INF 3430

Test og design for testbarhet

Innhold

- Verifikasjon og testing
- Design for testbarhet
  - Ad hoc forbedringer
  - Strukturelt design for test
  - Built-in self test
  - Boundary scan (IEEE1149.1)

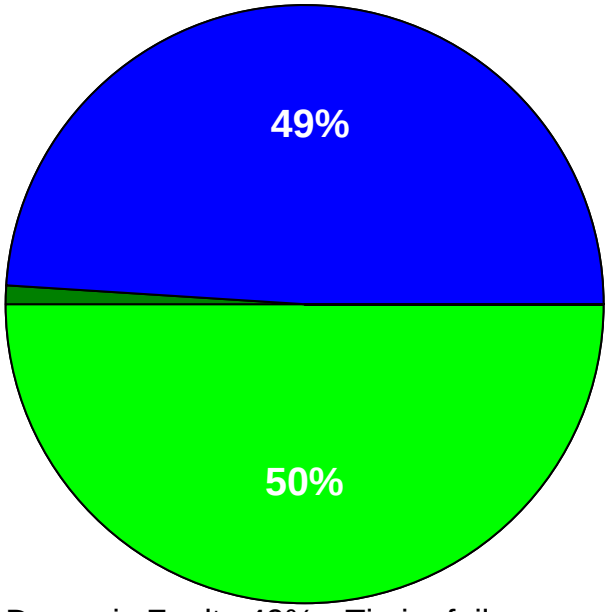
Verifikasjon og testing

- Ved verifikasjon sjekkes om et design er korrekt
- Ved testing antar vi at designet er korrekt, men at det kan ha fabrikasjonsfeil.

- To typer testing:
  - Funksjonell testing:
    - Funksjonell testing for å se at systemet virker korrekt
    - Funksjonell testing kan ta veldig lang tid fordi alle tenkelige tilstander til et system skal testes
  - Strukturell testing
    - Strukturell testing for å finne ut om systemet inneholder en feil
    - Strukturell testing kan ta kortere tid enn funksjonell testing fordi det ofte skal få testvektorer til for å teste om det finnes feil.

Behov for testing

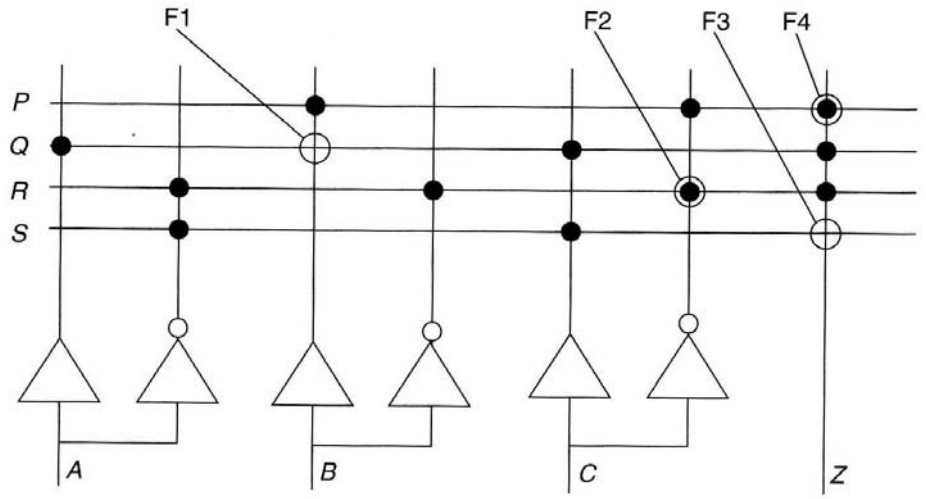
- Feil blir introdusert ved produksjon
  - Kortslutninger
    - loddeperler
  - Manglende forbindelser
    - Overets på PCB
  - Feil komponenter
  - Feil innsatte komponenter
  - Defekte komponenter
- Feil blir introdusert ved bruk/feil bruk
  - Overstiger spesifikasjoner
  - EMI (ElectroMagnetic Interference)
- Inni IC'er kan man ha en rekke feil
  - "Electron migration" fører til at baner bryter
  - Silisium og oksyd defekter kan medføre kortslutninger og feilfunksjon i transistorer
  - "Latch-up" pga. transiente strømmer kan forårsake "Stuck at" 0/1.
  - Inneholdet i hukommelser kan bli ødelagt pga. alfa-partikler eller EMI



Dynamic Faults 49% - Timingfeil
Intermediate Faults 1% - EMI
Static Faults 50% - Kortslutninger/brudd

”Single Stuck at” feil

- Z=P+Q+R
- P=B*/C
- Q=A*C
- R=/A*/B*/C
- S=/A*C



- Feil F1 er en tilleggsforbindelse som medfører at
  - Q=A*B*C
- F2 er et brudd som medfører at
  - R=/A*/B
- F3 medfører at termen S opptrer i Z
- F4 medfører at termen P forsvinner fra Z

Feilorientert testmønster

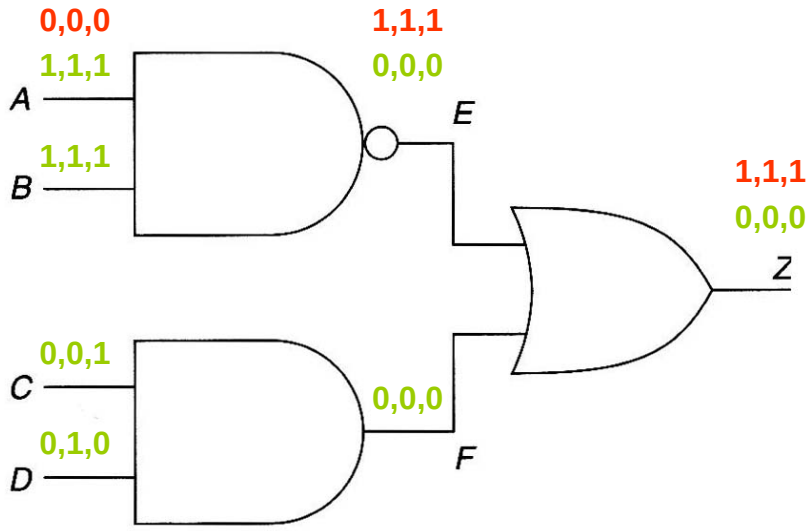
- Å lage et feilorientert testmønster vil si å lage en sett med inngangsverdier slik at interne feil kan bli avdekket ved å monitorere utgangene.
- For å finne ut om et system er testbart må man se på to forhold:
  - Kontrollerbarhet: Kan vi kontrollere alle noder på en slik måte at feil blir avdekket?
  - Observerbarhet: Kan vi observere og skille mellom en node med feil og en uten feil?

Feilorientert testmønster

- For å lage et minimum med testmønstre benytter man en feilorientert strategi:
  - Lager en feilliste. Dvs. lister opp alle noder som kan være ”Stuck-at 0/1”
  - Skriver en test
  - Sjekker feildekningsgrad.
  - Sletter dekket feil fra listen
  - Repeterer inntil ønsket feildekningsgrad er oppnådd
    - Det er ikke sikkert vi kan eller ønsker å finne alle feil
    - Kan være meget tidkrevende å finne testmønstre

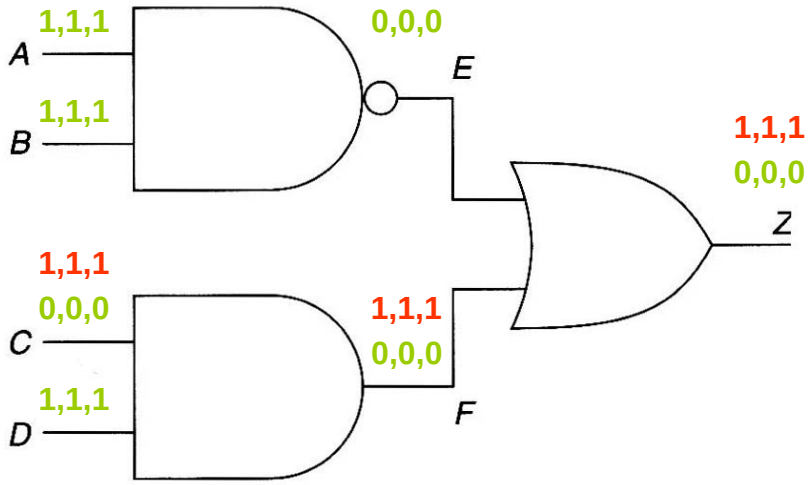
Feilorientert testmønstergenerering

- Kretsen til venstre kan ha 14 ”stuck-at 0/1 feil:
  - A/0, A/1, B/0, B/1, C/0, C/1, D/0, D/1, E/0, E/1, F/0, F/1, Z/0, Z/1
- For å finne et testmønster for A/0 må vi lage en kombinasjon av de andre nodene slik at utgangen er følsom for endringer på A
- En løsning er:
  - A=1, B=1,C=0,D=0, Z=0 eller kortfattet 1100/0
- Andre muligheter er
  - 1101/0 og 1110/0
- Dersom A/0 vil Z=1



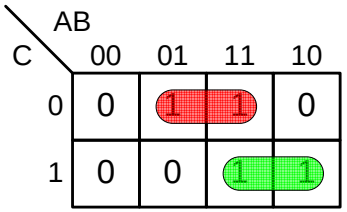
Feilorientert testmønstergenerering

- Skal sjekke for E/1
- Kan benytte 1101/0 som er det samme som et av testmønstrene for å finne A/0 feil
- Det betyr at samme testmønster kan avdekke flere feil

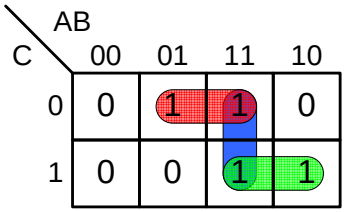


Udetekterbare feil

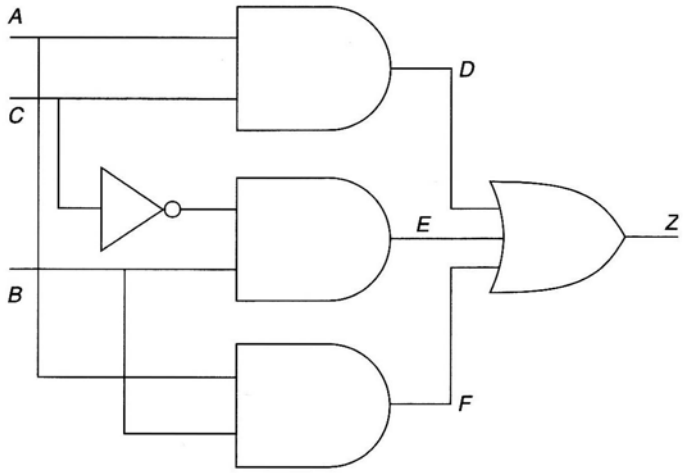
- For å unngå Hazards er det vanlig å legge til redundante termer
- Skal finne en test for F/0
  - Da må F settes til 1 ved at A=B=1
  - For å forplante verdien til F til utgangen Z må D=E=0.
  - For å sette D=0 må A=1 og C=0
  - For å sette E=0 må B=1 og C=1
  - Dette gir en inkonsistens, som betyr at det ikke finnes testmønster for å avdekke F/0 feilen.
- Man har etter dette tre valg:
  - Man aksepterer at kretsen ikke er 100% testbar
  - Ta vekk redundant logikk
  - Innføre en ekstra kontrollinngang for å tvinge D=0 når A=C=1



$Z=AC + B\overline{C}$



$Z=AC + B\overline{C} + AB$

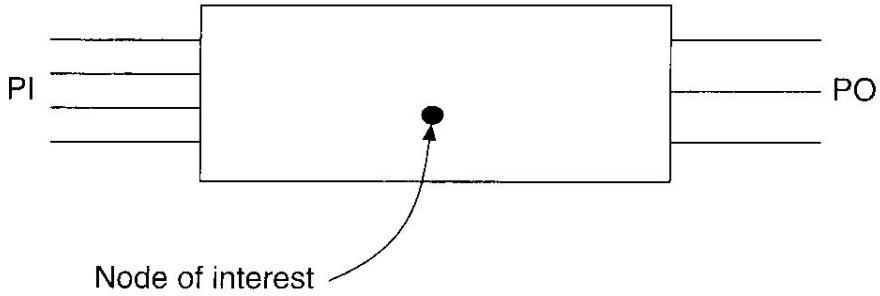


Automatisk genererte testmønstre

- Det finnes kommersielt tilgjengelige verktøy for å lage testmønstre.
- Disse verktøyene benytter algoritmer som automatiserer prosessen og som kan gjøre den nødvendige prøving og feiling for å finne et minst mulig sett av testvektorer.
- D-algoritmen og PODEM er eksempel på to algoritmer som benyttes

Design for testbarhet

- Som nevnt er en krets testbarhet er bestemt av to forhold:
  - Kontrollerbarhet
    - Man skal kunne kontrollere en intern node fra kretsens innganger
  - Observerbarhet
    - Man skal kunne observere verdien av en intern node ved å observere kretsens utgang

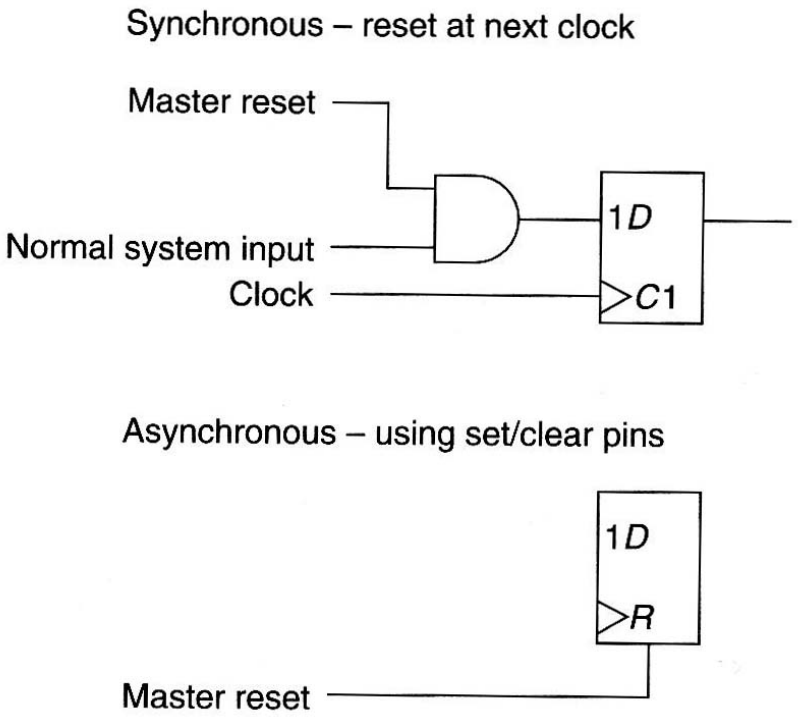


Hvordan kan man forbedre testbarheten?

- Ad hoc forbedringer
- Strukturelle design for test
- Innbygd selvtest (BIST-Built-In Self-Test)
- Boundary Scan (IEEE1149.1)

Ad hoc forbedringer

- Ad hoc forbedringer kan omfatte
  - Ta vekk redundant logikk fordi denne kan medføre udetekterbare feil
  - Synkrone system er mye lettere å kontrollere enn asynkrone.
    - I et synkront system kan man kontrollere klokken og eventuelt stoppe denne for å ”fryse” en gitt situasjon
  - Initialisering av sekvensielle design for å starte testing fra en kjent tilstand

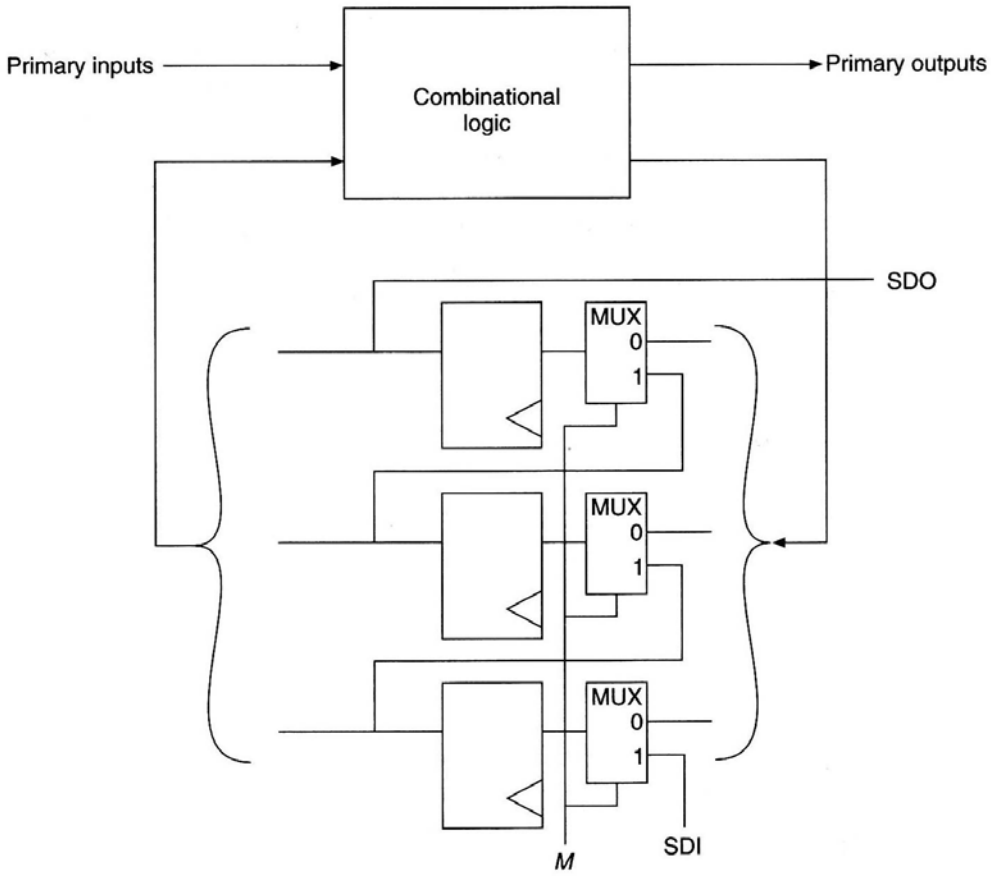


Strukturelle design for test

- Må tenke testbarhet fra starten fordi testlogikk må tas med i designet
- Et synkront design kan sees på som en tilstandsmaskin med neste tilstandslogikk, utgangslogikk og tilstandsregister.
- Vi har ikke direkte kontroll av innganger til kombinatoriske blokker og vi kan ikke direkte observere tilstandsvariable.
- I Scan-in Scan-out (SISO) prinsippet gjøres tilstandsvariable tilgjengelige ved å knytte dem sammen i et skiftregister ved bruk av multipleksere

Scan-in Scan-out (1)

- Testing av flip-flop'er
  - Sett M=1 og test flip-flop'er ved å klokke data gjennom skiftregisteret
  - En sekvens av 1 og 0 klokkes inn på SDI
  - Sekvensen 00110 tester alle transisjoner på flip-flop'ene
- Testing av kombinatorikk
  - Sett M=1 og sett tilstand på flip-flop'ene etter n cycler
  - Sett M=0. Sett opp innganger. Observer utgangene
  - Sett M=1 for å skifte innholdet av flip-flopene til SDO etter n-1 klokke cycler

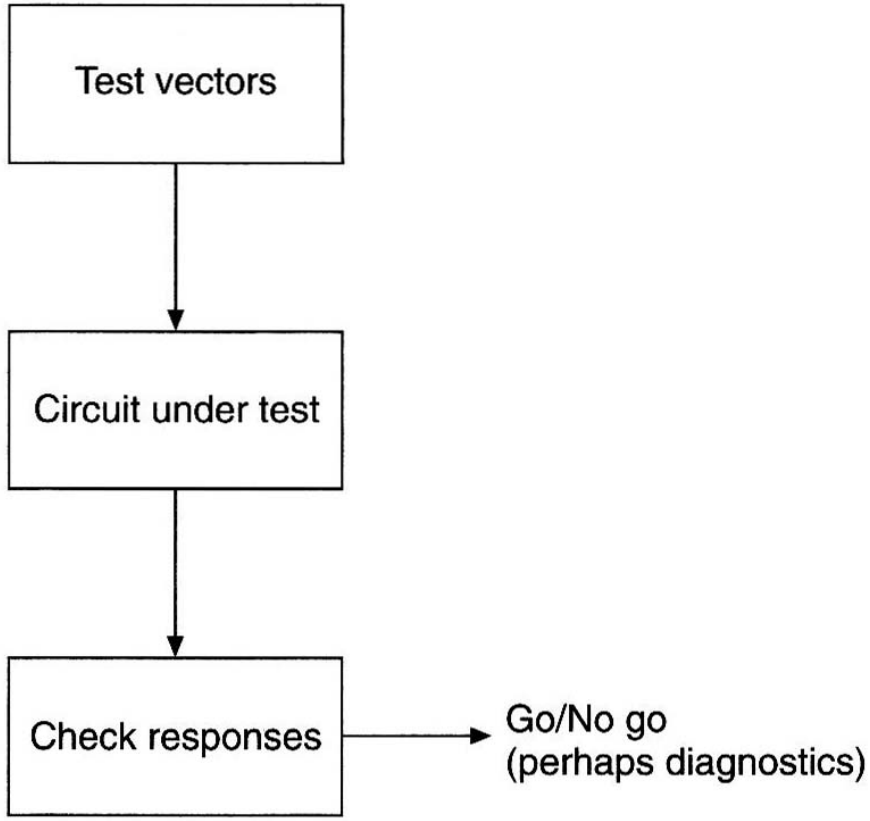


Scan-in Scan-out (2)

- Fordeler
  - Enkelt å teste sekvensielle kretser
  - Fullt testbar dersom kretsen er fri for redundant logikk (er ikke redd for hazards dersom kombinatorisk logikk står koblet mellom registre)
  - Trenger bare testmønstre som er nødvendige for den kombinatoriske logikken.
- Ulemper
  - Medfører ekstra hardware
  - Minst en ekstra pinne for M. SDI og SDO kan eventuelt deles med andre ved bruk av multipleksere
  - En ekstra multiplekser foran hver flip-flop.
  - Ekstra ledninger er nødvendig for Scan-path.
    - Pga. routingdelay kan rekkefølgen av scan-registre bestemmes etter at kretsen er plassert og routet
- Scan-in Scan-out er blitt en populær og vel utbredt design metode

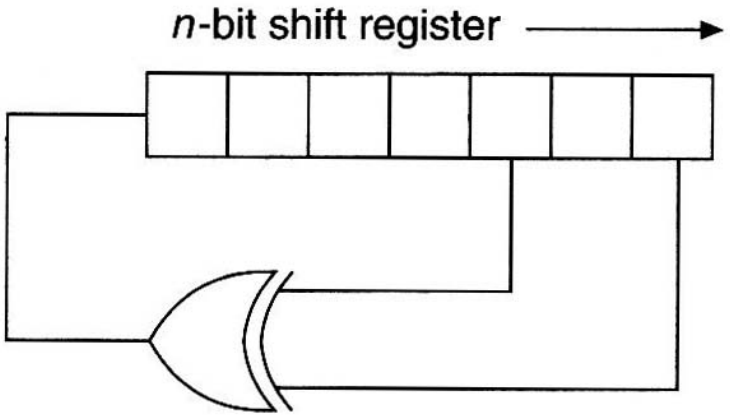
Built-In Self-Test (BIST)

- Prinsippet for BIST er å lage hardware som genererer testvektorer og sjekker responsen internt i kretsen.



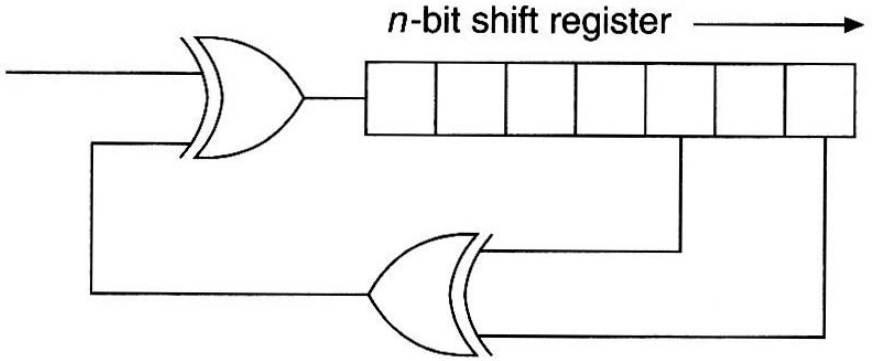
BIST-Generering av vektorer

- Hvordan kan man lage testvektorer?
  - Å benytte en teller vil medføre mye ekstra logikk ettersom antall innganger øker
  - Bruk av LFSR (Linear Feedback Shift Register) lager alle mulige kombinasjoner av 1 og 0 (utenom bare 0'er) og lar seg implementere meget enkelt. LFSR kalles ofte for Pseudo Random generatorer.



BIST-sjekking av respons

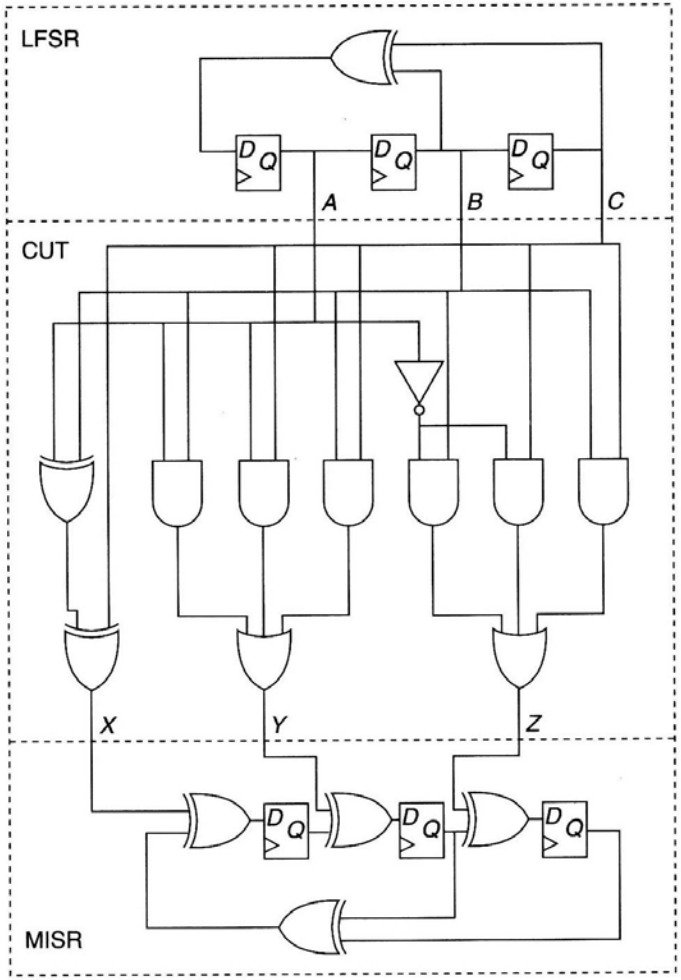
- Hvordan kan sjekke utganger?
  - Utgangene kan sjekkes ved å benytte f.eks. en LUT (look up table) og sammenligne utgangene fra denne med utgangen fra krets under test.
  - Utganger kan sjekkes ved at de klokkes inn i tilsvarende skiftregister som vist på figuren til høyre.
  - Etter et visst antall klokkesykler kan innholdet i skiftregisteret sammenlignes med en kjent verdi (signatur)



Multiple Input Signatur Register(MISR)

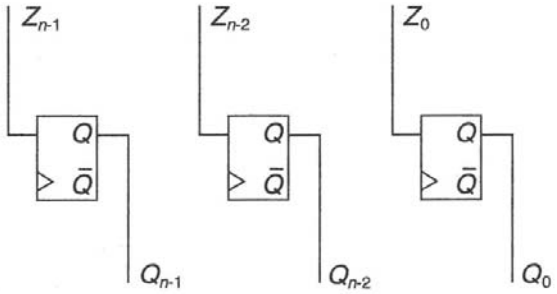
- Eksempel:
  X=A xor B xor C
  Y=AB+AC+BC
  Z=/AB+/AC+BC
- Dersom LFSR og MISR initialiseres til 111 får man følgende sekvens etter 8 klokkesykler
  - Kan benytte inneholdet i MISR (000) som signatur på feilfri krets
- Ulempe
  - Kan vises at det er en 2^{-n} sannsynlighet for at en stuck-at 0/1 lager samme signatur som feilfri krets. (n er antall registre i MISR). Dette fenomenet kalles aliasing
  - I eksemplet vil A/0 lage signaturen 000

LFSR output	CUT output	MISR
abc	*xyz*	
111	111	111
011	011	100
001	101	001
100	100	001
010	101	000
101	010	101
110	010	100
111	111	000

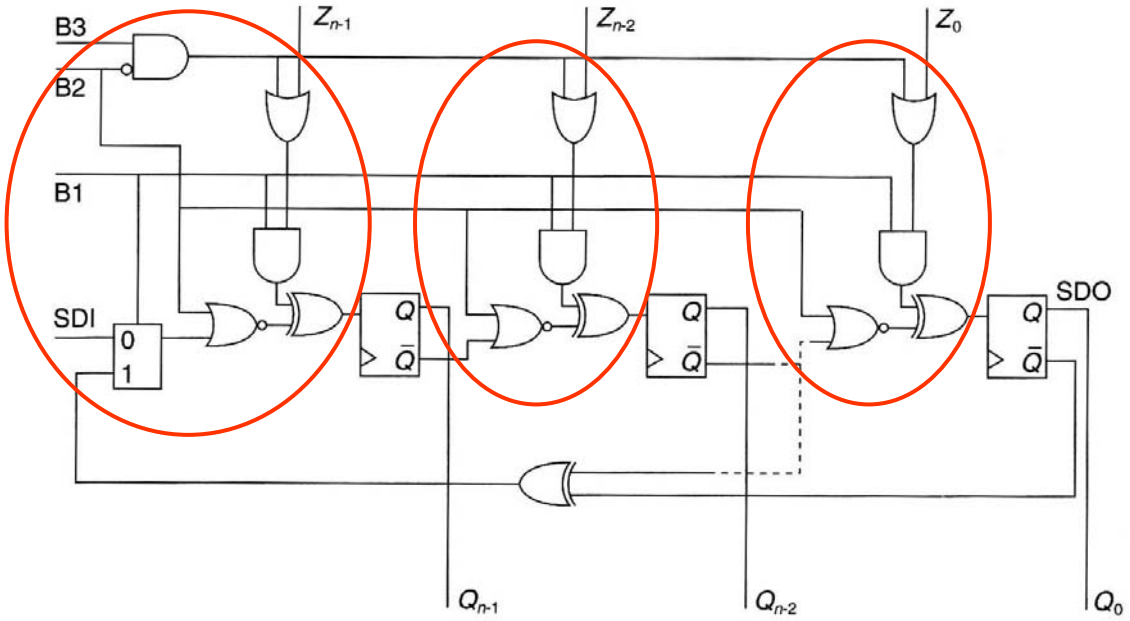


Built-In Logic Block Observation (BILBO)

- Normal krets



- BIST som er bygd opp av LFSR og MISR medfører ekstra registre i tillegg til de som er for normal virkemåte.
- For å gjenbruke eksisterende registre kan man legge in en BILBO struktur i kretsen og så benytte denne for BIST

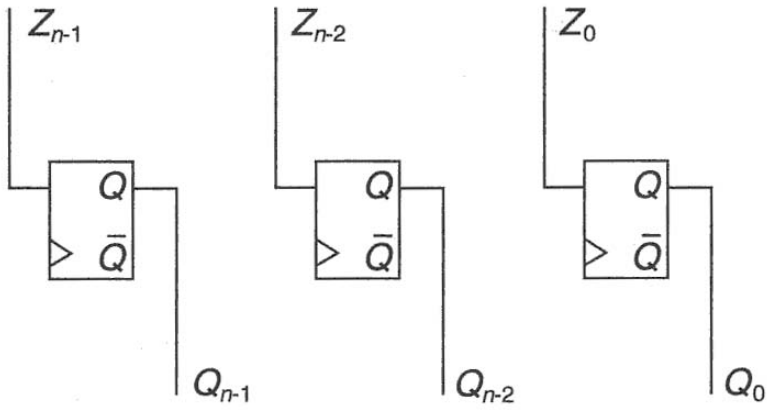


BILBO (2)

- Krever tre kontrollinnganger og virkemåte kontrolleres etter følgende sannhetstabell:

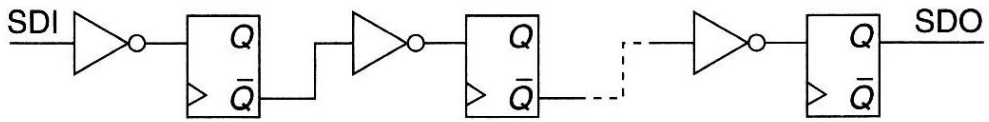
B1	B2	B3	Mode
1	1	–	Normal
0	1	–	Reset
1	0	0	Signature analysis MISR
1	0	1	Test pattern generation LFSR
0	0	–	Scan

- Normal mode

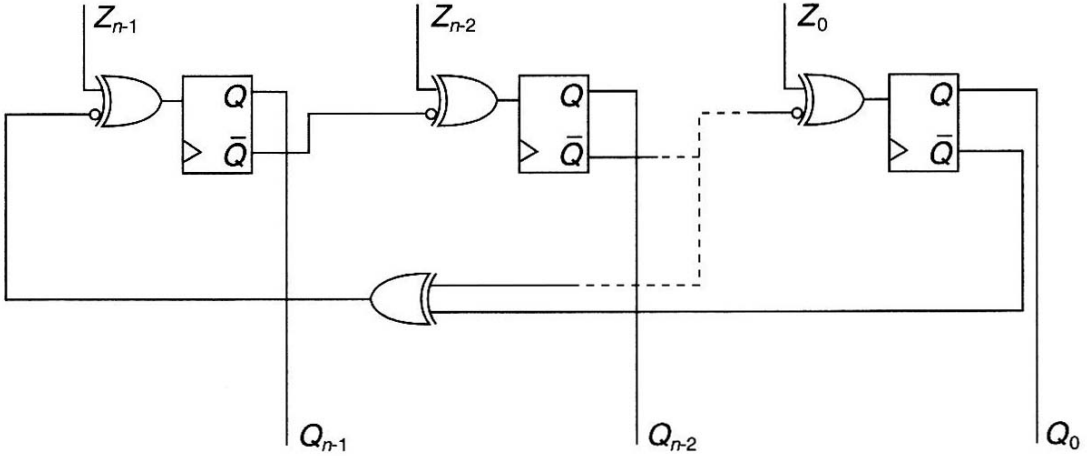


BILBO (3)

- Scan mode

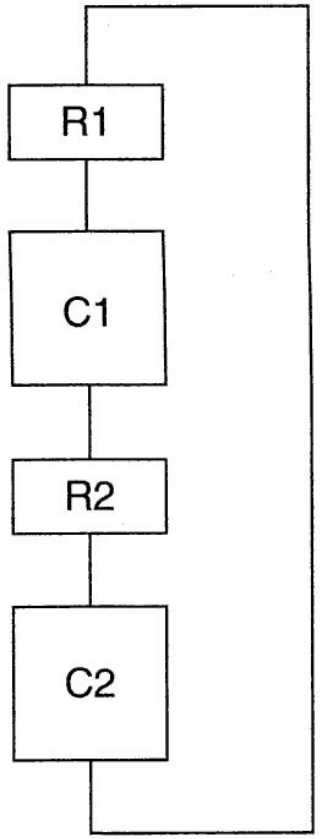


- LFSR/MISR mode



BILBO-eksempel

- R1 og R2 er BILBO strukturer
- C1 og C2 er kombinatoriske logikkblokker
- For å teste C1 så
  - Konfigureres R1 som LFSR og er Input til C1
  - R2 konfigureres som MISR og sjekker signatur fra C1
- For å teste C2 så
  - Konfigureres R2 som LFSR og er Input til C2
  - R1 konfigureres som MISR og sjekker signatur fra C2



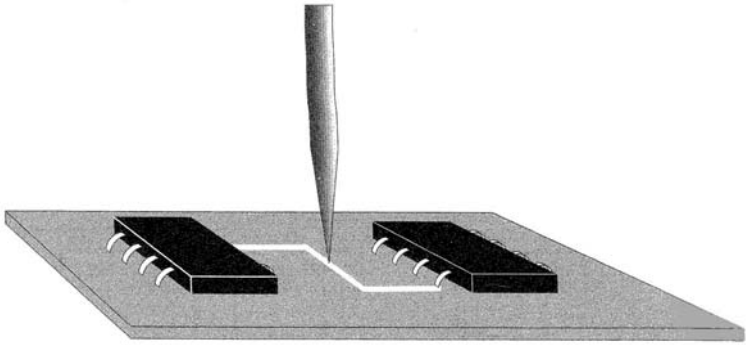
BILBO (5)

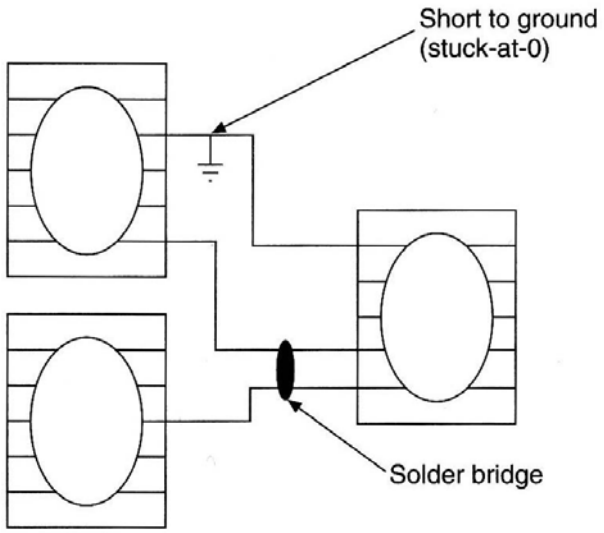
- Hvordan tester vi BILBO selv?
  - Kan teste registrene i BILBO ved å initialisere dem i Scan mode og kjøre et scan for å teste flip-flop'ene.
  - Medfører at vi har en eller annen form for BILBO kontroller.
    - Hvordan skal vi teste denne kontrolleren?
      - En ny kontroller?
        - » Hvordan skal vi teste denne? Osv
- Starter med å teste en liten del av systemet.
  - Dersom denne lille delen virker, benyttes denne til å teste mer av systemet.
    - Og dersom dette virker benyttes dette til å teste mer, osv.

Boundary Scan (IEEE1149.1)

- Tidligere var det vanligst å benytte In-Circuit testere for å teste forbindelsene på et kretskort, men
  - upraktisk fordi pakker er blitt veldig små
  - ikke mulig å teste IC'er som er montert fordi de er koblet sammen med andre IC'er.
  - ofte svært mange lag i PCB slik at baner er svært vanskelig tilgjengelige.

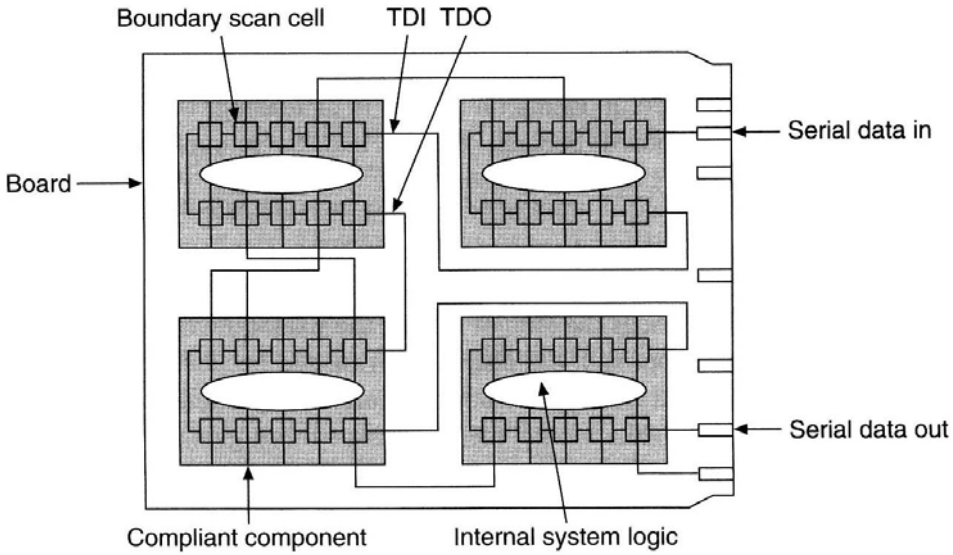
- Boundary scan er en teknikk for både å teste forbindelser mellom komponenter på et kretskort og teste IC'er som er montert på et kretskort.





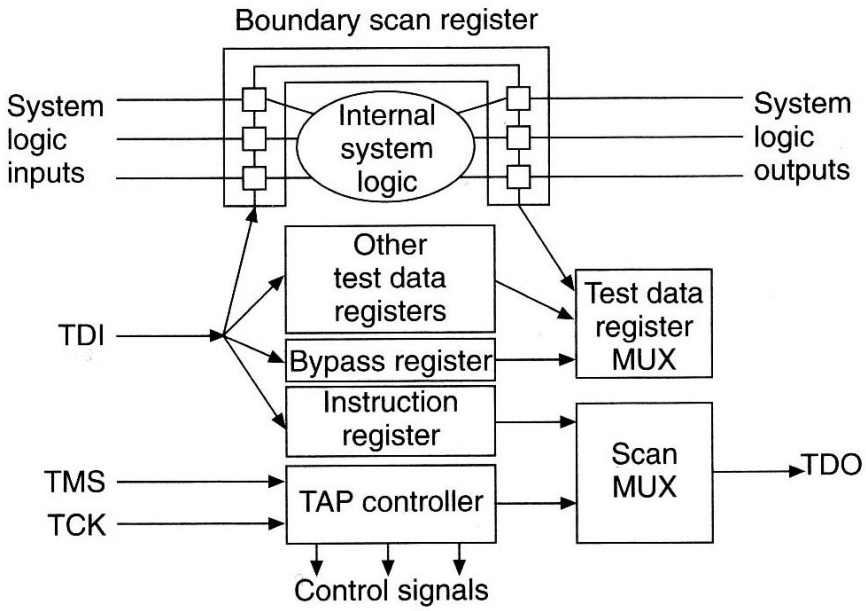
Boundary scan kjede

- Boundary scan standarden er et resultat av arbeidet til Joint Test Action Group (JTAG). Derfor omtales ofte Boundary scan som JTAG.
- Komponenter kobles sammen i en kjede og danner et langt skiftregister.



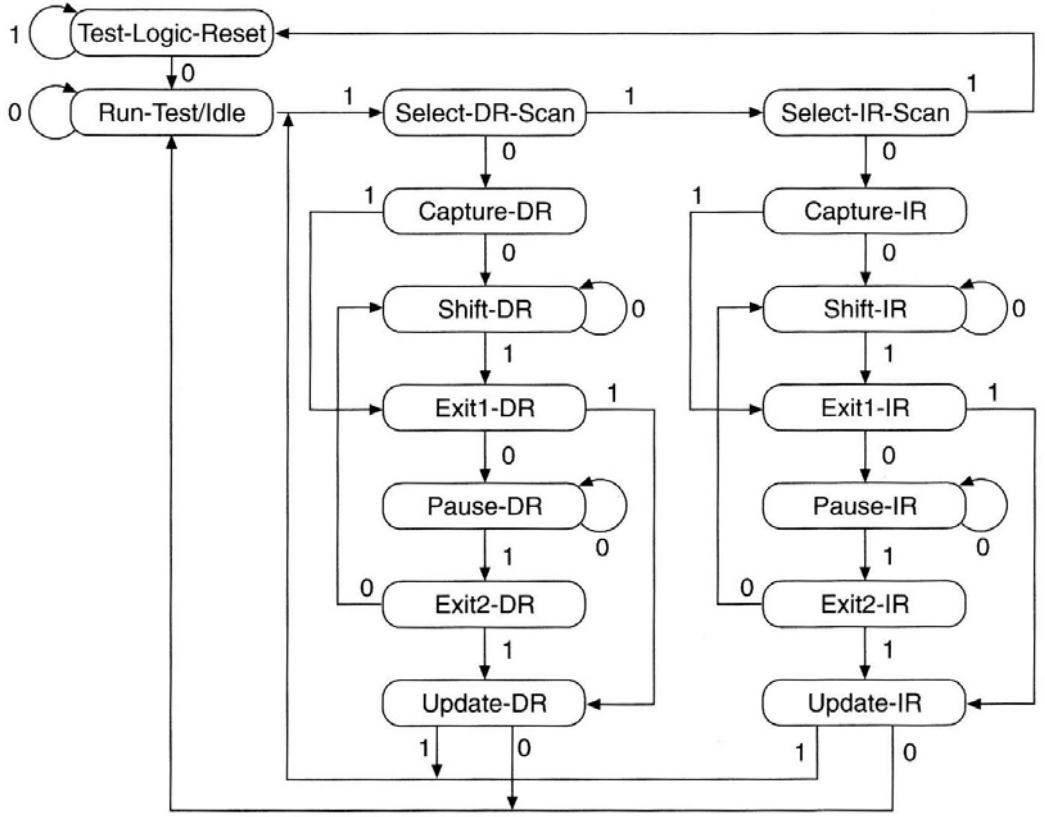
Test Access Port-TAP

- Inneholder en Test Access Port (TAP) som har følgende signaler:
  - TCK (Test Clock)
  - TMS (Test mode select)
  - TDI (Test data in)
  - TDO (Test data out)
  - TRST (Test reset-opsjon)



TAP-kontroller

- TAP (Test Access Port) controller
  - 16 tilstanders tilstandsmaskin som kontrollerer testen
  - TCK og TMS er innganger
  - Kontrollsignaler (utganger) fra TAP-kontrolleren lager kontrollsignaler for andre registre.

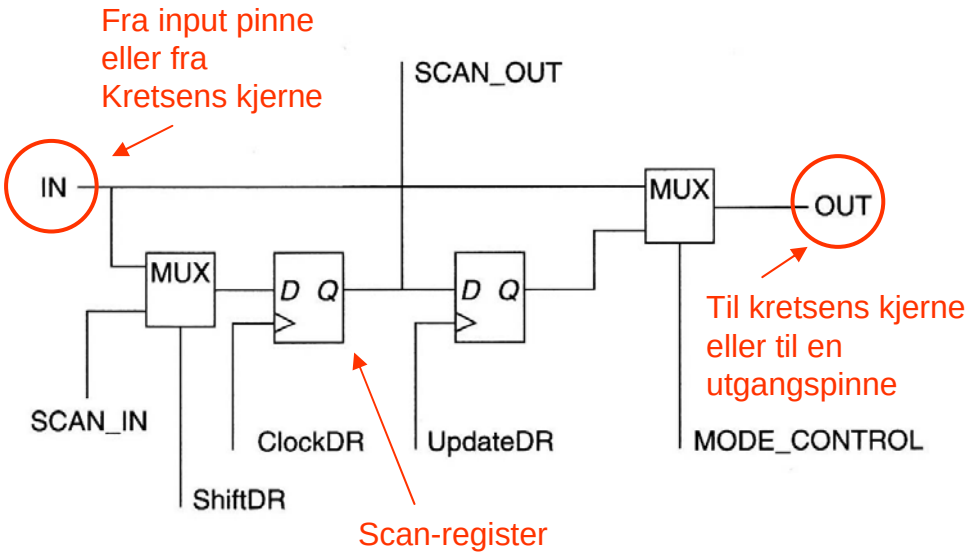


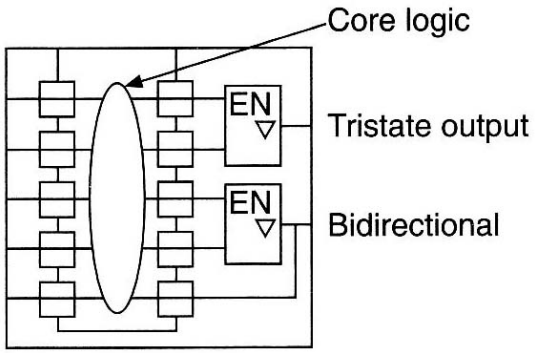
Test data registre

- Test data registre
  - Alle innganger og utganger må kobles sammen i en scan-path.
  - Benytter strukturer som i figur på neste side for å bygge opp scan-kjeden
  - I tillegg må det være et BYPASS register på et bit, som benyttes til å koble forbi en krets.
  - En krets kan også ha tilleggsregistre
    - Identification register
    - Configuration register (brukes for å programmere kretser som f.eks. FPGAer/Mikrokontrollere)

Boundary scan cellen

- Boundary scan cellen har fire modi
  - Normal mode
    - Normal flyt av data fra IN til UT
  - Scan mode
    - ShiftDR velger scan input som klokkes til Scan-registeret ved ClockDR
  - Capture mode
    - ShiftDR velger IN. Data klokkes i scan-path registeret ved ClockDR
  - Update mode
    - Etter Capture eller Scan kan data i scan-registeret klokkes til OUT ved UpdateDR.
- Utenom for tristate utganger er det bare Boundary scan cellen som skiller kjernen av kretsen fra pinnene.





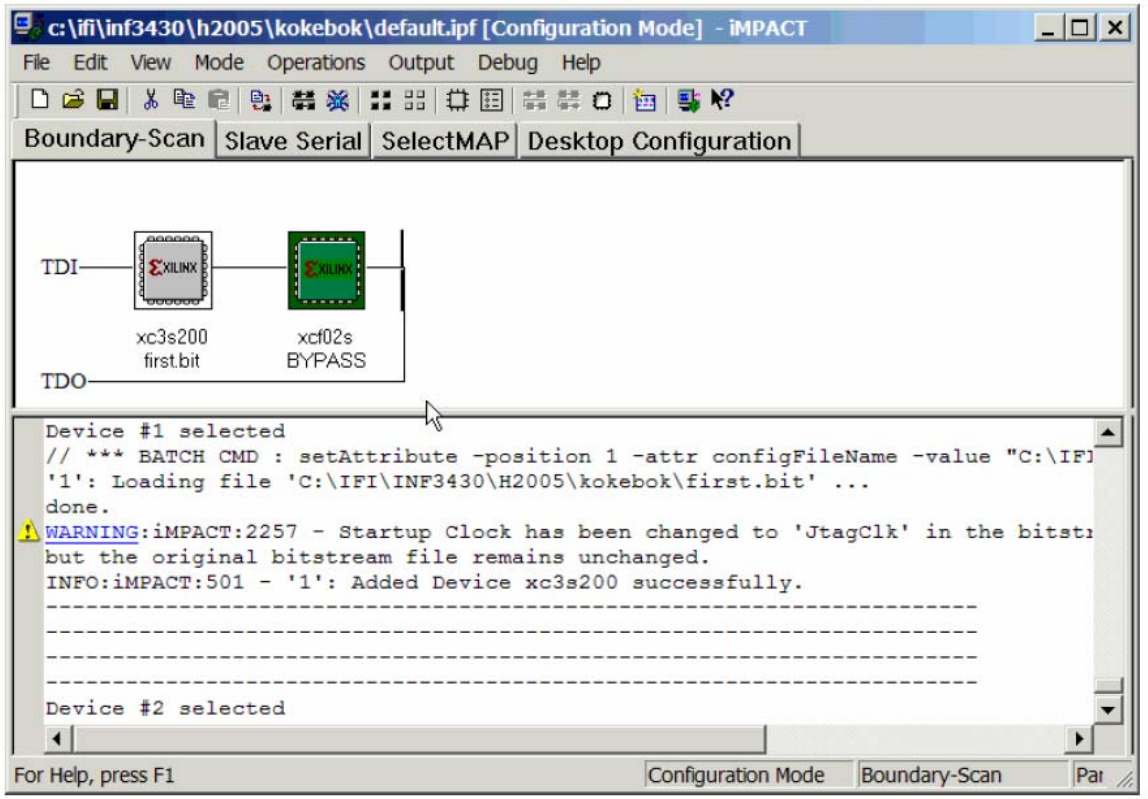
Instruksjonsregister (1)

- Inneholder minst to bit fordi tre tester er obligatoriske, men kan inneholde flere dersom flere tester er implementert
- Følgende tester er obligatoriske
  - EXTEST-tester forbindelser mellom IC'er
    - Data er sent fra utgangs boundary scan registeret fra en krets gjennom pinner og pad'er, langs ledninger, gjennom pad'er og pinner og til input boundary scan cellen til annen krets.
  - SAMPLE/PRELOAD-Setter opp utgangspinner
    - Utføres før og etter EXTEST for sette opp utganger og for å fange inn inngangssignaler
  - BYPASS
    - Kortslutter scankjeden til en krets

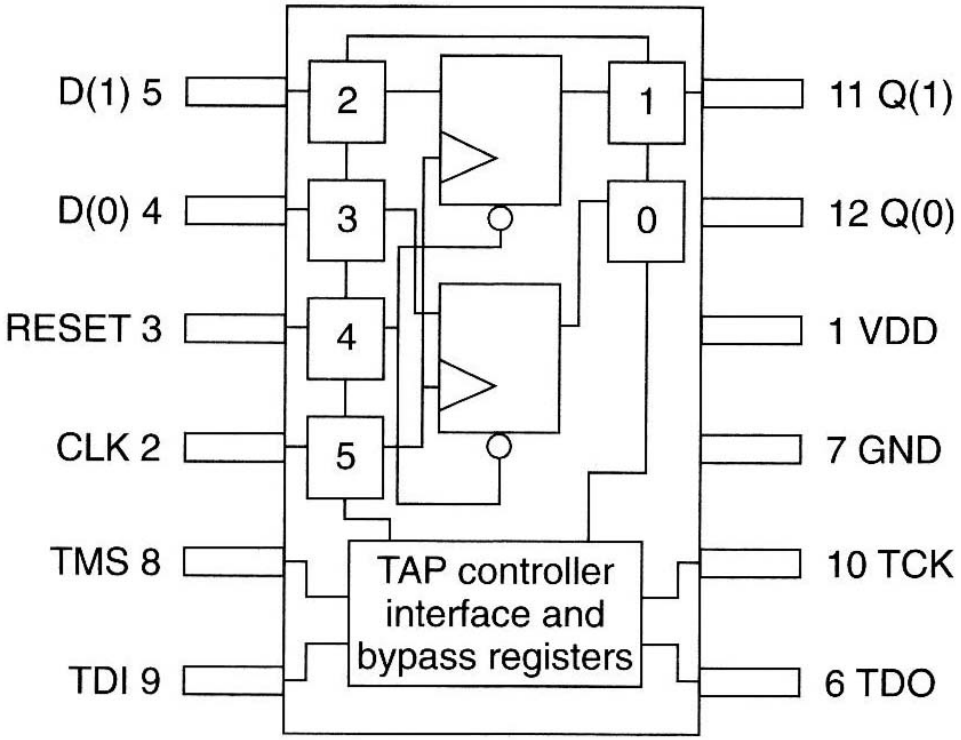
Instruksjonsregister (2)

- Følgende instruksjoner kan være aktuelle
  - RUNBIST
    - Starter å kjøre en BIST test
  - INTEST
    - Tester internt kretsløp i en krets
  - IDCORE, USERCODE
    - Returnerer ID for kretsen, og bruker ID for en programmerbar krets
  - CONFIGURE
    - Benyttes til konfigurasjon av f.eks. en FPGA eller programmering av en mikrokontroller.
    - Konfigurasjonsregistreret kan være knyttet opp mot TAP interfacet

Boundary scan på Spartan-3 testkort



IC med Boundary scan



BSDL-Boundary Scan Description Language

- For å benytte Boundary Scan mellom forskjellige kompenenter er det definert et eget språk, BSDL , for å beskrive Boundary scan funksjonaliteten på en krets
- BDSL er et subsett av VHDL

BSDL

```
24: entity dff_2 is
25:   generic(PHYSICAL_PIN_MAP : string := "UNDEFINED");
26:   port (CLK : in BIT; RESET : in BIT;
27:         Q : out BIT_VECTOR(1 to 2); D : in BIT_VECTOR(1 to 2);
28:         GND, VDD : linkage BIT;
29:         TDO : out bit; TMS, TDI, TCK : in BIT);
30:
31: -- Note. To compile using a VHDL compiler, the next line should read:
32: -- use work.STD_1149_1_2001.all;
33:
34:   use STD_1149_1_2001.all;
35:   attribute COMPONENT_CONFORMANCE of dff_2 : entity is
36:     "STD_1149_1_2001";
37:
38:   attribute PIN_MAP of dff_2 : entity is PHYSICAL_PIN_MAP;
39:   constant DIL_PACKAGE : PIN_MAP_STRING :=
40:     "CLK:2, RESET:3, Q:(12,11), D:(4,5), GND:7, VDD:1," &
41:     "TDO:6, TMS:8, TDI:9, TCK:10";
42:
43:   attribute TAP_SCAN_IN of TDI : signal is true;
44:   attribute TAP_SCAN_MODE of TMS : signal is true;
45:   attribute TAP_SCAN_OUT of TDO : signal is true;
46:   attribute TAP_SCAN_CLOCK of TCK : signal is (20.0E6, BOTH);
47:
48:   attribute INSTRUCTION_LENGTH of dff_2 : entity is 2;
49:   attribute INSTRUCTION_OPCODE of dff_2 : entity is
50:     "Bypass (11), Extest (00), Sample (01)";
51:   attribute INSTRUCTION_CAPTURE of dff_2 : entity is "01";
52:
53:   attribute BOUNDARY_LENGTH of dff_2 : entity is 6;
54:   attribute BOUNDARY_REGISTER of dff_2 : entity is
55:   --num   cell  port   function  safe
56:     "5    (BC_1, CLK,   input,    X)," &
57:     "4    (BC_1, RESET, input,    X)," &
58:     "3    (BC_1, D(1),  input,    X)," &
59:     "2    (BC_1, D(2),  input,    X)," &
60:     "1    (BC_1, Q(2),  output2,  X)," &
61:     "0    (BC_1, Q(1),  output2,  X)";
62: end dff_2;
```