

2. Übung: Scan Chain Insertion

Name(n): _____

Punkte: _____

1 Manueller Einbau einer Scan-Kette

Typischerweise werden in ASICs Teststrukturen zur Überprüfung des Produktionsprozesses eingebaut. Dabei werden Scan-Ketten in den Entwurf eingefügt, um die Beobachtbarkeit und Steuerbarkeit einer komplexen digitalen Schaltung zu erhöhen bzw. zu gewährleisten. Die Testmustererstellung (ATPG: *Automatic Test Pattern Generation*) kann dadurch automatisiert werden. Diese Teststrukturen und -muster werden zur Überprüfung der Korrektheit simuliert (*Pattern Resimulation*).

Es sei der folgende Code gegeben:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity scan_test is
5
6     port (
7         clk_i          : in  std_ulogic; -- clock
8         res_i          : in  std_ulogic; -- reset
9
10        --test_mode_i : in  std_ulogic; -- bypass scan test design rule
11                                   -- violations
12        scan_enable_i : in  std_ulogic; -- shift mode
13
14        a_i           : in  std_ulogic; -- primary input a
15        b_i           : in  std_ulogic; -- primary input b
16        z_o           : out std_ulogic  -- primary output
17    );
18
19 end scan_test;
20
21 architecture rtl of scan_test is
22
23     signal n, m, p, q : std_ulogic;
24
25 begin
26
27     process (clk_i, res_i)
28     begin
29         if res_i = '0' then
```

```

30     n <= '0'; m <= '0'; p <= '0'; q <= '0';
31     elsif clk_i'event and clk_i = '1' then
32         n <= a_i xor b_i;
33         m <= n and b_i;
34         p <= m or not n;
35         q <= not p;
36     end if;
37 end process;
38
39 z_o <= q;
40
41 end rtl;

```

Der auskommentierte Port `test_mode_i` ist normalerweise während des gesamten Tests auf logisch '1' und wird zur Umgehung von DfT-Verletzungen verwendet. Da es sich beim gegebenen Code um einen vollsynchronen Entwurf ohne solche Verletzungen handelt, ist dieser Eingang hier nicht nötig.

Der Einbau der Scan-Ketten und die Generierung der Testmuster erfolgt in der Industrie automatisiert. In dieser Übung sollen die Schritte jedoch zum besseren Verständnis manuell nachvollzogen werden. Gehen Sie dabei folgendermaßen vor:

1. Synthetisieren Sie die Schaltung und erstellen Sie eine Gatternetzliste. Der RTL-Code liegt in `$MHE3_HOME/templates/vhdl` bereit.
2. Tauschen Sie die Flipflops in der Netzliste gegen scan-fähige Flipflops aus (Tipp: `amsdoc`).
☐ Welche Flipflops wählen Sie dafür?
3. Fädeln Sie die Flipflops in einer Scan-Kette, beginnend vom Eingang `a_i` bis zum Ausgang `z_o`, auf. Verwenden Sie dabei bevorzugt jenen Ausgang des Flipflops, der noch keine anderen Gatter treibt. Begründen Sie zusätzlich, warum es sinnvoll ist, nicht belegte Ausgänge der Flipflops zu verwenden!
☐
4. Verbinden Sie `scan_enable_i` mit den dafür vorgesehenen Eingängen der Flipflops.

1.1 Erstellen des Testmusters

Erstellen Sie nun ein Testmuster, mit dem ein Stuck-at-1-Fehler am Eingang des Flipflops `p` erkannt werden kann. Die Register `m` und `n` müssen also mit geeigneten Werten geladen werden, sodass im

- ☐ *Mission Mode* (also `scan_enable_i = '0'`) am D-Eingang des Flipflops `p` eine '0' anliegt.
- Geben Sie zusätzlich mindestens zwei weitere, zu dem berechneten Fehler äquivalente, Stuck-at-Fehler an und begründen Sie, warum diese Fehler äquivalent sind!
- ☐

1.2 Pattern Resimulation

Schreiben Sie eine Testbench, die einen Scan-Test simuliert:

1. Zuerst sollen über die Scan-Kette in die Register n und m die passenden Werte geschoben werden.
2. Betreiben Sie dann die Schaltung für einen Taktzyklus lang im *Mission Mode*.
3. Lesen Sie über die Scan-Kette den Wert des Registers p aus und prüfen Sie, ob der erwartete Wert berechnet wurde.

1.3 Simulation eines Produktionsfehlers

Erstellen Sie eine Kopie der Netzliste mit Scan-Kette, in der Sie den Produktionsfehler durch eine Zuweisung eines festen Werts simulieren. Überprüfen Sie, ob Ihr Scan-Test den Fehler erkennt, und dokumentieren Sie dies entsprechend. Führen Sie eine ähnliche Simulation auch für einen äquivalenten Stuck-at-Fehler durch, und dokumentieren Sie auch dieses Ergebnis!

2 DfT-Verletzungen

Fassen Sie die in der Vorlesung besprochenen DfT-Verletzungen zusammen und erklären Sie für jede, warum sie im Zusammenhang mit DfT ein Problem darstellt!

1 Manueller Einbau einer Scan-Kette

Als Flipflops werden DFSC1 und DFSC3 verwendet, da diese die Pendanten zu DFC1 und DFC3 sind.

Es ist sinnvoll nicht belegte Ausgänge der Flipflops zu verwenden, da sich dadurch die Treiberstärke aufteilt.

1.1 Erstellen des Testmusters

```
a)
n m p q
1 0 0 0

b)
q = 0
p = !q
l = !0

b_i = 1
p = ((a_i xor b_i) and b_i) or !(a_i xor b_i)

bei a_i = 0
((0 xor 1) and 1) or !(0 xor 1) = 1 or !0 = 1

bei a_i = 1
((1 xor 1) and 1) or !(1 xor 1) = 0 or !0 = 1
```

2 Source Code

2.1 Manuelle ScanChain

../syn/netlist/scan_test_mod.vhd

```
32 component DFSC1
33     port( D, C, RN, SD, SE : in std_ulogic; Q, QN : out std_ulogic);
34 end component;
35
36 component DFSC3
37     port( D, C, RN, SD, SE : in std_ulogic; Q, QN : out std_ulogic);
38 end component;
39
40 signal N0, N2, n1, n2_port, n5, n6, n7, n9, n10, n11, n12 : std_ulogic;
41
42 begin
43
```

```

44     n_reg : DFSC1 port map( D => N0, C => clk_i, RN => res_i, SD => a_i, SE
        => scan_enable_i, Q => n9, QN => n11)
45     ;
46     m_reg : DFSC3 port map( D => n5, C => clk_i, RN => res_i, SD => n9, SE
        => scan_enable_i, Q => n10, QN => n6)
47     ;
48     p_reg : DFSC3 port map( D => N2, C => clk_i, RN => res_i, SD => n10, SE
        => scan_enable_i, Q => n2_port, QN =>
49         n7);
50     q_reg : DFSC1 port map( D => n7, C => clk_i, RN => res_i, SD => n2_port,
        SE => scan_enable_i, Q => z_o, QN => n1)
51     ;
52     U7 : NOR20 port map( A => n11, B => n12, Q => n5);
53     U8 : NAND20 port map( A => n6, B => n9, Q => N2);
54     U9 : XNR20 port map( A => n12, B => a_i, Q => N0);
55     U10 : CLKIN0 port map( A => b_i, Q => n12);

```

2.2 Manuelle ScanChain mit Fehler

../syn/netlist/scan_test_mod_fail.vhd

```

32     component DFSC1
33         port( D, C, RN, SD, SE : in std_ulogic; Q, QN : out std_ulogic);
34     end component;
35
36     component DFSC3
37         port( D, C, RN, SD, SE : in std_ulogic; Q, QN : out std_ulogic);
38     end component;
39
40     signal N0, N2, n1, n2_port, n5, n6, n7, n9, n10, n11, n12 : std_ulogic;
41
42 begin
43
44     n_reg : DFSC1 port map( D => N0, C => clk_i, RN => res_i, SD => a_i, SE
        => scan_enable_i, Q => n9, QN => n11)
45     ;
46     m_reg : DFSC3 port map( D => n5, C => clk_i, RN => res_i, SD => n9, SE
        => scan_enable_i, Q => n10, QN => n6)
47     ;
48     p_reg : DFSC3 port map( D => N2, C => clk_i, RN => res_i, SD => n10, SE
        => scan_enable_i, Q => n2_port, QN =>
49         n7);
50     q_reg : DFSC1 port map( D => n7, C => clk_i, RN => res_i, SD => n2_port,
        SE => scan_enable_i, Q => z_o, QN => n1)
51     ;
52     U7 : NOR20 port map( A => n11, B => n12, Q => n5);
53     U8 : NAND20 port map( A => n6, B => n9, Q => open);
54     U9 : XNR20 port map( A => n12, B => a_i, Q => N0);
55     U10 : CLKIN0 port map( A => b_i, Q => n12);
56
57     N2 <= '1';

```

2.3 Testbench

../src/scan_test_tb.vhd

```

32     -- stimulation process
33     stimuli : process is
34     begin
35         scan_enable <= '1' after 0 ns,
36                 '0' after 100 ns,

```

```

37         '1' after 120 ns;
38     a <= '0' after 0 ns,
39         '0' after 20 ns,
40         '0' after 40 ns,
41         '0' after 60 ns,
42         '1' after 80 ns,
43         '0' after 100 ns;
44
45     b <= '0';
46
47     wait for 120 ns;
48     assert z = '1' report "q wrong" severity error;
49     wait for 20 ns;
50     assert z = '0' report "p wrong" severity error;
51     wait for 20 ns;
52     assert z = '0' report "m wrong" severity error;
53     wait for 20 ns;
54     assert z = '0' report "n wrong" severity error;
55
56     wait;
57 end process;

```

3 Simulation

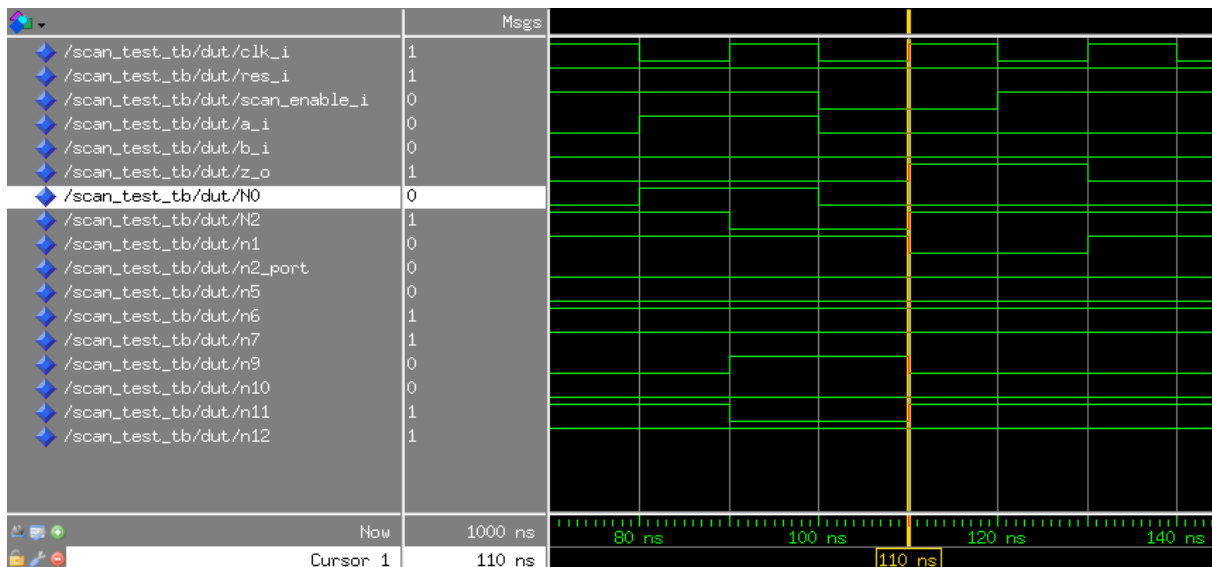


Figure 1: Waveform von Simulation ohne Fehler.

4 DfT-Verletzungen

4.1 No asynchronous design style

Keine asynchronen Elemente (Latch). Diese Latches könnten Daten speichern, was nicht erlaubt ist.

4.2 No on-chip generation of clock signals

Für die Flipflops in unserem Design sollte kein selbst erzeugter Takt (das heißt Ausgang eines Flipflops in unserem Design) verwendet werden, da es da beim Umschalten zum Mission Mode

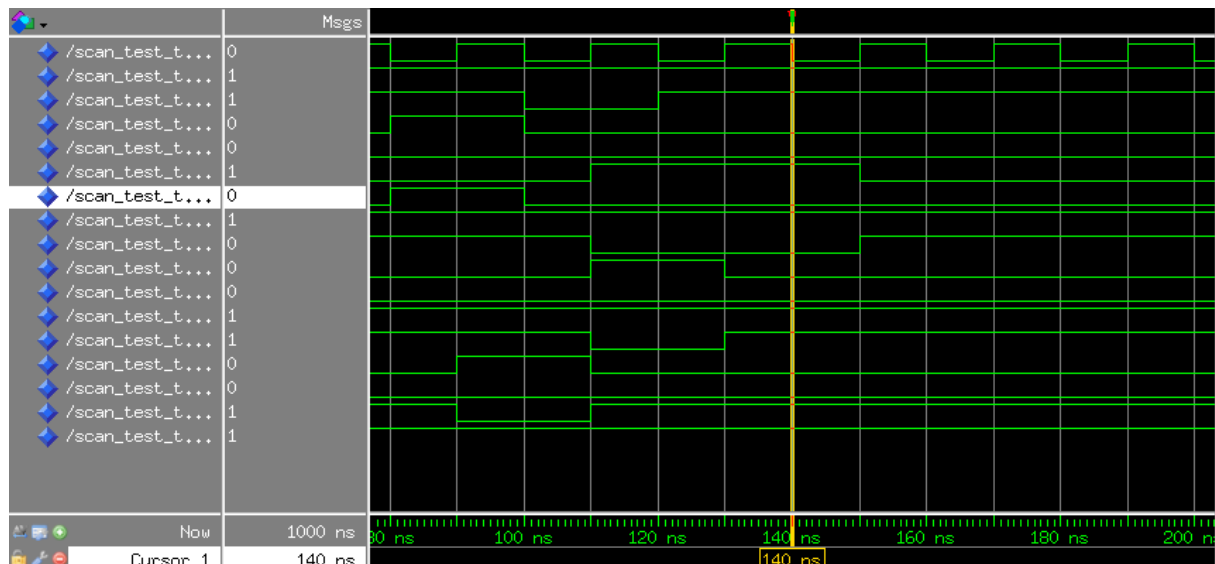


Figure 2: Waveform von Simulation mit Produktionsfehler. Assertion löst aus.

zu Fehlern kommen kann.

4.3 Special attention when more clock domains

Falls mehrere Clock Domains vorhanden sind muss man spezielle Maßnahmen ergreifen, damit die Scanketten richtig funktionieren. Man könnte zum Beispiel für jede Domain eine eigene Scankette machen.

4.4 No gating of clock signals

Clocksignale sollten nicht von anderen Signalen abhängig sein, da ansonsten beim Hineinschieben des Testpattern in die Scankette Probleme auftreten können.

4.5 No combinatorial feedback loops

In den kombinatorischen Teilen sollten keine Feedback Loops sein, da es sonst sein kann, dass dort Latches entstehen welche wiederum Werte speichern können, was nicht erlaubt ist. Wenn die Werte aus der Kombinatorik ins Flipflop geladen werden, dann kann nicht bestimmt werden welchen Wert diese haben.

4.6 No one-shot delays

Wenn bei einem Gatter ein Signal an einem Eingang anliegt und bei einem zweiten Eingang das invertierte anliegt, sollte das vermieden werden und stattdessen der negative Q-Ausgang des vorhergehenden Flipflops verwendet werden.

4.7 No on-chip generation of asynchronous control signals

Zum Beispiel die Reset Eingänge der Flipflops sollten nicht durch Kombinatorik in unserer Schaltung erzeugt werden, da dadurch Werte, die beim Schieben durch die Scankette laufen, Flipflops ungewollt resetten könnten.

4.8 Avoid tristate signals, check for bus contention

Busses sollten immer auf ein definiertes Potential gezogen werden, da sonst nicht vorhersehbar ist welche Werte sich einstellen, was das Testen unmöglich macht.