Reset and Initialization, the Good, the Bad and the Ugly

Ping Yeung, Mentor Graphics
Kaowen Liu, MediaTek Inc

Abstract

One daunting challenge of developing a low-power SoC design is how to verify its power-up, reset and initialization sequences.

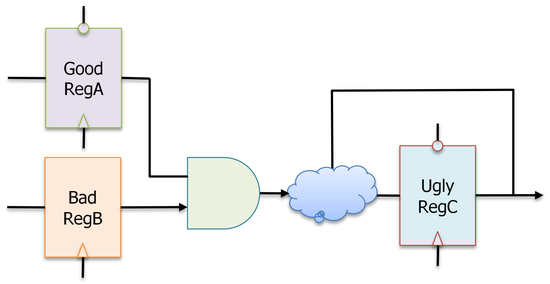
Our initialization verification methodology classifies design registers into:

GOOD registers - are initialized properly,

BAD registers - are not initialized

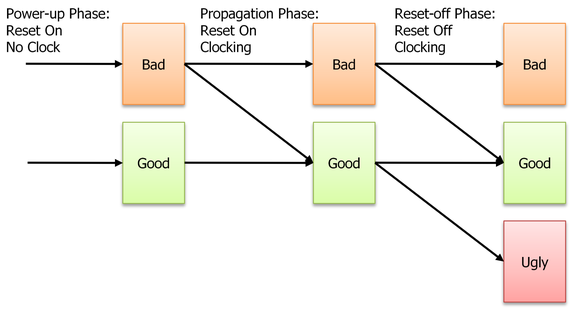
UGLY registers - are initialized, but are subsequently corrupted

The Register Status



Problem	Status of {RegA, RegB, RegC}	Status of {RegA, RegB, RegC}
Reset Propagation Problem: reset signals failed to reach RegA and RegC correctly	{bad, bad, bad}	{bad, bad, bad}
Reset Timing Problem: **reset signals to RegA and RegC did not meet protocol.**	{bad, bad, good} RegA is not initialized correctly.	{bad, bad, ugly} RegC is corrupted by RegA.
X-propagaton Problem: **reset signals to RegA and RegC assert correctly**	{good, bad, good} RegB is not initialized correctly.	{good, bad, ugly} RegC is corrupted by RegB.
Power Sequence Problem: **RegB is powered down**	{good, bad, good} RegA fails to isolate it	{good, bad, ugly} RegC is corrupted by RegB
Clock Gating Problem: **RegB is controlled by gated clock that is turned off**	{good, bad, good} RegA fails to gate it	{good, bad, ugly} RegC is corrupted by RegB

The Initialization Sequence



The power-up phase:
- Reset signals are asserted and the clocks are not running,
- Good registers with asynchronous or synchronous resets are cleared.
- Bad registers without any reset signal are undefined.

The propagation phase:
- Clocks start to toggle but the reset signals are still asserted.
- Cleared values from good registers are propagated forward by clocks to bad registers.
- Some bad registers turn good.

The reset-off phase:
- Clocks are toggling, and reset signals are de-asserted.
- Bad registers may be in the fan-in cone of good registers
- If these bad inputs are not guarded, good registers are corrupted, and turn ugly.

Methodology

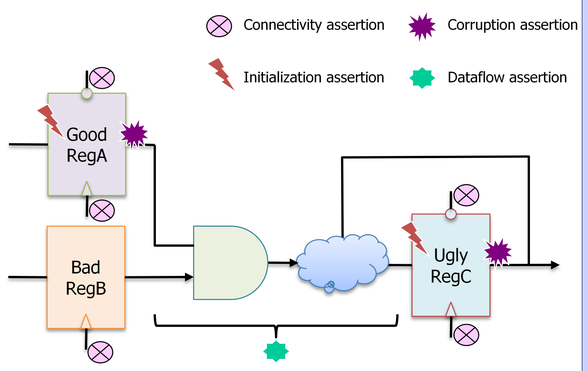
The Good Registers:
- Static verification is used to verify the clock and reset trees to the good registers.
- Assertions are generated to ensure the control signals to the clocks and resets are well behaved.
- Formal verification is used to identify registers that are initialized implicitly by the good registers.

The Bad Registers:
- They are potential X sources if their unknown values are allowed to propagate,
- They will corrupt the rest of the design, and should be loaded with fresh data before use.
- X propagation is enabled during functional simulation to monitor if bad values have been propagated and caused corruption in critical design elements such as clock signals, select-expressions, registers, and FSMs.

Methodology

The Ugly Registers:
- A good register can be corrupted by its fan-in logics when unknown values are propagated and sampled.
- Assertions are generated to monitor all good registers in the design.
- The properties are written to ensure that once the registers are initialized, they will stay good and will not be corrupted.
- With formal verification, we are able to stress the design early and uncover corner-case scenarios that will cause corruption later on.

Assertions for Different Registers



Register	Condition	Assertion
All Registers	Clock is connected correctly from top to blocks/registers	Connectivity assertion e.g. (dut.a.enable) |-> (dut.clk === dut.a.b.c.gclk)
Good registers	Reset is connected correctly from top to blocks/registers	Connectivity assertion e.g. $rose(dut.rst) |-> ##[1:3] (dut.a.b.c.rst)
Good registers	Register is being initialized correctly	Initialization assertion e.g. (dut.a.b.rst) |-> (dut.a.b.rega === `IVALUE)
Bad registers	Uninitialized value is being guarded from propagating	Dataflow assertion e.g. $isunknown(dut.a.b.regb) |-> ! $isunknown({dut.a.b.fanouts}) $isunknown(dut.a.b.regb) |-> ! $isunknown({dut.a.b.outports})
Ugly registers	Good registers should not turn ugly	Corruption assertion e.g. (! dut.a.b.rst) |-> ! $isunknown(dut.a.b.regc)

Results

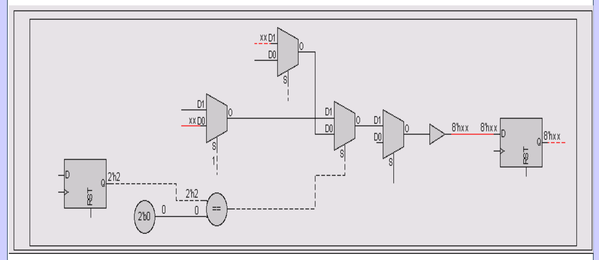
Design complexity	Design 1	Design 2	Design 3
Number of register bits	305	47016	43622
Number of latch bits	0	592	0
Number of RAMs	2	0	64
Number of asynchronous resets	3	13	16
Number of synchronous resets	2	33	35
Number of clocks	3	5	12

Register Status information	Design 1	Design 2	Design 3
Good registers	38%	50%	34%
Bad registers	58%	9%	66%
Ugly registers	4%	41%	<1%

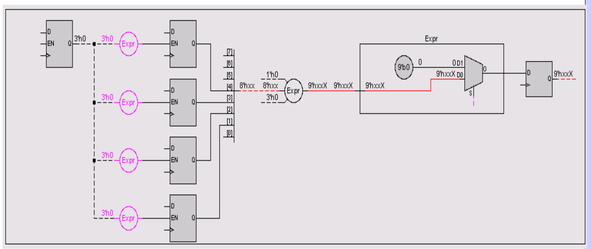
Design 1:
- a common situation where 1/3 of the registers are initialized
- formal verification finds a small percentage of registers that can be corrupted and turned ugly.

Design 2:
- a lot of register files in the design. When the incoming data is unknown, the register in the register files will be corrupted one after another.

Design 3:
- Bad registers in data-path are well guarded. A few good registers can potentially be corrupted under some specific scenarios below:



A good register was corrupted by an X source (an X assignment inside the default case branch).



A good register was corrupted by a group of uninitialized bad registers. The TX registers were not enabled. As a result, they were holding their corresponding un-initialized values.