
# Boundary Scan

# System M-1

# Handbuch

Doc. Version 2017-07-04-1

**Author: Mario Blunk**

Abstrakt: *Einführung in System M-1. Betriebsanleitung, Installation und Wartung, Grundlagen zur Testgenerierung mit Boundary Scan nach Std. IEEE 1149.1 auch bekannt als JTAG*

Schlüsselwörter: *Boundary Scan, OpenSource, JTAG, IEEE1149.1, Netzliste, Partliste, Teileliste, Import, hierarchisches Design, Datenbank, UUT, Prüfling, Modul, Prefix, Steckverbinder, Bautteil, Netz, Klasse, pull-down, pull-up, Kurzschluß, Unterbrechung, Testabdeckung, Primärnetz, Sekundärnetz, FPGA, CPLD, programmierbare Logik, Zelle, Input, Output, High-Z, open-drain, open-collector, Scanpfad, Scanport, TAP, Scankette, Einzelschritt, Breakpoint, Stromverbrauch, Verzögerung, Pause, Timeout, Abschaltung, Speicher, RAM, FLASH, I²C, SPI, Microwire, sicherheitsrelevant, Skripte, Batch, bash, Linux, Ada, gtkada, gnat, gcc, git, Versionskontrolle, ASC-II, intrusive, non-intrusive, Development Mode, Produktionsmodus, graphische Bedienoberfläche, GUI*

# Inhalt

# 1. Einführung

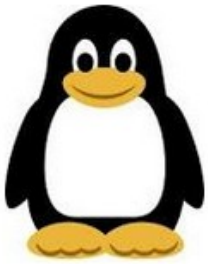
## *1.1. Was ist Boundary Scan ?*

Boundary Scan (deutsch: Tasten am Rand) ist ein adapterloses strukturelles Prüfverfahren welches:

1. Fertigungsfehler bis auf Pin/Pad-Level detektiert.
2. das Prüfen von ICs, bestückten Leiterplatten und ganzen Systemen ermöglicht.
3. elektronischen Zugriff auf physisch nicht kontaktierbare Netze und Pins/Pads gewährt.
4. den elektrischen Zugang zum UUT/Prüfling auf 5 Signale reduziert.
5. den UUT/Prüfling unter Betriebsspannung prüft.
6. in System Programmierung (ISP) ermöglicht.
7. auf im Silizium eingebetteten elektronischen Testpunkten basiert.
8. nach IEEE 1149.x standardisiert ist (seit Anfang 1990er Jahre).

## *1.2. Warum Boundary Scan mit System M-1 ?*

***Mit Boundary Scan sparen Sie Geld und Zeit !***



- Keine Lizenz – OpenSource !
- Kein Wartungsvertrag !
- Bereits in der Enwurfphase eines Prototypen werden Fehler festgestellt.
- Während der Inbetriebnahme von Prototypen werden auf strukturellem Level Fertigungsfehler festgestellt (Kurzschlüsse, Unterbrechungen, fehlende Bauteile).
- Gerätesysteme (Racks, Backplanes, Einschübe, …) können strukturell geprüft werden.
- Vereinfachte, beschleunigte Reparatur
- Weniger mechanische Ausrüstung (keine Adapter, weniger Funktionstests)
- Schnelle, automatische Testgenerierung (Build-Prozesse, Scripting)

Lesen Sie mehr hier: http://www.blunk-electronic.de/bsm/how_to_test.pdf

### *1.3. Die Technolgie nach IEEE 1149.1 im Prüfling/UUT (Kurzfassung)*

Um den eigentlichen Logikkern eines IC (an dessen Grenze/Boudary) befindet sich ein Register aus Testpunkten. Mit diesen kann ein Pin/Pad gemessen oder angesteuert werden.

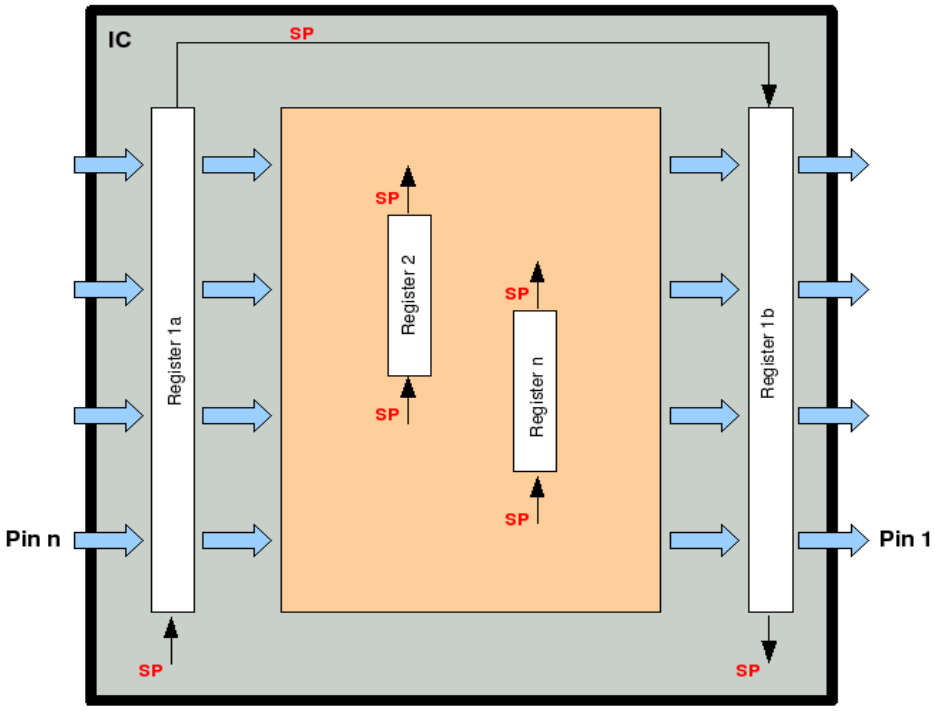



*figure 1: Ein boundary scan fähiger IC von innen*

Zur Bauteilerkennung, zum Programmieren oder Debuggen können auch im Kern Register eingebaut sein.

Über einen seriellen Scanpfad (SP) werden Testdaten in die Register geschoben oder aus diesen gelesen.

Auf MCMs (Multi-Chip-Module) können Testdaten zwischenden den ICs und der Außenwelt ausgetauscht werden:

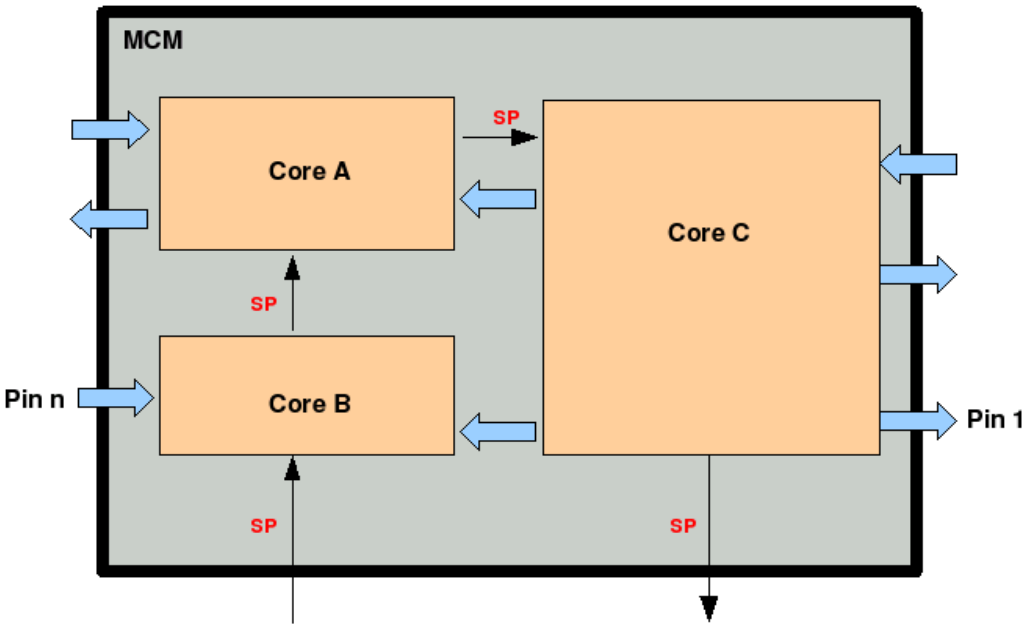



*figure 2: ein boundary scan fähiges Multi-Chip-Modul von innen*

Zwischen ICs und MCMs einer bestückten Leiterplatte werden Testdaten ausgetauscht:

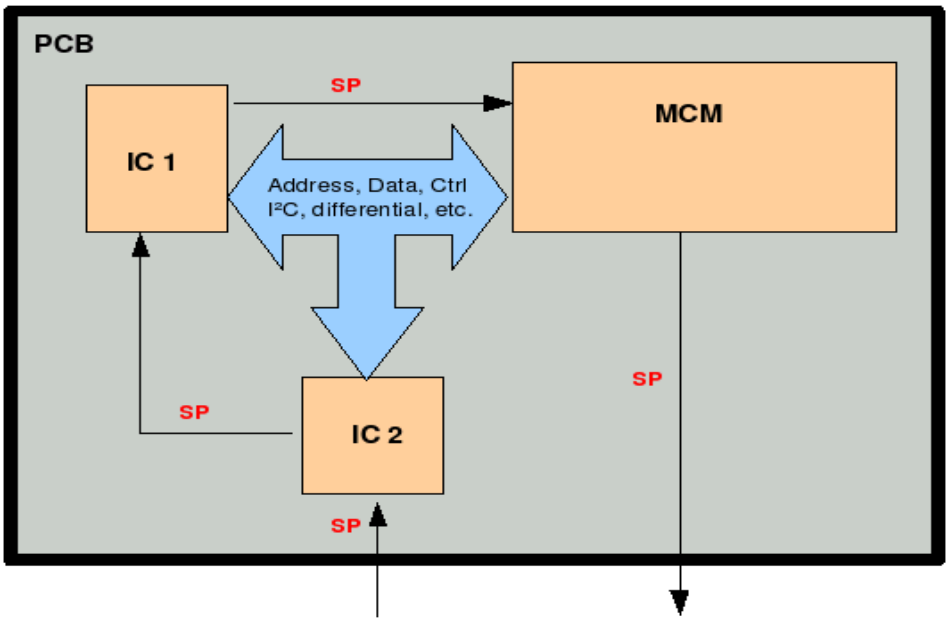



*figure 3: eine boundary scan fähige Leiterplatte mit einem Scanpfad*

In Gerätesystemen können die Zwischenverbindungen (Backplane, Verdrahtungen, …) geprüft werden:

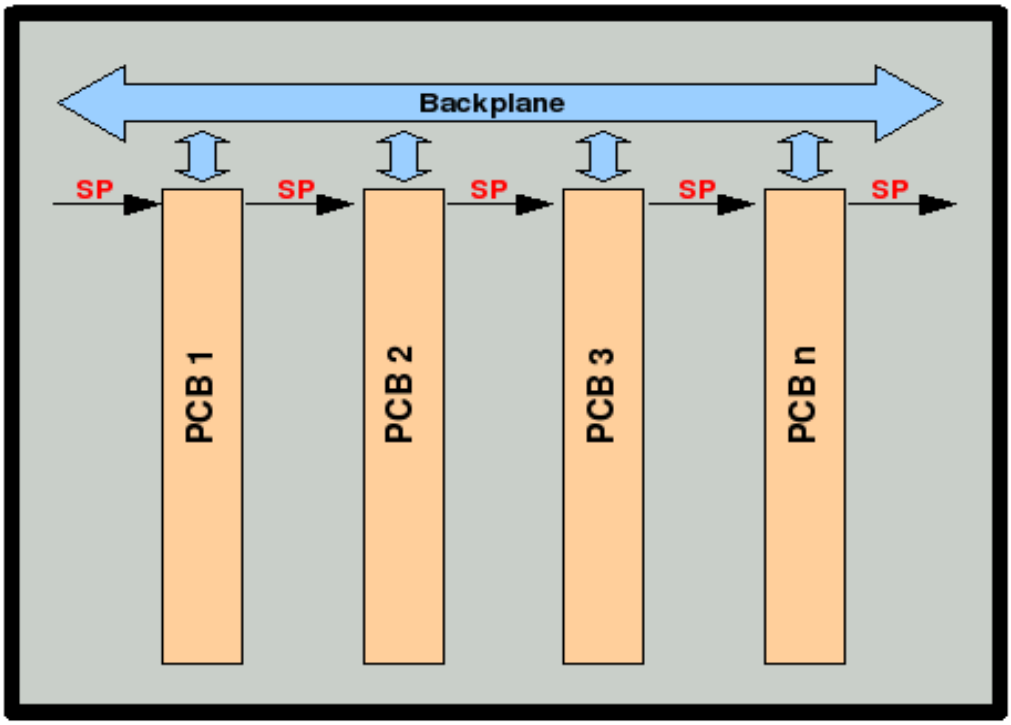



*figure 4: ein boundary scan fähiges Gerätesystem*

Dies ist sehr wertvoll für den **System-Test im Feld**:

- medizinische Gerätesysteme
- Steuerungen in Windkraftanlagen
- Fahrzeugsteuerungen
- Industriesteuerungen

Die „Verdrahtung“ auf der Leiterplatte:

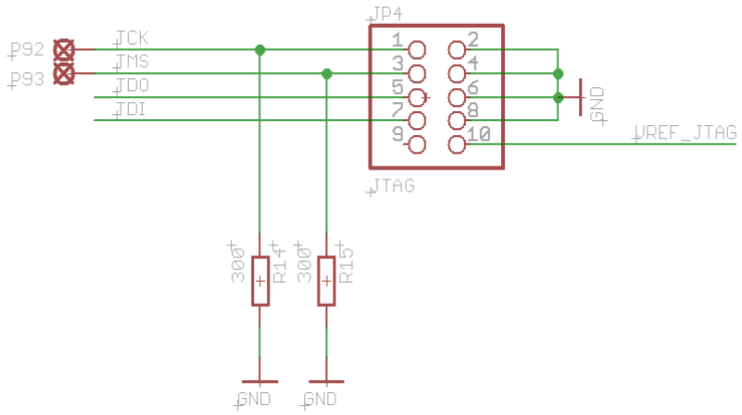



*figure 5: Scanpfad Steckverbindung*

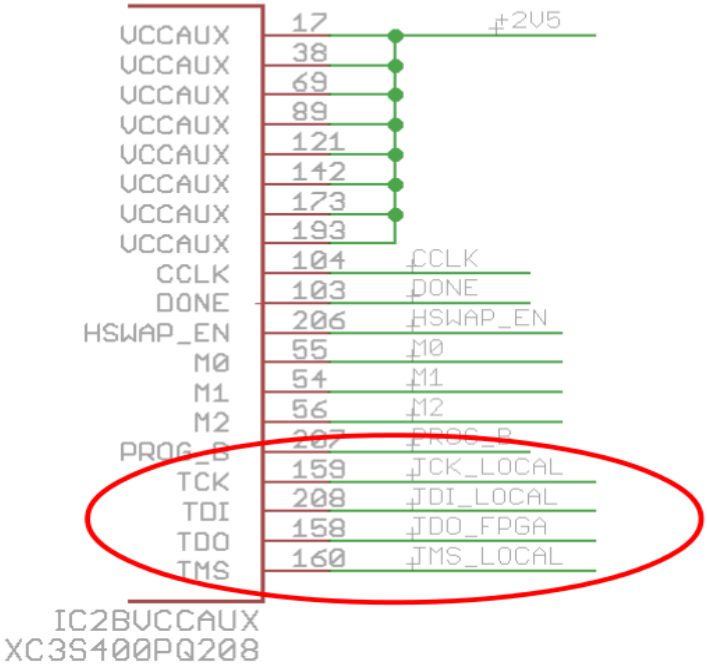



*figure 6: Scanpfad am IC*

### 1.4. Finanzielle Investition

Um gleich zum Punkt zu kommen: *System M-1* ist primär nicht käuflich zu erwerben, sonder wird gemietet. Damit entschärft sich die Frage nach dem Kaufpreis etwas:

- Der Kunde mietet das *System M-1* solange es nötig ist oder sie/er kann es kaufen.
- Es gibt keine verwirrende Features vs. Lizenz-Tabelle für die Software (weil OpenSource). Der Kunde bekommt eine Entwicklungs-, Reparatur und Ausführstation in einem Paket.
- Kosten entstehen durch die Miete oder den Kauf des Boundary Scan Controllers (BSC)+ PC, Schulungen, Einrichtung der Testprogramme und Support.

*Bitte kontaktieren Sie die Firma Blunk electronic für ein Angebot.*

### 1.5. System-Überblick

Das System besteht aus:

- einem gewöhnlichen Linux-PC,
- dem sogenannten Boundary Scan Controller (im Folgenden bezeichnet mit BSC),
- dem eigentlichen Prüfling (im Folgenden auch mit UUT bezeichnet).

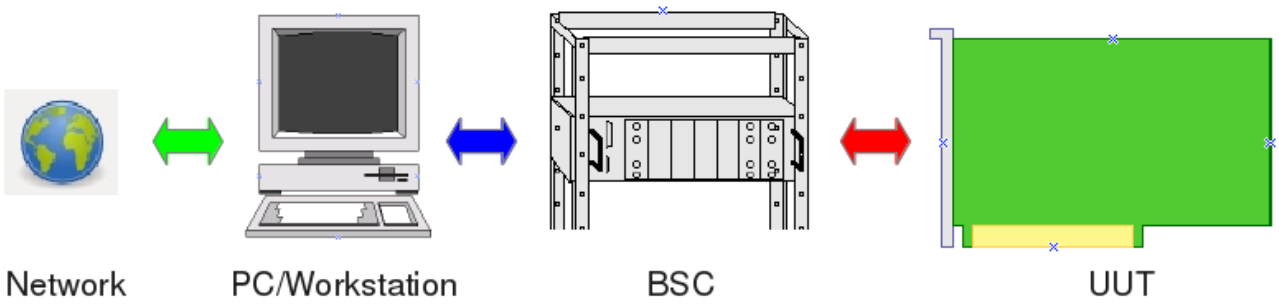



*Figure 7: System M-1 components*

Mit dem PC werden projekte verwaltet und Testprogramme generiert. Der BSC führt die Testprogramme aus indem serielle Daten zum UUT gesendet und von diesem empfangen werden.

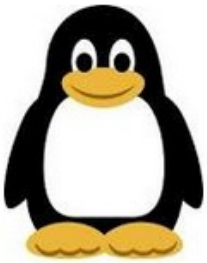
### *1.6. Funktionen*

Einige wichtige Eigenschaften sollen hier aufgeführt werden:

- *System M-1* umfaßt eine fertig installierten PC für die Testgenerierung. Der Kunde ist von Installation und Implementierungen befreit.

- Wartung und Support können per Fernsteuerung über das Netzwerk (lokales Netz oder Internet), was Reisekosten und Vorort-Service reduziert.

- Das Betriebssystem Linux©, welches auf dem PC läuft, zeichnet sich durch sehr stabilen, virus-immunen Betrieb aus – eine vernünftige Wahl für den Einsatz in Produktivumgebungen.



- Skripting/Stapelverarbeitung für automatische Generierprozesse

- Kritische Teile der Software des *System M-1* sind in der Programmiersprache Ada 2005 geschrieben, was für sicherheitsrelevante Anwendungen elementar ist.


- Die Software ist **offen für kundenspezifische Anwendungen oder graphische Bedienoberflächen** (GUI) !

- Die Software ist modularisiert. Ein Update kann für ein einzelnes Modul durchgeführt werden, statt für das ganze System Jedes Modul kann mit externer Software gesteuert werden, um z.B. in andere Testsysteme wie Flying-Probe-Test, In-Circuit-Test, AOI, System Test, QM integriert zu werden

- Die Software auf dem Linux-PC ist OpenSource (GNU General Public License).

- Die Mehrheit der Dateien, die von den individuellen Softwaremodulen erzeugt werden, sind streng ASC-II formatiert (also Klartext). Sie sind somit vom Menschen lesbar und editierbar. Damit ist Versionskontrolle, z.B. mit Git (https://git-scm.com) möglich.

- Das gesamte Softwarepaket des *System M-1* wird fix und fertig installiert ausgeliefert. Der Kunde muß sich also nicht befassen mit der Installation vom

  - ✔ Betriebssystem
  - ✔ Hardware Plattformen
  - ✔ Gerätetreibern
  - ✔ Lizensen

- Grundsätzlich bietet der BSC eine Stromüberwachung von 3 unabhängigen Versorgungsspannungen um die UUT vor Überstrom zu schützen.

- Grundsätzlich bietet der BSC vollständige galvanische Trennung des UUT bei Überlast, Test FAIL oder im Non-Test-Modus. Diese Forderung ist elementar wenn der BSC mit anderen Testsystemen wie ICT oder FPT interagiert.

- Der BSC hat ein robustes 19 Zoll Gehäuse mit robusten Steckverbindern für schnellen und sicheren Prüflingswechsel.

- Der BSC kann unabhängig vom PC betrieben werden. Für schnelle Vorprüfungen ist dies sinnvoll. Der Bediener braucht nur die START und STOP-Tasten an der Frontseite des BSC drücken. Die beiden LEDs zeigen PASS oder FAIL eindeutig an (siehe figure 8 Seite 13 und Abschnitt 9 Tests-Ausführung (run) auf Seite 33).

Standardfunktionen eines Boundary Scan System nach IEEE1149.1, die sein **müssen**, sind im Folgenden gelistet, ohne näher auf Einzelheiten einzugehen:

- Import von CAD-Daten (Netz- und Partlisten) aus CAE-Systemen (z.B. Altium/Protel, KiCad, ZUKEN, EAGLE , …)
- Minimaler UUT-Zugang via einen 5-adrigen seriellen Testbus (sogenannter TAP)
- Einstellbare TAP/Testbusspannung (1.8V bis 3.3V)
- Fehlererkennung bis auf Pin-Ebene (stuck-at, shorts, opens, fehlende Pull-Widerstände)
- Scanpfad-Test oder Infrastruktur-Test
- Verbindungs-Test oder Interconnect Test
- Speicher/Cluster-Interconnect Test (RAM, FLASH)
- Clock Test
- Pin-Toggle Test
- System Test

# 2. Betrieb und Komponenten

## *2.1. Lizensierung*

Wie in Abschnitt 1.4 Finanzielle Investition Seite 9 ausgeführt, ist KEINE Lizenz nötig um die Software zu betreiben. Sie bekommen ein Vollsystem ohne Einschränkungen.

Betreffend der Hardware, also des BSC, bezahlen Sie für dessen Miete oder Kauf.

## *2.2. Software*

Die Software des *System M-1* ist im Wesentlichen eine Zusammenstellung einzelner ausführbarer Linux binaries (ELF) – eine sogenannte Toolchain. Für die Testgenerierung werden diese per Kommandozeile plus Parameter gestartet. Im Produktionsmodus werden Tests per graphische Oberfläche gestartet (siehe Abschnitt 14 Produktions-Modus Seite 39).

Auf UNIX oder Linux basierenden Betriebssystemen ist es üblich, eine Applikation via Kommandozeile oder graphische Oberfläche (GUI) bedienen zu können. Wichtige taktische Funktionen zur Test Generierung und Ausführung sind in diversen ausführbaren Dateien implementiert, welche individuell gestartet werden können durch:

a) lokale Kommandozeile (Konsole oder Terminal)

b) ein sehr schlankes GUI zur Testausführung

c) ein anwenderspezifisches, kundenorientiertes GUI

d) eine externe Appliktion (wie z.B. Flying-Probe-Test, In-Circuit-Test, …) via System-Aufrufe.

e) per Netzwerk (ssh, remote desktop, vnc, ...)

Die Tatsache von individuellen ausführbaren Dateien macht das System-Update und die Fehlersuche einfach. Eine fehlerhafter Binärdatei kann einfach ausgetauscht werden anstatt das gesamte System zu aktualisieren.

Zusätzlich soll hier darauf hingewiesen werden, daß das System **offen** ist für jede Form von Erweiterung durch den Anwender oder Anbieter von Drittsystemen (FPT, ICT, AOI, …). Es hat sich außdem gezeigt, daß graphische Bedienoberflächen einer Vielzahl von Wünschen und Forderungen ausgesetzt sind und in ihrer Entwicklung und Bedienbarkeit ineffizient und überzüchtet sind (siehe Abschnitt 2.2.2).

### 2.2.1. Versionskontrolle und Softwaretests

Mit Ausnahme der vom BSC ausgeführten Binärdatei (des Testes) sind **alle** Zwischenstufen der Testentwicklung reine ASCII-Dateien (Klartext). Dies ist ein elementares Kritierium für Versionskontrolle (z.B. mit *Git )* Softwaretests.

### 2.2.2. Graphische Bedienoberfläche (GUI)

Zur Testgenerierung ist keine graphische Oberfläche verfügbar. Alle Funktionen sind verfügbar durch Starten einer ausführbaren Linux-Datei (ELF) mit Parametern.

Für den Einsatz in einer Fertigungslinie, d.h. zum Ausführen von Tests, ist ein sehr schlankes GUI verfügbar. Es ist bewußt auf die nötigsten Bedienelemente in dieser Umgebung zugeschnitten. Siehe Abschnitt 14 Produktions-Modus auf Seite 39 für mehr.

### *2.3. Hardware*

Der BSC (Boundary Scan Controller) befindet sich in einem robusten 19"-Gehäuse mit robusten Steckverbindern für schnellen und sicheren Prüflingswechsel (figure 8). Er kann autonom vom PC operieren. Insbesondere für **schnelle Vortests**, in denen der Bediener nur die START/STOP-Tasten an der Frontseite drückt und auf das Aufleuchten der PASS/FAIL-LEDs reagiert, ist dies von Vorteil.

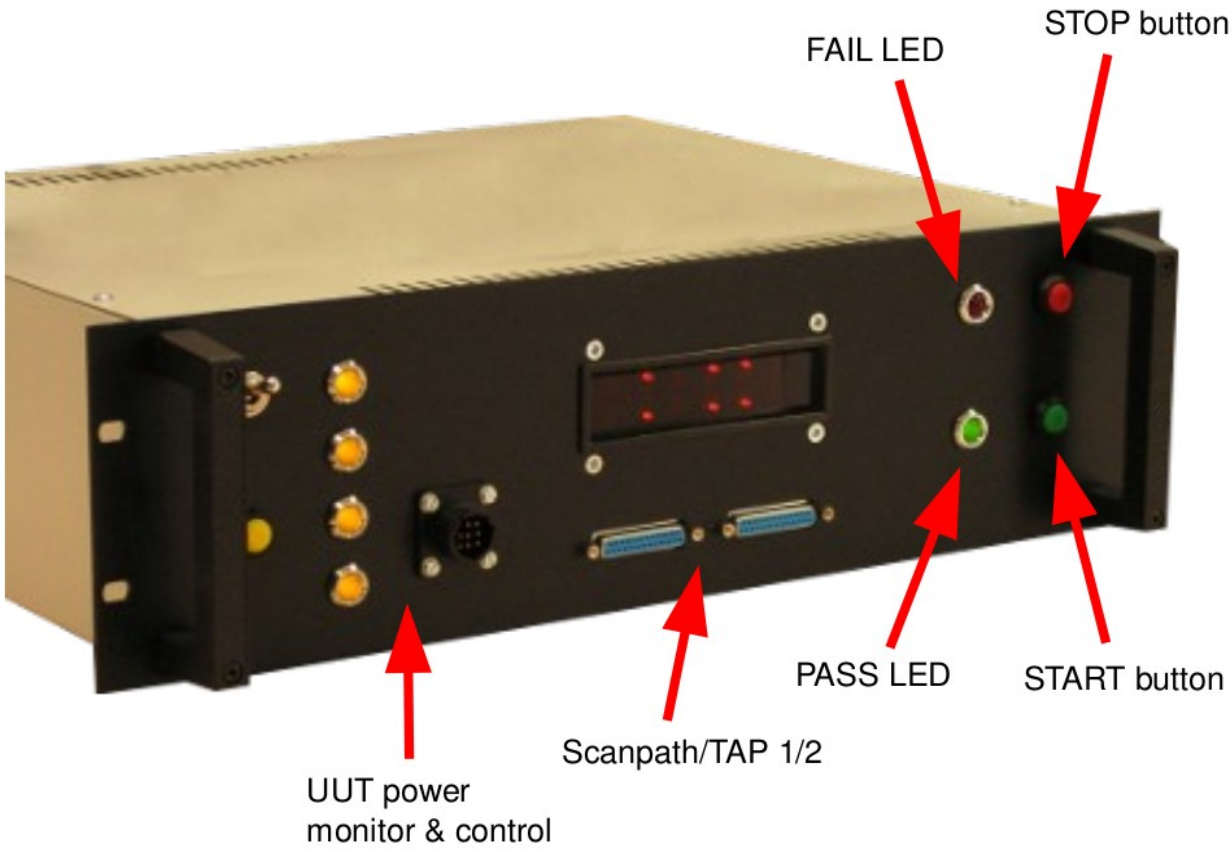



*figure 8: BSC Frontplatte*

## 3. Wo bekommt man die Software ?

Die Software wird mit dem System geliefert oder kann von GitHub gezogen werden:

https://github.com/Blunk-electronic/M-1

https://github.com/AdaCore/gtkada/archive/gtkada-3.4.1.tar.gz (nur für GUI nötig)

# 4. Installation

Im Allgemeinen wird die Software mit dem System fertig installiert ausgeliefert. Der Anwender braucht sich also mit der Installation nicht befassen. Zur Evaluation, also zum selber-Installieren müssen diese Komponenten auf dem Basisystem verfügbar sein oder installiert werden:

1. `gcc` und `gnat` (C und Ada-Compiler)
2. Fix g-sercom.adb (siehe www.blunk-electronic.de/pdf/compile_ada_rtl.pdf) wenn ältere Ada Runtime Bibliotheken wie Version 4.8 verwendet werden.
3. Ein Verzeichnis, in welches folgende Repositories geklont werden:
4. Repository M-1 von https://github.com/Blunk-electronic/M-1.git
5. Entpacken und Installieren von `gtkada` https://github.com/AdaCore/gtkada/archive/gtkada-3.4.1.tar.gz . Es gilt die darin vorgeschlagene Installtionsanleitung. Im Wesentlichen sind das diese Aktionen:

   - Installation des Paketes GTK3-devel

   - ./configure

   - make

   - make install
6. Ausführung des Shell-Skripts `install` im Verzeichnis `M-1` .

# 5. Support

Grundsätzlich ist Boundary Scan supportintensiv. Für Schulungen, Beratung und Testprogramm-Generierung kontaktieren Sie uns bitte unter www.blunk-electronic.de.

*Wir freuen uns auf die Zusammenarbeit mit Ihnen !*

# 6. Testprogramm-Entwicklung Ablaufplan

figure 9 stellt den allgemeinen Ablauf mit seinen grundlegenden Operationen zur Generierung von Testprogrammen oder der In-System-Programmierung (ISP) dar. Jedes Werkzeug, daß der Anwender direkt (oder indirekt) startet, erweitert oder modifiziert die UUT-Datenbak (im Folgenden mit UDB bezeichnet). Die UDB wiederum ist der Datenpool auf dem die Testgenerierung basiert. Jede hier dargestellte Aktion wird in den folgenden Abschnitten genauer beschrieben. Es ist ratsam diesen Ablaufplan beim Lesen dieses Handbuches griffbereit zu haben.

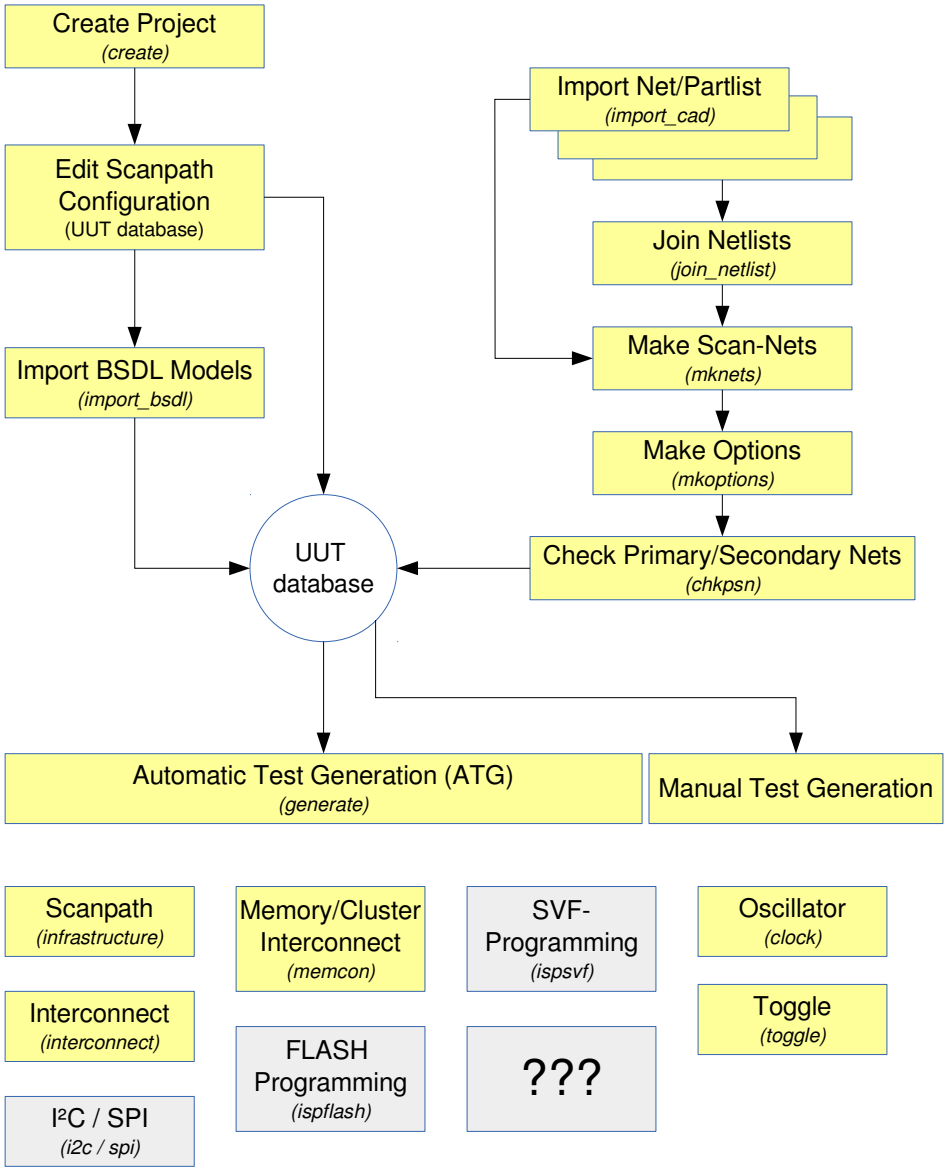



*figure 9: Ablaufplan Testgenerierung (graue Box heißt: in Entwicklung)*

### *6.1. Projekt anlegen (create)*

Der erste Schritt ist ein Projet anzulegen. Öffnen Sie ein Terminal und führen Sie das Kommando

```
$ bsmcl create my_project
```

aus. Es erzeugt ein Verzeichnis my_project welches die Projektstruktur beinhaltet. Wechseln Sie dann in das Verzeichnis my_project und sehen sich dort etwas um.

### *6.2. Einrichtung der UUT Datenbank (UDB)*

Sobald ein Projekt angelegt wurde, ist eine leere UDB verfügbar. Ihr Name setzt sich zusammen aus dem Projektnamen und der Endung .udb. In diesem Beispiel heißt die UDB des Projektes *my_project* myproject.udb.

Öffnen Sie die UDB mit einem Texteditor und gehen Sie zum Abschnitt SubSection chain_1. Hier beschreiben Sie den ersten Scanpfad Ihres Prüflings (oder UUT). Der Scanpfad (oder Scankette) ist das Rückgrat an dem alle boundary scan fähigen ICs – weiterhin bezeichnet als BICs – aufgereiht sind. Am Anfang enthält diese SubSection nur Kommentare (mit „--“ angeführt). Im Beispiel hier unkommentieren Sie einfach die fünfte Zeile um IC301 zum einzigen BIC des Scanpfades #1 zu machen. Die Kommentare in diesem Beispiel sollen beim Verstehen dieser Sektion helfen.

```
SubSection chain 1
        --UUT_TDI_1 / BSC_TDO_1
        --device, package, path to bsdl model, option [ remove_pin_prefix p ]
        --NOTE 1: use lower case letters for package names
        --NOTE 2: position 1 is closest to BSC TDO !
        --IC301 pc44 models/BSDL/xc9536_pc44.bsd
        --UUT_TDO_1 / BSC_TDI_1
EndSubSection
```

*Text 1: UUT Datenbank SubSection chain*

Diee SubSection spezifiziert die Reihenfolge der BICs in einem bestimmten Scanpfad, deren Gehäuse, den Speicherort des BSDL-Modelles und weitere Optionen. Der BSDL-Import kann mit Hilfe von Optionen beeinflußt werden. Siehe Abschnitt 6.3.1 Seite 17.

**Die Mehrheit der folgenden Aktionen basiert auf dieser Konfiguration.**

### *6.3. Import von BSDL-Dateien (import_bsdl)*

Nachdem ein Scanpfad spezifizert wurde, sollten die BSDL-Modelle in die Datenbank importiert werden. Das wird mit folgendem Kommando durchgeführt:

```
$ bsmcl import_bsdl my_project.udb
```

Ihre Datenbank wird durch die Sektion registers erweitert. Dieser Abschnitt enthält Informationen aus den BSDL-Modellen aller spezifizeriten BICs (siehe Text 1 Seite 16). Details zur BSDL-Syntax finden Sie im IEEE1149.1 Standard (siehe Referenz (3)).

Von diesem Stadium der Datenbank ausgehend, kann bereits der grundlegende Test, der sogenannte Scanpfad-Test (oder Infrastruktur-Test) generiert werden. In diesem Fall lesen Sie bitte in Abschnitt 6.8.1 Seite 25 weiter.

**WICHTIG**: BSDL-Modelle können das Verhalten von ICs im Pre- oder Post-Configuration-Mode darstellen. Besonders programmierbare Logik wie FPGAs oder CPLDs erforder spezielle Behandlung wenn Boundary Scan-Tests nach dem Konfigurieren durchgefürht werden. Ein Blick in das BSDL-Modell offenbart mitunter die Lösung.

#### 6.3.1. Option Pin-Prefix entfernen

Mit der Option „option remove_pin_prefix“ können Prefixe, die im BSDL-Modell den Pin-Namen vorangestellt sind, gelöscht werden. Text 2 zeigt, wie diese Option angewendet wird, um das Prefix „P“ beim Import des BSDL-Modelles eines Spartan-3 zu entfernen.

```
SubSection chain 1
       IC301 pq208 models/xc3s400_pq208.bsd option remove_pin_prefix P
EndSubSection
```

*Text 2: Pin-Prefix entfernen*

### *6.4. Import Netz- & Partliste (import_cad)*

Mit der Datenbank in ihrem gegenwärtigen Stand können Sie noch nicht viel anstellen. Wir werden die Datenbank nun um die sogenannte Netzliste des Prüflings (UUT) erweitern. Jedes CAE-System kann eine Netzliste, den Schaltplan des Prüflings als ASCII-Datei, exportieren. Optional kann auch eine Partliste erzeugt werden. Die Netzliste representiert alle elektrischen Verbindungen der UUT. Gewöhnlich sind dies Dateien ASCII basierend, was bedeutet, daß sie mit einem Texteditor geöffnet, gelesen oder modifiziert werden können (Ja, es gibt Situationen, in denen Sie eine Netzliste bearbeiten müssen...).

Das Projektverzeichnis unseres Dummy-UUT `my_project` enthält ein Unterverzeichnis, genannt `cad` wohin Sie die Netz- und Partliste kopieren sollten oder symbolische Links zu den eigentlichen Speicherorten anlegen. Ein Link hat den Vorteil, daß dieser immer auf den aktuellen Stand der Zieldatei führt.

Das Kommando zum Import erfordert die Spezifizierung des Formates und des Speicherortes der Netz- und Partliste. Wenn es keine Partliste gibt, lassen Sie diese einfach weg. Innerhalb des Projektverzeichnisses `my_project` wird der Import auf diese Weise gestartet:

```
$ bsmcl import_cad eagle6 cad/netlist.txt cad/partlist.txt main
```

Für andere Netzlistenformate geben Sie einfach das Kommando

```
$ bsmcl import_cad
```

ein. Es gibt die gegenwärtig unterstützten CAD-Formate aus. Da *System M-1* quelloffen, also OpenSouce ist, sind Sie hiermit eingeladen, eigene Netzlistenimporter zu schreiben. Das können Sie mit eine beliebigen Programmiersprache machen. Wichtig ist, daß dabei das Skeleton-File entsteht:

Das Ergebnis des Netzlistenimportes ist das sogenannte Skeleton (deutsch: Skelet), die Datei `skeleton.txt`. Durch die Dateiendung wird deutlich, daß es sich hier wieder um eine ASCII-Datei handelt. Sehen Sie sich diese doch einmal mit einem Texteditor an: Sie enthält den gesamten Prüfling in Klartext in einem standardisierten Format.

### 6.4.1. Hierarchisches Design

Wenn Sie mit einem UUT arbeiten, der nur aus einem Modul (also einer Leiterplatte) besteht, importieren Sie grundsätzlich immer als Top-Modul. In diesem Fall können Sie mit Abschnitt 6.5 Boundary Scan fähiges Netze erzeugen (mknets) Seite 20 weiterlesen.

#### *6.4.1.1. Top-Module & Sub-Module*

Besteht der Prüfling aus mehreren Leiterplatten oder ist es ein Gerätesystem, muß eine Leiterplatte as Top-Modul (oder Haupt-Modul) bestimmt werden. Alle anderen Platinen werden Sub-Module.

Wird nach dem Netz- oder Partliste das Argument „main“ übergeben, wird das Skeleton als Top-Modul angelegt. Das heißt, es entsteht die Datei `skeleton.txt` .

Wird das Argument „sub“ gefolgt vom Namen des Sub-Modules, wird ein Skeleton namens `skeleton_my_submdule.txt` erstellt:

```
$ bsmcl import_cad protel cad/X15.net sub X15
```

Jeder Import eines Sub-Modules erzeugt ein eigenes Skeleton-File, welches den Namen des Sub-Modules trägt, z.B. `skeleton_X15.txt` . Alle Bauteile und Netze darin tragen das Prefix `X15` im Namen. So heißt dann der Widerstand R5 im Sub-Modul `X15` dann folglich `X15_R5`. Zum Hintergrund: Prefixe an Bauteil- und Netznamen ist nötig, damit später die Testgeneratoren unterscheiden können, welches Bauteil oder Netz sich auf welchem Modul befindet.

Sollte Sie dieser Abschnitt verwirren, spielen Sie einfach etwas mit dem Importer (siehe auch Abschnitt 6.4 Import Netz- & Partliste (import_cad) Seite 18), importieren Sie Sub-Module und sehen Sie sich das Ergebnis an.

Nächste Station: Verbinden von Skeletten !

### 6.4.2. Join Skeletons (join)

Um Top- und Sub-Module miteinander zu verbinden, starten Sie z.B. dieses Kommando:

```
$ bsmcl join_netlist skeleton_X15.txt
```

Es verdindet das Sub-Modul `X15` mit dem Top-Modul Skeleton. Das Ergebnis ist ein Skeleton welches die Netze und Teile beider Module enthält. Dieser Schritt kann mit weiteren Sub-Modulen wiederholt werden bis alle Module vereint sind in der Datei `skeleton.txt`.

### *6.5. Boundary Scan fähiges Netze erzeugen (mknets)*

Die Bezeichnung *skeleton* spricht für sich. Das Skeleton-File muss mit Leben gefüllt werden indem es in die Datenbank überführt wird. Letztere liefert Informationen darüber welcher Pin (oder welches Pad) BIC (Boundary Scan fähiger IC) mit welchem Nezt verbunden ist. Das Kommando

```
$ bsmcl mknets my_project.udb
```

liest das Skeleton und importiert es in die spezifizierte Datenbank (Zur Erinnerung: Es können mehrere Datenbanken gehalten werden, die jeweils eine bestimmte Konfiguration des Prüflings/UUT repräsentieren.). Die Datenbank wird nun um einen weiteren Abschnitt, die Sektion *netlist* erweitert. Sehen Sie sich doch die Datenbank einmal mit einem Texteditor an, rollen Sie zu Abschnitt *netlist* und inspizieren Sie das Ergebnis:
Jedes Netz ist nun eine Subsektion. Der Subsektion-Kopf sagt Ihnen etwas über die Netzklasse. Standardmäßig sind alle Netze in Klasse NA, was für “not assigned” steht. Ein Netz der Klasse NA ist vollständig von jeder Testgenerierung ausgeschlossen. Bitte betrachten Sie dies als Sicherheitsmaßnahme.
Dem Subsektion-Kopf folgen alle Bauteile, welche mit diesem Netz verbunden sind, gelistet anhand ihres Namens, Bauteilklasse (im Moment noch ein Platzhalter '?'), Gehäuse, Wert und Pin/Pad. Ein Pin/Pad eines BIC liefert mehr Informationen, die sogenannte *cell info*. Die *cell info* ist von den Daten aus dem Datenbank-Abschnitt *registers* abgeleitet, welcher wiederum aus den Spezifikationen des BSDL-Modelles generiert wurde (siehe 6.3 Import von BSDL-Dateien (import_bsdl) Seite 17).

```
SubSection SDA class PU
-- name class value package pin [port | in_cell: id type func safe | out_cell: id type func safe
  RN300 NA 8x1k8 SIL9 3
  R400 NA 160 0207/10 1
  JP403 NA MON2 2X20 6
  IC301 NA XC9536 PLCC-S44 33 pb01_11 | 20 BC_1 INPUT X | 19 BC_1 OUTPUT3 X 18 0 Z
 EndSubSection
```

In diesem Stadium der Sektion *netlist* existiert jedes Netz für sich selbst, ohne jede Interaktion mit anderen Netzen. Der nächste Schritt unterstützt den Anwender beim Hinzufügen der Zwischenverbindungen (Interconnections).

## *6.6. Optionen generieren (mkoptions)*

Häufig sind Netze durch Serienwiderstände, Lötbrücken, Drähte, Induktivitäten, Steckverbinder u.s.w. miteinander verbunden. Der Bediener muß diese Bauteile dem System deklarieren.

### 6.6.1. Editieren der Datei mkoptions.conf

Die Textdatei mkoptions.conf ist der Ort, an dem Bauteile deklariert werden, die Nezte miteinander verbinden. Um nicht die gesamte Datei neu zu schreiben, befindet sich eine gut vorbereitete mkoptions.conf bereits in Ihrem Projektverzeichnis. Sie müssen also nur Änderungen entsprechend des Prüflings vornehmen. Öffnen Sie diese also nun mit einem Texteditor und sehen sie sich an:

```
Section connectors
 -- device A  device B
 -- module_a_X1 module_b_X3 [mapping pin_range first to last]
 -- Mapping can be one_to_one (default) or cross_pairwise.
 -- Examples:
 -- module_a_X1 module_b_X3 one_to_one
 -- module_a_X1 module_b_X3 cross_pairwise pin_range 1 to 40
EndSection


Section bridges
 -- N*
 -- RN1  array 1-8 2-7 3-6 4-5
 -- RN4* array 1-8 2-7 3-6 4-5
 -- R3*
 -- R99
 -- module_a_R7
EndSection
```

*Text 3: mkoptions.conf*

Kommentare beginnen mit „--“. Es gibt einen Abschnitt “section connectors” in welchem Sie angeben, welches Paar von Steckverbindern zwei Module miteinander verbindet. Beachten Sie das Prefix ! In diesem Beispiel hat das Sub-Modul module_a einen Steckverbinder X1 wessen Gegenstück X3 sich auf Sub-Modul module_b befindet. Eine weitere Verbindung besteht zwischen module_c, J55 und J1 auf dem Top-Modul. Zur Erinnerung: Das Top-Modul hat keine Prefixe. Die Annahme hier ist, daß die Steckverbinder eins-zu-eins ineinander gesteckt sind, was normalerweise bei Board-to-Board-Verbindern der Fall ist.

Der Abschnitt „Section bridges” enthält Bauteile, die wie eine Brücke zwischen Netzen fugieren. Es spielt dabei keine Rolle, ob diese Teile Widerstände, Induktivitäten, Drähte

oder ähnliches sind. Wichtig ist, ob ein bestimmtes Teil zu eine Verbindung zwischen Netzen führt. In unserem Beispiel befinden sich viele Lötbrücken deren Name mit dem Buchstaben „N“ beginnt. Die Notation N* addressiert alle Bauteile dieser Art. Dedizierte Deklarationen wie der Widerstand R99 sind ebenso möglich. Ein Spezialfall sind Arrays. Meißt haben wir es mit Widerstandsarrays (auch Widerstandsnetzwerke) zu tun. Sie spezifizieren hier den Namen, gefolgt vom Schlüsselwort array und den Pins, an denen die internen Widerstände angeschlossen sind.

### 6.6.2. Generierung der Optionen-Datei

Nach dem Editieren der Datei mkoptions.conf führen Sie das Kommando

```
$ bsmcl mkoptions my_project.udb
```

aus. Das Ergebnis ist die sogenannte Optionen-Datei, welche auf *.opt endet und die vollständige Netzliste Ihres UUT enthält - einschließlich aller Zwischenverbindungen die in mkoptions.conf spezifiziert wurden. Standardmäßig hat die Optionen-Datei den selben Namen wie die UUT-Datenbank. Optional kann auch ein anderer Name angegeben werden:

```
$ bsmcl mkoptions my_project.udb another_options_file.opt
```

### 6.6.3. Routing Tabelle

Als ein Nebenprodukt werden die miteinander verbundenen Netze automatisch in eine CSV-Datei geschrieben, welche den selben Namen wie die Datenbank hat. So können Sie nun die Datei my_project.csv mit einem Tabellenkalkulationsprogramm öffnen.

### 6.6.4. Struktur der Optionen-Datei

Die Optionen-Datei ist wie folgt strukturiert:

- Statistiken über Brücken, Netzwerke und Anzahl von Netzen

- Netzliste mit primären und sekundären Netzen

*Mkoptions* entscheidet welches Netz ein primäres und welches ein untergeordnetes sekundäres Netz wird. Enthält ein Netz einen boundarry-scan-fähigen Treiber-Pin, wird es zum Primärnetz. Andere Netze, egal ob diese scanfähige Pins haben oder nicht, werden Sekundärnetze. Der Gedanke hinter diesem Mechanismus ist, im Vorausblick auf die Testgenerierung, festzulegen, welches Netz treibt (oder sendet) und welche Netze dem übergeordneten Primärnetz „zuhören“.

#### *6.6.4.1.Editieren der Optionen-Datei*

Die Optionen-Datei spiegelt den gesamten Prüfling/UUT (einschließlich seiner Netz-Zwischenverbindungen und Abhängigkeiten) wieder. Nun sind Sie gefordert, die Netzklassen zu editieren. Standardmäßig befinden sich alle Netze in Klasse NA, was bedeutet “not assigned”. Es handelt sich hierbei um eine Sicherheitsmaßnahme, die verhindert, daß Treiber gegen andere unbekannte Signalquellen treiben oder

angeschlossene Peripherie schalten. Netzklassen werde NICHT automatisch zugewiesen. Sie sind nun gefordert, anhand des Schaltplanes, Netzklassen zuzuweisen. Mit anderen Worten, studieren Sie den Schaltplan des Prüflings und entscheiden Sie sorgsam, welches Netz wie getestet werden soll.

Verfügbare Netzklassen sind:

1. NA (not assigned) → Netz vollständig von der Testgenerierung ausgeschlossen (default)
2. NR (no restriction) → Netz kann ohne Einschränkung getestet werden
3. PU (pull-up) → Netz hat Pull-Up-Verhalten
4. PD (pull-down) → Netz hat Pull-Down-Verhalten
5. DH (drive-high) → Netz soll permanent H-getrieben werden (Sofern scanfähiger Treiber vorhanden)
6. DL (drive-low) → Netz soll permanent L-getrieben werden (Sofern scanfähiger Treiber vorhanden)
7. EH (expect high) → Auf dem Netz soll permanent H erwartet werden (unabhängig von der Signalquelle)
8. EL (expect low) → Auf dem Netz soll permanent L erwartet werden (unabhängig von der Signalquelle)

### *6.7. Check Primärer/Secundärer Netze und Klassen (chkpsn)*

*Chkpsn* führt einen Machbarkeitscheck der Angaben in der Optionen-Datei durch. Wann immer Sie denken, einen Check der Optionen-Datei zu machen, geben Sie folgendes Kommando ein:

```
$ bsmcl ckpsn my_project.udb
```

Es liest die Optionen-Datei (gleicher Name wie die Datenbank aber mit Endung *.opt). Sind die Angaben darin korrekt und umsetzbar, werden sie in die Datenbank des UUT importiert. Dieses Kommando kann jederzeit wiederholt werden um Schritt für Schritt mehr Netze für die Testgenerierung freizugeben.

Es ist im allgemeinen sinnvoll, ein Netz (oder Gruppen) nach dem anderen freizugeben, einen Test zu generieren und auszuführen, um dann weitere Netze zu für die Testgenerierung freizugeben.

Anmerkung: Zur Zeit wird die Routing-Tabelle (generiert durch *mkoptions*) nicht mit den manuell zugefügten primär-sekundär-Abhängigkeiten aktualisert (siehe Abschnitt 6.6.4.1).

## *6.8. Testgenerierung*

Als allgemeine Regelung: Die Testgenerierung (oder Testmuster-Generierung) basiert grundsätzlich auf eine gültigen, konsistenten UUT Datenbank. Wie oben erwähnt, können Sie mehrere Datenbanken innerhalb Ihres Projektverzeichnisses halten. Jede davon spiegelt eine bestimmte UUT-Konfiguration wieder. Wird ein Testgenerator gestartet, muß der Name der gewünschten Datenbank als Argument übergeben werden, zusammen mit dem Test-Profil und Optionen.

Das Ergebnis einer Testgenerierung ist ein Verzeichnis, den Namen des Tests gleichnamig, welches den Quellcode des Tests enthält. Der Quellcode wird im Folgenden als *sequence file* bezeichnet, ist ASCII basierend, kann als Programmmiersprache verstanden werden und kann zum Feintunig editiert werden (siehe Abschnitt 6.8.6 Manuelle Testgenerierung Seite 29). In den meisten Fällen kann er direkt kompiliert werden mit dem Compiler *compseq* (siehe Abschnitt 7 Kompilieren von Test (compile) Seite 32).

### **6.8.1. Scanpfad-Test** (infrastructure)

Der erste Test, der gewöhnlich generiert wird, ist der sogenannte Scanpfad-Test (oder Infrastruktur-Test). Sein einziger Sinn ist einen stabilen Scanpfad festzustellen. Der Scanpfad ist das Rückgrat, durch welches die Testdaten zwischen Prüfling und Boundary Scan Controller transportiert werden. Dieser Test ist von seiner Natur her **non-intrusive**, d.h. er kann ausgeführt werden ohne den Normal-Betrieb des UUT zu stören.

Der Scanpfad-Test erfordert keine Netzliste. Er kann direkt nach dem Import der BSDL-Modelle generiert werden (siehe Abschnitt 6.3 Import von BSDL-Dateien (import_bsdl) Seite 17).

Geben Sie dieses Kommando ein, um einen Scanpfad-Test zu generieren:

```
$ bsmcl generate my_project.udb infrastructure my_infra_test
```

### 6.8.2. **Interconnect-Test** (intercon)

Ohne Interconnect-Tests ist Boundary Scan wertlos. Mit anderen Worten: Der Interconnect-Test ist das Arbeitspferd von Boundary Scan. Dieser Test ist **intrusiv**, d.h. er stimuliert Treiber-Pins direkt und mißt an Input-Pins und verifiziert somit die Verbindungen zu und zwischen BICs (Boundary Scan fähige ICs). Die Vorbedingung für diesen Test ist ein intakter Scanpfad und wenigstens ein BIC. Gewöhnlich hat ein BIC scanfähige Pins. Es gibt Ausnahmen.

Mit nur einem IEEE1149.1-fähigen IC auf Ihrem Prüfling, mit BIC bezeichnet, können eine Reihe von Test gemacht werden, selbst wenn es keinen weiteren BIC zum Datenaustausch gibti. Mit einem einsamen BIC können periphere Komponenten (Relais, Motoren, Ventile, Leuchten, Displays, Schalter, Jumper,…), non-scan ICs oder Schaltungsteile angesteuert oder gelesen werden, können Kurzschlüsse oder fehlende Pull-Widerstände festgestellt werden.

Die Testabdeckung liegt im Bereich gering bis moderat.

Die Situation bessert sich wenn weitere BICs sich auf dem Prüfling befinden, weil dann deren Zwischenverbindungen die Testabdeckung erhöhen.

Mit diesem Kommando wird der Interconnect-Test-Generator gestartet:

```
$ bsmcl generate my_project.udb interconnect my_interconnect_test
```

**ACHTUNG: Der Interconnect-Test ist intrusive !**

**ACHTUNG:** Siehe Abschnitt 6.3 Seite 17 für Besonderheiten betreffend dem Verhalten von programmierbarer Logik.

### 6.8.3. **Memory Interconnect-Test** (memcon)

Wie in Abschnitt 6.8.2 Interconnect-Test (intercon) gesagt, die Verbindungen zwischen BICs sind wichtig, um die Testabdeckung zu erhöhen. Wenn Speicherbausteine (wie RAMs) mit BICs verbunden sind, kann die Testabdeckung weiter gesteigert werden. Via Boundary Scan können bestimmte Speicheradressen mit Testdaten geladen werden. Durch Zurücklesen dieser können fehlerhafte Adress, Daten oder Steuerleitungen detektiert werden.

Wichtig für die Generierung eines Speicher-Verbindungstests ist ein Modell des Speichers. *System M-1* verwendet ASCII-basierende Textdateien worin das Verhalten des Speichers beschrieben wird. Zur automatischen Testgenerierung wird die Datenbank, das Test-Profil, der Name des Speicherbausteins, das Speichermodell und das physische Gehäuse als Parameter übergeben:

```
$ bsmcl generate mmu.udb memconnect my_ram_test IC2 models/U256.txt SO28
```

**ACHTUNG: Der Speicher-Verbindungs-Test ist intrusive !**

### 6.8.4. **Clock-Test** (clock)

Oft muß ein Taktsignal getestet werden. Dabei reicht es in den meißten Fällen aus, festzustellen ob Pegelwechsel stattfinden. Die eigentliche Frequenz oder das Tastverhältnis sind zweitranging. Um also einen Clock-Test zu generieren müssen die Parameter Datenbank, Test-Profil, Ziel-Netz, Ziel-BIC, Anzahl der Tastungen und Intervalle spezifiziert werden:

```
$ bsmcl generate mmu.udb clock osc IC1 7 5 0.1
```

Das hier gezeigte Beispiel generiert einen Test, der im Netz “osc”, an IC1 pin 7, 5 Proben mit einer Pause von 0.1 Sekunden nimmt. Stellt der Test keinen Pegelwechsel fest, wird der Test “osc” mit dem Ergebnis FAIL beendet. Der Clock-Test ist **non-intrusive**, d.h. er greift in den Normal-Modus des UUT nicht ein. Dieser Test kann ausgeführt werden, während der Prüfling sich im operativen Betrieb befindet !

Anmerkung: Wird ein Clock-Test mit einer Datenbank generiert, die DH- oder DL-Netze enthält, schlägt die Test-Ausführung fehl. Statisch getriebene Netze der Klasse DH ode DL erfordern den **intrusiven** Modus.

### 6.8.5. **Toggle-Test** (toggle)

Zur Erst-Inbetriebnahme eines Prüflings muß häufig ein einzelnes Netz H oder L geschaltet werden um eine LED, ein Relais oder Messungen vorzunehmen. In einer Bestückunslinie hingegen müssen oft LEDs getestet werden. Im Gegensatz zum Clock-Test (siehe oben), ist der Toggle-Test intrusive, d.h. er greift in den Betrieb des UUT ein und stimuliert das Zielnetz. Zum Start des Testgenerators, wird die Datenbank, das Test-Profil, der Test-Name, das Ziel-Netz, die Anzahl der Zyklen sowie die L- und H-Dauer spezifiziert:

```
$ bsmcl generate mmu.udb toggle my_osc_test OSC 10 0.1 0.2
```

Das Beispiel hier läßt das Netz “OSC” zehn mal mit einer L-Zeit von 0.1 und einer H-Zeit von 0.2 Sekunden schalten. Der Treiber-Pin wird dabei vom Testgenerator ausgewählt.

**ACHTUNG: Der Toggle-Test ist intrusiv !**

### 6.8.6. Manuelle Testgenerierung

Im Allgemeinen, nach der automatischen Testgenerierung oder wenn ein Test von Hand geschrieben werden muß, ist das Ergebnis eine in Klartext geschriebene ASCII-Datei mit der Endung *.seq. Der Inhalt dieser Sequenz-Datei kann als Hochsprache verstanden werden.

In den meisten Fällen kann die vom Testgenerator erzeugte Sequenz-Datei als Ausgangspunkt zum Schreiben eines eigenen Tests verwendet werden. Dies erfordert fortgeschrittene Kenntnisse der Technologie Boundary Scan.

tbd

#### *6.8.6.1.Befehlssatz*

| Befehl | Bespiel | Bedeutung |
|---|---|---|
| connect | connect port 1 | verbindet den Scanport 1 mit demUUT |
| disconnect | disconnect port 1 | trennt den Scanport 1 vom UUT |
| | | |
| power up | power up 2 | schließt Stromversorgung Kanal 2 |
| power down | power down 2 | trennt Stromversorgung Kanal 2 |
| | power up gnd | verbindet Stromversorungung Kanal GND (keine Stromüberwachung !) |
| | power down all | trennt alle Kanäle der Stromversorung |
| imax | imax 1 0.2 timeout 2 | stellt max. Strom von Kanal 1 auf 0.2A mit Zeitlimit von 2s |
| delay | delay 0.5 | Pause von 0.5s |
| set | set IC1 drv boundary 4=0 | weist Zelle 4 im Boundary- Register von IC1 den Treiberwert 0 zu |
| set | set IC1 exp boundary 5=1 | weist Zelle 5 im Boundary- Register von IC1 den Erwartungswert 1 zu |
| sir | sir id 4 | Befehls-Scan mit id 4 |
| sdr | sdr id 5 | Daten-Scan mit id 5 |
| | sdr id 5 option retry 5 delay 1 | Daten-Scan mit id 5, max. 5 Wiederholungen und einer Pause von 1s |
| trst | trst | Scanpfad-Reset (gleichzeitig harter und weicher Reset) |
| htrst | htrst | harter Scanpfad-Rest |
| strst | strst | weicher Scanpfad-Reset |
| | | |

*Tabelle 1: Kurzreferenz Sequenze Befehlssatz*

#### *6.8.6.2.Stromüberwachung und Genauigkeit*

Die Überwachung der Stromaufnahme des Prüflings kann mit dem Befehl imax konfiguriert werden. Der Befehl benötigt in fester Reihenfolge die Parameter Kanal, Maximal-Strom, Zeitlimit. Beispiel für Kanal 1 mit Stromlimit 0,2A und Zeitlimit 2s:

imax 1 0.2 timeout 2

Der relative Fehler mit der der Maximalstrom vom BSC erfaßt wird beträgt ca. 25%.

## 7. Kompilieren von Test (compile)

Jeder Test mit seiner eigenen Sequenz-Datei muß in eine ausführbare Binärdatei übersetzt werden. Die Binärdatei hat die Endung *.vec und kann NUR vom Boundary Scan Controller ausgeführt werden. Es handelt sich hierbei um die einzige nicht-ASCII-Datei in Ihrem Projekt. Sie befindet sich nach dem Kompilieren im Verzeichnis des jeweiligen Testes. Zum Starten des Compilers *compseq* müssen Sie die Datenbank angeben und den Namen des Test:

```
$ bsmcl compile mmu.udb my_osc_test
```

## 8. Laden von Tests in den BSC (load)

Nach dem Kompilieren muß die Binärdatei in den Boundary Scan Controller geladen werden. Das einfache Kommando

```
$ bsmcl load my_osc_test
```

bewerkstelligt dies. Sie können Binärdateien in den BSC laden bis dessen Speicher voll ist.

Anmerkung: Nach jedem Neu-Kompilieren eines Testes muß dessen Binärdatei wieder in den BSC geladen werden.

Mit dem Start des Uploads wird ein gerade laufender Test automatisch abgebrochen.

## 9. Tests-Ausführung (run)

Sobald ein Test in den BSC geladen wurde, kann er mit dem schlichten Kommando

```
$ bsmcl run my_osc_test
```

gestartet werden.

Detektierte Fehler werden als Report im Terminal angezeigt (siehe Text 4 Seite 34).

```
BOUNDARY SCAN TEST SYSTEM M-1 Command Line Interface Version 025
================================================================
action          : RUN
Test/step 'intercon' started ...
waiting for test/step end ...

Test FAILED! Diagnosis:
failed scanpath   : 1
step id (dec)     : 7
sxr length (dec)  : 242 (one-based)
sxr fail pos (dec): 9 (one-based)
scan type         : SDR
device position   : 3
device name       : IC303
register          : BOUNDARY
failed bit pos.   : 8 (zero-based)
expected          : HIGH
net class         : PU

secondary net     : CT_D0

JP402 pin 2
IC303 pin 15
IC302 pin 3

primary net       : D0

RN401 pin 2
JP406 pin 2
JP404 pin 2
IC302 pin 15
IC301 pin 1
IC203 pin 11
IC202 pin 11
IC201 pin 13
IC200 pin 13


stuck at LOW or Pull-Up resistor missing !

Test/Step intercon FAILED
```

*Text 4: Testprotokol*

## *9.1. Einzelschritt-Modus*

Wird bei Test-Start die Schrittweite mit angegeben, hält der Test nach jedem Schritt an.

### 9.1.1. Schrittweite SXR

SXR steht für Scan-Data/Instruction-Register. Das Bespiel

```
$ bsmcl run my_osc_test sxr
```

bewirkt, daß der Test unmittelbar nach jedem Daten- und Befehls-Scan angehalten wird. Nach jedem Schritt wird der Status des Tests im Terminal angezeigt.

Im Halt-Zustand warten die TAP-State-Maschinen des UUT im SXR-End-State (mit TCK auf L) wie in „section info“ der Sequenz-Datei festgelegt.

### 9.1.2. Schrittweite TCK

TCK steht für Test Clock. Wird ein Test mit dieser Schrittweite ausgeführt, erfolgt der Test-Halt nach JEDEM Pegelwechsel des TCK Signales. Nach jedem Schritt wird der Status des Tests im Terminal angezeigt.

Während des Daten- oder Befehlstransportes (DR/IR-Scan), ist diese Betrachtungsweise hilfreich:

BP (bearbeitete Bits, bei eins beginnend) erhöht sich um 1 sobald ein TCK-Zyklus beendet ist. Ein TCK-Zyklus beginnt mit einer HL-Flanke (wobei TDOs und TMSs ihre Zustände ändern können) und endet mit einer LH-Flanke (alle TDIs und TMSs werden getastet). Also inkrementiert BP bei jeder LH-Flanke des Signals TCK. Ein Bit wird as „bearbeitet“ betrachtet, wenn der TCK-Zyklus beendet ist.

```
$ bsmcl run my_osc_test tck
```

### 9.1.3. Start-Taste am Frontpanel

Der im Einzelschritt gestartete Test kann in dieser Betriebsweise fortgesetzt werden, indem die grüne Start-Taste an der Frontplatte des BSC gedrückt wird. Jede Betätigung bewirkt einen weitern Schritt in der eingestellten Schrittweite.

### 9.1.4. Aufheben des Einzelschritt-Modus

Der Einzelschritt-Modus wird abgeschaltet, indem das Kommando zum gewöhnlichen Start eines Test eingegeben wird:

```
$ bsmcl run my_osc_test
```

Der Test wird dann ohne weitere Halts bis zu seinem Ende ausgeführt.

Der Einzelschritt-Modus wird bei Testende automatisch abgeschaltet, sodaß weiter Tests oder ein Neustart im Normal-Modus ausgeführt werden.

## *9.2. Breakpoints*

Breakpoint (Haltepunkte) erlauben dem Bediener, ein Programm an einer definierten Position anzuhalten um den Zustand des Prüflings zu untersuchen. Auf diese Weise können Fehler im Testprogramm eingegrenzt und behoben werden.

### 9.2.1. Setzen des Breakpoints

Ein Haltepunkt wird gesetzt, indem die SXR ID und optional die Bit-Nummer (bei eins beginnend) spezifiziert wird. Wird die Bit-Nummer weggelassen, hält der Test unmittelbar nach dem angegebenen SXR an. Dabei wird der in der Sequenz-Datei angegebene End-Zustand mit TCK auf L eingenommen. Dieses Beispiel setzt eine Haltepunkt nach SXR Nummer 6:

```
$ bsmcl break 6
```

Dabei wird nicht unterschieden ob es sich um einen Daten- oder Befehls-Scan handelt.

Soll nach nach einer bestimmten Bit-Position im spezifizierten SXR angehalten werden, ist dieses Beispiel dienlich:

```
$ bsmcl break 6 217
```

Der Haltepunkt ist nicht auf einen bestimmten Test festgelegt. Jeder Test wird von nun an an dieser Stelle angehalten.

Es kann nur ein Haltepunkt gesetzt werden.

### 9.2.2. Löschen des Haltepunktes

Zum Löschen des Haltepunktes geben Sie einfach das Kommando

```
$ bsmcl break 0
```

ein. Daraufhin wird kein Test mehr durch einen Haltepunkt angehalten.

### 9.3. Autonomer BSC-Betrieb

Für schnelle und einfache Vortests, nach dem Schema GEHT/GEHT-NICHT muß nur der grüne START-Knopf an der BSC-Vorderseite gedrückt werden. Auf diese Weise wird der zuletzt via Terminal ausgeführte Test gestartet. Die grüne oder rote LED zeigt das Testergebniss, also PASS oder FAIL an.

## 10. Abfrage eines Datenbank-Objektes

Mitunter müssen während der Generierung von Tests die Eigenschaften von Objekten wie z.B. ein Netz oder ein BIC (Boundary Scan fähiger IC) erfragt werden. Insbesondere sogenannte „shared control cell conflicts" bedürfen detaillierter Informationen über die betroffene Control-Zelle.

Das Modul *udbinfo* assistiert Ihnen an dieser Stelle und bewahrt Sie davor sich in Schaltplänen, Bestückungszeichnungen, Netzlisten und BSDL-Dateien zu verlieren.

### 10.1. Netze

Benötigen Sie z.B. detaillierte Informationen über das Netz „ADR15", geben Sie das Kommando

```
$ bsmcl udbinfo my_database.udb net ADR15
```

ein.

### 10.2. Boundary Scan ICs (BICs)

Die Eigenschaften eines BIC wie z.B. des IC303 erfragen Sie auf diese Weise:

```
$ bsmcl udbinfo my_database.udb bic IC303
```

### 10.3. Shared Control Cells

Die wohl wichtigste Funktion dieses Moduls ist die Anfrage, ob eine bestimmte Control-Zelle von mehr als einem Pin oder Netz verwendet wird. Das folgende Beispiel erkundigt nach Zelle 4 von IC303:

```
$ bsmcl udbinfo my_database.udb scc IC303#4
```

# 11. Anzeige der System-Konfiguration

Um einen kurzen Bericht zur System-Konfiguration zu bekommen, geben Sie diese Kommando ein:

```
$ bsmcl configuration
```

Es gibt die Namen von Basisverzeichnissen und den Namen der Schnittstelle zum BSC aus (z.B. /dev/ttyUSB0).

# 12. Anzeige der Firmware-Version

Die im BSC implementierte Firmware-Version wird mit diesem Kommando erfragt:

```
$ bsmcl firmware
```

# 13. Stapelverarbeitng/Scripting

Alle bisher beschriebenen Kommandos können auf sehr einfache Weise miteinander kombiniert werden (auch mit anderen Anwendungen) und wiederholt werden, indem die traditionelle, äußerste mächtige Linux/UNIX-Shell verwendet wird. Automatische Build-Prozesse können somit realisiert werden.

Anmerkung: Skripte innerhalb *System M-1* müssen mit der Endung *.sh enden.[1]

Der Dummy-Prüfling, welcher im M-1 Repository angeboten wird (Unterverzeichnis `uut`), enthält diverse Bash-Skripte mit denen Tests sehr schnell generiert werden können:

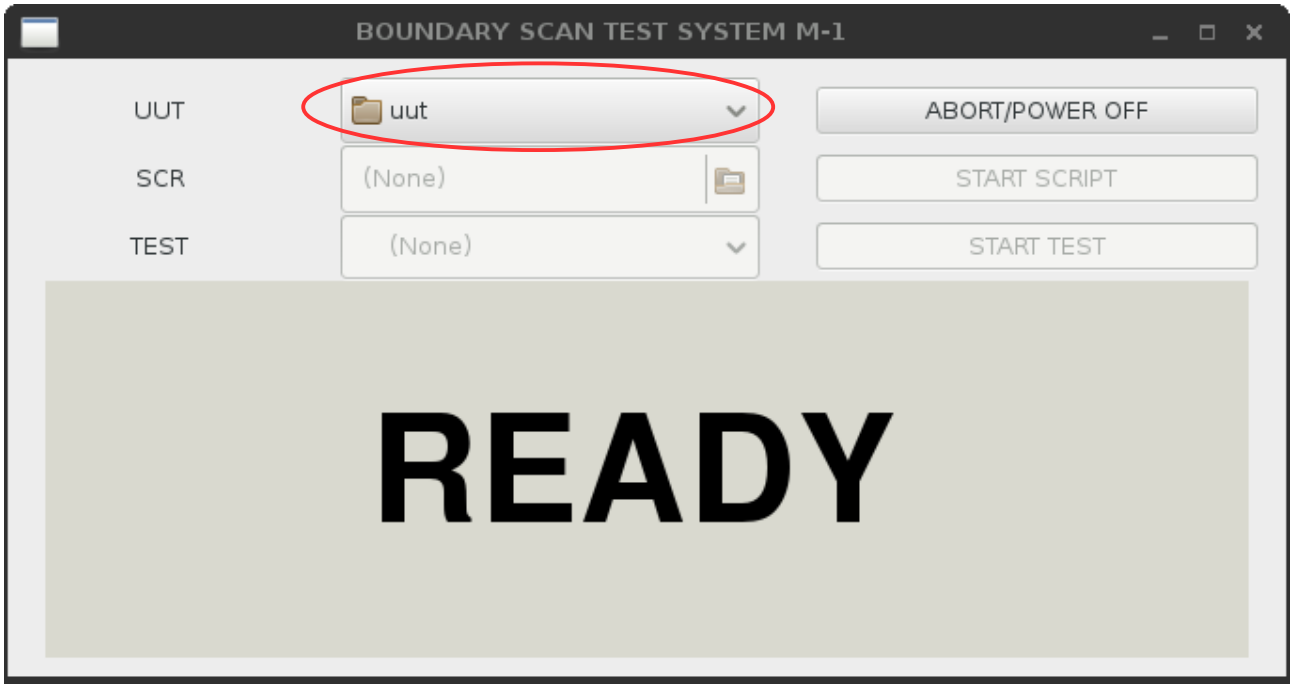
1. build_nets.sh
2. build_tests.sh
3. load_tests.sh
4. run_tests.sh

1 Linux- und UNIX Systeme bestehen im Allgemeinen nicht auf diese Endung. Sie erleichtert aber das Auffinden von Skript-Dateien.

# 14. Produktions-Modus

Es ist ratsam, die graphische Bedienoberfläche zu verwenden, wenn das *System M-1* in einer Bestückungslinie verwendet wird. Das GUI ist bewußt auf die nötigsten Bedienelemente zum Baugruppen- und Systemtest reduziert.

Nach dem Starten des GUI wird das Projekt (also der Prüfling oder die UUT) ausgewählt, indem der Projekt-Selektor (Screenshot 1) geklickt wird.




*Screenshot 1: Start-Up*

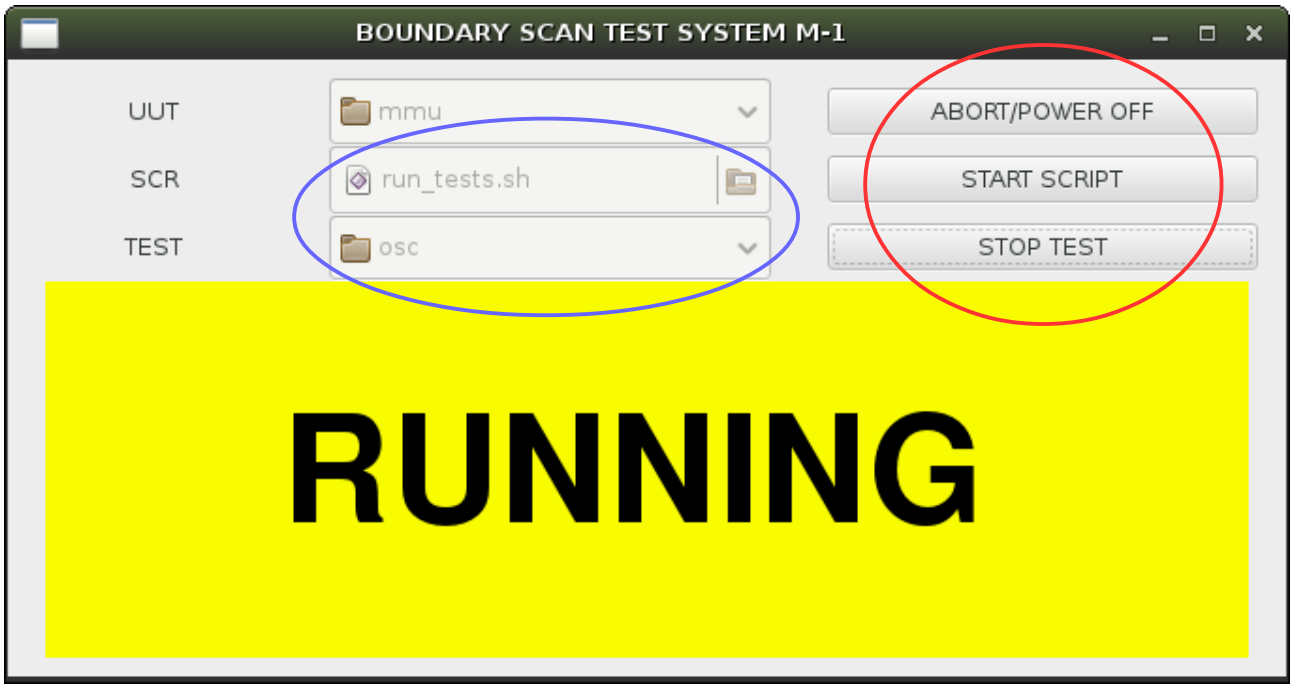
### *14.1. Test / Skript Start*

Nachdem der Prüfling ausgewählt wurde, kann ein Skript (was ein Stapel von mehreren Tests oder anderen Aktionen ist) oder ein Test ausgewählt und gestartet werden (Screenshot 2). Während ein Skript oder ein Test läuft, zeigt das Statusfenster ein leuchtend gelb/schwarzes “RUNNING” an.

### *14.2. Test / Script Abbruch*

Ein Skript oder Test kann abgebrochen werden, indem auf den Button „ABORT/POWER OFF”, “STOP SCRIPT” oder “STOP TEST” geklickt wird.

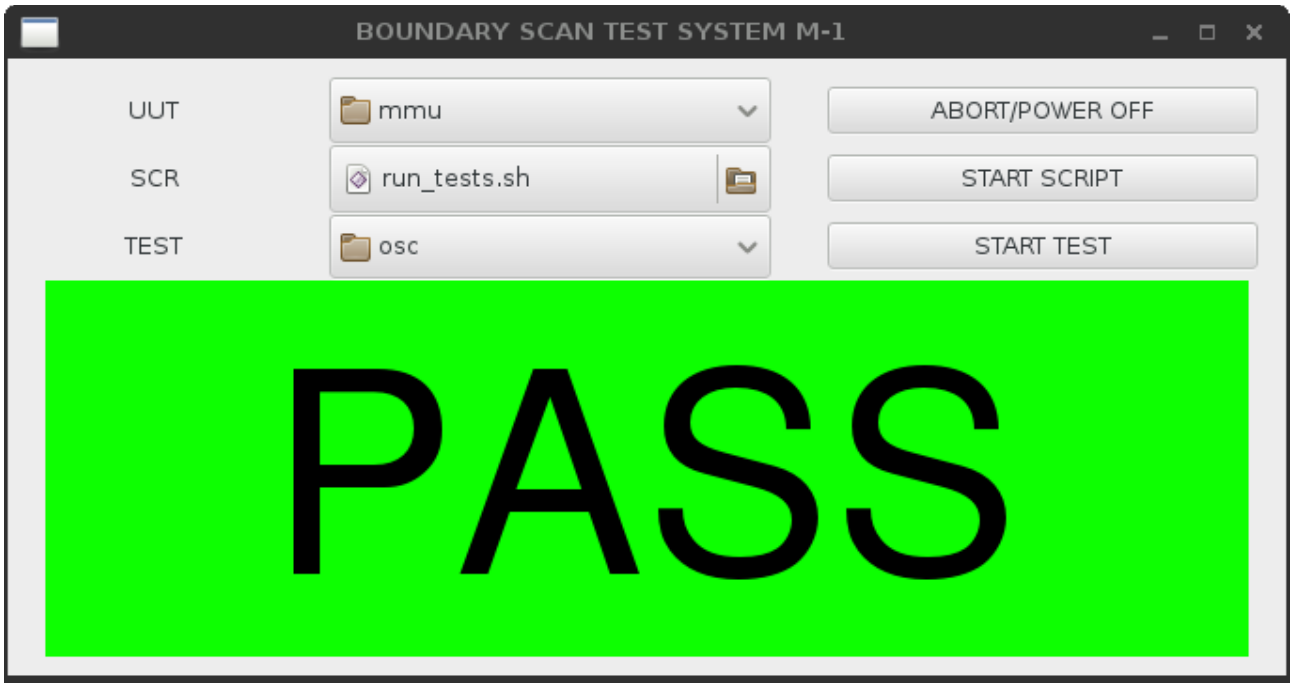
ACHTUNG: Es kann bis zu zwei Sekunden dauern, bis die Software den Abbruch einschließlich Abschalten des UUT vollzogen hat. Der schnellste Abbruch wird erzwungen, wenn der rote Taster “STOP” an der Frontplatte des Boundary Scan Controllers gedrückt wird.




*Screenshot 2: Test/Script running*

### *14.3. Test / Script PASS*

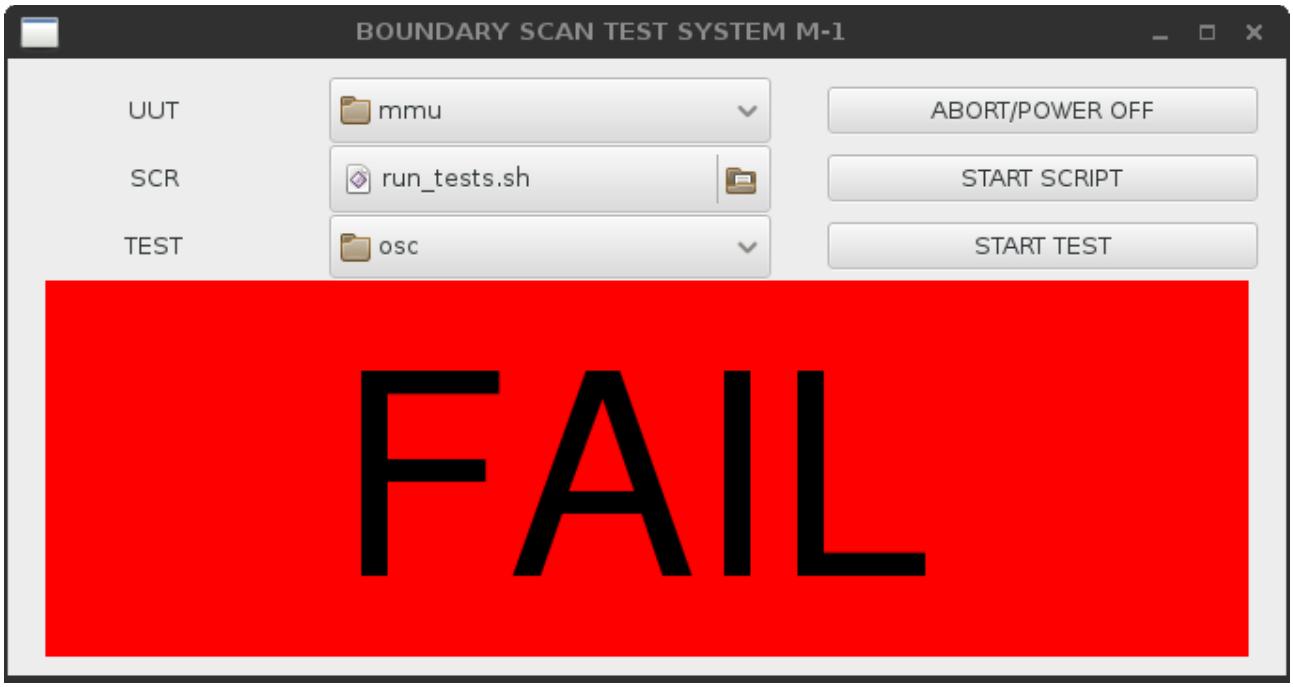
Das offensichtliche Ergebnis eines erfolgreichen Skripts oder Tests ist das leuchtend grüne/schwarze “PASS” im Statusfenster.




*Screenshot 3: Test/Script passed*

### *14.4. Test / Script FAIL*

Ein gescheitertes Skript oder ein fehlgeschlagener Test wird durch das alarmierende rot/schwarze “FAIL” signalisiert. Für den Produktions-Modus ist diese Aussage ausreichend, um die guten von den schlechten Prüflingen zu unterscheiden. Für Reparatur und Fehlersuche wird zusätzlich im Terminal-Fenster ein detaillierter Fehlerbericht über betreffende Netze und Bauteile angezeigt (siehe Text 5 Seite 43).




*Screenshot 4: Test/Scrip failed*

```
BOUNDARY SCAN TEST SYSTEM M-1 Command Line Interface Version 025
================================================================
action          : RUN
Test/step 'intercon' started ...
waiting for test/step end ...

Test FAILED! Diagnosis:
failed scanpath    : 1
step id (dec)      : 7
sxr length (dec)   : 242 (one-based)
sxr fail pos (dec): 9 (one-based)
scan type          : SDR
device position    : 3
device name        : IC303
register           : BOUNDARY
failed bit pos.    : 8 (zero-based)
expected           : HIGH
net class          : PU

secondary net      : CT_D0

JP402 pin 2
IC303 pin 15
IC302 pin 3

primary net        : D0

RN401 pin 2
JP406 pin 2
JP404 pin 2
IC302 pin 15
IC301 pin 1
IC203 pin 11
IC202 pin 11
IC201 pin 13
IC200 pin 13


stuck at LOW or Pull-Up resistor missing !

Test/Step intercon FAILED
```

*Text 5: Fehlerbericht*

# 15. Referenzen


(1) http://www.blunk-electronic.de/bsm/Boundary_Scan_Basics.pdf

(2) http://www.blunk-electronic.de/bsm/how_to_test.pdf

(3) The Institute of Electrical and Electronics Engineers, Inc. , 3 Park Avenue, New York, NY 10016-5997, USA; IEEE Std 1149.1-2001 "*IEEE Standard Test Access Port and Boundary-Scan Architecture*"

(4) Niklaus Wirth (Wirthsches Gesetz): „A Plea for Lean Software": E. Perratore, T. Thompson, J. Udell, and R. Malloy: „Fighting fatware", Byte, Vol. 18, No. 4, April 1993, pp. 98-108 auf https://github.com/sysprv/demo-corporate-documentation-public/blob/master/Niklaus%20Wirth%20-%20A%20Plea%20for%20Lean%20Software.pdf


# 16. Haftungsauschluß

Dieses Dokument wurde nach bestem Wissen und Gewissen erstellt. Ich übernehme keine Verantwortung für Ungenauigkeiten oder Fehler in diesem Dokument. Ich behalte mir das Recht vor, Änderungen zu jeder Zeit und ohne Ankündigung vorzunehmen und verpflichte mich nicht dieses Dokument zu aktualisieren.

Ich übernehme keine Verantwortung für direkte oder indirekte Schäden, die durch die Verwendung des *Systems M-1* entstehen.

Mario Blunk


*Blunk electronic / Owner : Dipl. Ing. Mario Blunk / Buchfinkenweg 3 / 99097 Erfurt / Germany / Phone +49 361 6022 5184*