Optimizing Functional Safety and High Reliability for FPGA-based Design

Nilesh Shilankar & Manohar Reddy Vadicherla

SYNOPSYS®

SYSTEMS INITIATIVE™

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Presenters

Nilesh Shilankar

Applications Engineering, Staff Engineer

Nilesh is working as Staff Applications Engineer at Synopsys and look after the FPGA based prototyping and implementation customers in global support team. He has done Bachelor of Engineering in 2012 and Post graduate Diploma in VLSI from C-DAC.

Presenters

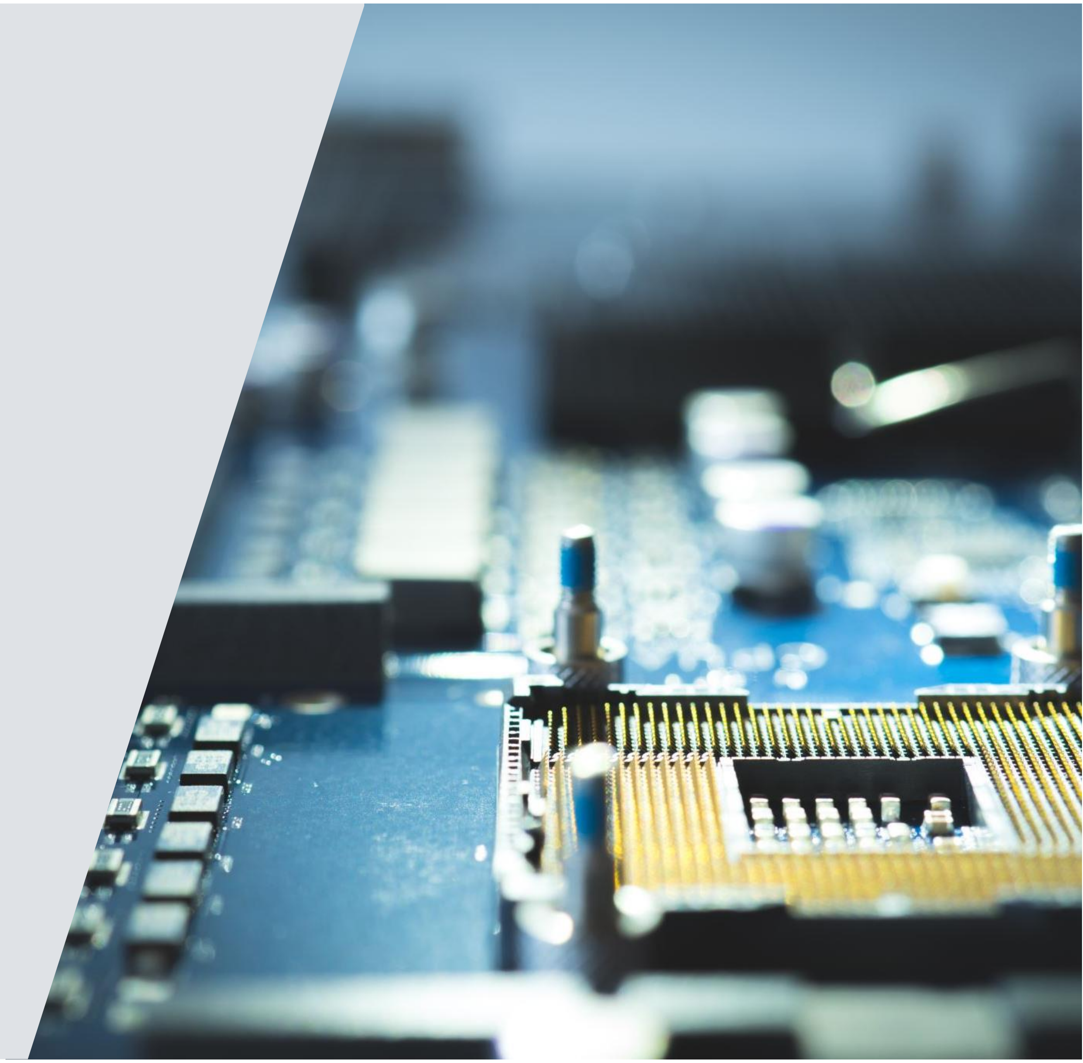
Manohar Reddy Vadicherla

Applications Engineering, Sr. Manager

Manohar is working as Sr. Manager, Applications Engineering at Synopsys and manages a global Support and Product Engineering team. He holds a bachelor degree and holds 20+ years of experience in FPGA design and verification domain.

Agenda

- Applications of FPGA
- Introduction and need of Safety and High Reliability
- Synopsys Synplify® Overview
- Random Fault Mitigation Techniques
- Debug for High Reliability & Functional Safety
- High Reliability Certification
- Summary
- Q&A



FPGA Applications

HPC, Data Center (AI accelerators)

Networking

Mobile/5G

Automotive

Industrial Automation

Enterprise Software

Healthcare

Memory

Government & Aerospace

Financial Services

PC/Gaming

AR / VR

Need for Safety & High Reliability in FPGA Designs

- FPGA are used in many safety critical applications areas:
  - High radiation environments
  - Human Safety
  - Military and Aerospace
  - Medical Instrumentations
  - Robotics in medical and industrial Applications
  - Critical Communication service that require high uptime!
  - Automotive and Industrial

Need for Safety and Reliability Pervades All Electronic Systems

Mil/Aero

- Avionics
- Satellite
- Flight Systems

- Driver Assistance
- Lane Departure
- Pedestrian Detection

Automotive

Communications & Networking

- Maintain high uptime
- Quality of Service

- Motor Control
- Programmable Logic Controller
- Robot Safety

Industrial

Medical

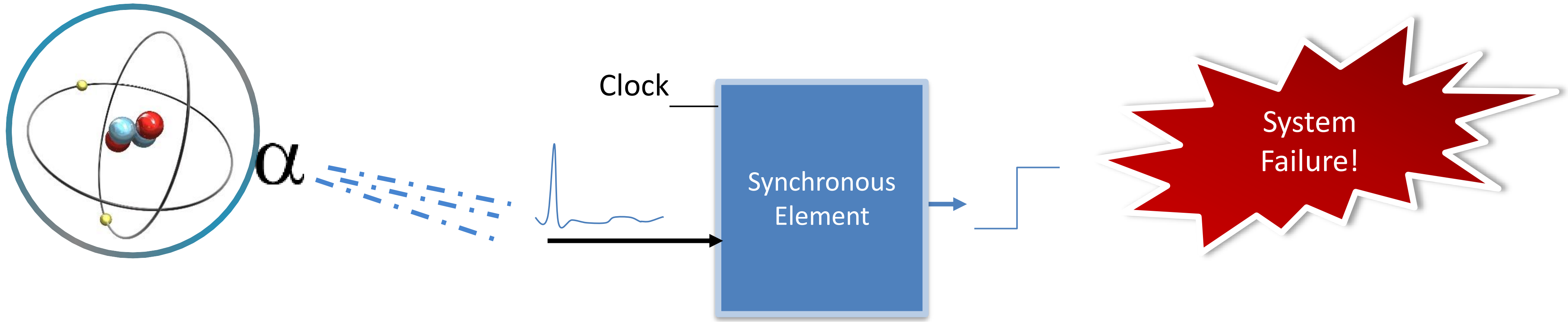
- Magnetic Resonance Imaging
- Ultrasound
- Portable Medical

System Faults in FPGA Design

Single Event Transient and Single Event Upset

Glitches clocked into a synchronous element can cause operation errors

- Use an error detection and recovery scheme to fix the error
- Can impact memories, registers and state machines



SET (Single Event Transient)

A current spike or “glitch” in a signal that occurs due to ionization or electromagnetic radiation

Soft Error (Single Event Upset)

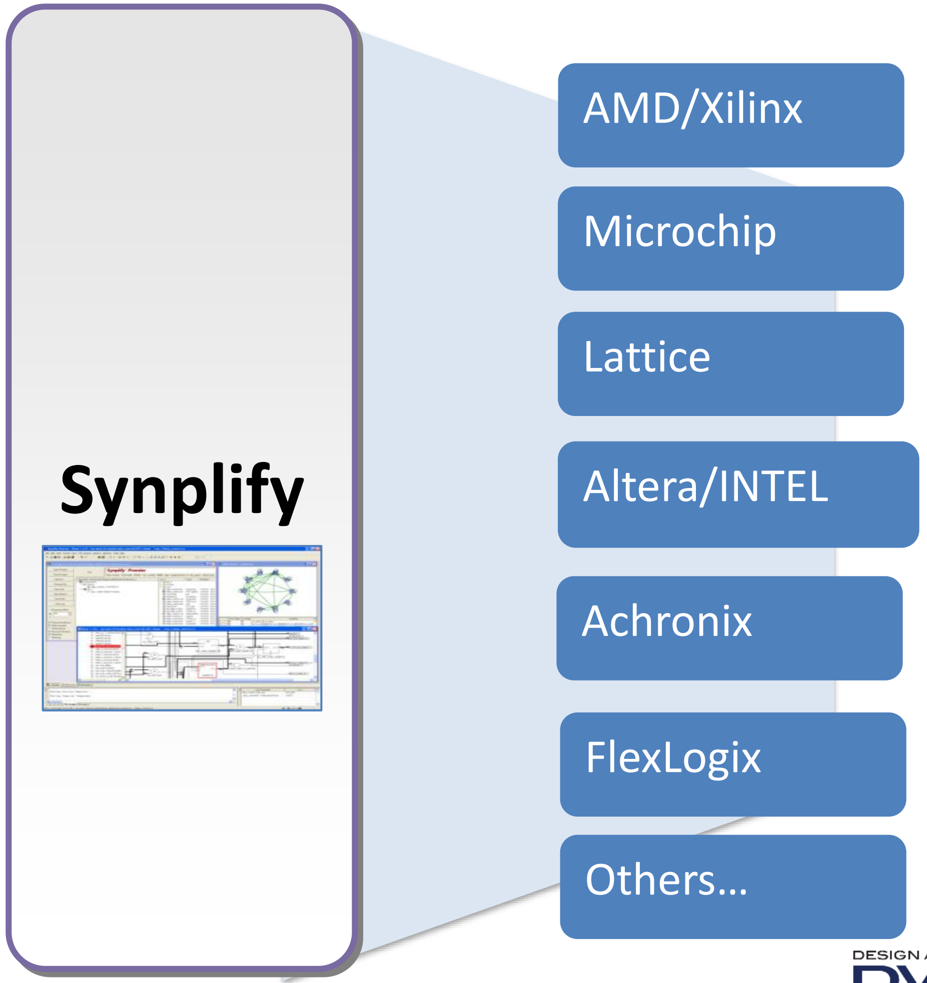
Incorrect signal (SET) captured by a synchronous element that impacts its internal state and / or output

Synplify-FPGA Synthesis Overview

Faster time to market with accelerated synthesis and integrated debug

- Widely adopted synthesis solution
- Best in QoR with Advanced synthesis technology
- Fast Runtime!
- Fastest design bring-up and debug turnaround
- Versatile IP handling with encryption
- Support for latest & legacy FPGA device families

- Safety and mission critical design automation
  - Error detection & mitigation insertion
  - Triplicated logic physical separation
  - Duplicate with Compare (DWC)
  - Hamming3 for FSM
  - Error Flag Insertion
  - Arbitrary Hardware Fault Injection



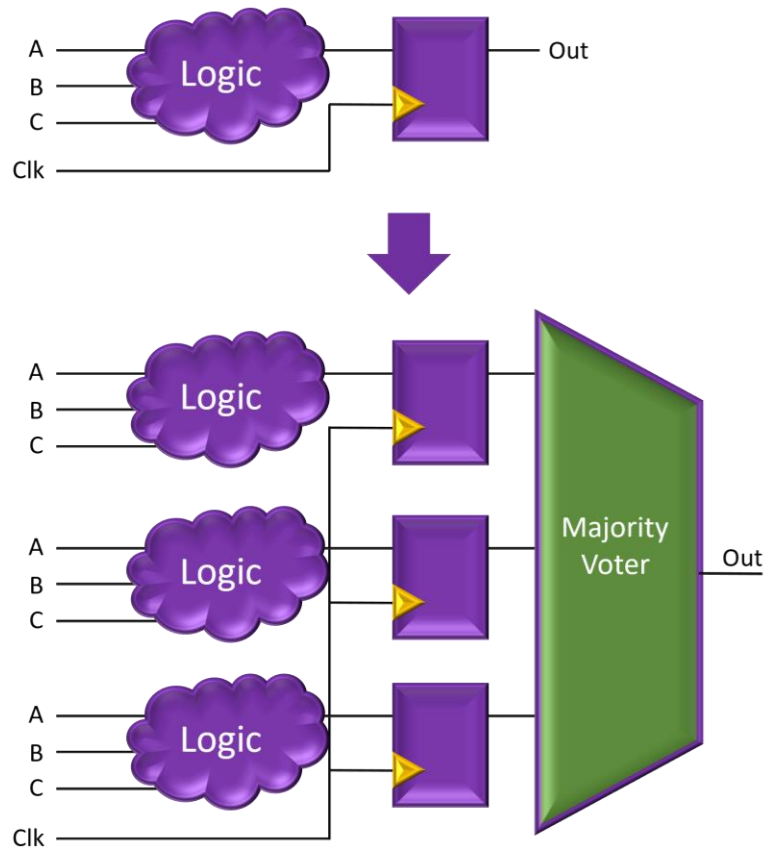
SYNOPSYS SYNPLIFY – RANDOM FAULT MITIGATION TECHNIQUES

Synplify – Random Fault Mitigation Techniques

Logic and Registers	**Local TMR**	RadHard device registers
	Block TMR	SRAM devices
	Distributed TMR	SRAM devices
	Physical separation	For Block TMR, Distributed TMR
	Gate Level TMR	Implement TMR on mapped designs
	Hamming -3 register	
Safe/Fault Tolerant FSM	**Hamming-3**	Automatic Error Detection & Correction
	Safe Encoding FSM	Detection with Recovery / Reset
	Safe Case FSM	Avoid Lockup
Memories	**TMR of BRAMs**	Triplicate/self correct w/ voting logic
	Create Error Flag	Drive CRAM scrubbing
	Use ECC RAM primitives	
Global Routing, IP, Clocks, I/Os	**Duplicated with Compare**	Replicate and correct
	Distributed TMR	For I/Os
	Block TMR	For Clocks, routing, and IP
Debug	**Error Monitoring**	
	Fault Simulation	
	Hardware based Fault injection	

Triple Modular Redundancy (TMR) Technique

- No HDL source code modification
- Fully tool automated solution via tool attribute
- Select and Triplicates only designated portion of design
- Utilizes majority voter to generate error-corrected output
- 3 TMR techniques: Local TMR, Distributed TMR, and Block TMR



Triple Modular Redundancy (TMR) Implementations

Local TMR (LTMR)

Protect Registers, including CE feedback

- Registers triplicated and feedback correction provided

Distributed TMR (DTMR)

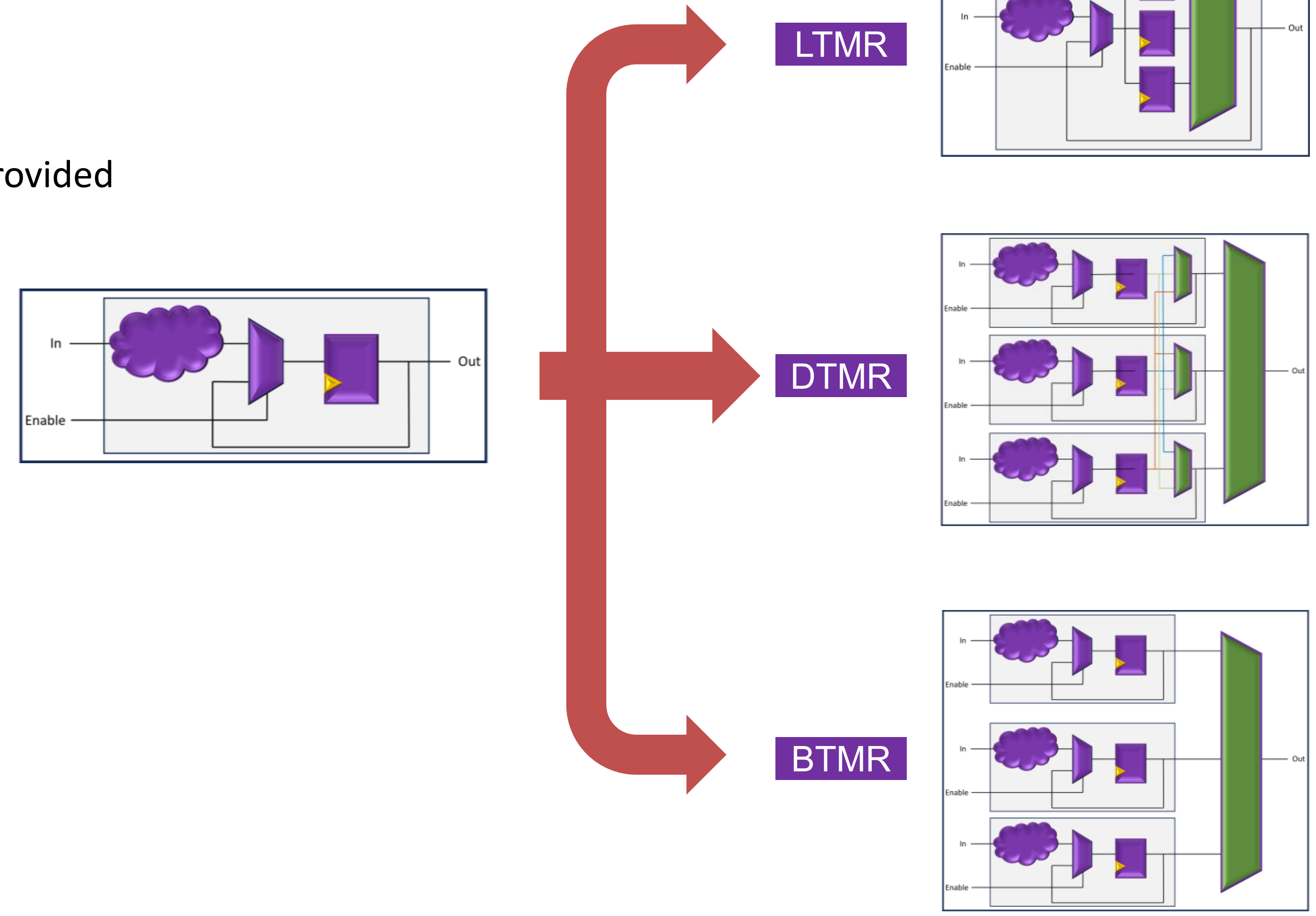
Protect I/O's & Logic, including CE feedback

- Alters Logic block internals, adds internal voters and performs feedback correction
- Physically separates triplicates on die

Block TMR (BTMR)

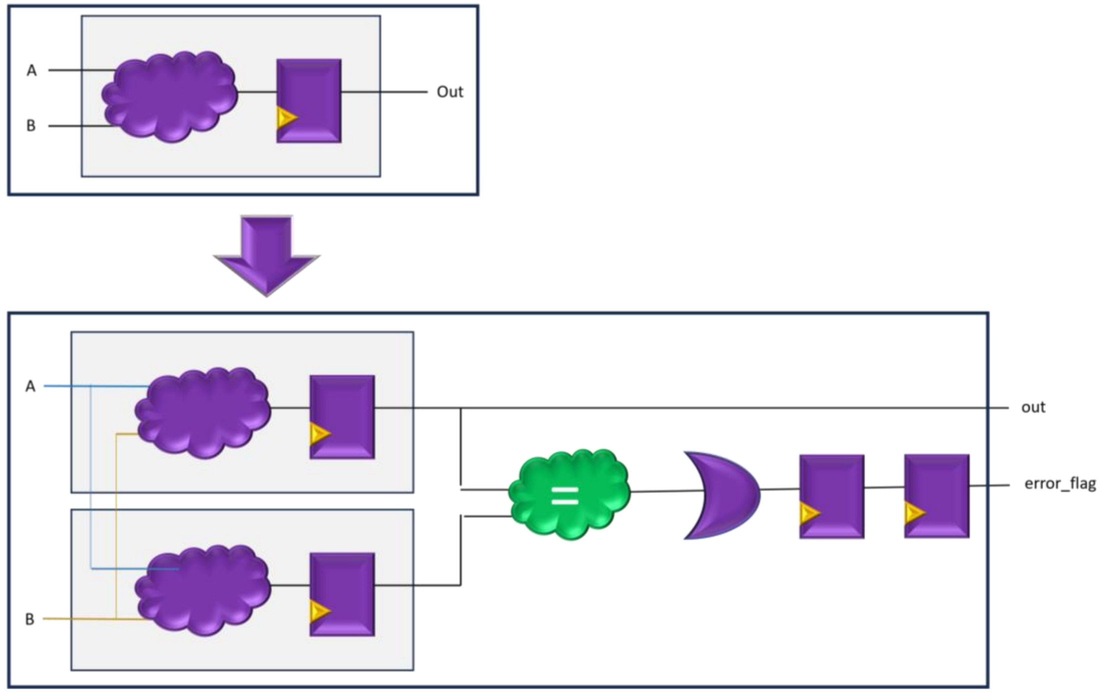
Protect synchronous IP, Clocks, Routing

- Block internals unaltered
- Physically separates triplicates on die



DWC (Duplicate with Compare)

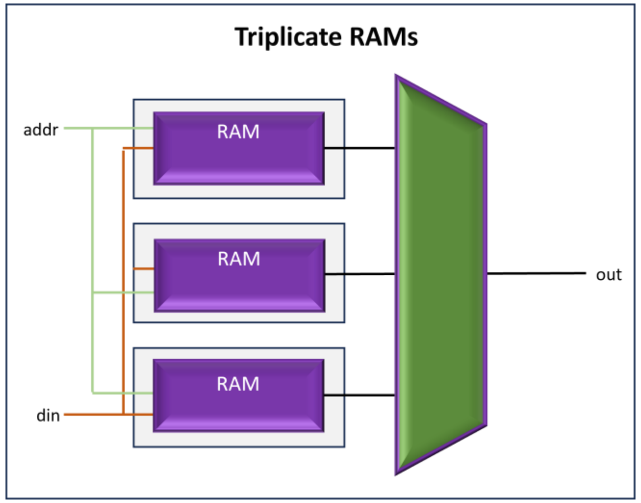
- Provides detection but not correction – less area than TMR
- Two identical copies of A (A0, A1) are created, and outputs compared
- Output from one of the replicated copies drives the original output OUT

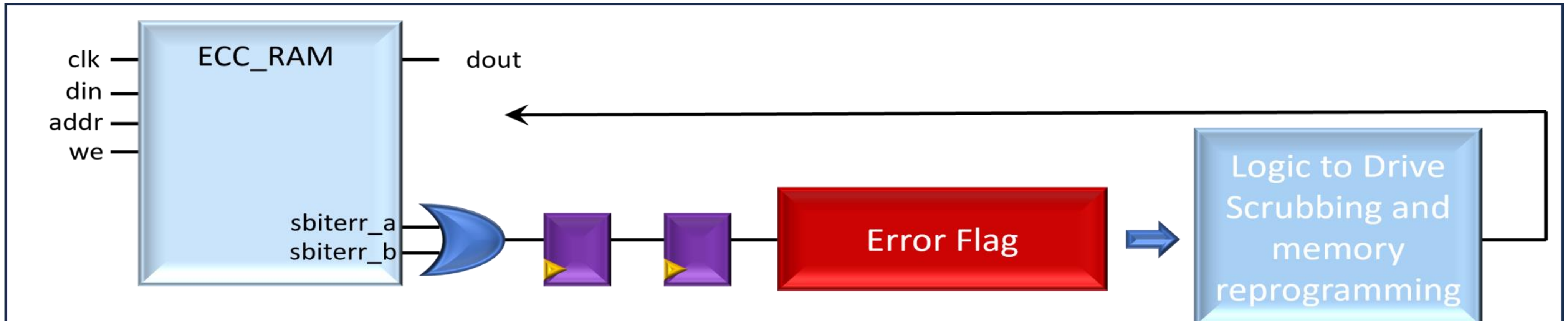


Synplify – Memory Protection

Memories | Debug

- Infer ECC on BlockRAM and URAM with built-in ECC
  - Data encryption and decryption within RAM IP
  - Provides single-bit and double-bit error ports
- Prevents false data from being propagated





Create error monitors to trigger RAM scrubbing

Safe/Fault Tolerant FSM

Single Event Upset (SEU) FSM Mitigation Techniques

Error Recovery

Safe Encoding FSM

Recovery logic that takes the state machine to “rst” state

Safe Case FSM

Recovery logic that takes the state machine to “default” or “others” clause state

```
always @ (posedge clk)
    if (rst)
        present_state <= s0;
    else
        present_state <= next_state;
always @ (in1,present_state)
    case (present_state)
        s0 :………..
        s1 : ……..
        s2 : ……
        s3 : ………… //no transition into S3
        default : next_state <= s3;
   endcase
```

SEU = Single Event Upset

Error Correction

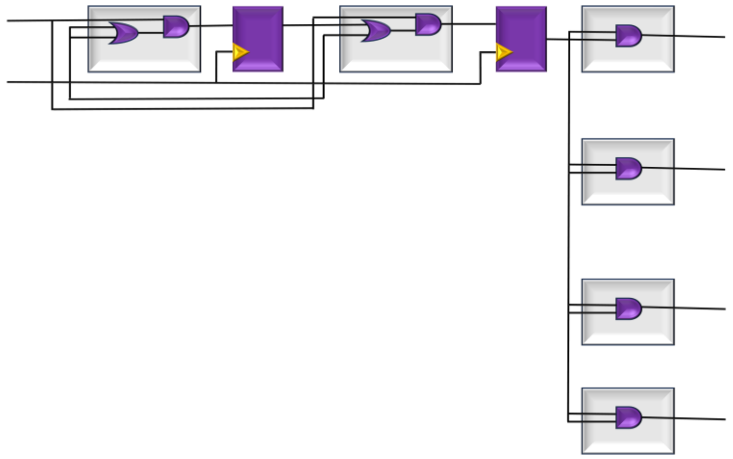
Hamming Dist3 FSM

Single-bit Error Correction (SEC) and Double-bit Error Detection (DED) with recovery to a “default” or “others” clause state

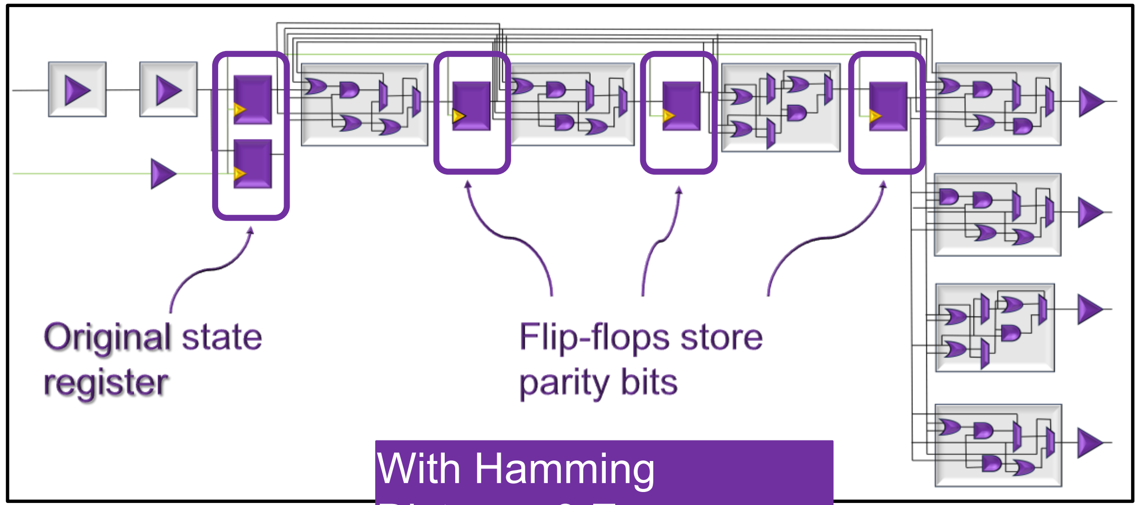
Safe/Fault Tolerant FSM

FSM Error Correction using Hamming Distance 3

Error mitigation technique to generate fault-tolerant FSMs with Hamming-3 encoding



Without Error Correction

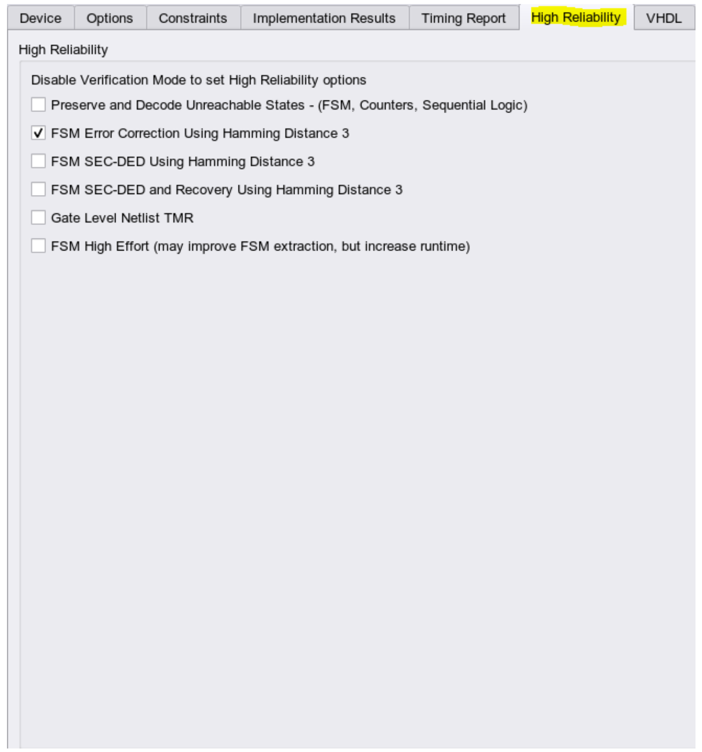


With Hamming Distance 3 Error Correction
Comparator circuits and parity flip-flops added

Implements FSM ***double-bit error detection*** (DED) and ***single-bit error correction*** (SEC) logic using Hamming-3 encoding

Synplify High Reliability Options for FSM

1. Preserve and Decode Unreachable States (FSM, Counters, Sequential Logic)
2. FSM Error Correction Using Hamming Distance 3
3. FSM SEC-DED Using Hamming Distance 3
4. FSM SEC-DED and Recovery Using Hamming Distance 3
5. Gate Level Netlist TMR
6. FSM High Effort (may improve FSM extraction, but increases runtime)



High Reliability

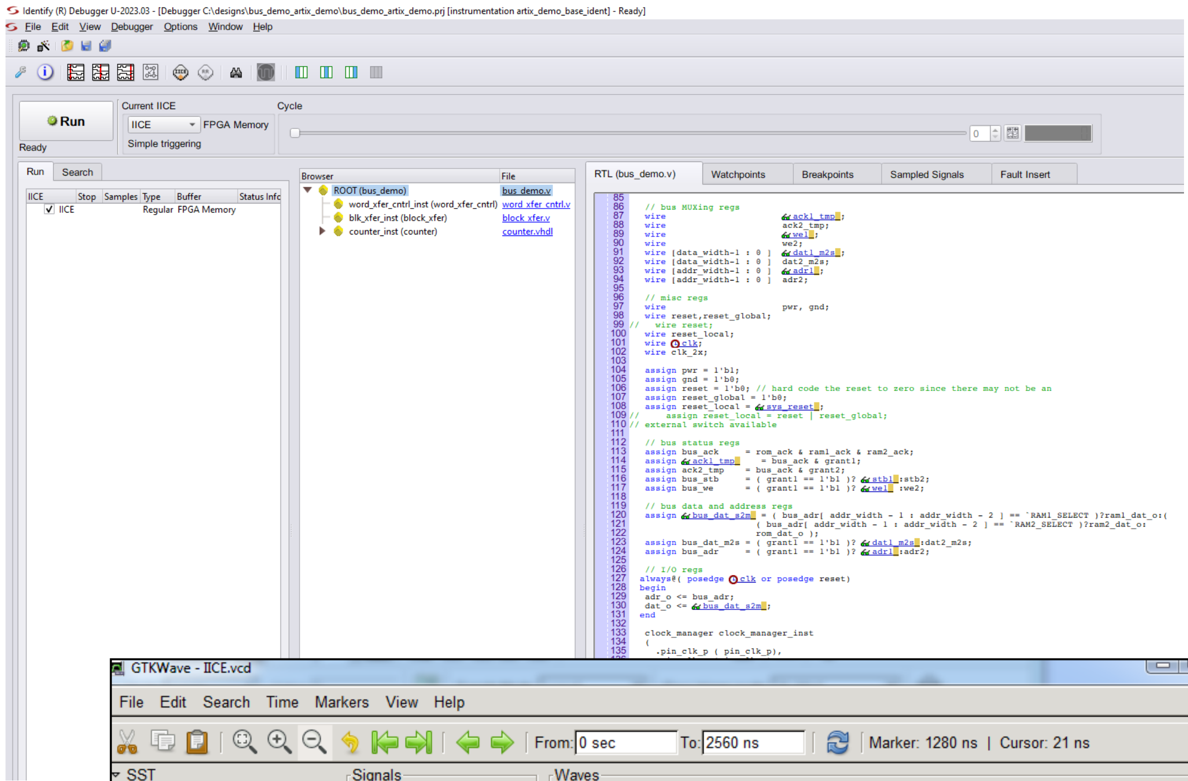
Disable Verification Mode to set High Reliability options

- [ ] Preserve and Decode Unreachable States - (FSM, Counters, Sequential Logic)
- [✓] FSM Error Correction Using Hamming Distance 3
- [ ] FSM SEC-DED Using Hamming Distance 3
- [ ] FSM SEC-DED and Recovery Using Hamming Distance 3
- [ ] Gate Level Netlist TMR
- [ ] FSM High Effort (may improve FSM extraction, but increase runtime)

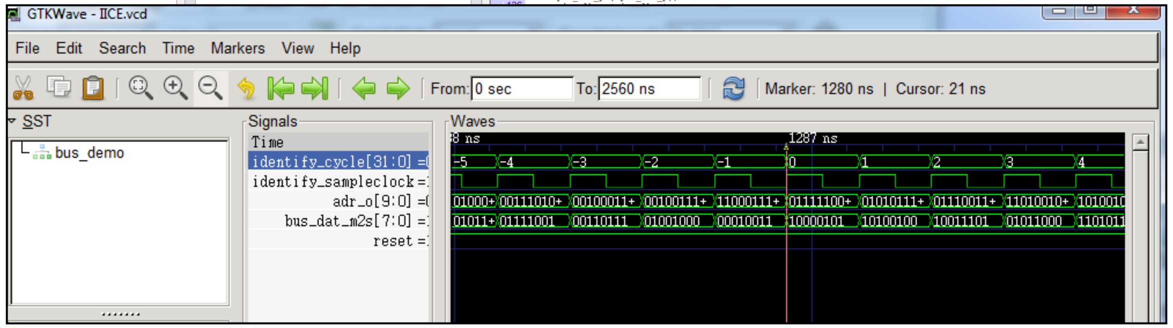
SYNOPSYS SYNPLIFY – DEBUG FOR HIGH RELIABILITY & FUNCTIONAL SAFETY

Synplify - Debugging for High Reliability Applications

Simulator-like Visibility into FPGA Hardware Operation


Identify (R) Debugger U-2023.03 - [Debugger C:\designs\bus_demo_artix_demo\bus_demo_artix_demo.prj [instrumentation artix_demo_base_ident] - Ready]
File Edit View Debugger Options Window Help
Run
Current IICE
IICE
FPGA Memory
Simple triggering
Cycle
Ready
Run
Search
IICE Stop Samples Type Buffer Status Info
IICE Regular FPGA Memory
Browser
File
ROOT (bus_demo) bus_demo.v
word_xfer_cntrl_inst (word_xfer_cntrl) word_xfer_cntrl.v
blk_xfer_inst (block_xfer) block_xfer.v
counter_inst (counter) counter.vhdl
RTL (bus_demo.v)
Watchpoints
Breakpoints
Sampled Signals
Fault Insert

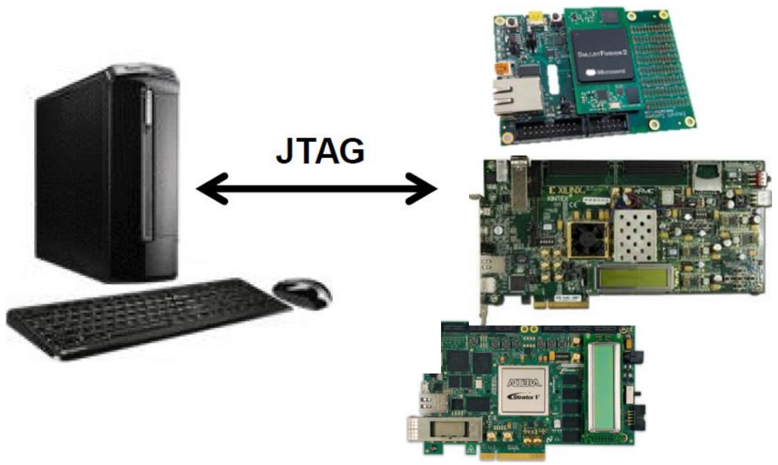
```
  // bus MUXing regs
  wire                           ack1_tmp;
  wire                           ack2_tmp;
  wire                           we1;
  wire                           we2;
  wire [data_width-1 : 0 ]       dat1_m2s;
  wire [data_width-1 : 0 ]       dat2_m2s;
  wire [addr_width-1 : 0 ]       adr1;
  wire [addr_width-1 : 0 ]       adr2;

  // misc regs
  wire                           pwr, gnd;
  wire reset,reset_global;
//   wire reset;
  wire reset_local;
  wire clk;
  wire clk_2x;

  assign pwr = 1'b1;
  assign gnd = 1'b0;
  assign reset = 1'b0; // hard code the reset to zero since there may not be an
  assign reset_global = 1'b0;
  assign reset_local = sys_reset;
//     assign reset_local = reset | reset_global;
// external switch available

  // bus status regs
  assign bus_ack      = rom_ack & ram1_ack & ram2_ack;
  assign ack1_tmp      = bus_ack & grant1;
  assign ack2_tmp     = bus_ack & grant2;
  assign bus_stb      = ( grant1 == 1'b1 )? stb1:stb2;
  assign bus_we       = ( grant1 == 1'b1 )? we1 :we2;

  // bus data and address regs
  assign bus_dat_s2m = ( bus_adr[ addr_width - 1 : addr_width - 2 ] == `RAM1_SELECT )?ram1_dat_o:(
                     ( bus_adr[ addr_width - 1 : addr_width - 2 ] == `RAM2_SELECT )?ram2_dat_o:
                     rom_dat_o );
  assign bus_dat_m2s = ( grant1 == 1'b1 )? dat1_m2s:dat2_m2s;
  assign bus_adr     = ( grant1 == 1'b1 )? adr1:adr2;

  // I/O regs
 always@( posedge clk or posedge reset)
 begin
  adr_o <= bus_adr;
  dat_o <= bus_dat_s2m;
 end

  clock_manager clock_manager_inst
  (
    .pin_clk_p ( pin_clk_p),
```



From: 0 sec To: 2560 ns Marker: 1280 ns | Cursor: 21 ns

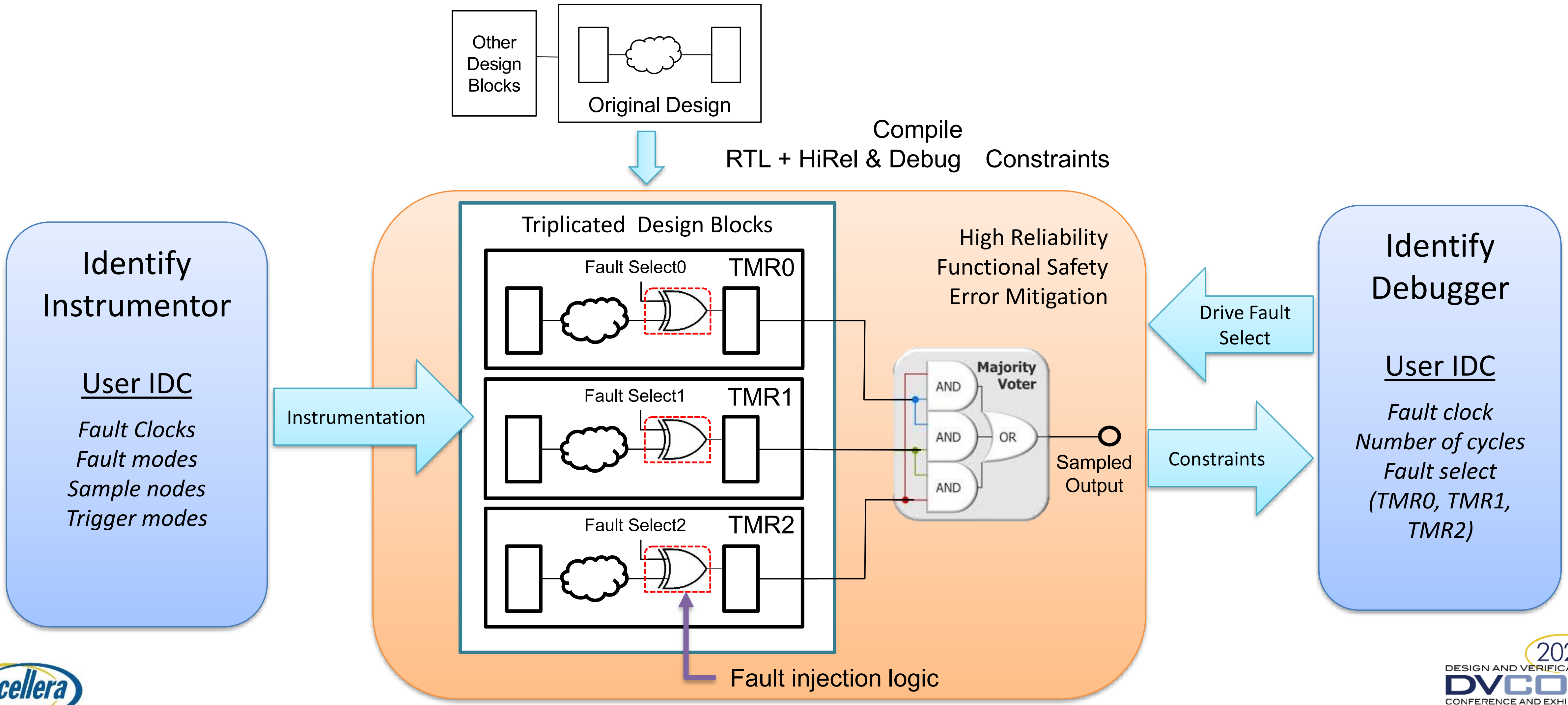
Signals	Waves
Time	
identify_cycle[31:0] =	-5 -4 -3 -2 -1 0 1 2 3 4
identify_sampleclock =	
adr_o[9:0] =	01000+ 00111010+ 00100011+ 00100111+ 11000111+ 01111100+ 01010111+ 01110011+ 11010010+ 1010010
bus_dat_m2s[7:0] =	01011+ 01111001 00110111 01001000 00010011 10000101 10100100 10011101 01011000 1101011
reset =	

- Internal visibility in target system at full speed
- Trigger on signals and control path
- Instrument and debug design directly in RTL source code
- Explore state spaces not easily reached by simulation
- Support for routing signals to external Logic Analyzer
- Automation via full TCL support

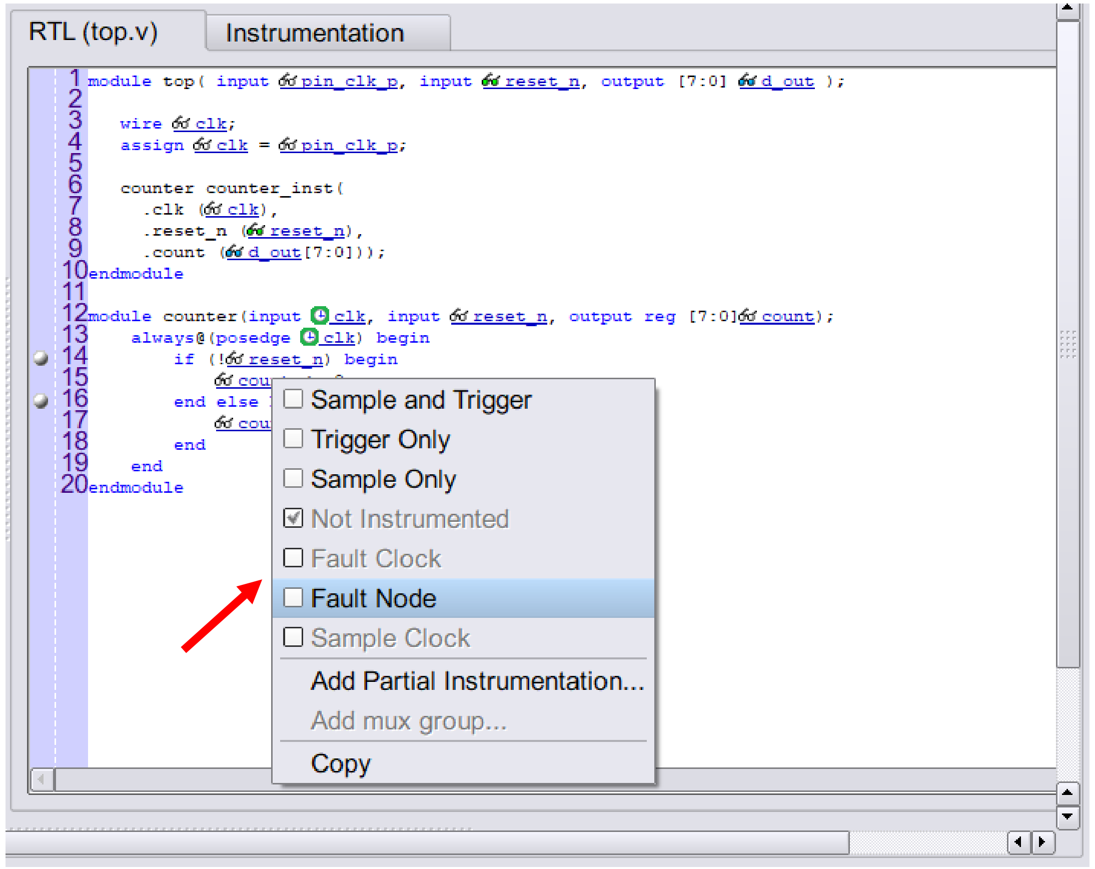


Automated Error Detection and Mitigation with Fault Injection Flow

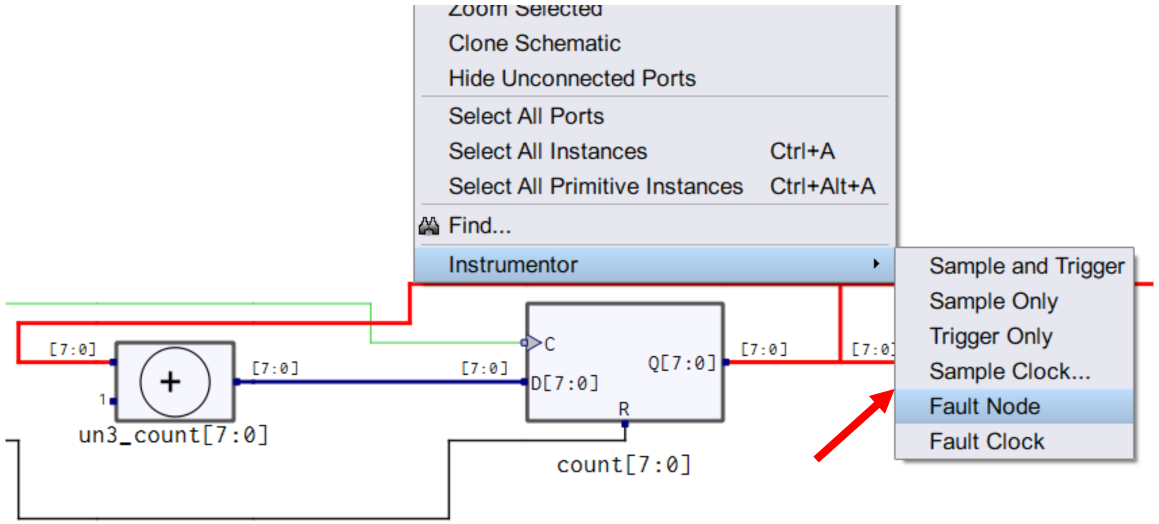
Voter effectiveness testing



Configuring Fault Nodes using Identify Instrumentor

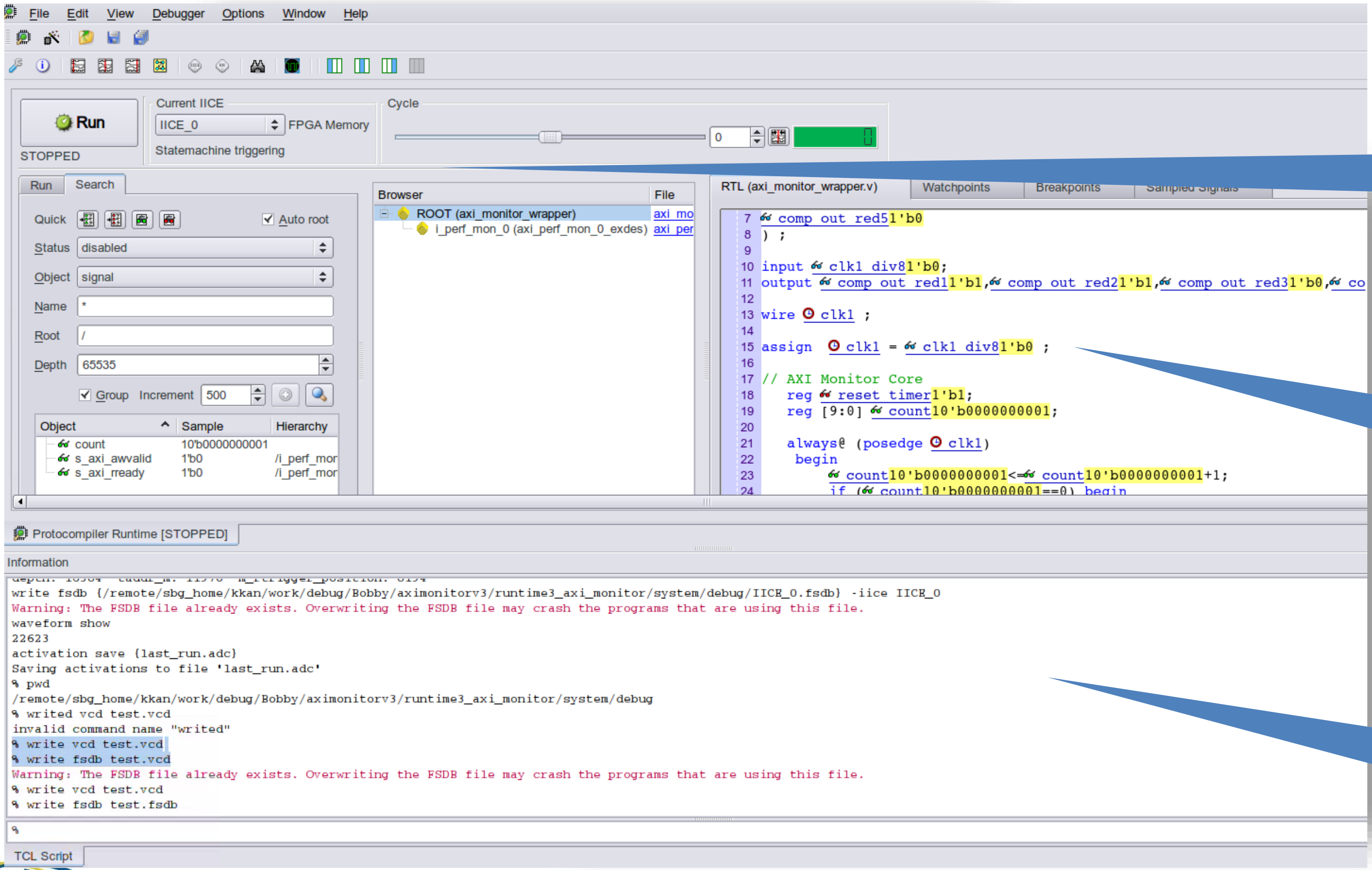


```
module top( input pin_clk_p, input reset_n, output [7:0] d_out );

   wire clk;
   assign clk = pin_clk_p;

   counter counter_inst(
     .clk  (clk),
     .reset_n (reset_n),
     .count  (d_out[7:0]));
endmodule

module counter(input clk, input reset_n, output reg [7:0]count);
    always@(posedge clk) begin
        if (!reset_n) begin
           cou
        end else
           cou
        end
    end
endmodule
```



Fault Nodes designate nets on which faults will be injected Via tool Schematic View

Using Identify Debugger for Data Debug on the FPGA



Quick ✓ Auto root
Status: disabled
Object: signal
Name: *
Root: /
Depth: 65535
✓ Group Increment 500

Object	Sample	Hierarchy
count	10'b0000000001	
s_axi_awvalid	1'b0	/i_perf_mor
s_axi_rready	1'b0	/i_perf_mor

ROOT (axi_monitor_wrapper) axi_mo
i_perf_mon_0 (axi_perf_mon_0_exdes) axi_per

```
comp out red51'b0
) ;

input clk1 div81'b0;
output comp out red11'b1, comp out red21'b1, comp out red31'b0, co

wire clk1 ;

assign clk1 = clk1 div81'b0 ;

// AXI Monitor Core
   reg reset timer1'b1;
   reg [9:0] count10'b0000000001;

   always@ (posedge clk1)
    begin
        count10'b0000000001<= count10'b0000000001+1;
        if ( count10'b0000000001==0) begin
```

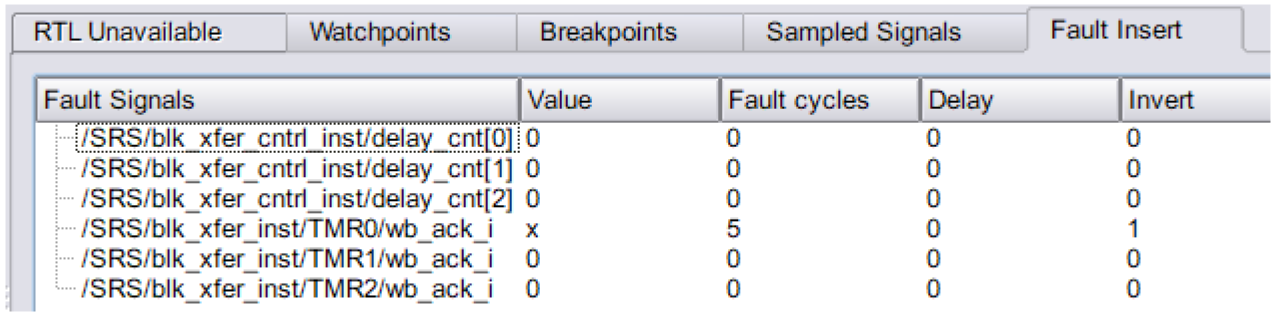
```
write fsdb {/remote/sbg_home/kkan/work/debug/Bobby/aximonitorv3/runtime3_axi_monitor/system/debug/IICE_0.fsdb} -iice IICE_0
Warning: The FSDB file already exists. Overwriting the FSDB file may crash the programs that are using this file.
waveform show
22623
activation save {last_run.adc}
Saving activations to file 'last_run.adc'
% pwd
/remote/sbg_home/kkan/work/debug/Bobby/aximonitorv3/runtime3_axi_monitor/system/debug
% writed vcd test.vcd
invalid command name "writed"
% write vcd test.vcd
% write fsdb test.vcd
Warning: The FSDB file already exists. Overwriting the FSDB file may crash the programs that are using this file.
% write vcd test.vcd
% write fsdb test.fsdb
%
```

Use cycle control to traverse sample buffer

View data values from FPGA annotated onto source code or schematic

Dynamically change trigger conditions

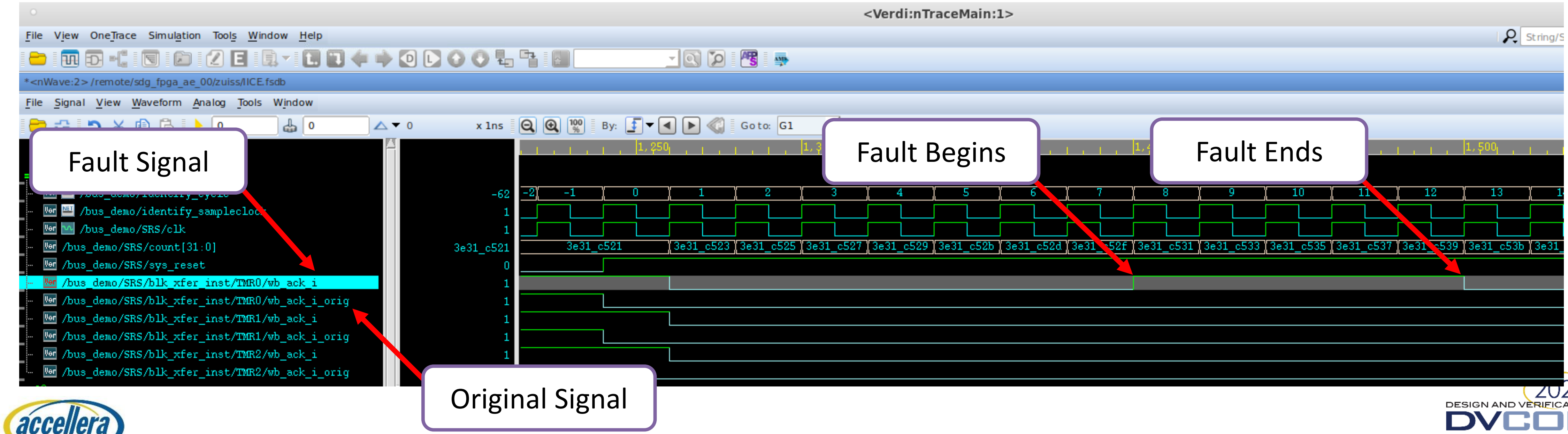
Viewing Faults



RTL Unavailable | Watchpoints | Breakpoints | Sampled Signals | Fault Insert

Fault Signals	Value	Fault cycles	Delay	Invert
/SRS/blk_xfer_cntrl_inst/delay_cnt[0]	0	0	0	0
/SRS/blk_xfer_cntrl_inst/delay_cnt[1]	0	0	0	0
/SRS/blk_xfer_cntrl_inst/delay_cnt[2]	0	0	0	0
/SRS/blk_xfer_inst/TMR0/wb_ack_i	x	5	0	1
/SRS/blk_xfer_inst/TMR1/wb_ack_i	0	0	0	0
/SRS/blk_xfer_inst/TMR2/wb_ack_i	0	0	0	0

- Faults are configured in Identify Debugger
- Waveforms can be viewed in Synopsys Verdi



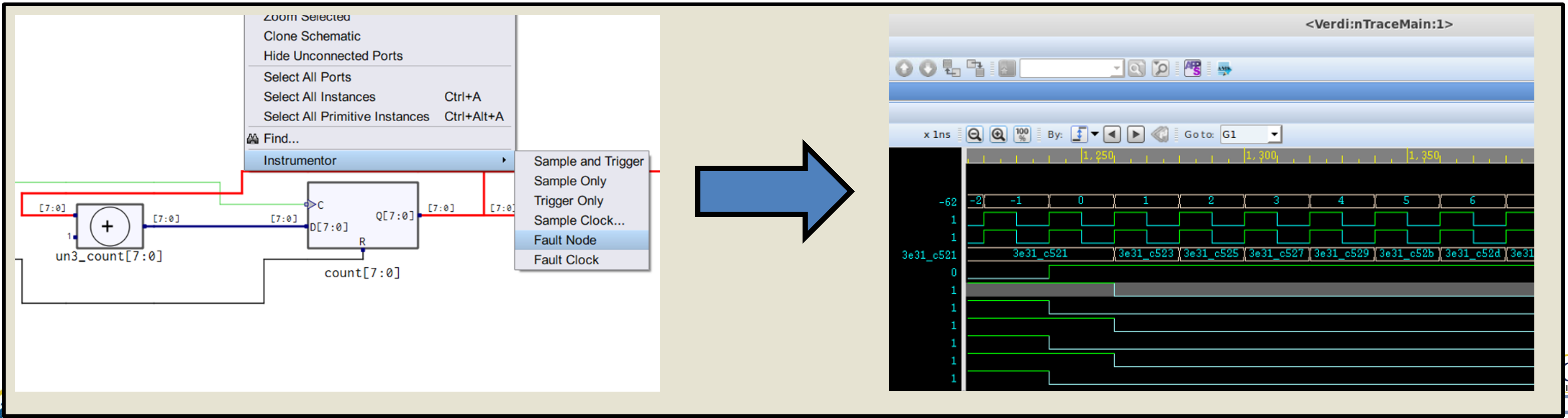
<Verdi:nTraceMain:1>
File View OneTrace Simulation Tools Window Help
*<nWave:2> /remote/sdg_fpga_ae_00/zuiss/ICE.fsdb
File Signal View Waveform Analog Tools Window
x 1ns
Go to: G1
Fault Signal
Fault Begins
Fault Ends
Original Signal

```
/bus_demo/identify_cycle                    -62
/bus_demo/identify_sampleclock                1
/bus_demo/SRS/clk                             1
/bus_demo/SRS/count[31:0]             3e31_c521
/bus_demo/SRS/sys_reset                       0
/bus_demo/SRS/blk_xfer_inst/TMR0/wb_ack_i      1
/bus_demo/SRS/blk_xfer_inst/TMR0/wb_ack_i_orig 1
/bus_demo/SRS/blk_xfer_inst/TMR1/wb_ack_i      1
/bus_demo/SRS/blk_xfer_inst/TMR1/wb_ack_i_orig 1
/bus_demo/SRS/blk_xfer_inst/TMR2/wb_ack_i      1
/bus_demo/SRS/blk_xfer_inst/TMR2/wb_ack_i_orig
```

Fault Injection Summary

- Identify Instrumentor
  - Instrument Safety Critical portions of a design
- Identify Debugger
  - Assert faults on internal signals of a running FPGA
- Capture and view waveforms
  - Quickly verify effects of High Reliability features

Quickly verify High Reliability in hardware using Fault Injection



SYNOPSYS SYNPLIFY – HIGH RELIABILITY CERTIFICATION

Synplify Enables DO-254 Compliant Process

Traceability

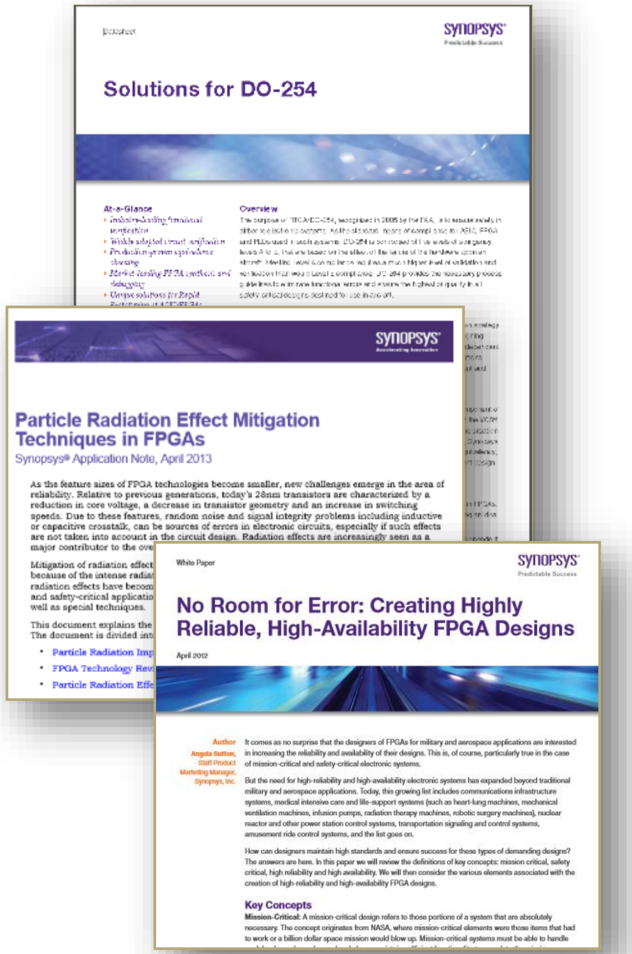
Schematics for Documentation
Timing Reports and Log files
Safe FSM and TMR Reports

Repeatability

Reproducible synthesis results
Lock down pre-verified blocks
Best Practices App note for Safety-critical design

Equivalence

Identify/Simulator Integration
Sequential preservation control for equivalence assurance and requirements trace



Datasheet

Solutions for DO-254

Particle Radiation Effect Mitigation Techniques in FPGAs
Synopsys® Application Note, April 2013

White Paper

No Room for Error: Creating Highly Reliable, High-Availability FPGA Designs

April 2012

Author
Angela Sutton,
Staff Product
Marketing Manager,
Synopsys, Inc.

It comes as no surprise that the designers of FPGAs for military and aerospace applications are interested in increasing the reliability and availability of their designs. This is, of course, particularly true in the case of mission-critical and safety-critical electronic systems.

But the need for high-reliability and high-availability electronic systems has expanded beyond traditional military and aerospace applications. Today, this growing list includes communications infrastructure systems, medical intensive care and life-support systems (such as heart-lung machines, mechanical ventilation machines, infusion pumps, radiation therapy machines, robotic surgery machines), nuclear reactor and other power station control systems, transportation signaling and control systems, amusement ride control systems, and the list goes on.

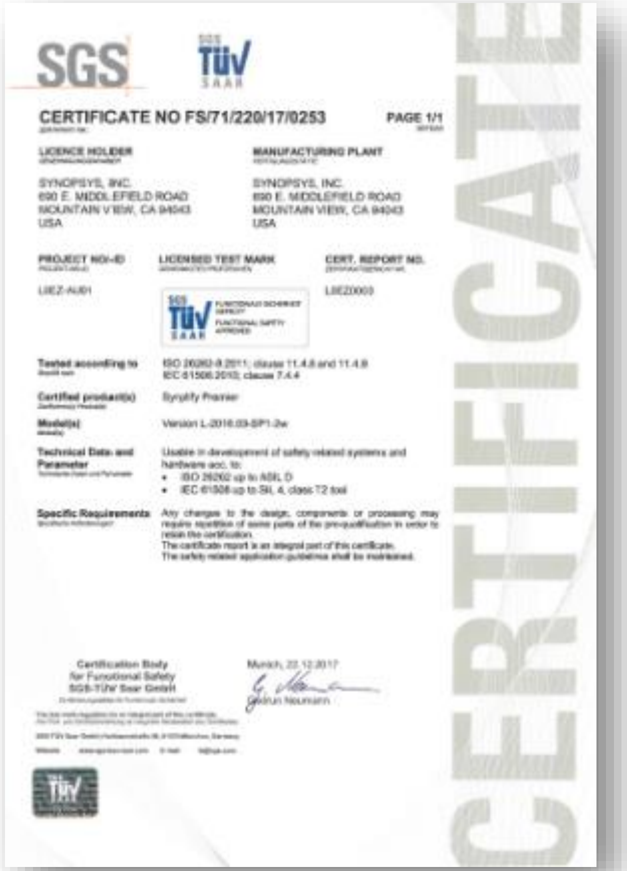
How can designers maintain high standards and ensure success for these types of demanding designs? The answers are here. In this paper we will review the definitions of key concepts: mission critical, safety critical, high reliability and high availability. We will then consider the various elements associated with the creation of high-reliability and high-availability FPGA designs.

Key Concepts

Mission-Critical: A mission-critical design refers to those portions of a system that are absolutely necessary. The concept originates from NASA, where mission-critical elements were those items that had to work or a billion dollar space mission would blow up. Mission-critical systems must be able to handle

Synplify-Certification for High Reliability

- Deployment of functional safety synthesis
  - Aerospace and Defense, Automotive and Industrial
- Strong investment in functional safety
  - R&D and certification
- Synplify functional safety certifications
  - Version: L-2016.03-SP1-2w & later
  - ISO26262, up to ASIL D
  - IEC61508, up to SIL 4



CERTIFICATE NO FS/71/220/17/0253 PAGE 1/1

LICENCE HOLDER	MANUFACTURING PLANT
SYNOPSYS, INC. 690 E. MIDDLEFIELD ROAD MOUNTAIN VIEW, CA 94043 USA	SYNOPSYS, INC. 690 E. MIDDLEFIELD ROAD MOUNTAIN VIEW, CA 94043 USA

PROJECT NO/-ID	LICENSED TEST MARK	CERT. REPORT NO.
LBEZ-AU01	FUNCTIONAL SAFETY APPROVED	LBEZ0003

Tested according to	ISO 26262-8:2011; clause 11.4.8 and 11.4.9 IEC 61508:2010; clause 7.4.4
Certified product(s)	Synplify Premier
Model(s)	Version L-2016.03-SP1-2w
Technical Data and Parameter	Usable in development of safety related systems and hardware acc. to: • ISO 26262 up to ASIL D • IEC 61508 up to SIL 4, class T2 tool
Specific Requirements	Any changes to the design, components or processing may require repetition of some parts of the pre-qualification in order to retain the certification. The certificate report is an integral part of this certificate. The safety related application guidelines shall be maintained.

Certification Body
for Functional Safety
SGS-TÜV Saar GmbH

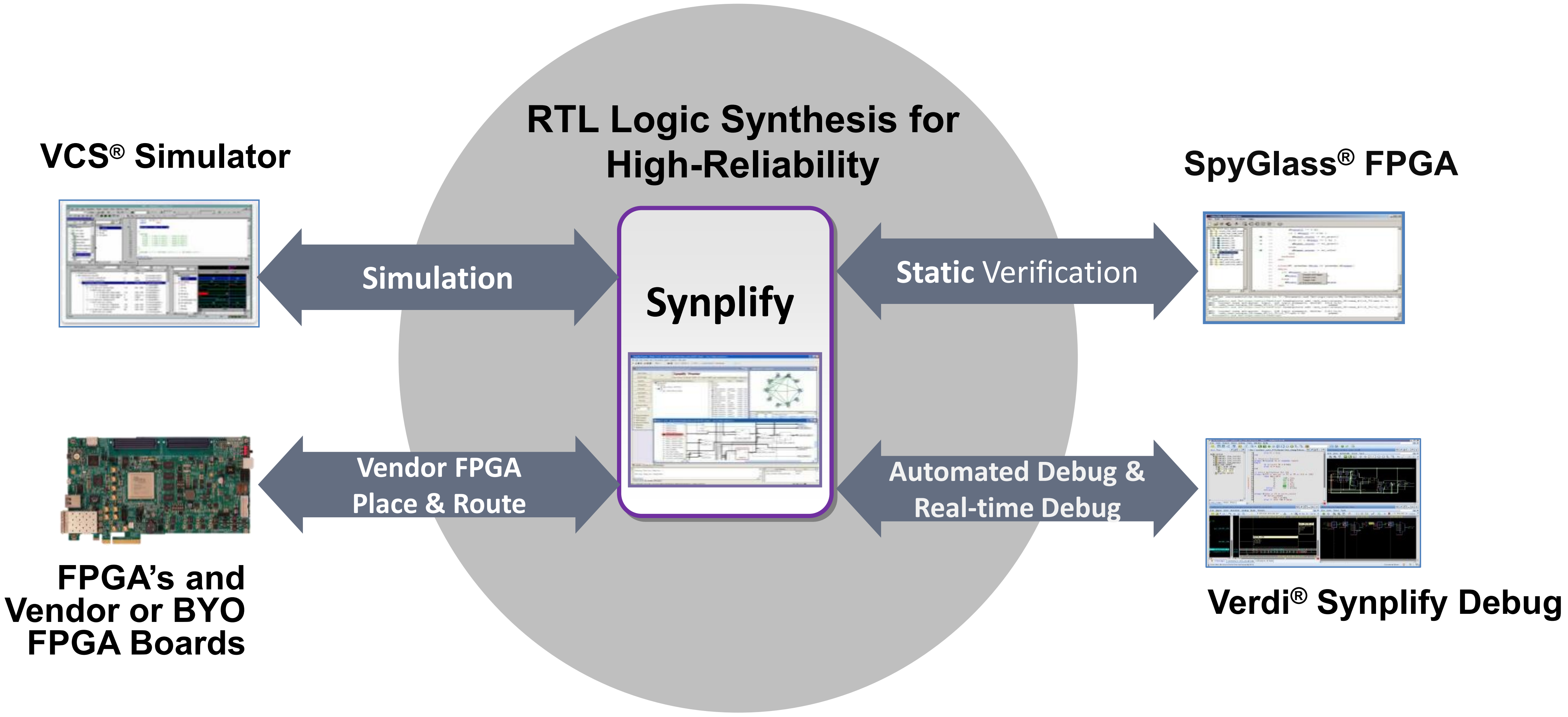
Munich, 22.12.2017

Gudrun Neumann

CERTIFICATE

https://www.sgs-tuev-saar.com/en/certification-database.html

Tight Flow Integrations for Easy Design Bring-up


RTL Logic Synthesis for High-Reliability
VCS® Simulator
SpyGlass® FPGA
Simulation
Synplify
Static Verification
Vendor FPGA Place & Route
Automated Debug & Real-time Debug
FPGA's and Vendor or BYO FPGA Boards
Verdi® Synplify Debug

Synplify High Reliability & Functional Safety Summary – Key Values

Fast Design Setup
Language support
IP handling
Constraint setup

High Reliability & FuSa

TMR, Hamming3
ECC, Error Flag,
Fault Injection

Fast Bring-Up
Continue on Error
Fast synthesis
Constraint checking

Easy System Debug
In-system debug
Bitstream to RTL
Incremental re-route

High Performance
Accelerated runtime
Advanced Synthesis
Post P&R re-synthesis
Exploratory P&R

Quality Support

Q&A

Thank You

For more information visit www.synopsys.com/fpga