Registers and outputs at reset

```
!rst_n |-> data === '0 && addr === '0
```

Restrictions on control signals (input & output)

Known control signals	`!$isunknown({ctrl})`
Mutex on control signals	`$onehot0({a,b,c})`
Restrictions on enums	`cmd inside {WRITE,READ}`

Restrictions on data signals (input & output)

Known data in transactions	`ctrl |-> !$isunknown({addr, data})`
Address ranges	`addr < SIZE`
Math functions, data relations	`input == (sqrt*sqrt)`
iff or <-> for IEEE 1800-2005	`(empty == '1) === (fill == '0)`
Thermometer code: 0..01..1	`$onehot0(code+1)`
Gray code	`$onehot0(code ^ $past(code))`

Handshaking protocols

Single beat req	`req |=> !req`
Single beat req until ack	`req |=> !req throughout ack[->1]`
No spurious ack	`ack |-> req`
No spurious ack	`!ack |-> !req`
Req stays high until ack	`$rose(req) |-> req[*0:$] ##0 ack`
Req stays high until ack	`$fell(req) |-> $past(ack)`
Data stability	`!en |=> $stable(data)`
Data sampling	`!$stable(data) |-> $past(valid && ready)`
Formal: ack is high eventually	`##[0:$] ack`

Invariants

Pipeline	`$past(past, 5) === now`
Formal: Pipeline	`##5 $past(past, 5) === now`
Internal registers never x	`!$isunknown({data, addr})`
Overflow/underflow	`!(full && enq)`
FSM transitions	`state == DECODE |->` `  $past(state inside {RESET, STORE})`
Data integrity	See other side
Unique	See other side

Timing

Setup	`p_min_time(data, posedge clk, 1ns)`
Hold	`p_min_time(posedge clk, data or posedge clk, 1ns)`
Clock period	`p_min_time(posedge clk, posedge clk, 10ns)`
Rate limit	`(en, cnt = n-1) |=> (!en, cnt--)[*0:$] ##1 en && cnt <= 0`
CDC stability	`!$stable(d_in)|=> $stable(d_in)[*2]`
Glitches	See other side
Multi clock data integrity	See other side

Single Clock Template

```
a_meaningful_name: assert property (
  @(posedge clk) disable iff (!rst_n)
  <... your assertion here ...>
) else $error("Error: <...>", $sampled(<...>));
```

Data integrity

```
longint wcnt, rcnt;
always @(posedge clk) if (write) wcnt++;
always @(posedge clk) if (read) rcnt++;
property p_data_integrity;
  longint cnt;
  logic [N-1:0] data;
  @(posedge clk) disable iff (!rst_n)
  (write, cnt = wcnt, data = wdata) |->
    (read && (rcnt === cnt))[->1] ##0 (rdata === data);
endproperty
```

No data creation

```
wcnt >= rcnt
```

Unique

```
genvar g, h;
generate for (g = 0; g < N; g++) begin
  for (h = 0; h < g; h++) begin
    a_unique: assert property (
      @(posedge clk) disable iff (!rst_n)
      array[g] != array[h]
    );
  end
end endgenerate
```

Timing

```
property p_min_time(start, stop, duration);
  time start_time;
  @(start) (1,start_time = $time) |=> @(stop)
    (($time - start_time) >= duration);
endproperty
```

Glitches
(fast to slow domain)

```
property p_no_glitch;
  logic data;
  @(d_in) (1, data = !d_in) |=>
    @(posedge clk) (d_in === data);
endproperty
```

CDC data integrity
(remember no data creation)

```
longint wcnt, rcnt;
always @(posedge wclk) if (write) wcnt++;
always @(posedge rclk) if (read) rcnt++;
property p_data_integrity;
  longint cnt;
  logic [N-1:0] data;
  @(posedge wclk) disable iff (!rst_n)
  (write, cnt = wcnt, data = wdata) |=> @(posedge rclk)
    (read && (rcnt === cnt))[->1] ##0 (rdata === data);
endproperty
```

Elaboration time assertion

```
generate if (N < 1) begin
  elab_error_if_incorrect_n non_existing_module();
end endgenerate
```

Silence assertion

```
$assertkill(0, path.to.dut.a_assert_name)
```