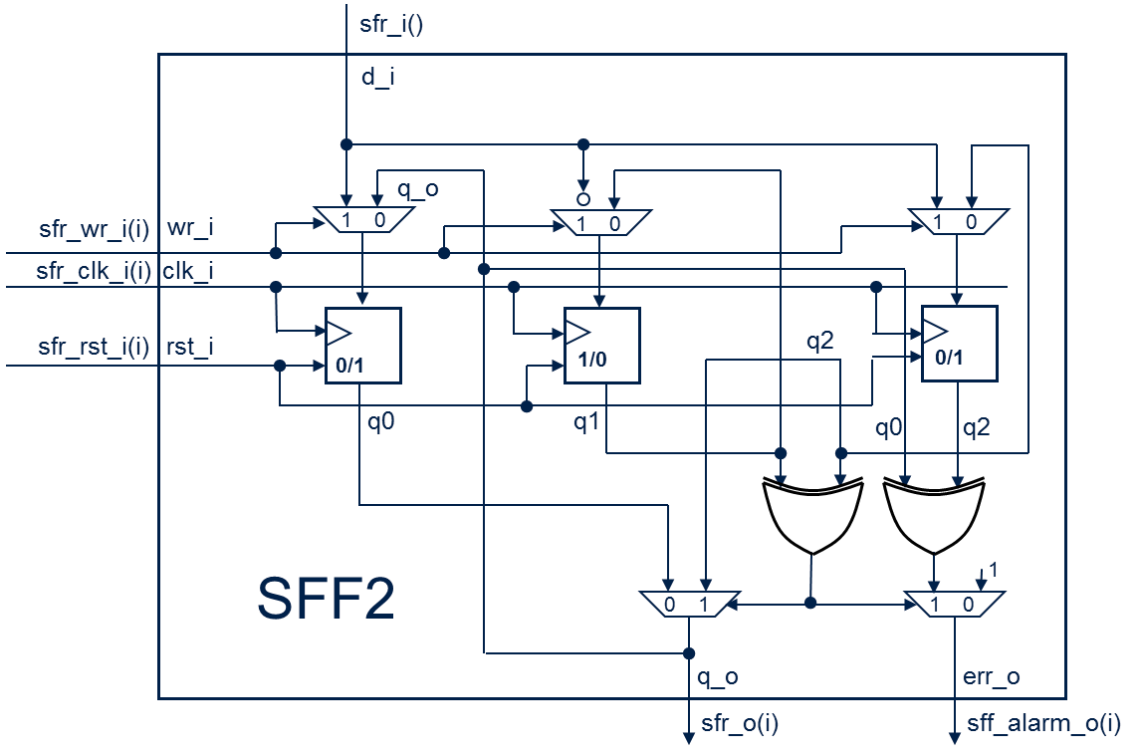



- Inputs: data, write enable
- Outputs: uncorrected data, alarm

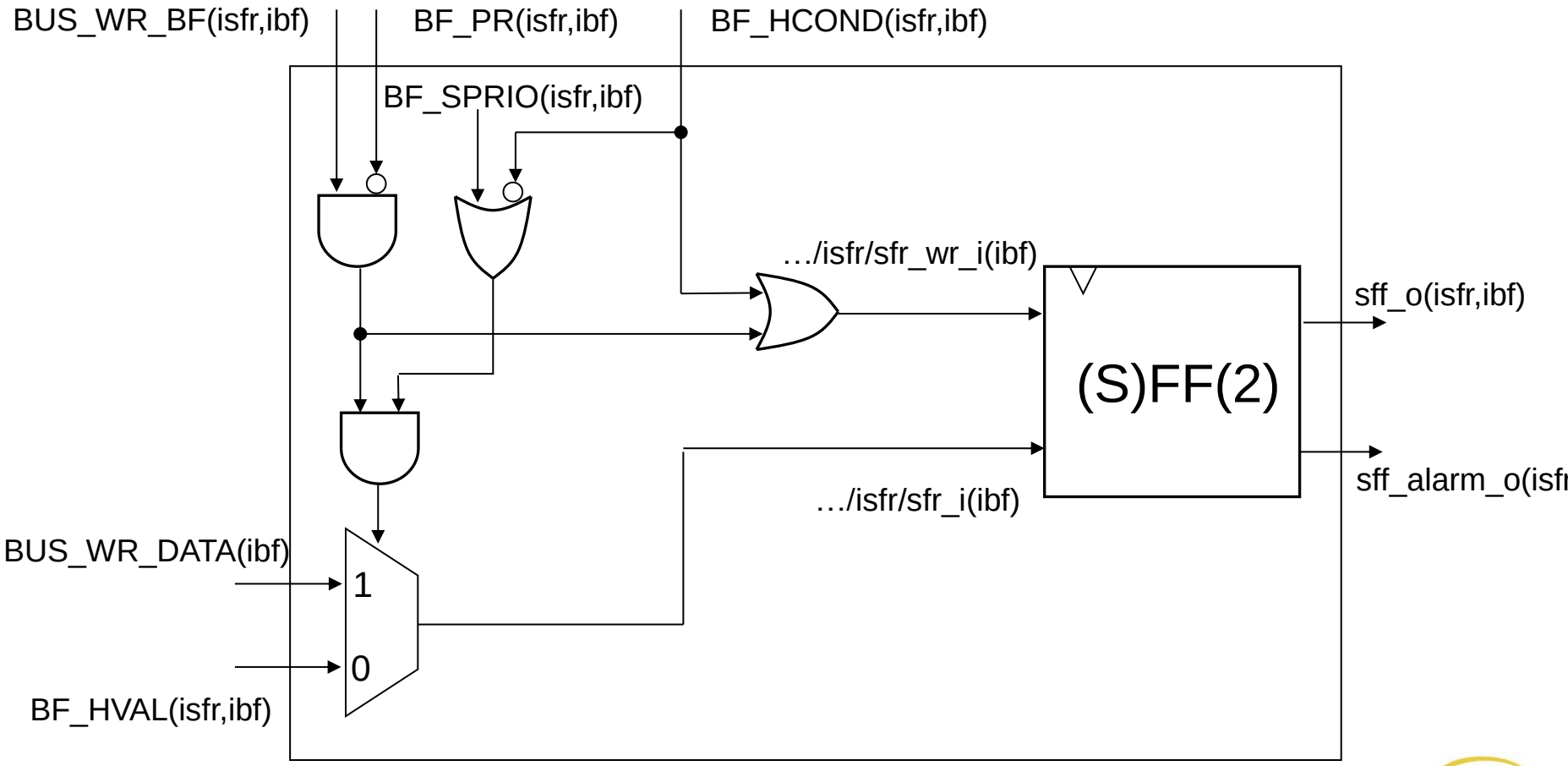
# Correction




- Inputs: data, write enable
- Outputs: corrected data, alarm

# Plugging Together



# Formal Register Properties

- Black-Box:
  - Write + read from Bus:
    Previous write data
    identically read,
    if no intermediate write and
    if no reset

```
j := BFih(SFRX00,ERR);
trd := twr + 3;
val := BUS_WR_DATA(j)@twr+1;
during[twr,trd+1]:
  RST(RSTCL(SFRX00)) = 0;
at twr:
  BUS_WR_BF(SFRX00,ERR);
during[twr+1,trd]:
  not BUS_WR_BF(SFRX00,ERR);
at trd:
  BUS_RD_BF(SFRX00,ERR);
|- at trd+1: BUS_RD_DATA(j) = val;
```

- White-Box:
  - HW Write:
    HW-write condition true,
    HW-write data depending on internal signals

```
val := BF_HVAL(SFRX00,BFK)@twr+1;
during[twr,twr+2]:
RST(RSTCL(SFRX00)) = 0;
at twr+1:
  BF_HCOND(SFRX00,ERR);
at twr:
  not BUS_WR_BF(SFRX00,ERR) \/
  BF_PR(SFRX00,ERR) = 1
|- at twr+2: BF(SFRX00,BFK) = val;
```

```
BF_HCOND(SFRX00,ERR) :=  <Overflow>;
BF_PR(SFRX00,ERR) := inst0/inst02/inst021/sfrx00(31) = 1;
BF(SFRX00,ERR) = inst0/inst02/inst021/sfrx00(28);
```

# Aggregate Register Properties

- List of registers:
  - All rwh-bit-fields checked at once

```
isfr:= any(SFR_indices); ibf:=1..number_BFs(i);
trd := twr + 1; val := BF_HVAL(i,j)@twr+1;
BF_TYPE(isfr,ibf) = rwh /\ BF_LAT(isfr,ibf) = 1;
during[twr,trd+1]:
  RST(RSTCL(isfr)) = 0;
at twr:
  not BUS_WR_BF(isfr,ibf) \/
  BF_PR(isfr,ibf) = 1 \/
  BF_SPRIO(isfr,ibf) = 0;
at twr+1:
  BF_HCOND(isfr,ibf);
|- at trd+1: BF(isfr,ibf) = val;
```

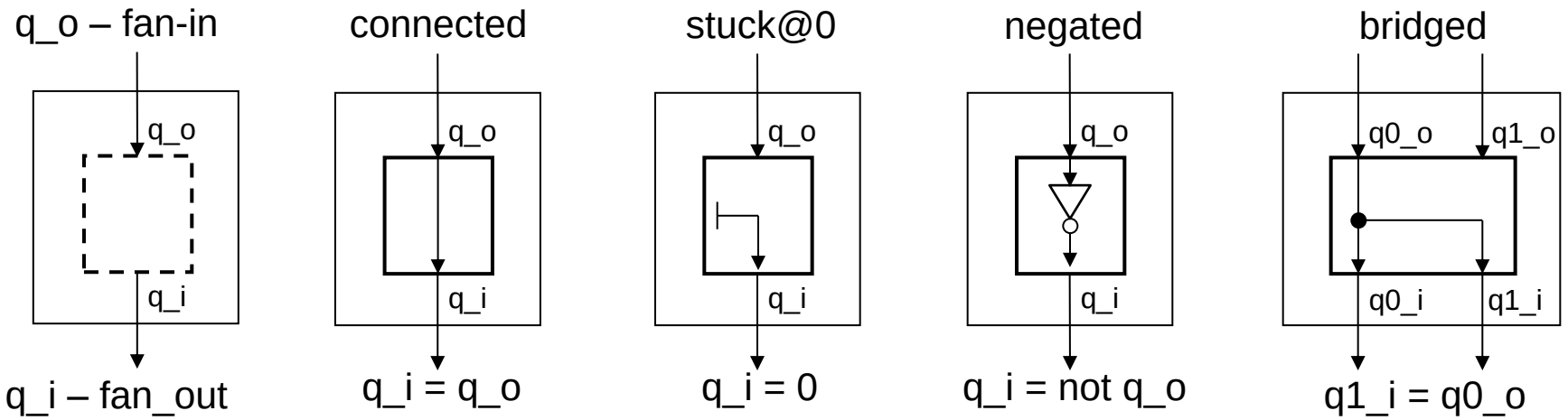
# Register Safety Properties

- List of registers:
  - Any error in any bit-field of any register causes alarm

```
isfr:= any(SFR_indices); ibf:=1..number_BFs(i);
during[te,te+1]:
  RST(RSTCL(isfr)) = 0;
at te:  be(isfr,ibf);
|- at te+1:  alarm = 1;
```

# Fault Models

- Cut signals into fan-in (output) and fan-out part
- Assume fault in property assumption

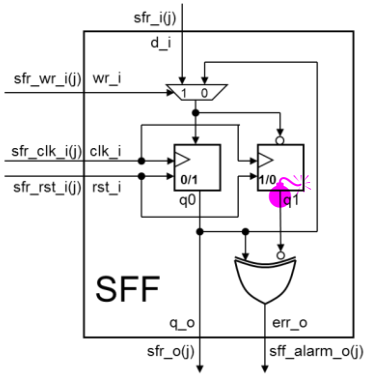


# Vectorization for Negation Model

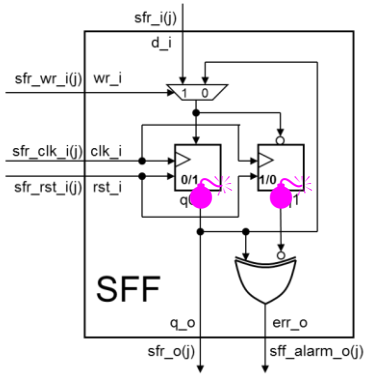
- Concatenate all corresponding bits
- Construct difference vectors
- Count 1-positions in difference vectors for duplicated FlipFlops

```
sff_q0_o :=  inst0/inst00/inst000/inst_sffj0/q0_o & … inst0/inst02/inst021/inst_sffj5/q0_o  … ;
sff_q0_i  :=  inst0/inst00/inst000/inst_sffj0/q0_i &  …inst0/inst02/inst021/inst_sffj5/q0_i  … ;
sff_q1_o :=  inst0/inst00/inst000/inst_sffj0/q1_o &  …inst0/inst02/inst021/inst_sffj5/q1_o  … ;
sff_q1_i  :=   inst0/inst00/inst000/inst_sffj00/q1_i &  …inst0/inst02/inst021/inst_sffj5/q1_i  … ;
be(SFRx000,ibf) := ((sff_q0_o xor sff_q0_i)(ibf) xor (sff_q1_o xor sff_q1_i)(ibf) = 1;
```

sbe: detected




dbe: undetected

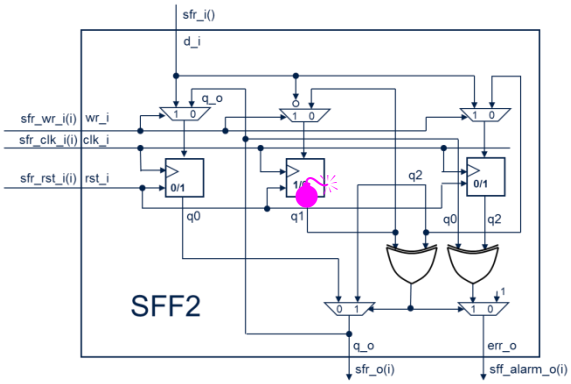


- Concatenate all corresponding bits of triplicated Flip-Flops

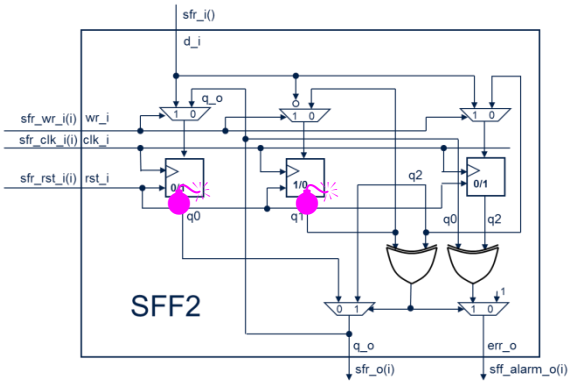
```
sff2_q0_o :=    inst0/inst00/inst000/inst_sff2i0/q0_o &  …
                inst0/inst02/inst021/inst_sff2i5/q0_o  … ;

sff2_q1_o :=    inst0/inst00/inst000/inst_sff2i0/q1_o &  …
                inst0/inst02/inst021/inst_sff2i5/q1_o  … ;

sff2_q2_o :=    inst0/inst00/inst000/inst_sff2i0/q2_o &  …
                inst0/inst02/inst021/inst_sff2i5/q2_o  … ;

sff2_q0_i :=    inst0/inst00/inst000/inst_sff2i0/q0_i &  …
                inst0/inst02/inst021/inst_sff2i5/q0_i  … ;

sff2_q1_i :=    inst0/inst00/inst000/inst_sff2i0/q1_i &  …
                inst0/inst02/inst021/inst_sff2i5/q1_i  … ;

sff2_q2_i :=    inst0/inst00/inst000/inst_sff2i0/q2_i &  …
                inst0/inst02/inst021/inst_sff2i5/q2_i  … ;
```

- Form vectors for single, double, triple bit errors

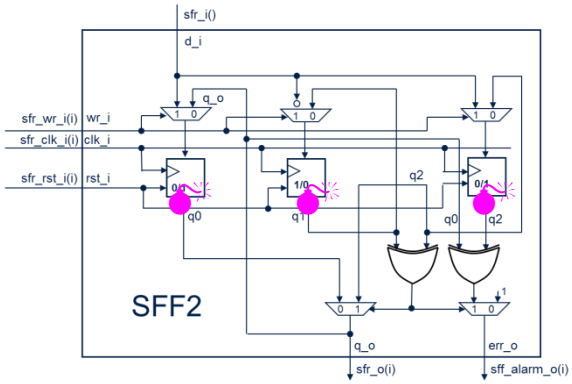
```
sbe_sff2 := ((sff2_q0_o xor sff2_q0_i ) and (sff2_q1_o = sff2_q1_i ) and (sff2_q2_o = sff2_q2_i )) or
            ((sff2_q0_o = sff2_q0_i ) and (sff2_q1_o xor sff2_q1_i ) and (sff2_q2_o = sff2_q2_i )) or
            ((sff2_q0_o = sff2_q0_i ) and (sff2_q1_o = sff2_q1_i ) and (sff2_q2_o xor sff2_q2_i ));
dbe_sff2 := ((sff2_q0_o xor sff2_q0_i ) and (sff2_q1_o xor sff2_q1_i ) and (sff2_q2_o = sff2_q2_i )) or
            ((sff2_q0_o xor sff2_q0_i ) and (sff2_q1_o = sff2_q1_i ) and (sff2_q2_o xor sff2_q2_i )) or
            ((sff2_q0_o = sff2_q0_i ) and (sff2_q1_o xor sff2_q1_i ) and (sff2_q2_o xor sff2_q2_i ));
tbe_sff2 :=  ((sff2_q0_o xor sff2_q0_i ) and (sff2_q1_o xor sff2_q1_i ) and (sff2_q2_o xor sff2_q2_i ));
```




sbe: corrected




dbe: detected




tbe: undetected

# Aggregate Safety Properties

| | | | |
|---|---|---|---|
| sbe(n) := | count1s(sbe_sff) = n; | sbe2(n) := | count1s(sbe_sff2) = n; |
| dbe(n) := | count1s(dbe_sff ) = n; | dbe2(n) := | count1s(dbe_sff2) = n; |

- Regular SFR property on fault-injected model

| | |
|---|---|
| <sfr_prop> := <sfr_prop_assumption> | \|- <sfr_prop_commitment> |

fail

| | |
|---|---|
| sbe(0) /\ dbe(0) /\ dbe2(0) /\ <sfr_prop_assumption> | \|- <sfr_prop_commitment> |

hold

| | |
|---|---|
| sbe(n0) /\ sbe2(n2) /\ dbe2(n3) /\ n0 + n1 + n2 + n3 > 0 | \|- sfr_red_alarm_o = 1 |

hold

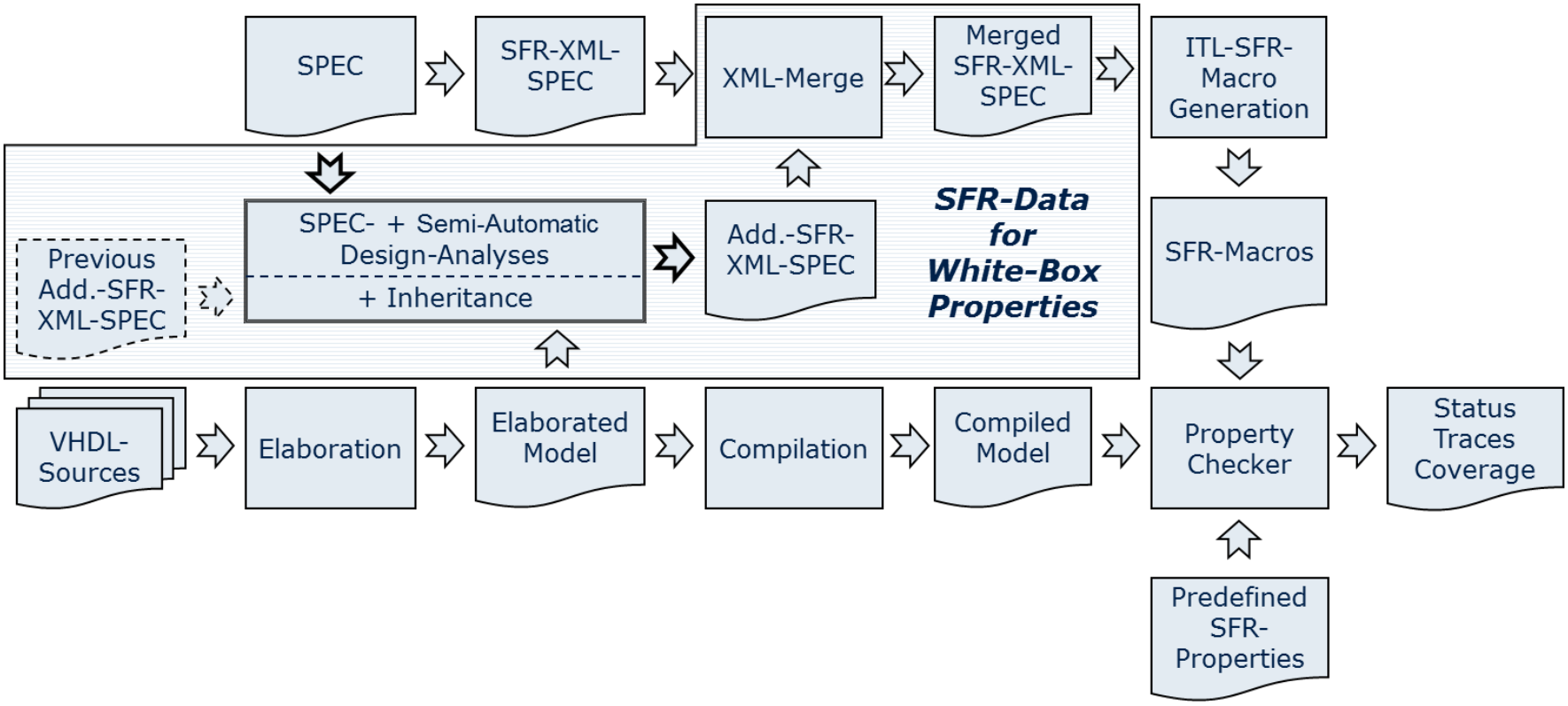
| | |
|---|---|
| <sfr_prop> := sbe(n0) /\ sbe2(n2) /\ dbe2(n3) /\ n0 + n1 + n2 + n3 = 0 | \|- sfr_red_alarm_o = 0 |

hold

| | |
|---|---|
| sfr_red_alarm_o = 0 /\ <sfr_prop_assumption> | \|- <sfr_prop_commitment> |

hold

# Automated Register Verification Flow



# White-Box versus Black-Box Props

| Operation | Black-Box | White-Box |
|---|---|---|
| Read pure rw-bit-fields | Write, then read;<br>compare bus read data with previous write data | Just read;<br>compare bus read data with internal contents |
| Read rwh-bit-fields | n/a: Internal hw-write after sw-write can modify register | ---"--- |
| Read rh-bit-fields | n/a: no value to compare with read value | ---"--- |
| Write rw-bit-fields | Like read property | compare bus write data with internal contents |
| Write rwh-bit-fields | n/a | ---"--- |
| Write rh-bit-fields | n/a: no value to compare with read value | ---"--- |
| HW-write | n/a: neither write value, nor internal contents accessible | compare hw write data with internal contents |

# Experience and Results

- Black-box properties
  - Cover pure rw-registers
  - Required macros are generated fully automatically
- White-box properties
  - Cover all function of all registers including hw writes
  - Additional macros to be partially edited,
    partially inheritance + default guesses
- Aggregate SFR-properties
  - Cover all bit-fields of all registers simultaneously
    130 SFRs completely verified by 30 black- and white-box properties in 35 minutes with up to 10 parallel proofs
- Safety properties generated fully automatically
  - All potential faults checked simultaneously by 20 properties with exhaustive fault-injection, built-in self-test logic covered as well very efficient: e.g. 3000 faults in 25 minutes
  - Macro generation in less than one minute

# Conclusions

- Formal register verification is most efficient and complete -> highest quality
- Formal safety verification completely checks hw safety-mechanism -> evidence for ISO26262 certification
- Maximum efficiency by
  - Automatic transformation of XML specifications
  - Uniform safety mechanisms
  - Corresponding pre-defined generic properties
  - Automatically generated + inherited design-specific macros

# Questions

Finalize slide set with questions slide