High Reliability & Functional Safety for FPGAs with Synplify & Radiant

Phil Simpson, Lattice Semiconductor & Khalid Khan, Synopsys

A Lattice Semiconductor White Paper.

March 2024

Learn more:

www.latticesemi.com

Contact us online:

www.latticesemi.com/contact
www.latticesemi.com/buy

TABLE OF CONTENTS

Background

Advances in design and manufacturing technology allow increased factory automation, where tasks are automatically performed by sophisticated equipment such as industrial robots. Manufacturing processes require fail-safe mechanisms to prevent human injury or costly downtime. With increased sophistication and automation of the manufacturing processes, there is an increasing need for error detection and recovery methods.

In addition to industrial applications, the rise of autonomous driving and driver assisted vehicles is creating a demand for functional safety design automation. For example, Advanced Driver Assistance Systems (ADAS) include features such as adaptive cruise control, automatic braking, and lane departure warning systems. These features require safe operation of vehicles in the event of random hardware or system failures. These types of failures require safety features that can detect errors and provide warnings to the driver, or automatically correct and maintain the vehicle in a safe operating state.

Lattice FPGA Devices

The Lattice Nexus™ FPGA development platform offers a new class of small density SRAM-based devices which provides a true differentiator for FPGAs destined for use in state-of-the-art systems performing mission-critical and safety-critical applications for the Commercial, Industrial, Communications, Defense, Aerospace, and automotive applications.

The Lattice Nexus platform based small FPGA devices deliver performance/power optimization using Programmable back bias enabled by insulated gate of FD-SOI technology. In addition, the devices have added capabilities that are necessary for high reliability designs. Critical area size reduction in Nexus provides 100X SER (Soft Error Rate) reliability improvement over competing FPGA devices. The Nexus based FPGAs include dedicated intellectual property (IP) blocks that automatically perform ECC (Error Correcting Code)-based memory scrubbing as a background process. Nexus FPGAs also have an SED/SEC block built into the configuration memory to facilitate rapid detection and correction of errors on a frame-by-frame basis without the need for external circuitry.

The Lattice Avant™ 16nm FinFET platform is the foundation for industry leading low power and small form factor mid-range SRAM-based FPGA families. The high-end features of the Avant platform make it suitable for networking controllers, PLCs, Edge computers, machine vision, and Industrial robotics applications in the Industrial market. The DSP performance makes it suitable for Automotive networking and software defined radio applications, as well as general wireless communications applications.

The Avant mid-range FPGA devices are built upon 16nm low power FinFET technology with the use of low leakage transistors to balance power and performance. The family includes leading edge reliability features with updated fast built-in Soft Error Detection, built-in Soft Error Correction and ECC on memory blocks and external memory interfaces. Avant devices have dedicated logic to perform Cycle Redundancy Code (CRC) checks for the entire bitstream, which runs in parallel along with ECC. There are two layers of SED/SEC, ECC logic to detect and correct single bit error per data frame and detect two-bit errors and CRC logic to detect multi-bit errors in the device.

FPGA Design Tools

The Radiant™ and Synopsys Synplify software provide a full range of features to enable successful design of Lattice FPGA devices in Safety Critical Applications.

The Radiant 2023.2 software provides support for the Nexus and Avant platforms and adds the macro-based safety critical design flow capability.

The Synopsys Synplify 2023.09 release is the first release of Synplify synthesis tool to provide support for the Lattice Nexus and Avant platforms. This release provides Advanced Triple Module Redundancy (TMR), Safe & Fault Tolerant Finite State Machine (FSM) support, memory correction and advanced debug capabilities such as Error monitoring, fault simulation and hardware fault injection for the Nexus, Avant, and mature Lattice platforms.

High Reliability Design Techniques

Macro Based Safety Critical Design Flow

- **What is a Safety Critical Design Flow**
  The goal of the safety critical design flow is to ensure that the design implementation of identified 'Secure' design block or blocks do not change and that the content of the 'Secure' design blocks can only be accessed by identified signals from within the design.

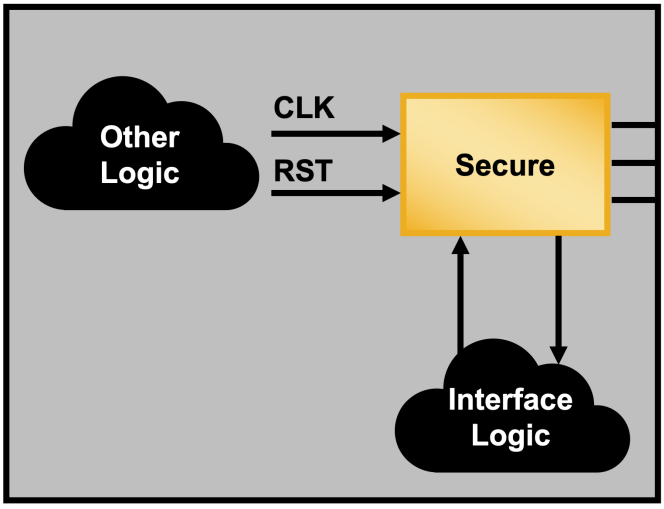

Figure 1:
Safety Critical Design

In the Safety critical Design flow, the 'secure' area only contains logic from the secure design block. It prohibits logic or routing from using this region with the exception of user approved signals, i.e. user defined interfaces to other blocks. Once the secure design block is complete (functionally correct and timing closed) the logic in the secure region can be locked at both the placement and routing level. This is achieved in the Radiant software using the macro design flow.

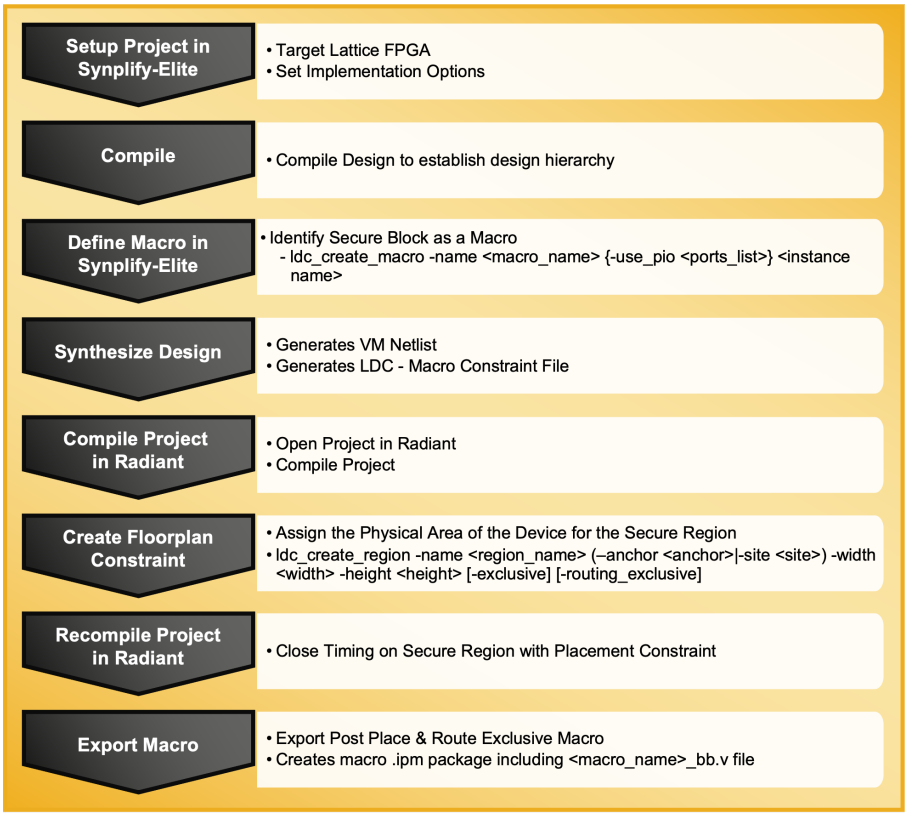

Figure 2:
Creating a Post P&R Macro for a secure region.

Once satisfied that the design is functionally correct and timing closed, the designer will export the post-place and route macro for the secure block as an exclusive macro. The exclusive macro is key to the safety critical design flow. This option prevents the rest of the design from using any unused resources, such as LUT's, BRAM, DSPs and routing within the secure region. Only signals that are connected at the RTL level and control signals such as clocks and resets can feed into the secure region. The exported macro will be used in all future compilations of the complete design. The <macro_name>.bbv file is used in place of the RTL design for the secure design block in all subsequent compiles in Synplify for changes in the rest of the design. Synplify will create a VM netlist with the macro as a black_box. This ensures that Synplify does not resynthesize the secure macro and any future compiles in Radiant uses the exported macro with locked place and route. This guarantees the same place and route on the 'secure' region and prohibits the use of resources in the macro region by other parts of the design, thus ensuring that there are no changes to the secure design block.

- **Error Detection & Correction**

For certain applications, detection of an error is critical. A standard practice to protect against these areas is to build redundant circuitry into the design. In many cases, this is achieved by performing TMR. In applications which have area constraints, a designer may use custom methods to correct errors. Such methods can include applying global reset to the design and scrubbing configuration memories on the FPGA. In these cases, duplication of logic susceptible to an error could be sufficient. A comparator can be built on the output of duplicated logic to detect error and designers can use an error flag to implement their custom error correction method. The area penalty using this method is lower compared to full triplication techniques as there are only two copies of the

modules instead of three. Synplify provides methods to duplicate modules and add comparator logic automatically to FPGA designs for state machines, memory, and logic. It also provides ways to access the comparator output to monitor errors and trigger necessary error mitigation logic.

The Nexus and Avant FPGAs include dedicated IP blocks that automatically perform ECC-based memory scrubbing as a background process.

Nexus FPGAs have an SED/SEC block built into the configuration memory, in order to facilitate rapid detection and correction of errors on a frame-by-frame basis without the need for external circuitry.

The Avant devices have an improved, hardware implemented, SED circuit which can be used to detect SRAM errors so they can be corrected, either automatically or after notification and consent. The SED hardware in Lattice Avant devices is part of the Configuration block. The SED hardware reads data from the FPGAs configuration memory and performs an ECC calculation on every frame of configuration data. With Automatic operation, once a single error is detected it is corrected, a notification is generated and SED resumes operation. With Consent operation, once a single error is detected the SED hardware halts and a fabric notification is generated. Upon consent from the fabric, the single error is corrected and the SED resumes operation. In all modes, if more than one-bit error is detected within one frame of configuration data, a fabric error notification is generated, and the SED continues operation.

Redundancy

One of the commonly used methods for protection is to add redundancy. TMR technique involves triplicating the logic and using single or multiple majority voter(s) to determine the correct output. This scheme provides protection if one of the copies of the triplicate logic is affected by an error, the remaining two copies help determine the correct output. The amount of logic to be triplicated depends on the type of logic that is being protected, e.g. Triplication of registers uses less area than triplication of entire blocks of combinational logic.

Distributed TMR

Logic blocks can contain sequential feedback loops. A fault introduced in sequential feedback loops can retain an error over multiple clock cycles even if the enclosing logic block is subjected to TMR. So it’s important to add a majority voter at the output of registers which are part of feedback loops. Distributed TMR, a feature available in Synplify, allows designers to triplicate blocks of logic and add majority voters to the output of blocks and in the sequential feedback loops.

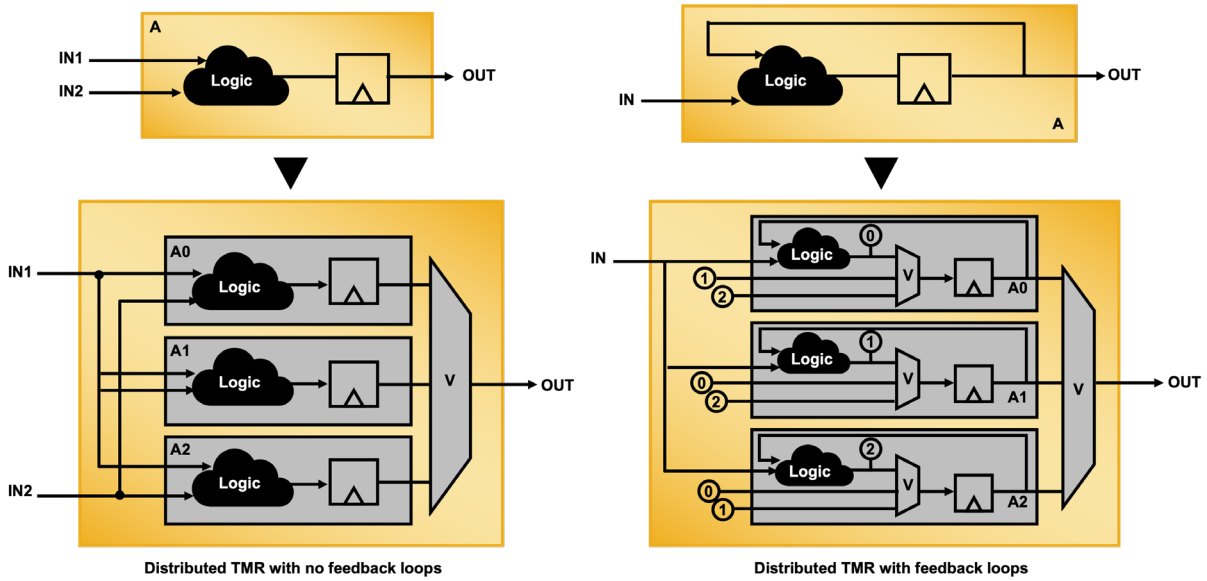

Figure 3:
Distributed TMR

Designs can have multiple interconnected blocks which require protection and by using Synplify, the designer can have the triplication and voter circuitry added automatically.

Multiple Module TMR

Some applications require extra protection for every path in the design. Synplify provides a method for this level of protection and can add a majority voter at every register in the triplicated copies of the block that are subjected to distributed TMR.

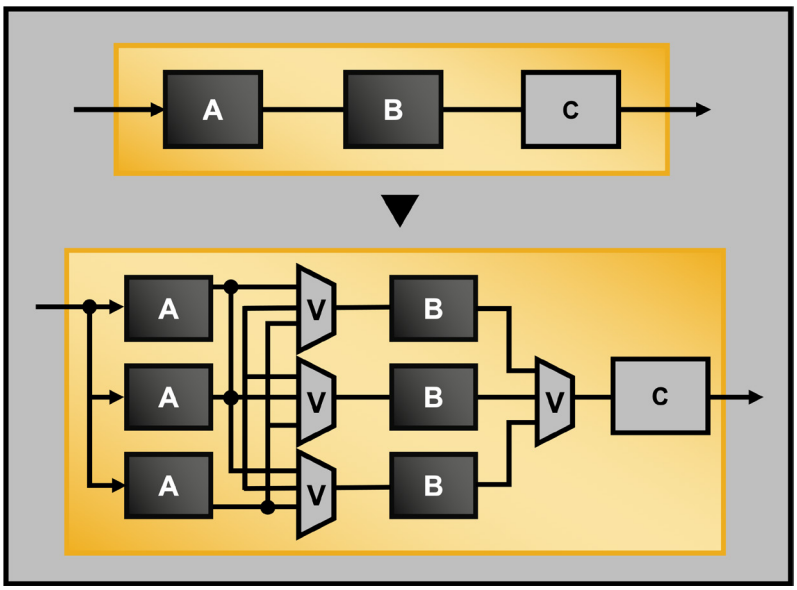

Figure 4:
Multi-module TMR

Block TMR

On the other hand, some users cannot afford the cost of voters at the sequential feedback loops in the design. In this case, the user can apply distributed TMR with voter logic only at the output of modules. This is referred to as block TMR.

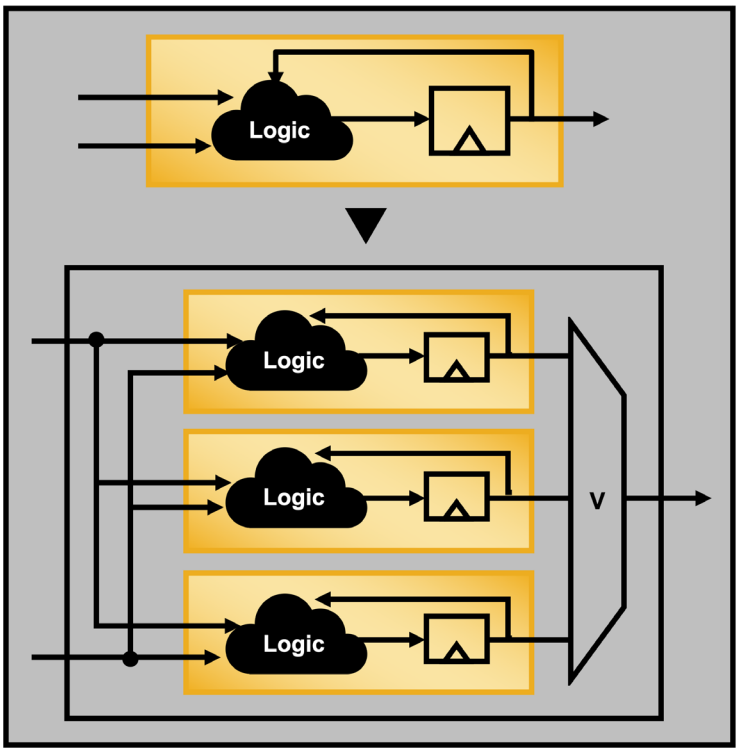

Figure 5:
Block TMR

With block TMR, if an error affects a register in feedback loop, it can cause accumulation resulting in a non-functional design. The designer can incorporate ways to detect and mitigate such errors by resetting the block into the design.

State Machines

Protecting FSM registers is critical for reliable design execution and allowing the state machine to operate in its legal states. A soft error can create an illegal state within an FSM which could cause deadlock and therefore the design to fail. Although the techniques described above such as distributed TMR or local TMR can help FSMs as well, it increases FPGA utilization and can impact design performance.

To address these types of issues, Synplify provides integrated methods to detect errors on a state machine register and reset to a valid state.

```
always @(state)
    begin
        case (state)
            zero:
                out = 4'b0000;
            one:
                out = 4'b0001;
            two:
                out = 4'b0010;
            default:
                out = 4'b0000;
        endcase
    end
always @(posedge clk or posedge reset)
    begin
        if (reset)
            state = zero;
        else
            case (state)
                zero:
                    state = one;
                one:
                    if (in)
                        state = zero;
                    else
                        state = two;
                two:
                    state = zero;
            endcase
```

Figure 6:
Example State Machine

Safe encoding: With safe encoding, if an FSM gets into an illegal state because of SEU, a reset signal is applied to the state machine to bring it into a reset state. The reset signal can be asynchronous or synchronous. Synplify supports syn_encoding = safe as an attribute on FSM. In the example in figure x, the FSM has 4 states (s0, s1, s2 and s3). If the FSM goes into an illegal state because of an error, logic is added to reset the state machine to state s0.

Safe state machine: If an FSM gets into illegal state, it goes into the default state described in the RTL in the following clock cycle. In the state machine shown above, the default state is s3 and the designer needs to implement the necessary logic to mitigate the effect of the errors.

Hamming-3 encoding: This is used to encode the state of an FSM and allow detection and correction of a one-bit error. If one of the state registers changes due to an error, hamming-3 encoding identifies the correct state. An option to detect bit errors in FSM and correct it in the following clock cycle is supported.

RAMs

The Nexus and Avant platforms support Error Code Correction logic for Block RAMs (EBR). In the case of Avant, the EBR has a built in ECC engine. The ECC engine supports a write data width of 64 bits, and it can be cascaded for larger data widths such as x128, ECC read may be performed in x1, x2, x4, x8, x16, x32, or x64 modes. The ECC parity generator creates and stores parity data for each 64-bit word written. When a read operation is performed, it compares the data with its associated parity data and reports back if any Single Event Upset (SEU) event has changed the data. Any single-bit data change is automatically corrected at the data output. In addition, two dedicated error flags indicate if a single-bit or two-bit error has occurred. The use of the ECC is an option that can be enabled in the BRAM via the IP. An example is shown in figure 7.

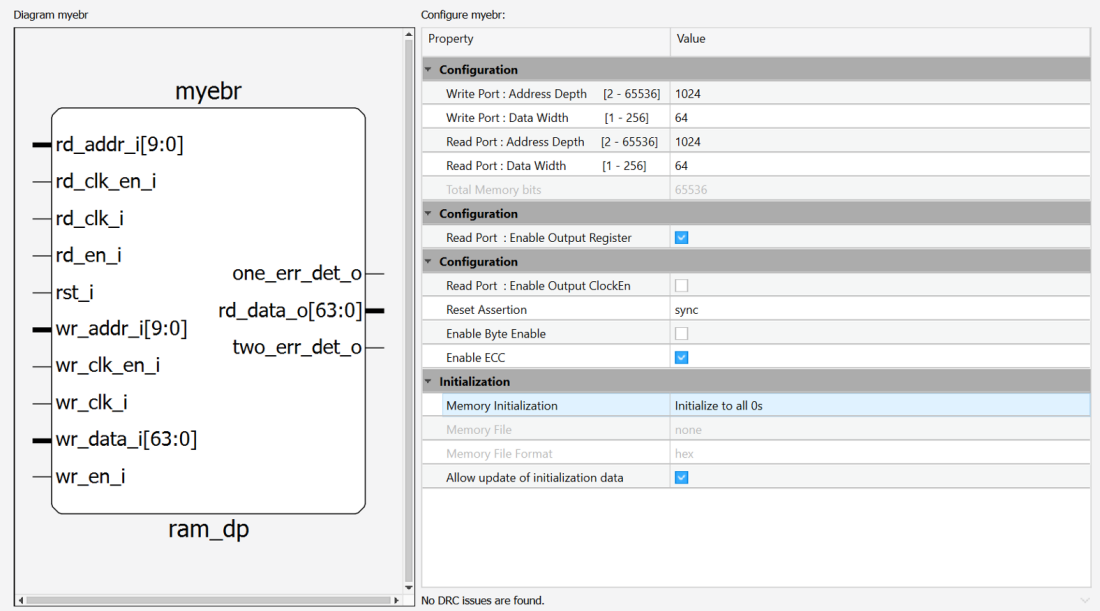

Figure 7:
Lattice Dual-Port Block RAM IP

Synplify TMR of Block RAM provides correction for multiple bit errors but requires three times more Block RAM resources. Since FPGAs have limited number of Block RAMs, this may not be feasible approach for all applications.

To use TMR for Block RAM, a user can enclose RAM in a module and apply Block or Distributed TMR or specify local TMR for the Block RAM.

Conclusion

A successful and fast implementation of a highly reliable FPGA design requires the right combination of:
1. FPGA silicon features applicable to high reliability applications.
2. Safety Critical Design Flow.
3. Automated high-reliability design flow techniques.

The Lattice Radiant 2023.2 software release together with Synopsys Synplify software provides the full range of features to enable successful high reliability design implementation in Lattice Nexus and Avant FPGA devices.