Finite State Machine Design Considerations

Design and Verification of Digital Systems

Juan-Bernardo Gómez-Mendoza[1]

September 29, 2025

[1]Universidad Nacional de Colombia

Agenda

1. Reset and Enable Conventions
2. Registered Outputs
3. Safe State Machines
4. Clock Domain Crossing

Reset and Enable by Convention

- Commit early to synchronous vs. asynchronous resets and signal polarities; wrap mismatched interfaces rather than complicating the FSM.
- Guarantee deterministic start-up: define state encodings and outputs that settle before downstream sampling.
- Instrument reset sequencing with immediate or SVA checks to spot release races during verification.
- Register clock-gate or enable controls so updates occur only when the enable is stable, avoiding missed transitions.
- Align documentation and verification environments with the implementation so lab bring-up mirrors simulation.

Pass Every Output Through Registers

- Drive external logic from registered signals to eliminate simulation-only hazards and spurious pulses in silicon.
- Provide clean launch edges for static timing analysis and simplify hold-time closure across the design.
- Use registered pulse generators or one-hot decoded stages when zero-latency responses are required.
- Protect intentional register stages with synthesis directives (for example `keep` or `dont_touch`) to prevent retiming.
- Reflect registered behaviour in scoreboards and sign-off checklists so verification accounts for latency.

Safety: Unreachable States

- Enumerate legal states explicitly and provide a default recovery path to a known safe state.
- Preserve recovery logic with safe-FSM attributes such as `(* syn_encoding = "safe" *)`, `(* keep = "true" *)`, or `(* syn_keep = 1 *)`.
- Apply formal and lint checks to prove forward progress and containment of illegal encodings.
- Force illegal states in simulation (DPI/backdoor) to validate the recovery path after synthesis.
- Plan for silicon observability via scan, status pins, or embedded logic analysers to detect escapes.

Clock Domain Crossing Essentials

- Treat every CDC path as metastability-prone: define ownership, handshakes, and wait requirements explicitly.
- Stretch pulses or convert to level-based protocols; move multi-bit data through Gray codes, FIFOs, or request/acknowledge schemes.
- Keep CDC assumptions consistent across waiver files, simulation constraints, and timing analysis.
- Account for added latency in the FSM micro-architecture when inserting synchronizers.

Two-Flip-Flop Synchronizer

- First stage absorbs metastability, second stage presents a clean sample to the destination domain.
- Constrain placement and timing: co-locate flops, mark with `ASYNC_REG`, and treat inputs as asynchronous.
- Remember the synchronizer introduces at least one destination-cycle delay.

Synchronizer RTL Example

```
// source_clk domain
always_ff @(posedge source_clk or negedge rst_n)
  if (!rst_n)
    event_async <= 1'b0;
  else if (fire_condition)
    event_async <= 1'b1;
  else if (ack_sync)
    event_async <= 1'b0;

// destination_clk domain
(* ASYNC_REG = "TRUE" *) logic event_meta;
(* ASYNC_REG = "TRUE" *) logic event_sync;
always_ff @(posedge destination_clk or negedge rst_n) begin
  if (!rst_n) begin
    event_meta <= 1'b0;
    event_sync <= 1'b0;
  end else begin
    event_meta <= event_async;
    event_sync <= event_meta;
  end
end
```

Verification Takeaways

- Model CDC latency and randomise clock phase relationships during simulation to mimic silicon.
- Back-annotate gate delays or metastability models to evaluate MTBF targets before tape-out.