Testverktyg för JTAG Boundary Scan

Erik Berggren

Kandidatarbete i datateknik

Testverktyg för JTAG boundary scan

Erik Berggren

LiTH-ISY-EX-ET--17/0463--SE

Handledare:

Kent Palmkvist
ISY, Linköpings Universitet
Markus Svensson
Syncore Technologies

Examinator:

Kent Palmkvist
ISY, Linköpings Universitet

Division of Computer Engineering
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Sammanfattning

Ett projekt har genomförts i python för att läsa och analysera nätlistor från eCAD programmet Altium. Projektet är en prototyp till en mjukvara som färdigutvecklad ska kunna användas till att automatisera kontakttest på mönsterkort mha JTAG Boundary Scan. Projektet undersöker hur stor andel av kontaktbanorna på några godtyckligt valda mönsterkort som är tillgängliga för Boundary Scan test och finner att i snitt 39% av kontaktbanorna är observerbara.

Innehåll

1 Inledning

1.1 Motivering

En viktig del av samtida elektronikutveckling är ta fram prototyper och testa kretsar innan fullskalig produktion. Typiska tester en kretskortstillverkare skulle kunna göra är att verifiera att kontaktbanor är intakta och av god kvalité, undersöka att logiken som implementeras på kretskortet utför önskad funktion och utvärdera energiaspekter såsom värmeutveckling. Det här projektet har undersökt hur man på ett enkelt sätt kan testa kontaktbanor på ett kretskort.

En traditionell metod för att undersöka kontaktbanorna (kopparbanorna i mönsterkortet samt lödpunkter) på ett kretskort är att lägga upp kortet på en nålfixtur. Nålarna är kopplade till mätinstrument och läggs emot kontaktpunkter på kortet för att mäta resistansen mellan punkterna.

I takt med att storleken på kretskort har minskat och ledningsbanor läggs i olika lager så har möjligheten att komma åt testpunkter för nålprobning minskat. Elektronikbranschen har svarat med att ta fram nya verktyg. Ett av verktygen som finns för ICT (In Circuit Test, på svenska ungefär *”elektriskttest för att verifiera krets”*) är JTAG (Joint Test Action Group) Boundary Scan [1], hädan efter "boundary scan". Boundary scan erbjuder virtuella testpinnar på kretskortet att driva eller observera signaler med. Boundary scan testpinnar är åtkommliga via ett seriellt gränsnitt och låter en kretskortstillverkare enkelt testa sina kort genom att koppla in sig på boundary scans standardport.

1.2 Syfte

Syncore Technologies vill undersöka möjligheten att testa sina kretskort för funktionalitet och integritet till självkostnadspris. Det finns kommersiella verktyg [2] som erbjuder ICT test med boundary scan men man vill ha ett egetutvecklat verktyg. Ett projekt annonserades således av Syncore för att ta fram någon sorts prototyp till ett ATPG verktyg att köra mot boundary scan. Den här rapporten sammanfattar arbetet med att utveckla verktyget.

Projektets huvudsakliga syfte blev att undersöka hur ett automatiserat test kunde sättas upp. En målsättning var att generera ett fungerande set testvektorer till en godtycklig krets.

1.3 Frågeställningar

Den ursprungliga projektannoseringen var vagt formulerad till ”Prospektera ett testverktyg med boundary scan”. Fyra mer specifika frågeställningar sattes istället för projektet.

Vad är ett bra format att hantera nätlistan med? Altium kan exportera nätlistor i många olika format, vad är ett format med brett stöd och vilka befintliga verktyg kan man eventuellt bygga vidare på?

Vilka noder kan anses testbara? Hur stor andel av näten är åtkommliga för boundary scan? Beroende på vilka komponenter som finns på ett kretskort; ska man kategorisera nät i anslutning

till olika komponenter på något sätt? Kan man använda boundary scan till att stryra komponenter och vilken respons kan man förvänta sig från dessa?

Hur genererar man testvektorer? Vilka bitmönster krävs för att hitta olika typer av fel? Hur kommer signaler propagera över kortet, vilken respons ska man förvänta sig från mer komplexa nätstrukturer?

Hur kan man verifiera att de genererade testvektorerna är korrekta? Finns det något sätt att kontrollera att generade mönstret är korrekt och med god feltäckingsgrad utan att köra mot riktigt hårdvara?

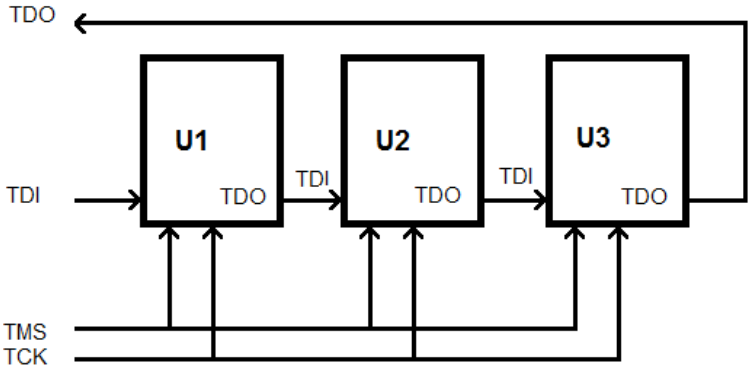
1.4 Avgränsningar

Under arbetets gång så upptäcktes att för att generera testvektorer som detekterar de flesta typer av fel så som avbrott/kortslutningar/stuck-at fel på riktiga kretskort med många nät så skulle arbetet få för stort omfång. Därför minskades den här biten till att bara sätta upp en slumpad (nonsens) bitsekvens med boundary scan registerns kedjans längd.

2 Teori

2.1 Boundary Scan

Boundary scan implementeras av microchiptillverkare i microkontrollers. Microkontrollerna löds i sin tur fast på kretskort. Boundary scan tillhandahåller virtuella testpinnar åt en korttillverkare att testa ledningsbanor och kretskortets funktionalitet emot. Boundary scan hanterar endast digitala signaler och är inte utrustat att läsa analoga pinnar på en mikrokontroller. Genom att sätta upp flera microkontrollers med boundary scan i en testkedja som i figur 1 kan tillverkaren göra tester på kretskortet utan att behöva koppla upp sig till mer än till boundary scans seriella gränssnitt.



Figur 1. Komponenter med Boundary scan stöd ihop kopplade i en testkedja. TDO, TDI, TMS och TCK är boundary scans standardiserade styrsignaler.

Boundary scan kan köras i två olika testlägen. Boundary scan är i ett läge helt transparent för logiken på kretskortet och observerar värdet på in/ut signaler från komponenterna på kortet. Enheten under test körs i sitt normala läge medan boundary scan sparar undan värdet på alla signaler det kommer åt. Observerade värden kan sen skiftas ut seriellt över TDO och analyseras av korttillverkaren.

Det andra av boundary scans testlägen låter operatören driva signaler på kretskortet medan alla boundary scan komponenters normala funktion inaktiveras. I det här läget kan testoperatören undersöka ledningsbanor på kretskortet utan att riskera att kortslutningar eller feldragna ledningsbanor skadar komponenterna på kortet. En enkel modell av ett sådant s.k. interconnect test kan ses i figur 2.

2.2 Felmodell

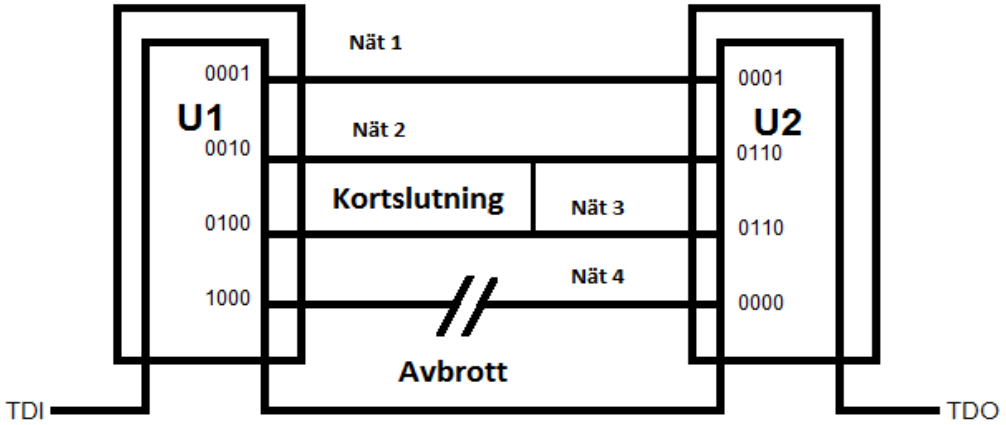
En felmodell [3] som täcker de vanligaste felen som kan uppkomma på ett mönsterkort kan formuleras enligt följande; Vi kategoriserar fyra typer av fel. Flera fel kan uppkomma samtidigt och oberoende av varandra.

Stuck-at 1 och stuck-at 0 är fel då drivaren på ett nät är fast i ett läge, konstant hög eller låg. För att upptäcka den här typen av fel ska testet driva signalen i nätet hög och låg en gång.

Avbrott är avbrott på nätet och artar sig som ett stuck-at fel. Samma strategi som upptäcker

stuck-at fel upptäcker också den här typen av fel.

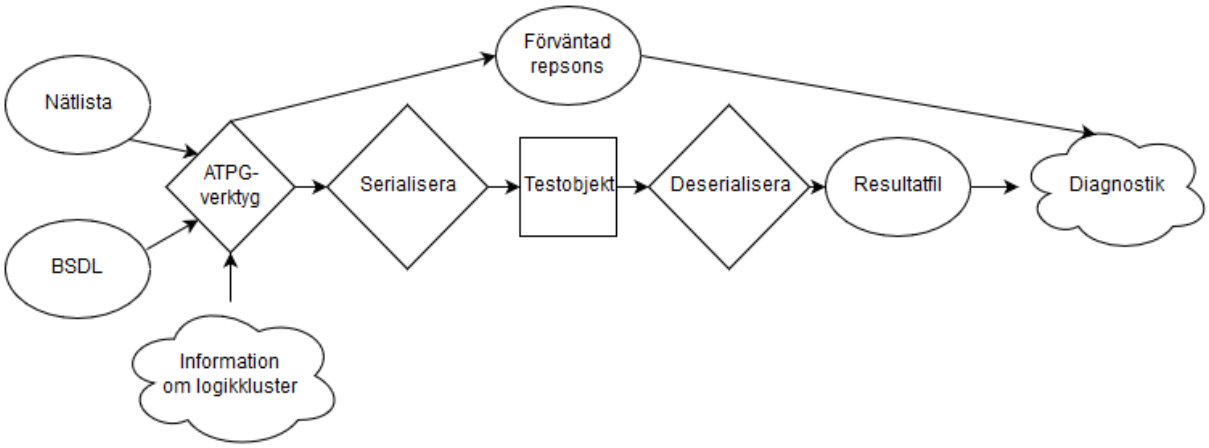
Kortslutningar kan modelleras som en AND- eller OR- grind där någon av de inblandade signalerna dominerar. För att upptäcka kortslutningar måste signaler på intill liggande pinnar drivas med olika värden vid något tillfälle.



Figur 2. Boundary scan interconnect test över fyra nät med olika typer av fel. Indata syns vid pinnarna på U1 och observerat utdata vid pinnarna på U2.

2.3 ATPG verktyg

Genom att låta en programvara analysera kretskortets strukturer och låta boundary scan driva signaler på kretskortet kan automatiserade tester för praktiskt taget vilket kort som helst göras. Denna programvara som skapar ett testmönster åt boundary scan kallas Automated Test Pattern Generator (ATPG). ATPG verktyget tar en nätlista och IC-kretsarnas BSDL (Boundary Scan Description Language) beskrivning som indata och genererar en matris av testvektorer som kan omvandlas till en seriell bitström av en s.k. serializer. Bitströmmen kan skiftas in i boundary scan kedjan, signalerna kan drivas ut över kretsen och observeras för att till slut analyseras och jämföras mot den förväntade responsen. Figur 3 illustrerar det här flödet som är typiskt för boundary scan testande [4]. N. Jarwala och C. W. Yau beskriver i sitt arbete [5] ett par olika mönster av testvektorer som ATPG verktyget kan generera och optimerar antalet testvektorer som krävs för god feltäckning vid boundary scan test.



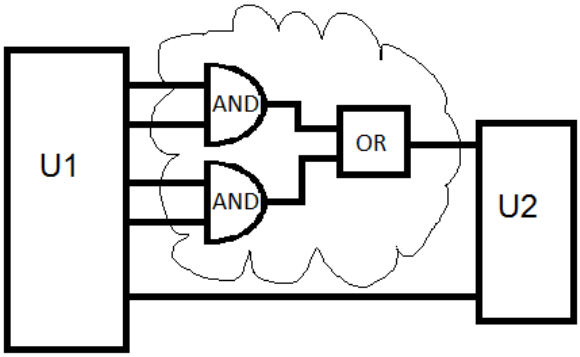
Figur 3. Flöde för ett automatiserat boundary scan test. ATPG verktyget generar vektorer som en serializer omvandlar till en bitström att driva över kresten (Testobjektet). En resultatfil genereras och jämförs mot den förväntade responsen.

Att i ett tidigt skede utnyttja boundary scan för att sätta upp automatiserade tester av kretskort låter tillverkare spara tid och pengar jämfört med traditionella bed-of-nails tester som kräver mera manuellt arbete.

2.4 Kretskort med mixade boundary scan/icke boundary scan komponenter

Det är ovanligt att ett kretskort består av enbart komponenter som stöder boundary scan. Om så vore fallet så är det enkelt att driva och observera alla signaler på kortet. När det finns grindar och komponenter som inte stöder boundary scan mellan två boundary scan komponenter som i figur 4. så måste APTG verktyget kompletteras med information om den s.k. klisterlogiken för att kunna generera testvektorer som har god feltäckning för mellanliggande näten.

Att utvärdera klisterlogik är ett komplext problem och antalet testvektorer som krävs ökar snabbt i takt med storleken på logikklustret. Det finns metoder som Roths D-algoritm [6] som löser den här typen av problem men någon analys av hur vektorer kan genereras för klisterlogik är utanför det här arbetets omfattning.

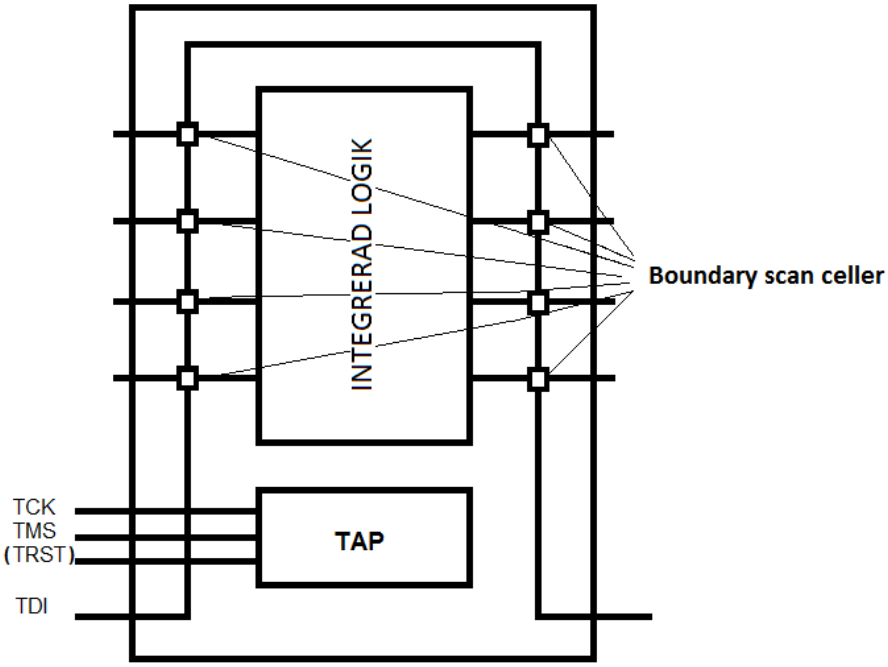


Figur 4. Information om den mellanliggande logiken måste tillhandahållas åt ATPG verktyget för att generera testvektorer som upptäcker fel i mellanliggande näten (molnområdet) ska kunna genereras.

2.5 En boundary scan implementation

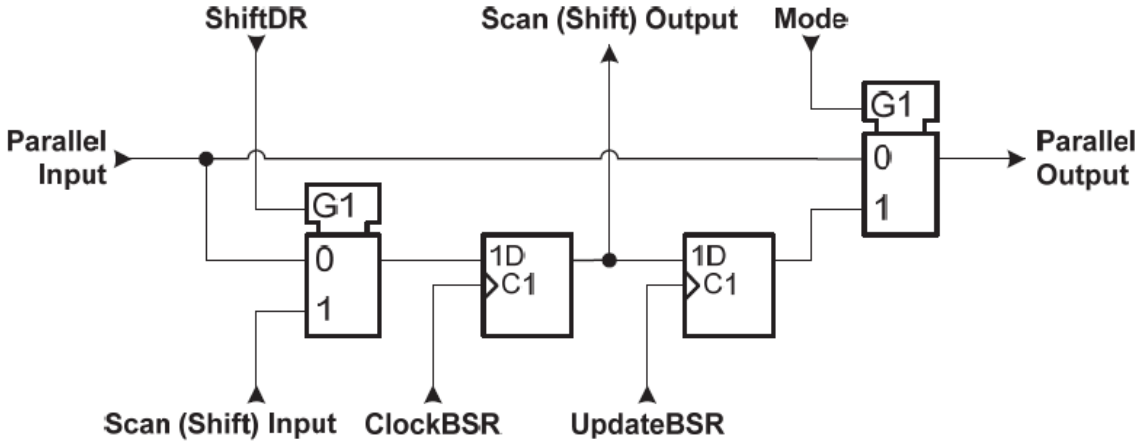
En boundary scan implementation som följer standarden [1] beskrivs i följande stycke. Implementationen är framtagen av JTAG och reviderades tre gånger innan den accepterades för första gången som standard för boundary scan av testability bus standards commitee IEEE 1988. I texten görs ingen skillnad på JTAG boundary scan och boundary scan som koncept, JTAG är en fungerande boundary scan implementation och signalerna och register namnen må vara specifika för JTAG boundary scan men skulle kunna ta andra namn eller heta vad som helst i andra implementationer, funktionaliteten som beskrivs bör dock vara den samma.

Enligt standarden består boundary scan av en kedja av boundary scan registerceller som tillsammans utgör det så kallade Boundary Scan Registret (BSR). Utöver BSR så finns det också styrlogik som styr cellernas beteende, alltsammans implementeras i hårdvara i moderna microkontrollers. Den seriella kedja som registercellerna sammanlänkas i kan betraktas som ett stort skiftregister där varje registercell kan spara värdet för signalen på pinnen som cellen är kopplad till. Data som cellerna lagrat kan sen läsas om den skiftas ut över boundary scans seriella utgång, TDO. En komponent med registerceller i anslutning till pinnarna visas i figur 5.



Figur 5. Komponent som implementerat JTAG boundary scan. I anslutning till pinnarna på komponenten sitter boundary scan registerceller.

En typisk boundary scan registercell består av en eller flera d-vippor i serie som är parallellkopplade till en pinne på en mikrokontroller. Signaler på pinnen kan sen multiplexas in på vipporna efter behov. En extra buffer gör att värdet från en tidigare scanning av pinnen kan sparas undan samtidigt som nya värden skiftas genom testkedjan. En typisk celldesign visas i figur 6.



Figur 6. Boundary Scan cell implementation (IEEE std 1149.1 - IEEE Standard for Test Access Port and Boundary-Scan Architecture, 2013, pp. 230)

Det kan förekomma flera celler i anslutning till varje pinne i en mikrokontroller. För tvåvägs pinnar, så krävs det två stycken av cellerna i figur 6 (en i vardera riktning), samt en kontroll cell som styr vilken av signalerna som drivs. Det är också möjligt att ersätta de två cellerna med en mer avancerad cell som fyller bägge funktionerna [7].

2.6 TAP(Test Action Port) kontrollern

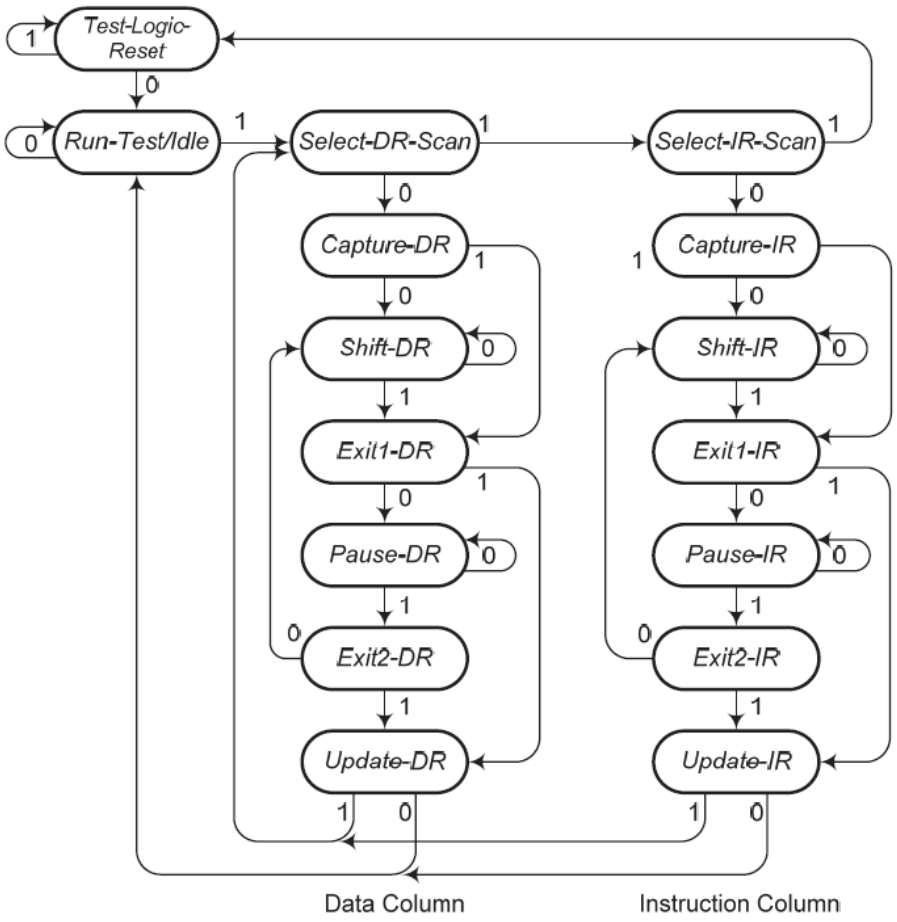
Utöver boundary scan registercellerna så implementeras i JTAG standarden också logik i microkontrollen för att generera signaler som styr cellernas beteende. Styrlogiken kallas i JTAG termer TAP (Test Action Port) och styrs via ett gränssnitt om fyra signaler (TMS, TDO, TDI, TCK).

TCK är kedjans egen klocka, på så sätt kan boundary scan köras på vilket kretskort som helst oberoende av vilka klocksignaler som kan tänkas driva komponenterna på kortet vid normal drift.

TMS (Test Mode Select) är kedjans huvudsakliga styrsignal, beroende på vilket värde TMS tar på stigande/sjunkande flank av testklockan så ändras boundary scan TAP-arnas tillstånd, se diagramet figur 7.

TDI (Test Data In) är en seriell ingång där instruktioner och test data kan skiftas in till boundary scan kedjan.

TDO (Test Data Out) är en seriell utgång där boundary scan skiftar ut testdata efter kört test.



Figur 7. Tillståndsmaskin för TAP-kontrollern (IEEE std 1149.1 - IEEE Standard for Test Access Port and Boundary-Scan Architecture, 2013, pp. 24)

JTAG standaren kräver att alla TAP:ar i testkedjan hålls i samma läge och ändrar tillstånd samtidigt. Figur 7 visar ett diagram över vilka tillstånd en TAP kan befinna sig i och hur TMS ändrar TAP:arnas tillstånd

Viktigaste tillstånden vid kontakttest (då EXTEST instruktionen är lagd i instruktionsregistret, se underrubrik *Exempel på testflöde*) på ett kretskort är Update-DR, Run-Test/Idle och Capture-DR. Update-DR laddar kedjan med data, som sen läggs ut på pinnarna vid Run-Test/Idle och sparas undan igen i Capture-DR tillståndet.

2.7 Instruktioner

Varje boundary scan komponent har stöd för minst tre stycken standard instruktioner. Vilken OP-kod som gäller för varje given instruktion går att läsa i kretsens BSDL beskrivning. Instruktionerna BYPASS, EXTEST och SAMPLE/PRELOAD är obligatoriska för att en komponent ska uppfylla standarden. Andra vanliga instruktioner är IDCODE och INTEST. Tabell 1 sammanfattar funktionen hos ett par vanliga instruktioner.

Namn	Beskrivning
BYPASS	Sätter TDI och TDO i anslutning till varandra via ett enbitars register så att komponenten exkluderas vid körning av testkedjan. Andra komponenter i kedjan kan således testas utan onödig overhead.
SAMPLE/PRELOAD	Sätter TDI och TDO i anslutning till boundary scan registret. Boundary scan komponenten behåller sitt normala körläge. I det här läget kan en scan operation ta en ögonblicksbild av signaler in/ut från komponenten. Den här instruktionen används också för att ladda data in till boundary scan registret inför en EXTEST instruktion.
EXTEST	Sätter TDI och TDO i anslutning till boundary scan registret. Komponentens pinnar kan läsas då TAP-kontrollern befinner sig i tillståndet "capture-dr" (se figur 3). Nya värden kan shiftas in i boundary Scan Registret i "shift-dr" tillståndet för att sen drivas ut på pinnarna genom att flytta TAP-kontrollerns tillstånd till "update-dr".
ICODE	Sätter TDI och TDO i anslutning till IDCODE registret. IDCODE registret är ett läsbart register som håller information om kretsen från tillverkaren (namn, modellnummer etc)
INTEST	Sätter TDI och TDO i anslutning till boundary scan registret. Det här läget låter användaren köra självtester på komponentens interna logik.

Tabell 1. Instruktioner för TAP-kontrollern.

2.8 Exempel på testflöde

För att köra ett EXTEST på kedjan ska först EXTEST eller BYPASS instruktionen laddas in på alla komponenter i kedjan.

Att ladda in en instruktion i TAP-kontrollern är i sig en procedur som styrs med TCK och TMS. Först måste man gå till Shift-IR tillståndet, skifta in instruktionen EXTEST/BYPASS via TDI till alla komponenter i testkedjan. Läsa instruktionen i Update-IR och gå till Capture-DR

Under EXTEST skiftas testdata (stimuli) in på TDI till boundary scan registercellerna i Shift-DR tillståndet. Stimuli läses efter ett antal skift in till drivarcellerna vid fallande flank på testklockan i Update-DR tillståndet. Responsen kan fångas i Capture-DR för att till slut skiftas ut och läsa i änden på TDO i Shift-DR tillståndet.

2.9 Nätlistor från Altium

En Verilog nätlista från Altium är en text fil som listar komponenter och elektriska nät i en krets. Nätlistan beskriver hur näten är ihopkopplade. Figur 8 visar ett beskuret utdrag ur en nätlistafil.

```
module D520X_SRAM
  (
   PSRAM_W_E,
   PSRAM_ZZ1,
   PSRAM_ZZ2
  );

undef           PSRAM_W_E;                          // ObjectKind=Port|PrimaryId=PSRAM.W\E\
input           PSRAM_ZZ1;                          // ObjectKind=Port|PrimaryId=PSRAM ZZ1
input           PSRAM_ZZ2;                          // ObjectKind=Port|PrimaryId=PSRAM ZZ2

wire [0 : 21] NamedSignal_MAIN_A;                   // ObjectKind=Net|PrimaryId=MAIN A[0..21]
wire [0 : 15] NamedSignal_MAIN_DQ;                  // ObjectKind=Net|PrimaryId=MAIN DQ[0..15]
wire          NamedSignal_MAIN_L_B;                 // ObjectKind=Net|PrimaryId=MAIN L\B\

(* Current_rating = "", Ibis_Model = "", Mfg_part_num = "IS66WVE4M16BLL-70BLI", Mount = "", Op_temp_range = "",
U_MEMORY_ISSI_IS66WVE4M16BLL U1300                  // ObjectKind=Part|PrimaryId=U1300|SecondaryId=1
      (
        .A1(NamedSignal_MAIN_L_B),                  // ObjectKind=Pin|PrimaryId=U1300-A1
        .A2(NamedSignal_MAIN_O_E),                  // ObjectKind=Pin|PrimaryId=U1300-A2
        .A3(NamedSignal_MAIN_A[0]),                 // ObjectKind=Pin|PrimaryId=U1300-A3
```

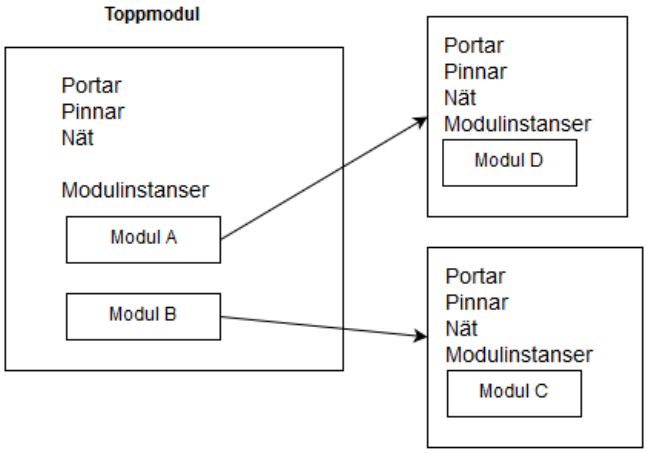
PORTAR

NÄT

MODULINSTANSER

Figur 8. Utdrag ur nätlista från Altium.

Komponenter i kretsen listas i filen som *moduler* med information om nät, portar och pinnar. Altium placerar moduler i en hierarki där en modul kan innehålla instanser av andra moduler. Dessa modulinstanser beskrivs senare i filen och kan i sin tur hålla andra moduler. Genom att ”platta till” strukturen och lyfta nät från en lägre nivå till ovanliggande modul i hierarkin, tills dessa att alla nät befinner sig på samma nivå, kan man bygga en platt modell av hela kretsen som är sökbar. Figur 9 visar ett enkelt exempel på modul hierarkin.



Figur 9. Exempel på modulhierarki ur nätlista

2.10 BSDL

BSDL är ett subset av hårdvarubeskrivningsspråket VHDL. BSDL beskriver boundary scan implementationen i en microkontroller. Särskilt intressant i det här projektet är informationen om hur boundary scan registret ser ut. I bsdl filen finns information om hur boundary scan registret är uppsatt under attributen BOUNDARY_REGISTER. Figur 10 visar hur cellerna numreras, vilken typ av cell som finnes, vilken port den sitter i anslutning till, funktion samt information om eventuella kontrollceller.

I BSDL filen finns också information om hur portarna, exempelvis. *PC13* i figur 8, mappas fysiskt till en pinne på mikrokontrollern.

```
attribute BOUNDARY_REGISTER of STM32F1_Med_density_LQFP64: entity is

--    num       cell    port            function        safe  [ccell  disval  rslt]

...

      "216      (BC_1,  *,              CONTROL,        1),                          " &
      "215      (BC_1,  PC13,           OUTPUT3,        X,      216,    1,      Z),  " &
      "214      (BC_4,  PC13,           INPUT,          X),                          " &

...
```

Figur 10. BSDL beskriving av boundary scan registret i en mikrokontroller från ST (utdrag)

2.11 Testbarhet i en nod

En nod är ett nät. Nätet kan vara testbart eller observerbart av boundary scan. Nät där det finns boundary scan celler i anslutning till ändarna är testbara av boundary scan. Nät med bara en boundary scan cell i enda änden är observbara. Det framtagna verktyget ska kunna hitta boundary scan celler på pinnarna i boundary scan komponenter och analyserar sedan näten och dess förgreningar som är kopplade på respektive pinnar.

3 Metod

3.1 Förstudie

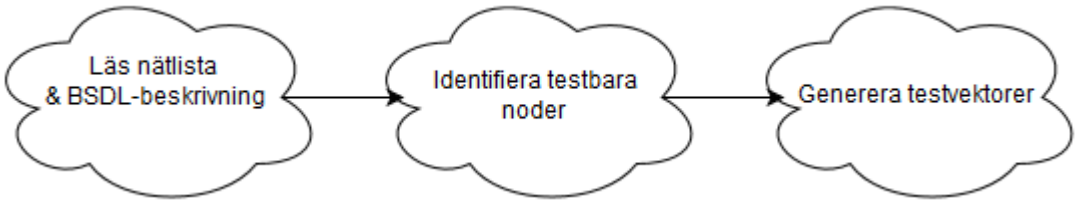
Nätlistor från Altium kan exporteras i många olika format. Användaren kan välja bland ett tjugotal olika format av hårdvarubeskrivningsspråk att exportera nätlistan som. För projektet valdes Verilog formatet. Valet av Verilog formatet motiverades med att formatet är vida spridet och har funnits en längre tid. Verilog är ett fullfjädrat hårdvaru beskrivingsspråk som klarar att beskriva en krets ner till grindnivå eller andra fritt valbara primitiv som t ex svarta lådor eller sytheserbar RTL (Register Transfer Level) kod.

EDIF (Electronic Design Interchange Format) var ett annat av nätlisteformaten som övervägdes. EDIF togs fram som ett tillverkarneutralt format i mitten av 80-talet och företagen som stödde formatet hoppades kunna förenkla översättningar mellan de många olika nätlisteformaten som då fanns. Senaste stora revisionen skedde 1996 och formatet känns utdaterat, därtill så är den mesta dokumentationen skriven kring -96, något som ledde till att formatet valdes bort.

De nätlistor i Verilog som Altium genererar är typiskt ett subset av Verilog där vissa strukturer utelämnas och en del information lämnas i form av kommentarer. Efter att läst in en nätlista så analyserades strukturerna i den givna kretsen.

ATPG-verktyg studerades för att ta fram ett arbetsflöde för projektet. Ett ATPG verktyg gör typiskt en sökning för att hitta nät som är åtkomliga från komponenter med boundary scan stöd. Data från sökningen körs sen tillsammans med data från boundary scan beskrivningen av komponenterna på kretskortet för att generera ett testmönster.

Arbetsflödet som lades upp för projektet illustreras i figur 11.

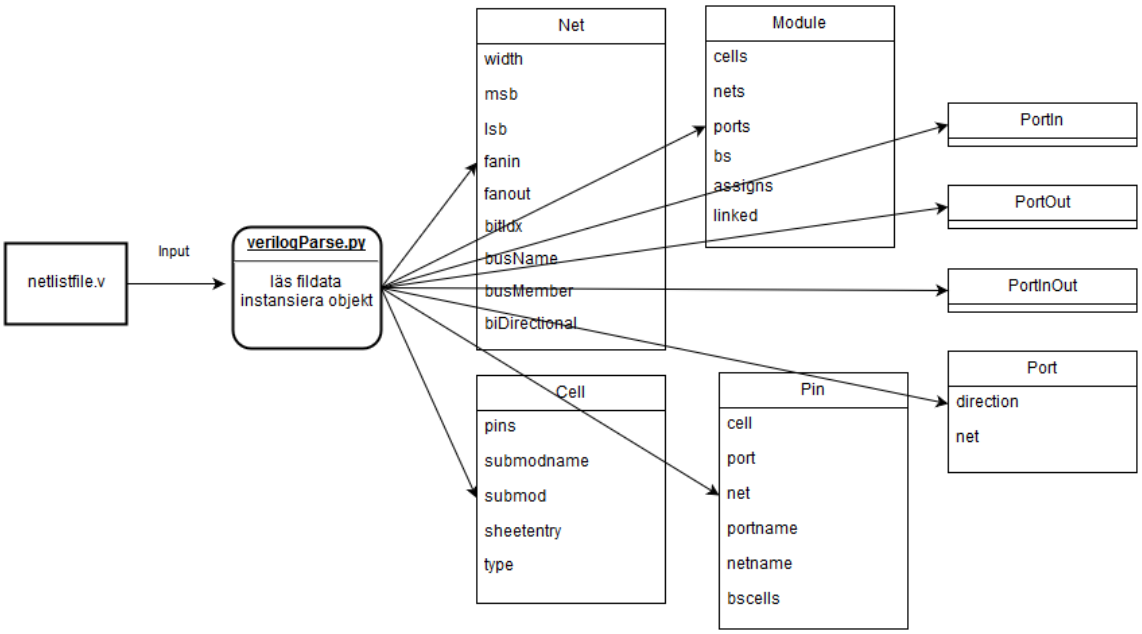


Figur 11. Arbetflöde

3.2 Implementation

Första steget i implementationsbiten av projektet var att läsa in information om kretsen från eCAD (electronic Computer Aided Design) programmet Altium. Altium är ett designverktyg för mönsterkort som låter användaren sätta upp kopplingsscheman, layouta samt simulera elektronikprojekt. Altium har en funktion för att exportera nätlistor från dessa projekt.

Ett öppet projekt för att parsa Verilog [8] användes som utgångspunkt för projektets parser. Funktionalitet för att hantera kommentarer lades till. Ändringar/tillägg i programmet gjordes för att passa parsning av Altiums nätslistor bättre. *verilogParser* klassen läser data från fil och instansierar datastrukturer som motsvarar nät, komponenter, pinnar och in/utportar i kretsscheman, figur 12.



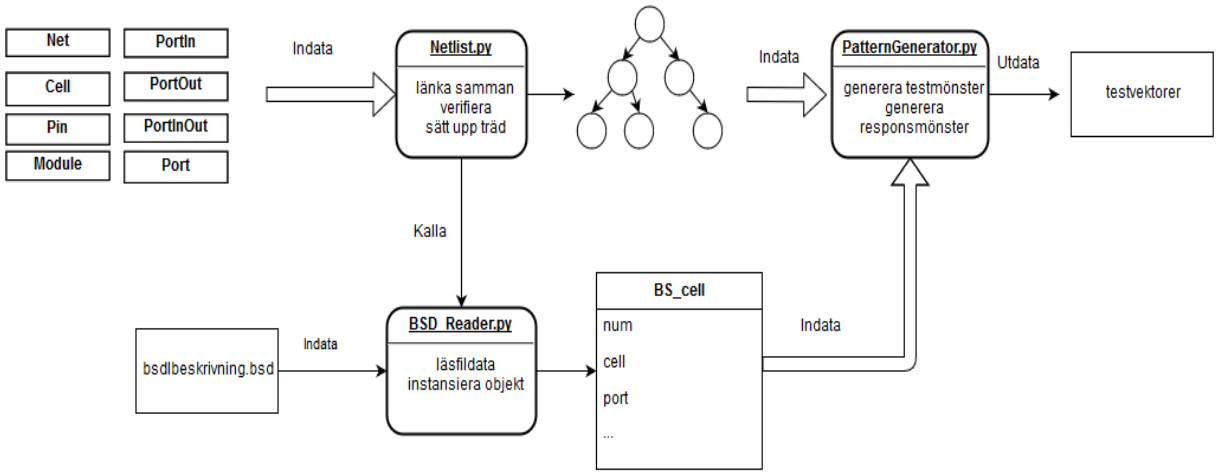
Figur 12. Strukturer som parsern instansierar

Klassen *Netlist* ansvarar för att strukturera den inlästa informationen från första steget och för att identifiera vilka nät som är åtkomliga för boundary scan test. Kopplingsscheman som exporteras från Altium delas vanligtvis upp i mindre bitar och exporteras som en s.k modul i Verilog. Klassen *Netlist* med medlemsfunktioner länkar samman alla dessa mindre moduler till en större struktur som representerar hela det elektriska nätet i kretsen. *Netlist* sätter upp det elektriska nätet i en trädstruktur efter att användaren har definierat en komponent som rotnod i trädet. Löv i trädet representeras av komponenter i kretsen och kanter av ledningsbanor mellan komponenterna. En sökning görs genom trädet efter vägar mellan rotnoden och komponenter som stödjer boundary scan. Varje sådan väg sparas undan. Vägar över kontakter, minnesbankar och andra aktiva komponenter filtreras bort, så att vägar mellan boundary scan komponenter över utvalda passiva komponenter såsom brytare, säkringar och motstånd blir kvar. En extra sökning av komponentträdet söks för att identifiera pull-up/down motstånd. Informationen om eventuella pull-up/downs sparas undan för senare bruk.

Information om vilka registerceller i boundary scan kedjan som sitter anslutna till respektive boundary scan komponents pinnar läses in med en annan python komponent. *BSD_Reader* klassen mappar inkopplade pinnar till boundary scan registerceller. I ett senare skede används denna information för att generera ett testmönster av signaler för boundary scan att driva över kretsen.

Klassen *PatternGenerator* ansvarar slutligen för att konstruera ett testmönster med hjälp av indata från bsdl beskrivningen och topologi informationen från tidigare steg. *PatternGenerator* är i vid författandet av rapporten inkomplett och förmår bara att generera en slumpad bitsekvens med boundary scan registerkedjans längd.

En sammanfattning av programflödet visas i figur 13.



Figur 13. Programflöde

3.3 Utvärdering

För att utvärdera prestandan i de framtagna scripten så gjordes en analys av hur stor andel av näten på en given krets som är observerbara i ett boundary scan test. Näten klassades efter hurvida de var observerbara eller icke-observerbara av boundaryscan. Hur stor andel av näten som fanns observerbara på några kretsar efter körning av det framtagna scriptet presenteras under rubriken Resultat.

Observerbara nät är nät som sitter i anslutning till pinne med minst en boundary scan registercell. Täckningsgraden definieras som antal av boundary scan observerbara nät genom totala antalet nät.

Matspänningsnät och jordnät kategoriserades för sig (räknades inte in bland de potentiellt testbara näten) och tar alltid namn på formen "PowerSignal_xxxxxxx" i nätlistor från Altium.

4 Resultat

Det framtagna scriptet kördes på olika nätlistor från tre olika mönsterkort. Kretsarna är på riktiga projekt med olika karaktär såsom varierande antal boundary scan komponenter, med eller utan RAM-minnen och olika yttre gränssnitt. Resultaten presenteras i tabell 2.

Kort	Antal Nät	BS observerbara nät	Täckning
1	534	141	26%
2	150	37	25%
3	127	84	66%

Tabell 2. Testbarhet olika kort

Kort ett har två boundary scan komponenter och är en "stor" krets i det avseendet att den har många gränssnitt, en del knappar och ett par signalförstärkare. Kretsen används för att spela in fågel och djurläten i naturen. Bland annat finns det minnen och USB-kontakter samt en ethernet utgång som kan användas för att tanka hem inspelad fågelsång. Värt att notera att det inte finns något JTAG gränssnitt utan pinnarna på boundary scan komponenterna avsett för JTAG används till andra ändamål.

Kort två har en boundary scan komponent som också är kretsens enda microkontroller. Denna mindre krets används i en luftavfuktare och har en del sensorer kopplat till sig. Krets två implementerar inte heller något JTAG gränssnitt även om mikrokontrollern stöder det.

Kort tre är ett utvecklingskort från mikrokontrollertillverkaren ST-Microelectronics. Test kortet har två mikrokontrollers som stöder boundary scan. Kretsen är relativt liten med 127 nät i den exporterade nätlistan. Här finns ett JTAG interface implementerat och en komplett testkedja som utgörs av de två mikrokontrollerna.

I genomsnitt så kan man direkt observera 39% av näten på de tre kretsarna, en relativ låg siffra som kan förklaras med att endast ett kort var designat med test i åtanke.

5 Diskussion

5.1 Resultat

För att maximera testtäckning så krävs det att korttillverkaren gör ett betänkande och väljer komponenter som stöder boundary scan och implementerar en testkedja på kretskortet. De kretsar som undersökes i det här arbetet hade i snitt under hälften av totala antalet ledningsbanor tillgängliga för boundary scan test. Projektet undersökte i slutändan endast tre olika kretskort, väldigt få om man ska dra några långt gående slutsatser, något som förklaras med att mycket av tiden gick åt till studium av testmetodik på mönsterkort och tiden att skriva kod/debugga underskattades.

5.2 Metod

Inga tester gjordes mot riktig hårdvara. En hårdvarudrivrutin till boundary scan som skriver det av projektet generade testmönstret och verifierar utkommande mönster är en tänkbar vidareutveckling på projektet. BSDL parsern är visserligen fungerande men skulle behöva utökas för att hantera alla tänkbara definitioner, i dagsläget ignorerar den del definitioner i .bsd-filen som inte behövs i projektets begränsade miljö. För att testa minneskluster så behöver ATPG verktyget ha kännedom om hur man adresserar och läser/skriver databanker på minnena, alternativ initierar ett self-test på minnesmodulerna. Det här kräver att väldigt många vektorer genereras och är något som det här projektet inte omfattar.

Precis som i fallet för att testa minneskluster så krävs det en del extra information som utvärderar funktionen i logikkluster mellan två boundary scan komponenter om dessa ska omfattas av ett interconnect test. Att analysera sådan s.k klisterlogik omfattas inte av arbetet men är en tänkbar framtida utvidgning.

5.3 Nästlisteformat

Altium kan exportera en nätlista ur en design i ett tjugotal olika format. Det breda stödet med många format kan förklaras med att Altium ska kunna användas tillsammans med många olika typer av design- och simuleringsverktyg.

Bara två format är fullfjädrade hårdvarubeskrivning språk och bägge har breddstöd inom EDA-industrin (Electric Design Automation); VHDL och Verilog. Andra format som XSPICE lämpar sig bättre för att användas med simuleringsverktyg eller är nichade för andra eCAD-program. Efter en första sållning så kom valet ner till hurvida man skulle använda VHDL eller Verilog.

VHDL och Verilog är likvärdiga när det kommer till att beskriva hårdvarustrukturer, vilket är det en nätlista gör. Vilket format som väljs i ett arbete handlar ofta om preferens. Några viktiga skillnader språken emellan är att VHDL är starkt typat, i ett avseende gör det programmering i VHDL robustare då inga implicita typkonverteringar får förekomma och kompilatorn kommer ge fel om en 2 bitars signal kopplas till en 16 bitars signal och dylikt. Verilog är svagt typat och flexiblare i detta avseende men ett större ansvar vilar hos den som programmerar att det blir rätt. Andra skillnader är att VHDL har stöd för användardefinerade typer medan Verilog saknar sådant stöd. När man jobbar med nätlistor torde det inte förekomma några användardefinerade typer, alla typer deklareras av Altium som nät eller bussar och liknande. Inom projektets ramar utnyttjades inte beteendebeskrivningsmöjligheterna hos något av språken. VHDL påminner om ADA i syntax

medan Verilog är mer likt C.

Pågrund av sin flexibilitet och likheter med C tillsammans med artikelförfattarens tidigare erfarenheter av C-programmering så valdes Verilog formatet.

5.4 Felmodell och testmönster

Projektet kom aldrig till att generera riktiga testmönster för en godtycklig krets. Problem med multidriva nät som är vanligt förekommande på riktiga kort gör att den enkla felmodell som beskrivs i 2.2 inte räcker till. Ett fel nära någon av drivararna, innan nätet grenar ut till en observerande boundary scan cell skulle inte kunna upptäckas av den enkla modellen i 2.2. ATPG verktyget bör då ha någon sorts tabell för att hålla kolla på vilka drivare som sitter på samma noder för att kunna generera ett lämpligt mönster. En mekanism för att hålla en drivare i multidrivna nät inaktiv behövs dessutom för att inte konflikter eller tröskelvärden på spänningar ska överstigas och skada kretsen. Allt som allt fanns inte tiden till att implementera något av dessa mekanismer och mönstergenereringen stannade därför vid att en slumpad bitsekevens (nonsensvärden) i boundary scan registerkedjans längd skapades.

5.5 Arbetet i ett vidare sammanhang

Att snabba upp och automatisera testfasen av elektronikutveckling tjänar två syften; fler konstruktioner kan testas under en given tid och mindre manuellt arbete behövs till testprocessen. Resurser kan prioriteras på annat håll om en välfungerande automatiserad testmiljö kan själv forma ett utförligt set testvektorer att köra mot en design. Mänskliga faktorn försvinner om en maskin tar fram testmönstret, risken att något överblickas eller felaktiga data nertecknas försvinner. En risk är att systemfel i programvaran istället konsekvent missar någon typ av diagnoserbart fel eller att andra buggar ger dåliga utdata. En noggrann granskning av programvaran bör således föreligga innan den tas i bruk på en större skala.

6 Slutsatser

I rapporten presenteras ett enkelt boundary scan ATPG verktyg utvecklat i python. Verktyget används för att analysera hur stor andel näten man kan testa på ett kretskort mha JTAG boundary scan. Verktyget ska betraktas som en grov prototyp och tar indata i form av nätlistor från eCAD programmet Altium och ett .bsd filformat från chiptillverkare.

Analysen av hur stor del av näten som var observerbara av boundary scan är snabbt utförd och besitter visst värde i det att man får en bild av hur lämpat ett kretskort är för boundary scan test. Är observerbarheten av nät låg på ett kretskort så kanske andra metoder av ICT test lämpar sig bättre. Projektet körde verktyget mot ett par olika kretskort och fann att i snitt så var 39% av näten på korten observerbara av boundary scan.

En av projektets målsättningar var att generera ett set med testvektorer med hög felteckningsgrad för en godtycklig design som indata. Det här målet uppnådes inte på grund av att komplexiteten i att analysera alla möjliga typer av nät och komponenter som kan förekomma på ett kretskort.

Mycket tid gick åt att läsa in sig på boundary scan standarden och olika strategier för att generera test vektorer. Att generera vektorer som har god feltäckning på stora kretsar visade sig vara ett svårt problem som påminner som Königbergs sju broar problemet så tillvida att man måste undersöka hur en signal propagerar genom grafen som det elektriska nätet utgör. Dessutom måste man ta i beaktande vilka komponenter som kan tänkas ändra dess värde och sen mata den här informationen in till APTG-verktyget.

Större avgränsning i början av projektet hade möjligtvis hjälpt projektet till ett bättre verktyg än det som togs fram. Nu lades ribban högt och målen uppnåddes inte riktigt.

Faktiska nyttan av det framtagna python scripten kan diskuteras, erfarenheterna att jobba med boundary scan får ses som det egentliga värdet som kom ur arbetet. Verktygets förmåga att läsa nätlistor och sätta upp datastrukturer kan säkert återanvändas i framtida projekt.

7 Referenser

[1] IEEE Std 1149.1 - 2013 IEEE Standard for Test Access Port and Boundary-Scan Architecture.

[2] Corelis Inc. “ScanExpress TPG” http://www.corelis.com/products-JTAG/ScanExpress_TPG_Test_Pattern_Generation.htm, (Hämtad 2016-10-02)

[3] A. Jutman, “Board-level testing and IEEE1149.x Boundary scan standard” [Powerpoint-presentation] http://www.gojtag.com/sites/default/files/BS%20testing%20-%20goJTAG%20v.11.02%20print.pdf (Hämtad 2016-12-03)

[4] H. Bleeker, P van den Eijnden, F de Jong. 1993. "Architechture of a Boundary-Scan Test Flow" *Boundary-Scan Test - A Practical Approach*. Kluwer Academic Publishers, pp. 166-167

[5] N. Jarwala, C. W. Yau, "A new framework for analyzing test generation and diagnosis algorithms for wiring interconnects," *Test Conference, 1989. Proceedings. Meeting the Tests of Time., International*, Washington, DC, 1989, pp. 63-70.

[6] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," in *IBM Journal of Research and Development*, vol. 10, no. 4, pp. 278-291, July 1966.

[7] Kenneth P. Parker. 2003. "The Boundary Scan Register" *The Boundary Scan Handbook*. Springer Science pp. 21-25

[8] P. "etep" Stevenson. "Python-based Verilog Parser" https://github.com/yellekelyk/PyVerilog (Hämtad 2016-06-09)