Boundary Scan System M-1

Manual

Doc. Version 2018-10-12-1

Author: Mario Blunk

Abstract: *Introduction into System M-1. Instructions on operation, installation and maintenance. Basics on test generation for Boundary Scan Testing acc. to Std. IEEE 1149.1 also known as JTAG.*

Keywords: *Boundary Scan, OpenSource, JTAG, IEEE1149.1, netlist, partlist, import, database, UUT, module, hierarchic design, prefix, connectors, part, net, class, pull-down, pull-up, short, open, test coverage, primary net, secondary net, FPGA, CPLD, programmable logic, cell, input, output, highz, open-drain, open-collector, scanpath, scanport, TAP, scanchain, single step, breakpoint, current consumption, delay, timeout, shutdown, memory, RAM, FLASH, I²C, SPI, Microwire, mission critical, scripting, bash, Linux, Ada, gtkada, gnat, gcc, git, version control, ASC-II, intrusive, non-intrusive, version control, git, development mode, production mode, graphical user interface, GUI, line break, line feed, DOS, UNIX*

Table of Contents

1. Introduction

1.1. What is Boundary Scan ?

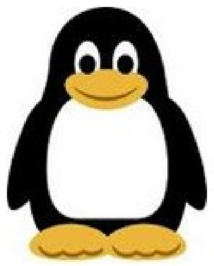
Boundary Scan (meaning: sample ath the boundary) is an adapter-less structural test technique which:

1. detects manufacutring faults down to pin/pad level.
2. enables testing of ICs, assembled PCBs and whole systems.
3. enables electrical access to physically not accessible nets, pins and pads.
4. reduces the electrical access to the UUT down to 5 wires.
5. tests the UUT while under power.
6. enables In-System-Programming (ISP).
7. bases on silicon embedded electronic testpoints.
8. is standardized acc. to IEEE 1149.x (since beginning of the 1990es).

1.2. Why Boundary Scan with System M-1 ?

Boundary Scan saves you money and time !

- No license – OpenSource !
- No maintenance contract !
- detection of design errors in early design stage
- While board-bring-up of prototypes shorts, opens, missing parts are detected on a structural low level.
- Whole systems (racks, backplanes, modules, …) can be tested on a structural low level.
- simplified and and speed up repair
- less mechanical equipment (no adaptors, less functional testing)
- rapid, automated test generation (build-processes, scripting)



Read more here: http://www.blunk-electronic.de/bsm/how_to_test.pdf

1.3. The Technique acc. to IEEE 1149.1 at the UUT (in brief)

Around (at the boundary) the actual logic core of an IC a ring of registers (with testpoints) is laid. They serve to take measurements and to control pins/pads:

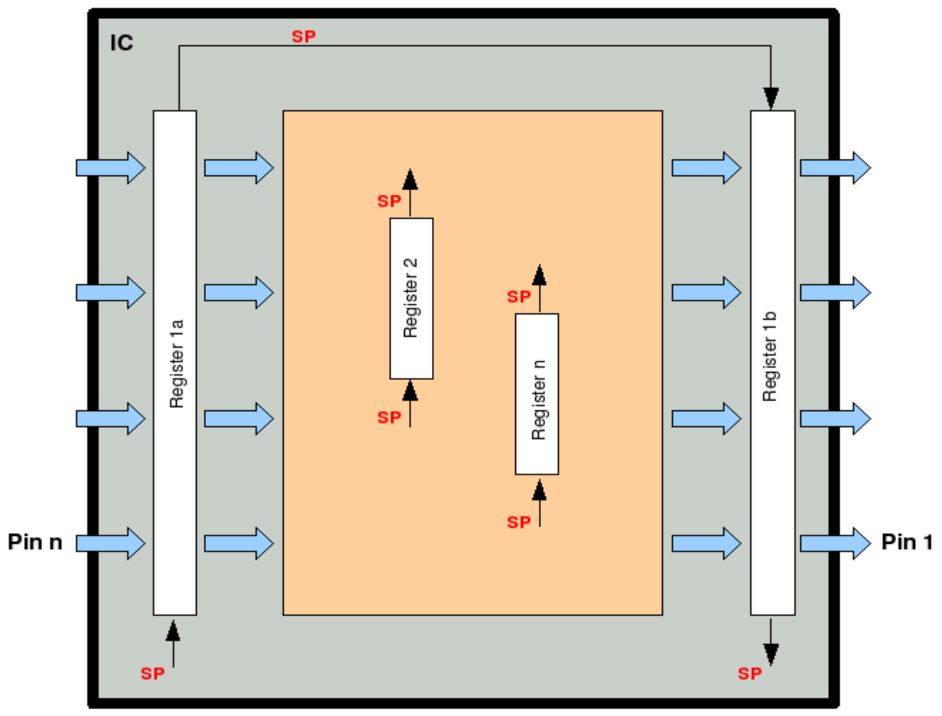


figure 1: a boundary scan capable IC inside

For device identification, for programming or debugging, other registers can be incorporated in the core.

Via a serial scanpath (SP) test data is shifted into the registers and read from them.

On MCMs (Multi-Chip-Modules) test data can be exchanged between its ICs and the surrounding circuitry:

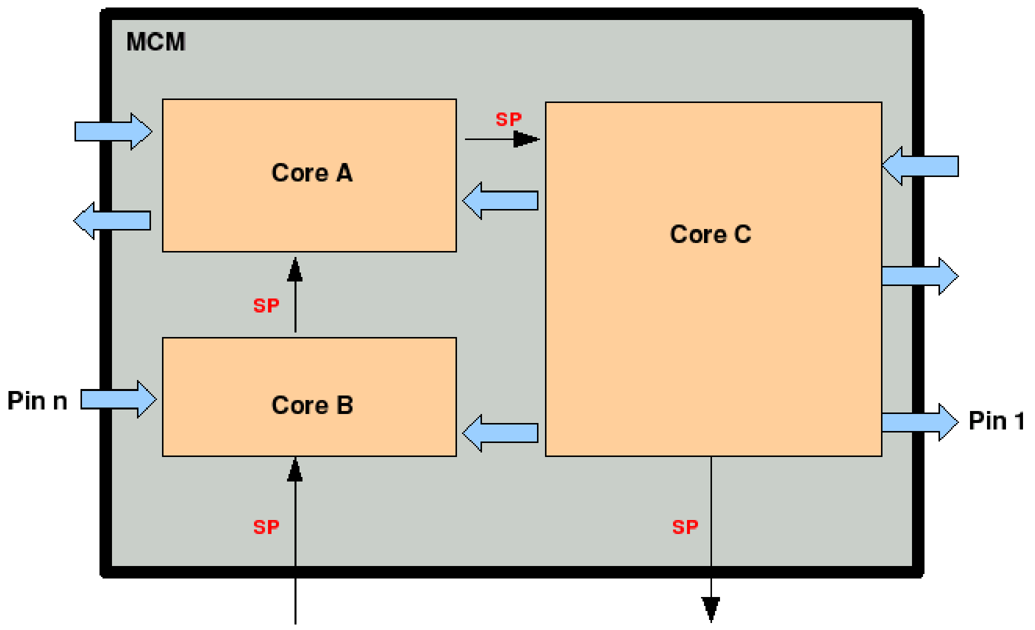


figure 2: a boundary scan capable Multi-Chip-Modul inside

Between ICs and MCMs of an assembled PCB, test data can be exchanged:

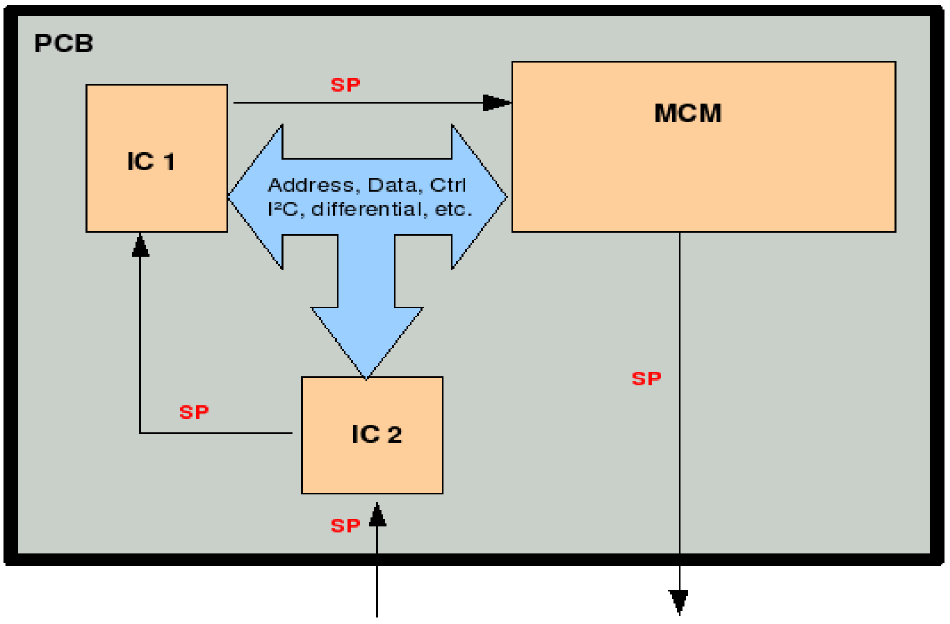


figure 3: a boundary scan capable PCB with a scanpath

Interconnections between modules can be tested (backplanes, wiring harnesses, …):

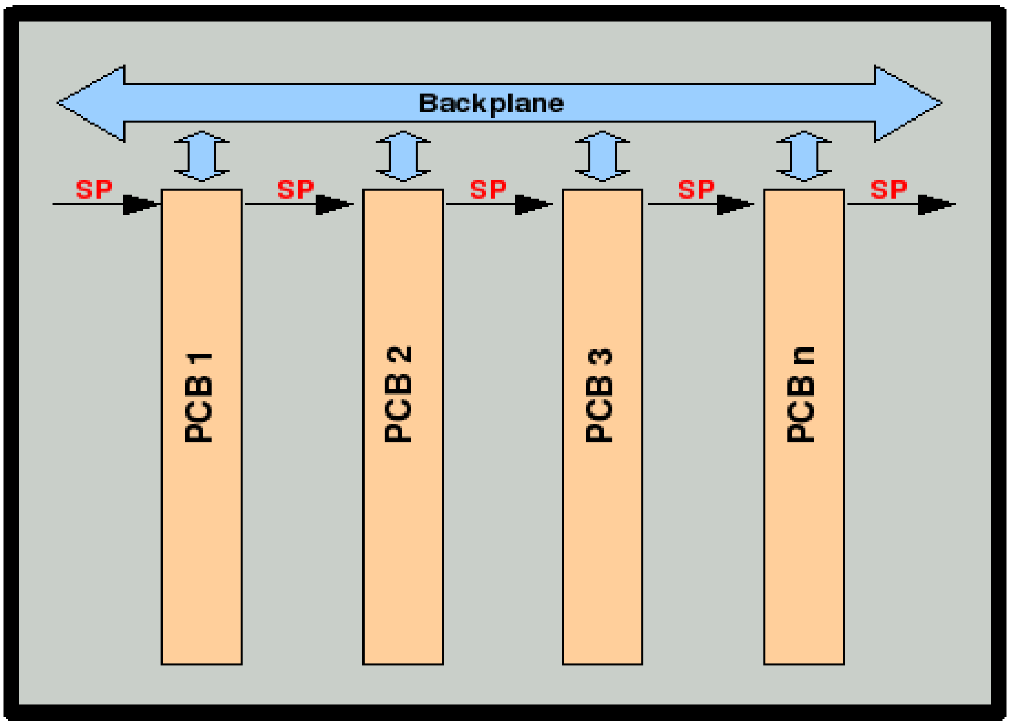


figure 4: a boundary scan capable system

This is very valuable when **system tests in the field** are to be conducted:

- medical instruments and systems
- wind power plants
- vehicles
- industrial controls

The „wiring“ on the PCB:

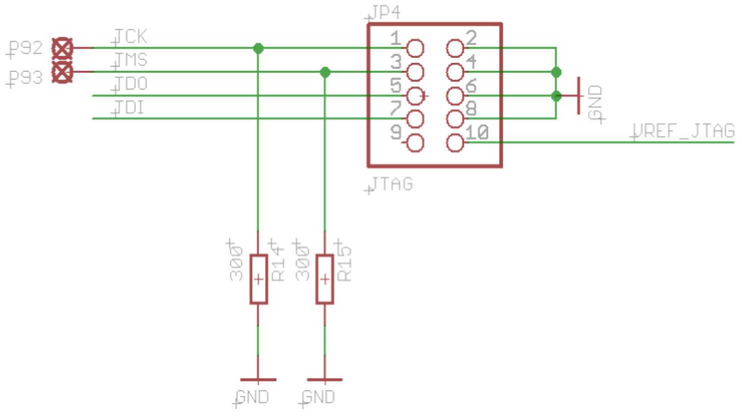


figure 5: scanpath connector

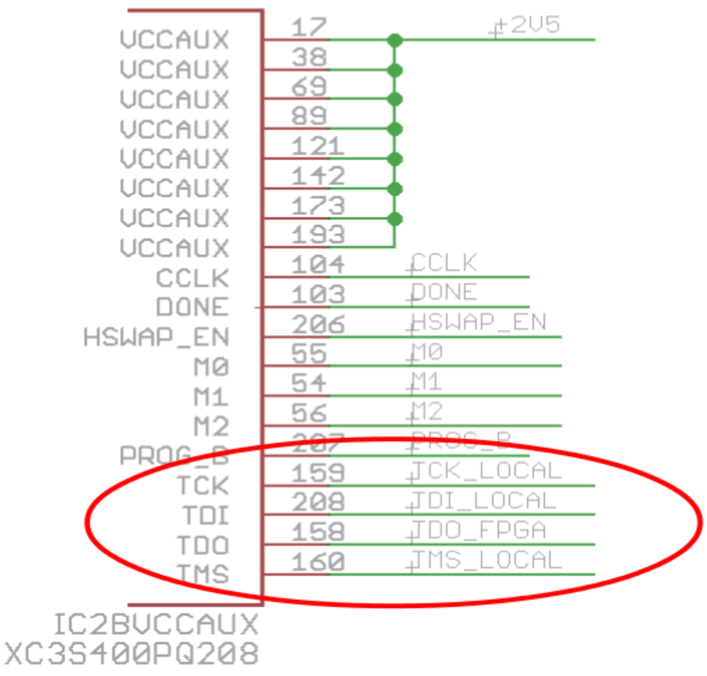


figure 6: scanpath terminals of an IC

1.4. Financial investment

To get right to the point: *System M-1* is primary not for sale, it is for rent ! This defuses the standard question about the price slightly:

- The customer may rent *System M-1* as long as required or may buy it.
- There is NO confusing features vs. license table (due to OpenSource). You get a development, repair and execution station in one package.
- Charges for training, test program setup and support apply.
- If the customer wants to buy it, she/he pays for the delivered hardware, that is the boundary scan controller (BSC) and the PC that comes along with it.

Please contact Blunk electronic for quotations.

1.5. System overview

The boundary scan system M-1 consists of:

- a regular Linux-PC with various software modules installed.
- a so called boundary scan controller (further-on referred to as BSC)
- the target hardware you want to test or debug → the unit under test (further-on referred to as UUT)

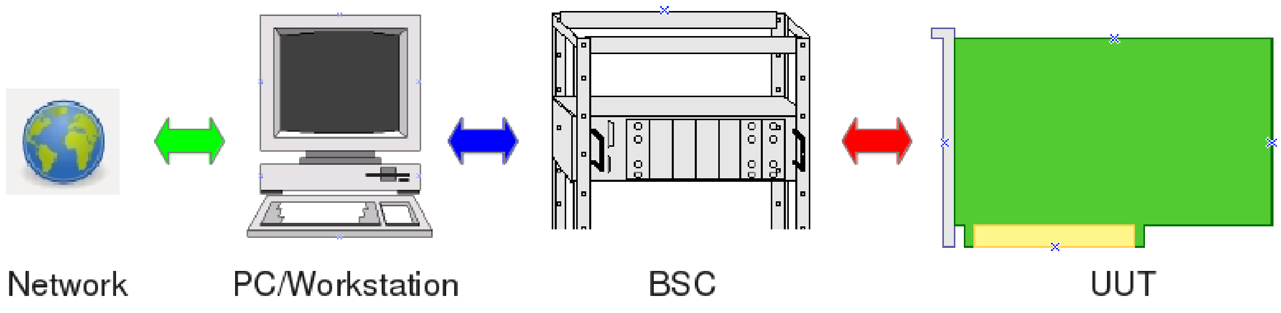


Figure 7: System M-1 components

On the PC you manage projects and generate test programs. The BSC executes test programs by sending serial data into the UUT and reading serial data from the UUT.

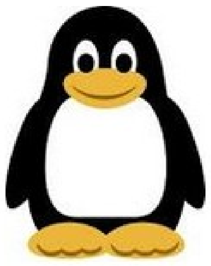
1.6. Features

There are some important features to point out:

- *System M-1* includes a ready made PC with all the software required for test development, debugging and execution. The customer is freed from any software installation issues.

- Maintenance and support can be done remotely via the web interface. The network access may be corporate inside or from outside. The latter reduces traveling and on-site support.

- The operation system running on the shipped PC is Linux© . Linux is highly virus immune, long-term stable and highly network capable – a reasonable choice for an assembly line deployment.



- Scripting for automated build-processes

- Crucial parts of the *System M-1* software have been written in ADA 2005 that perfectly fits the requirements of mission critical applications !

- The software is **open to user specific applications or graphical user interfaces** (GUI) !

- The software is well modularized. Updating is done regarding the very module affected, not all across the system. Each module can be accessed via external software i.e. in regard to system integration (Flying-Probe-Test, In-Circuit-Test, AOI, System Test, QM, …)

- Large parts of the software are open sourced (GNU General Public License).

- Most files that are generated by the individual software modules are strictly ASC-II formatted. They are human readable and editable thus allowing version controlling (like Git https://git-scm.com) and verifying.

- All the software of *System M-1* is shipped fully installed. The customer does not need to worry about issues like:

- ✔ operating system
- ✔ hardware platforms
- ✔ driver software
- ✔ licensing

- ➔ By default, the BSC provides power monitoring of 3 individual channels to protect the UUT from power failure.
- ➔ By default, the BSC provides a full galvanic separation from the UUT upon power failure, test FAIL or non-test-mode. The latter is a constraint for interaction with other test systems.
- ➔ The BSC comes in a robust 19 inch housing with rugged connectors for fast and safe UUT change.
- ➔ The BSC can operate autonomously from the PC. For fast pre-testing of UUTs the operator activity is reduced to pushing front panel buttons START and STOP. Two LEDs indicate PASS or FAIL (see figure 8 page 14 and section 9 Executing Tests (run) on page 34).

Standard functions a Boundary Scan System acc. to IEEE1149.1 **must** provide are listed here without going further into the details at this point:

- ➔ Importing CAD data from any CAE system (like Altium/Protel, KiCad, ZUKEN, TestExpert, EAGLE , …)
- ➔ Minimal UUT access via a 5 wire serial test-bus (so called TAP)
- ➔ Adjustable scanport/TAP voltage (1.8V to 3.3V)
- ➔ Fault detection down to pin-level (stuck-at, shorts, opens, missing pull-resistor)
- ➔ Scanpath Test (Infrastructure Test)
- ➔ Interconnect Test
- ➔ Memory/Cluster-Interconnect Test (RAM, FLASH)

- Clock Test
- Pin-Toggle Test
- System Test

2. Operation & Components

2.1. Licensing

As said in section 1.4 Financial investment page 9, there is NO license required for the software. You get a system fully featured.

Regarding the hardware, that is the BSC, you are charged for buying or renting the BSC.

2.2. Software

The software of System M-1 is basically a collection of single executable Linux binaries (ELF) – a so called toolchain. For managing projects, generating and executing test programs these executables get launched via command line. In production mode, means in an assembly line, tests are launched via graphical user interface (see section 14 Production Mode page 41).

A it is custom with UNIX and Linux applications, the system can be operated via a graphical user interface (GUI) as well as the command line. Major tactical functions for test execution and development/debugging are implemented in various executable files which can be launched individually by:

a) local command line (console or terminal)

b) a very lean GUI for launching tests

c) a customized, tailored GUI

d) a third party application (like Flying-Probe-Test, In-Circuit-Test, …) via system calls.

e) remotely via network (ssh, remote desktop, vnc, ...)

The fact of individual executable files eases system updates and bug fixing. A malfunctioning executable can easily replaced instead of updating the whole system.

In addition it should be pointed out that the system is **open** for any extension made by the customer or third parties. As graphical user interfaces are frequently subject to individual constraints it proved ineffective to ship an over-sophisticated GUI to customers (see section 2.2.2).

2.2.1. Version Controlling and Software Verification

A files generated as intermediate steps are purely ASCII based text files (plain text). Only the executable code that runs in the boundary scan controller (BSC) is binary. So version control with tools like *Git* or software verification can be accomplished.

2.2.2. Graphical User Interface (GUI)

For test program development there is no graphical user interface. All functions are available by launching an executable with dedicated arguments.

For deployment in an assembly line, for executing tests, there is a lean GUI available. It is made for productive environment with the most essential widgets. See section 14 Production Mode on page 41 for more.

2.3. Hardware

The BSC comes in a robust 19 inch housing with rugged connectors for fast and safe UUT change (figure 8). It can operate autonomously from the PC. For fast pre-testing of UUTs the operator activity this can be reduced to just pushing front panel buttons START and STOP. Two LEDs indicate PASS or FAIL.

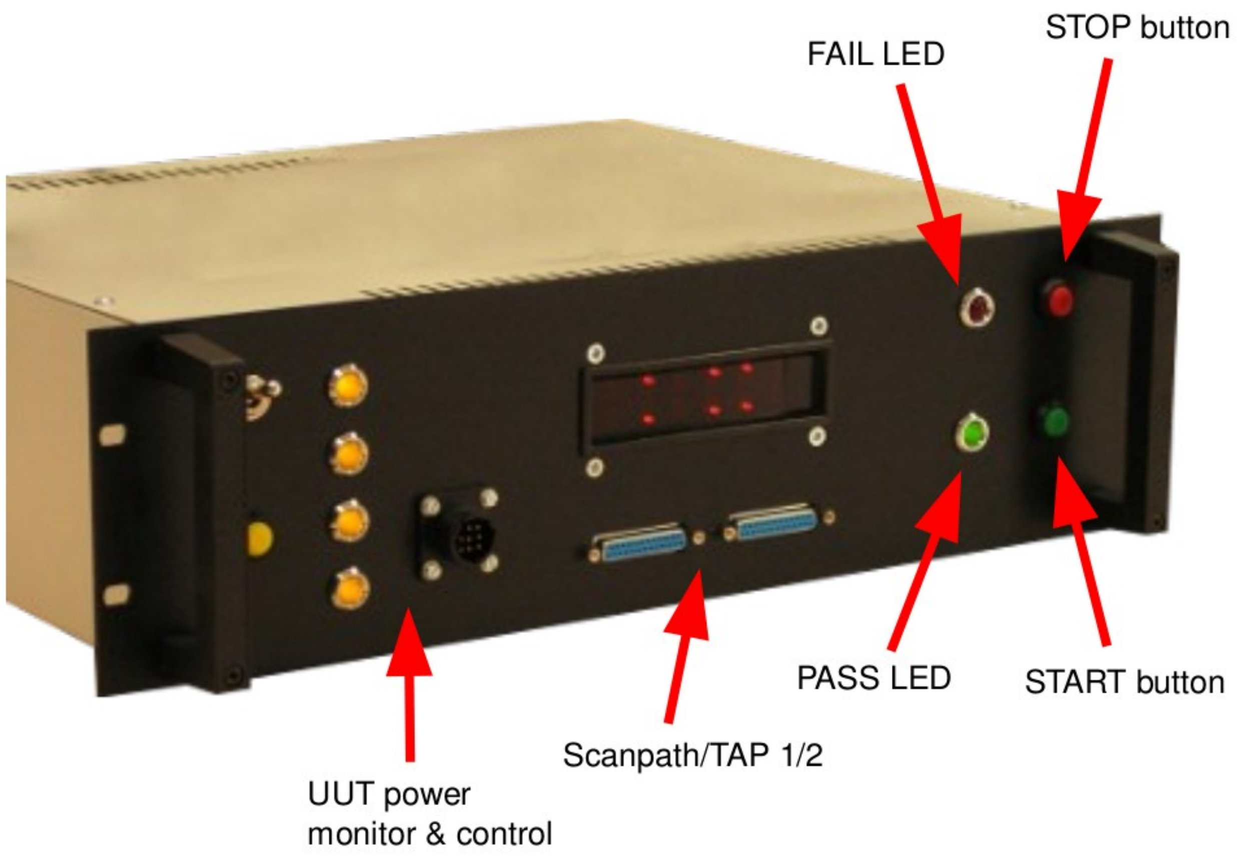


figure 8: BSC front panel

3. Where to get it

The software is provided by us or available at GitHub:

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

4. Installation

In general, the system is shipped fully installed. The user does not have to worry about installation issues. However, for evaluation the installation is as follows:

1. Make sure `gcc` and `gnat` (C and Ada-Compiler) are installed.
2. Create a directory where the following repositoriy will be cloned in:
3. The M-1 repository from https://github.com/Blunk-electronic/M-1.git
4. Change into directory M-1 and run the install script install.
5. Find help for installation of gtkada here

   https://github.com/Blunk-electronic/M-1/blob/master/src/bsmgui/readme.txt

5. Support

In general boundary scan is support intensive. For training, consulting, test program setup please contact us at www.blunk-electronic.de.

You are highly welcome !

6. Test Development Flowchart

figure 9 depicts the general work-flow with basic actions required to generate test patterns or in-system programming actions, further-on referred to as ISP. Every tool the user launches directly or indirectly modifies the UUT database, further-on referred to as UDB. The UDB in turn is the data pool test generation bases on. Every action outlined here will be described more detailed in the following sections. So have this chart at hand throughout this manual.

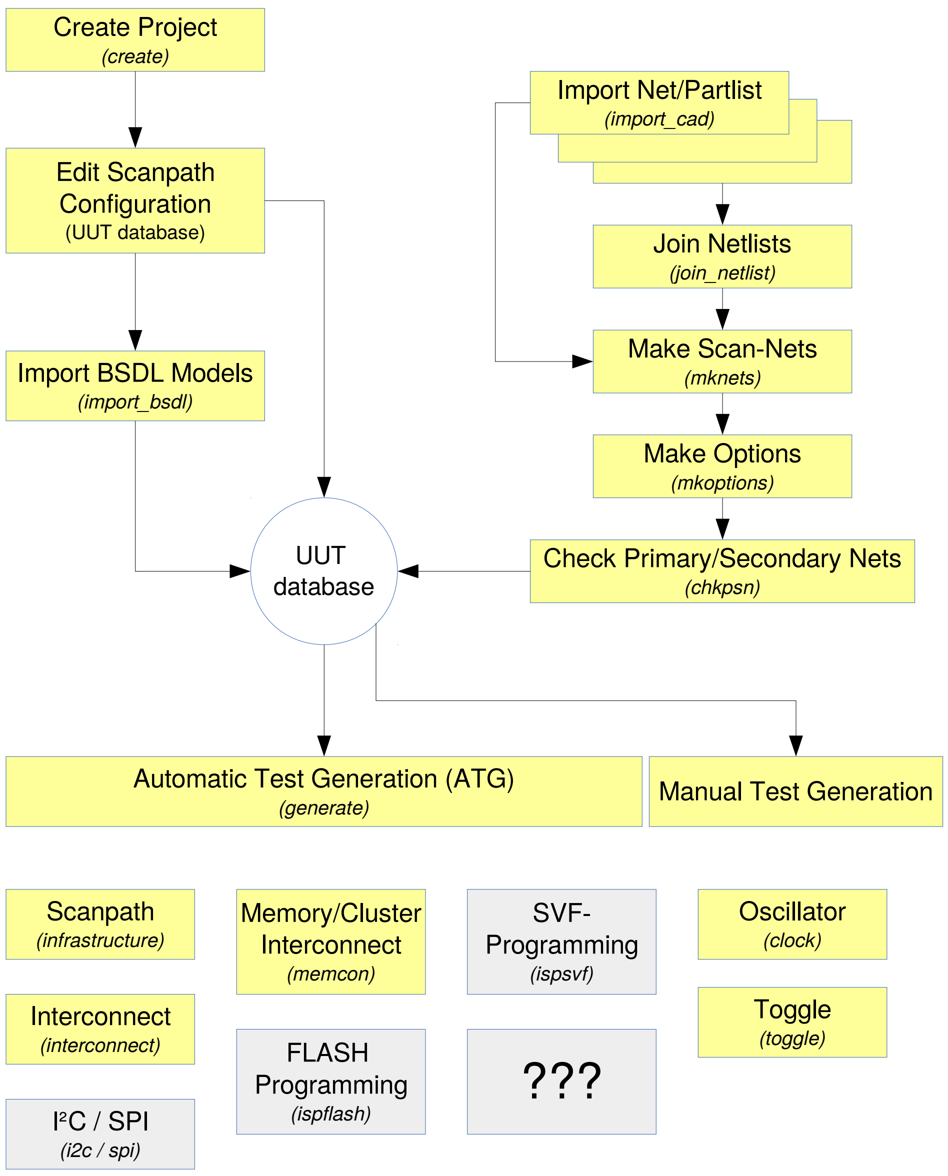


figure 9: General Workflow (greyed box means: under construction)

6.1. *Create Project* *(create)*

The first step is to create a project. Open the console terminal and launch the command:

```
luno@notebook1:~/tmp/bscan_projects> bsmcl create my_project
```

It creates a directory my_project that contains the project structure of that project. Change into the directory my_project and have a look at its contents.

6.2. *Setup UUT Database*

Once a project has been created, a bare UUT database is available. Its name is composed of the actual project name and the extension .udb. So the database of our example project *my_project* is myproject.udb.

Open the database with a text editor and go to SubSection chain_1. Usually here you describe the scanpath(s) of your UUT. The scanpath (or scan chain) is the backbone where all boundary scan capable ICs – furhter-on referred to as BICs - are chained. For the start the SubSection contains only comments (headed by double dash). In our example here uncomment the fifth line in order to make IC301 the only BIC of scanpath #1. The comments in this example explain how to read this section.

```
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 database SubSection chain

This SubSection specifies the order of BICs in a particular scanpath, the package, the path to the BSDL model and some options. The BSDL import can be controlled by options. See section 6.3.1 page 18.

Most of the following actions base on this configuration.

6.3. Import BSDL-Files *(import_bsdl)*

Once the scanpath has been specified the BSDL models should be imported into the database. Therefore run the command

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

Your database gets extended by the section registers. This section contains the BSDL information for all BICs as specified in Text 1 page 17. Read more on BSDL syntax in the IEEE1149.1 Standard, see reference (3).

From this state of the database, the most basic test , the so called Scanpath Test (or Infrastructure Test) can be generated. In this case proceed with section 6.8.1 page 26.

IMPORTANT: BSDL-Modells may reflect the behavior of ICs in pre- or post configuration mode. Especially programmable logic like FPGAs or CPLDs require special treatment if boundary scan testing after configuration is to be conducted. Have a look into the BSDL-file for advises.

6.3.1. Option to remove prefixes of pins

With option „option remove_pin_prefix“ prefixes, which are used in BSDL-models with pin names, can be removed. Text 2 shows how to use this option in order to remove the prefix „P“ while importing the BSDL-model of a Spartan-3 FPGA.

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

Text 2: remove pin prefix

6.3.2. Known Issues

6.3.2.1. Import freezes or outputs faulty Database

BSDL files are exported from CAE systems or other Boundary Scan tools. The line break (or line ending) might be DOS formatted (LF+CR). Thus the BSDL file must be converted to UNIX-Format (LF) previous to importing:

```
$ dos2unix models/xc3s400_pq208.bsd
```

6.4. Import Net & Part List *(import_cad)*

With the database in its current state you can't do much. Now we extend it with the so called netlist of the UUT. Every CAE design tool can export a netlist (an optionally a partlist) which reflect all the electrical connections of the board you want to test. Usually these files are strictly ASCII based, which means you can open them with any text editor, do net or part searching or even modifying (Yes there are cases when you are forced to edit a netlist file...).

The project directory of our dummy UUT my_project contains a sub folder named cad where you should copy net and partlists or where you can create symlinks to the acutal files. Links are favorable because they always point to the most recent version of the targeted file.

The command to import net and partlist requires specifying the format and the location of the netlist and partlist. If there is no partlist, just leave it off. Run the command from inside the project directory my_project:

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

For other netlist formats just type the command:

```
$ bsmcl import_cad
```

It outputs the formats currently supported. Since *System M-1* is widely opensouce, you are invited and free to write your own netlist importer. You may use any programming language to accomplish that. Important is to get as a result the skeleton file:

The outcome of the netlist import is a so called skeleton file, named skeleton.txt. By its extension it's clear this is also an ASCII file. Just have a look at it with a text edtor. The skeleton file contains the whole UUT netlist in a standardized format.

6.4.1. Known Issues

6.4.1.1. Import freezes or outputs faulty Skeleton

Net and partlists are exported from CAE systems. The line break (or line ending) is mostly DOS formatted (LF+CR). The net and partlist must be converted to UNIX-Format (LF) previous to importing:

```
$ dos2unix cad/netlist.txt
```

6.4.2. Hierarchic Designs

If you are working with just a single board UUT always import as top module. The following sections can safely be skipped in this simple case. Just continue reading in section 6.5 Make Boundary Scan capable Nets (mknets) page 21.

6.4.2.1.Top Module & Sub Modules

If your UUT consists of more than one board, or if it is a whole system, one board must be selected as master module, further-on referred to as top module. All other boards will the be imported as sub modules.

If the argument „main“ is passed after the net- or partlist, the skeleton is created as top-module. That is the file `skeleton.txt` .

If you provide the argument „sub“ instead, followed by the name of the sub-module, the skeleton with the name `skeleton_my_submdule.txt` is created:

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

Each import of a submodule creates a dedicated skeleton file that bears the name of the sub module like `skeleton_X15.txt` . All parts and nets of the sub module are therein prefixed with `X15`. So for example the resistor R5 in submodule `X15` gets renamed to `X15_R5`. The reason for prefixing parts and nets is that later the test generators will see all modules as one target. The only way to tell which parts and nets belong to which module is by giving the prefixes.

After all, if this section confuses you, just play with the import command (see section 6.4 Import Net & Part List (import_cad) page 19), import sub modules and have a look at the resulting skeleton files.

Next stop: Joining skeletons.

6.4.3. Join Skeletons (join)

In order to merge top module with one or more sub modules run for example the command:

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

It joins the sub module `X15` with the top module skeleton. The result is a skeleton file that contains the nets and parts of both modules. This step can be repeated with other sub modules many times until all modules are joined in a single `skeleton.txt`.

6.5. Make Boundary Scan capable Nets (mknets)

The designation *skeleton* speaks for itself. The skeleton file must be brought to life by transferring it into the database. The latter provides information on which pin (or pad) of a BIC (Boundary Scan capable IC) is connected to which net. The command

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

reads the skeleton file and imports it into the specified database (remember there can be many databases each of which represents a particular UUT configuration). The database gets extended by another section called *netlist*. Have a look into the database, scroll to section *netlist* and inspect the outcome:
Every net now is a subsection. The subsection header also tells about the net class. By default all nets are in class NA, which means “not assigned”. A net of class NA is completely excluded from any test generation. So please consider this default as safety measure.
After the subsection header all parts connected to the net are listed with their name, part class (currently a placeholder '?'), package, value and pin name. A BIC pin has more information, the so called *cell info*. The *cell info* is derived from the information in section *registers*, which in turn was built from the specifications of the BSDL files (see 6.3 Import BSDL-Files (import_bsdl) page 18).

```
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 this state of section *netlist* each net exists for itself, without any interconnections to other nets. The next step assists the user in adding interconnections.

6.6. Make Options (mkoptions)

Frequently, nets are connected with each other by series resistors, solder bridges, wires, inductors, connectors and so on. Since *System M-1* does not feature artificial intelligence, the user must declare parts that connect nets with each other.

6.6.1. Edit mkoptions.conf

The text file mkoptions.conf is where parts are specified, which connect nets. In order not to start from scratch, a well prepared file is already present in your project root directory. You only have to modify it according to your UUT. Open it with a text editor and have a look at it:

```
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

Comments start with double dash "--". There is a "section connectors" where you specify by which connector pair a module is connected with another module. Note the prefix ! In this example sub module module_a has a connector X1 which mates with sub module module_b connector X3. Another connection is made by module_c, J55 and J1 on the top module. Remember, the top module does not have prefixes. The assumption is that the pins meet each other one-to-one, which is usually the case with board-to-board headers an the like.

Section “bridges” contains parts that act like a bridge between two nets. It does not matter if these devices are resistors, wires, inductors or whatsoever. Important is whether a certain part results in a series connection. In our example here there are lots of solder bridges whose name starts with N. So the notation N* addresses all parts of this kind. Dedicated declarations like the resistor R99 are also possible. A special case are arrays. Mostly we deal with resistor arrays. Specify the array name, followed by the keyword array and the pins where the internal resistors are tied with.

6.6.2. Generate Options File

After editing mkoptions.conf run the command:

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

The outcome is a so called options file, ending in *.opt, ASCII based of course, that contains the full net list of your UUT including all the interconnections you just specified in mkoptions.conf. By default the options file has the same name as the UUT database. Optionally, the name can be specified as

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

6.6.3. Routing Table

As a byproduct, interconnected nets are automatically written in a CSV file that has the same name as the UUT database. So if you open my_project.csv with a spreadsheet software the routing from one net to the other can be seen.

6.6.4. Options File Structure

The structure of the options file is as follows:

- statistics about bridges, arrays and net count

- net list with primary and secondary nets

Mkoptions decides which net is to be a primary net and which net is a subordinated secondary net. If a net contains a boundary scan capable driver, it is selected as primary net. Other nets, regardless if they have scan capable pins or not, become secondary nets. The idea behind is to specify in advance, for later test pattern generation, which net is the driving net and which net(s) are listening to their superordinated primary net during test execution.

6.6.4.1.Editing the Options File

Since the options file reflects the whole UUT (including net interconnections and dependencies) you are now requested to edit the net classes. By default all nets are in class NA, which means “not assigned”. This default is a safety measure that prevents activating drivers that contend with other unknown signal sources or control peripheral components. There is NO automatic net class assignment. You are requested to dig into the schematic and to assign net classes. In other words, study the UUT schematic and decide carefully which net is to be tested in which manner.

Available net classes are:

1. NA (not assigned) → net completely excluded from test generation (default)
2. NR (no restriction) → net can be tested without any restriction
3. PU (pull-up) → net has pull-up behavior
4. PD (pull-down) → net has pull-down behavior
5. DH (drive-high) → net is to be driven permanently high (if scan capable driver available)
6. DL (drive-low) → net is to be driven permanently low (if scan capable driver available)
7. EH (expect high) → net is expected to be high (by whatever source)
8. EL (expect low) → net is expected to be low (by whatever source)

6.7. Check Primary/Secondary Nets and Net Classes *(chkpsn)*

Chkpsn checks if the assignments in your options file are correct and feasible. Whenever you feel it is good to check if the settings you made in the options file, run the command

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

It reads the options file (same name as the database but with extension *.opt). If your options settings are correct and acceptable, they are imported into the UUT database. This command can be repeated any time in order to release more nets to the test generation.

It is a good and safe approach to release on net (or a group thereof), generate a test, see the result and release the next net to the test generators.

Note: Currently, the routing table, generated by *mkoptions* does not get updated with the manually added primary-secondary net dependencies (see section 6.6.4.1).

6.8. Test Program Generation

As a general rule: Test program generation (or test pattern generation) bases strictly on a valid UUT database. As said earlier, you can have multiple databases living in the project root directory. Each of which represents a certain UUT configuration. When launching a test program generator, the name of the desired database is passed as argument, along with the test profile and other optional parameters.

The result of a test program generation is a directory, bearing the name of the actual test, that contains the source code of the test. The source code file is further-on called as *sequence file*. It is ASCII based, can be regarded as programming language and can be modified for fine tuning (see section 6.8.6 Manually Writing Tests page 30). Mostly it can be compiled right away by launching the compiler *compseq* (see section 7 Compiling Tests (compile) on page 33).

6.8.1. Scanpath Test (infrastructure)

The first test you usually generate is the so called scanpath test (or infrastructure test). Its only purpose is to ensure the scanpath that runs through your UUT is intact. The scanpath is the backbone where test data between UUT and Boundary Scan Controller travels through. This test is by nature **non-intrusive**, means it can be executed without affecting the UUT normal operation.

The scanpath test does not require netlist information. It can be generated right after importing BSDL files (see section 6.3 Import BSDL-Files (import_bsdl) on page 18).

Run this command in order to auto-generate a scanpath test:

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

6.8.2. Interconnection Test (intercon)

Without the ability to run interconnect tests, boundary scan is worthless. In other words: Interconnect testing is the workhorse of boundary scan. This test is **intrusive**, means it directly stimulates driver pins and reads receiver pins, thus verifying the connections to and between BICs. The precondition for this test is an intact scanpath and at least one BIC has boundary scan capable pins. Usually an IC that is part of the scanpath, has scan capable pins. There are exceptions.

With just one IEEE1149.1 capable IC, we refer to it as BIC, on your UUT you can do a lot of testing, even if there is no other BIC to exchange data with. With a solitary BIC you can drive or read peripheral components (like relays, motors, valves, displays, lights, switches, jumpers, ...), drive and read inputs from non-scan ICs or units, test shorts between pins, test pull-resistors.

The test coverage ranges from light to moderate.

Things get better if there are other BICs on the board, because more interconnections can be tested.

Run this command in order to auto-generate an interconnect ion test:

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

CAUTION: The Interconnection test is intrusive !

IMPORTANT: See section 6.3 page 18 on specials about behavior of programmable logic.

6.8.3. Memory Interconnections Test (memcon)

As said in section 6.8.2 Interconnection Test (intercon) the connections between BICs are important to increase test coverage. If there are memories like RAMs which are connected with BICs, the test coverage can be boosted even more. Via Boundary Scan certain memory locations can be written with certain data. Reading back the content allows detecting faulty address, data or control nets.

Important for generating a memory interconnet test is a model of the memory device. *System M-1* uses ASCII based text files wherein the device behavior is described. On automated test generation, the database, the test profile, the target device name, model file and the physical package is to be specified as shown in this example command:

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

CAUTION: The memory interconnections test is intrusive !

6.8.4. **Clock** (clock)

Frequently a clock signal is to be tested. In most cases it is fully sufficient to verify the signal is toggling. The actual frequency or duty cycle are of secondary importance. So in order to auto-generate a test that samples a clock signal, parameters like database, test profile, target net, target BIC, retry count and sample interval are to specify:

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

The example shown here generates a test that tests net “osc”, uses IC1 pin 7, samples 5 times with a pause in-between of 0.1 seconds. If no signal change can be detected, the net “osc” is faulty. The clock test is **non-intrusive**, means it does not affect the UUT operation. So the test can be executed while the UUT is in fully operational mode.

NOTE: If a clock test is generated based on a database with DH or DL nets, it is going to **fail** when executed. Static driven nets of class DH or DL require **intrusive** mode. In this case the source code of the test program must be modified so that instead of SAMPLE the EXTEST mode is used.

6.8.5. **Toggle** (toggle)

While debugging a UUT, frequently a single net is to be driven high or low in order to lit an LED, turn on a relay or to take measurements. In an assembly line frequently LEDs must be tested. In contrast to the clock test (see above), the toggle test is intrusive, means it halts the UUT and stimulates a targeted net. When launching the automated test generator, the database, the test profile, the test name, the target net, the cycle count and the low/high-time specify the outcome.

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

The example shown here makes the net “OSC” toggle 10 times with a low-time of 0.1 and a high-time of 0.2 seconds. The test generator cares for selecting the appropriate driver pin.

CAUTION: The toggle test is intrusive !

6.8.6. Manually Writing Tests

In general, after automatic test generation or when writing test programs manually, the outcome is a ASCII based text file with extension `*.seq`. This sequence file can be regarded as high level programming language.

In most cases, a sequence file generated by an automatic test generator, serves as starting point to write well tailored code. Since this requires advanced skills and knowledge about the Boundary Scan technology, this section will be expanded later.

tbd

6.8.6.1.Instruction Set

Instruction	Example	Meaning
connect	connect port 1	connects the scanport 1 with the UUT
disconnect	disconnect port 1	disconnects the scanport 1 from the UUT
power up	power up 2	connects power channel 2
power down	power down 2	disconnects power channel 2
	power up gnd	connects power channel GND (no current monitoring !)
	power down all	disconnects all power channels
imax	imax 1 0.2 timeout 2	sets maximum current of power channel 1 to 0.2A with a timeout of 2 sec.
delay	delay 0.5	pauses for 0.5 sec.
set	set IC1 drv boundary 4=0	assigns boundary register cell 4 of IC1 the drive value 0
set	set IC1 exp boundary 5=1	assigns boundary register cell 5 of IC1 the expect value 1
sir	sir id 4	instruction scan with id 4
sdr	sdr id 5	data scan with id 5
	sdr id 5 option retry 5 delay 1	data scan with id 5, a retry count of 5 with an intermediate delay of 1sec.
trst	trst	scanpath reset (both hard and soft)
htrst	htrst	hard scanpath reset
strst	strst	soft scanpath reset

Table 1: Short Reference Sequence Instruction Set

6.8.6.2. Current Monitoring and Accuracy

The UUT current consumption monitor can be configured with the instruction imax . The instruction requires in a defined order the parameters channel, maximum current, timeout. Example for channel #1 with current limit 0.2A and timeout of 2 seconds:

```
imax 1 0.2 timeout 2
```

The relative error when monitoring the current by the BSC is around 25%.

7. **Compiling Tests** (compile)

Every test with its dedicated sequence file needs to be compiled in order to make an executable binary file. The executable binary file extension is `*.vec` and can be run on the Boundary Scan Controller exclusively. This file is the only non ASCII file within the UUT directory. After compiling, this file lives in the corresponding test directory. When launching the compiler *compseq* you must specify the database the test was generated on and the test itself.

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

8. **Loading Tests into the BSC** (load)

After compilation the binary must be uploaded into the Boundary Scan Controller. See the following example.

```
$ bsmcl load my_osc_test
```

You can upload as many binaries until the controller RAM is full.

NOTE: Each time a test is re-compiled, it must be uploaded again.

On uploading a running test is aborted automatically.

9. **Executing Tests** (run)

Once a test has been uploaded it can be started right away by a simple command as shown below:

```
$ bsmcl run my_osc_test
```

If faults are detected, a report is displayed in your terminal. The test report shows the faulty pin or net of the UUT (see Text 4 page 35).

```
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: Test Report

9.1. Single Step Execution

If the step width is provided on launching a test, the test halts after every step.

9.1.1. Step Width SXR

SXR stands for Scan-Data/Instruction-Register. So SIR means instruction scan whereas SDR means data scan. The example

```
$ bsmcl run my_osc_test sxr
```

causes the test to halt right after every data or instruction scan. After every step the test status is displayed in the terminal window.

When halted the TAP-state-machines of the UUT wait in the end-state (with TCK cleared) as specified in section info of a test sequence file.

9.1.2. Step Width TCK

TCK means Test Clock. If a test is executed with this step width, it is halted after each change of the TCK signal. After every step the test status is displayed in the terminal window.

While shifting, this might be helpful:

BP (bits processed, one-based) increments when a TCK cycle is finished. A TCK cycle starts with a HL-edge (when TDOs and TMSs may change) and ends with a LH-edge (all TDIs and TMSs are sampled). So BP increments on every LH-edge of TCK. A bit is regarded as “processed” when the TCK cycle finishes.

```
$ bsmcl run my_osc_test tck
```

9.1.3. Push Button Start at Front Panel

Once a test has been started in any step mode, it can be carried on in that mode by pressing the green Start button at the PSC front panel. Each exertion causes the test to proceed one more step.

9.1.4. Turning off Single Step Mode

The single step mode is to be turned of by entering the regular command to start a test in normal mode:

```
$ bsmcl run my_osc_test
```

The test will now run until its end without further halts.

The single step mode is deactivated automatically once a test ends so that further tests run in normal mode.

9.2. Breakpoints

Breakpoints allow the user to halt a program at a well defined position in order to investigate the state of the UUT. This way a test program can be debugged.

9.2.1. Setting the Breakpoint

A breakpoint is set by specifying the sxr ID followed by the bit position(one-based) to halt at. The bit position to halt at is optionally. If left off, the execution just halts after the specified sxr in its end-state (TCK cleared) as defined in the sequence file. The following example sets a breakpoint right after SXR number 6:

```
$ bsmcl break 6
```

There is no distinction between data or instruction scan. If the test execution is to halt after a certain bit position, see the next example:

```
$ bsmcl break 6 217
```

The breakpoint is not related or bound to any particular test. Execution halts whenever any test has processed the specified sxr and bit position.

Only one breakpoint can be set.

9.2.2. Clearing the Breakpoint

To delete the breakpoint just do:

```
$ bsmcl break 0
```

The test execution will not halt at a any position any more.

9.3. Autonomous BSC Operation

For fast GO/NOGO pre-testing just press the green START button at the BSC front panel. The last test executed via terminal, get started this way. The green or red front panel LEDs indicated a PASS or FAIL result.

10. Query Database Item

Frequently during test program setup, properties of certain items like a net, a boundary scan IC must be inquired for. Especially so called “shared control cell conflicts” required detailled information about the affected control cell.

The module *udbinfo* serves this need and keeps you from getting lost among schematics, assembly drawings, netlists and BSDL files.

10.1. Nets

For example if you need detailed information about the net ADR15 type the following command:

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

10.2. Boundary Scan ICs (BICs)

To obtain properties of an IC like IC303 do:

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

10.3. Shared Control Cells

The most valuable feature of this module is the control cell inquiry. To see which nets share the same control cell see the following example. It inquires for the nets which share control cell 4 of IC303:

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

11. Show System Configuration

To get a short report about the system configuration type this command:

```
$ bsmcl configuration
```

It outputs names of base directories and the interface name (like /dev/ttyUSB0) the Boundary Scan Controller is connected with.

12. Show Firmware

The firmware implemented in the boundary scan controller can be inquired by the command:

```
$ bsmcl firmware
```

13. Batch Execution/Scripting

All commands shown in the sections above can be easily combined (also with other applications) and repeated using the traditional, powerful Linux/UNIX shell. Automated build processes can be implemented this way.

NOTE: Script files within *System M-1* must end in *.sh.[1]

The dummy UUT provided in the M-1 repository (sub folder `uut`) contains several bash scripts that generate tests very rapidly:

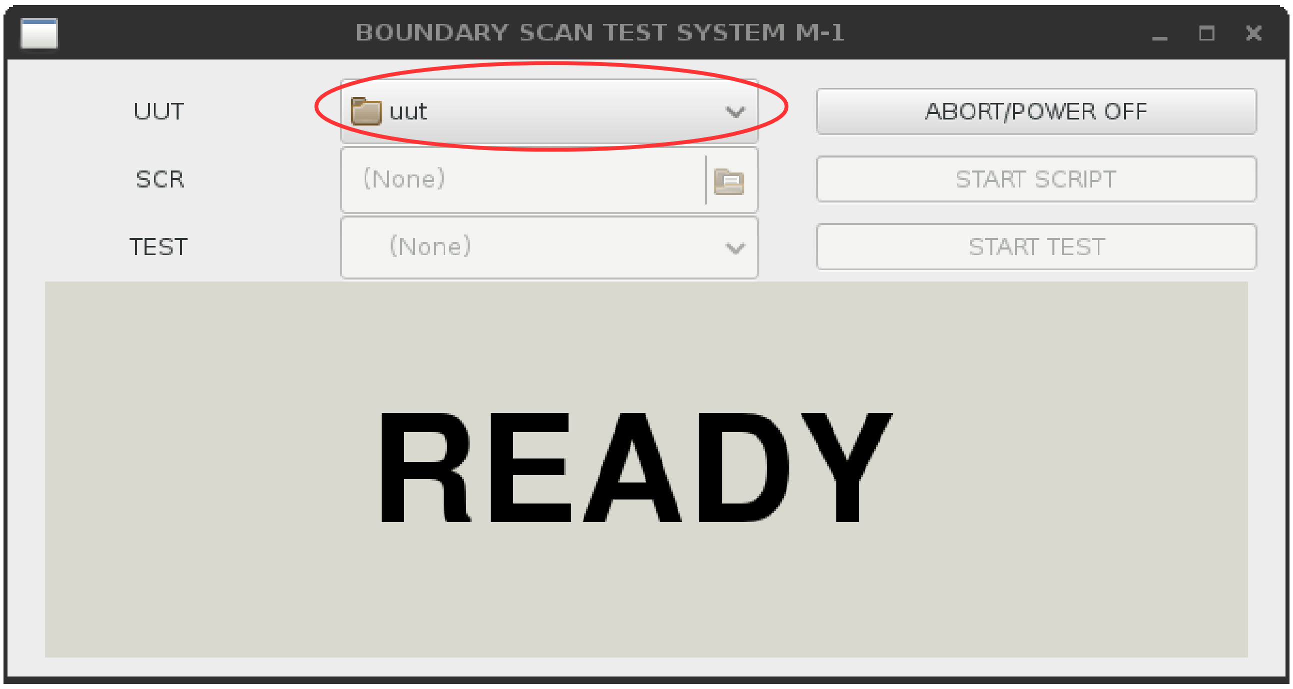
1. build_nets.sh
2. build_tests.sh
3. load_tests.sh
4. run_tests.sh

1 Linux and UNIX systems do not insist on this extension in general. The extension eases locating scripts.

14. Production Mode

It is highly recommended to use the graphical user interface when operating *System M-1* in an assembly line. The GUI is intentionally reduced to the most essential widgets required for board and system testing.

On start-up the GUI allows to select the UUT (or target) by clicking the button labeled with UUT (Screenshot 1).



Screenshot 1: Start-Up

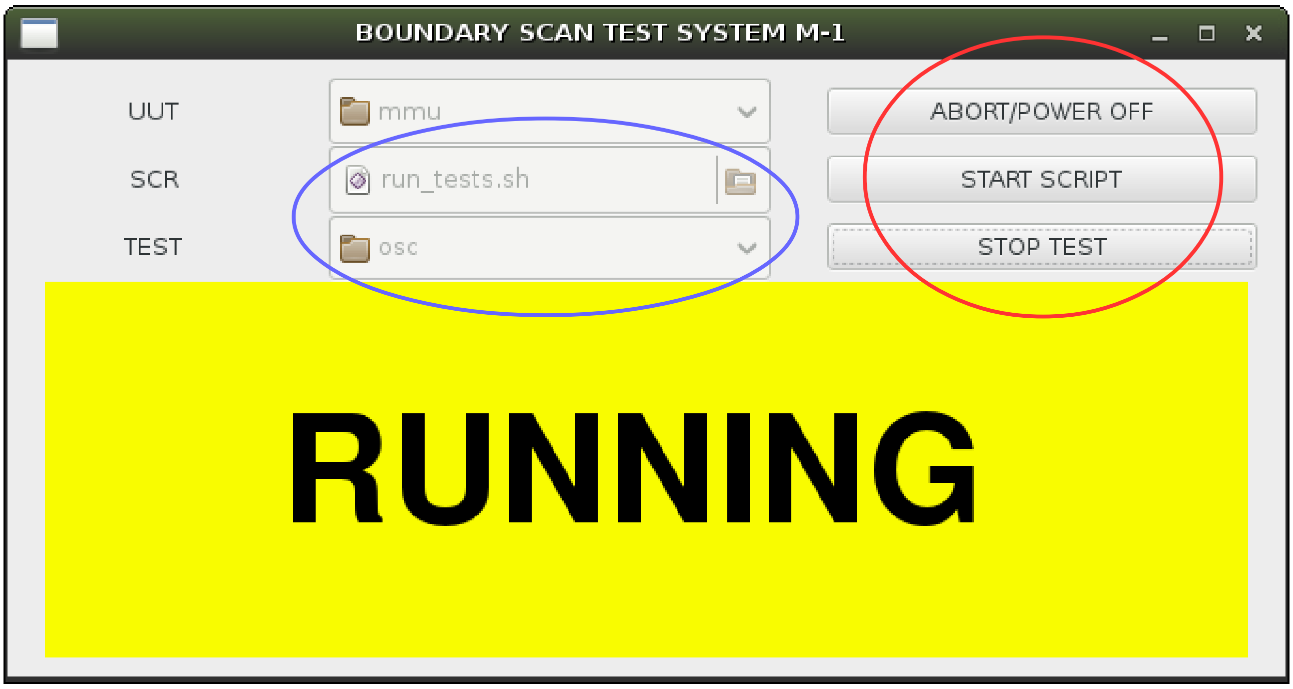
14.1. Test / Script Start

After setting the UUT, a script (which is a batch of tests or other actions) or a test can be selected and launched as shown in Screenshot 2. While a script or test is being executed, the status window shows a bright yellow/black "RUNNING".

14.2. Test / Script Abort

Aborting the script or test is possible by clicking "ABORT/POWER OFF", the "STOP SCRIPT" or "STOP TEST" button.

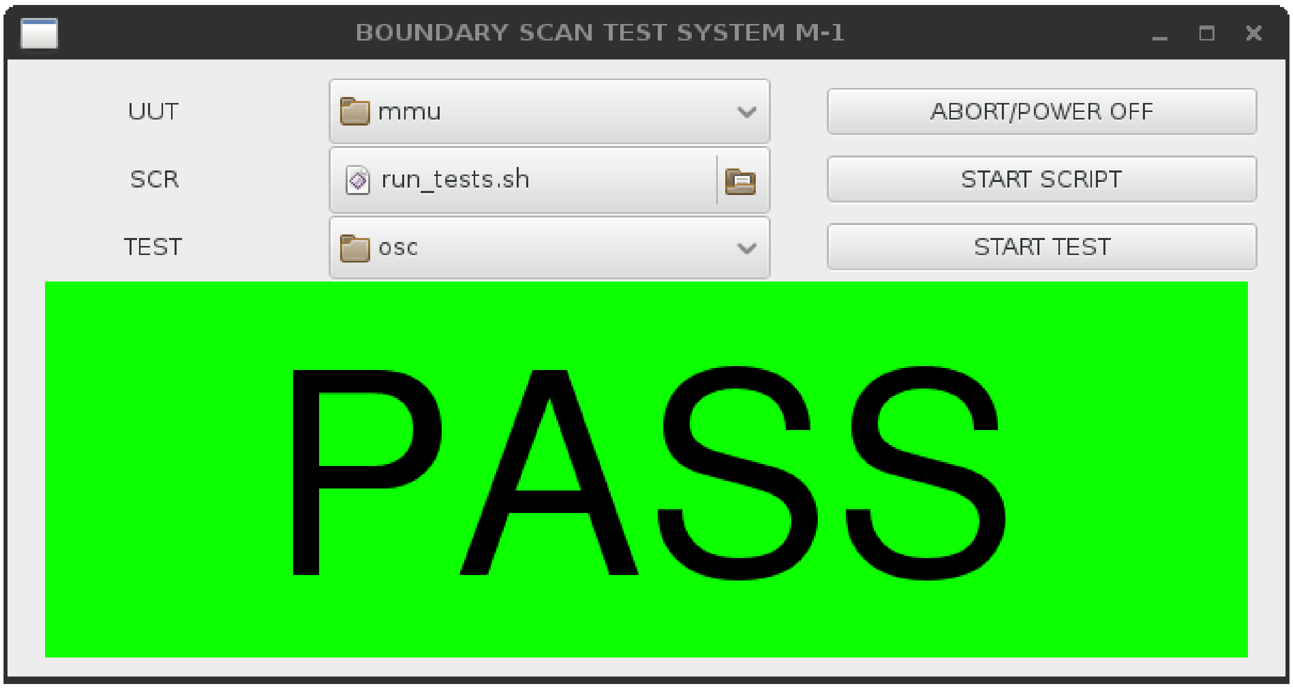
WARNING: It may take up to two seconds for the software to abort and shut down the UUT. The fastest abort can be forced by pressing the red "STOP" button at the front panel of the boundary scan controller.



Screenshot 2: Test/Script running

14.3. Test / Script PASS

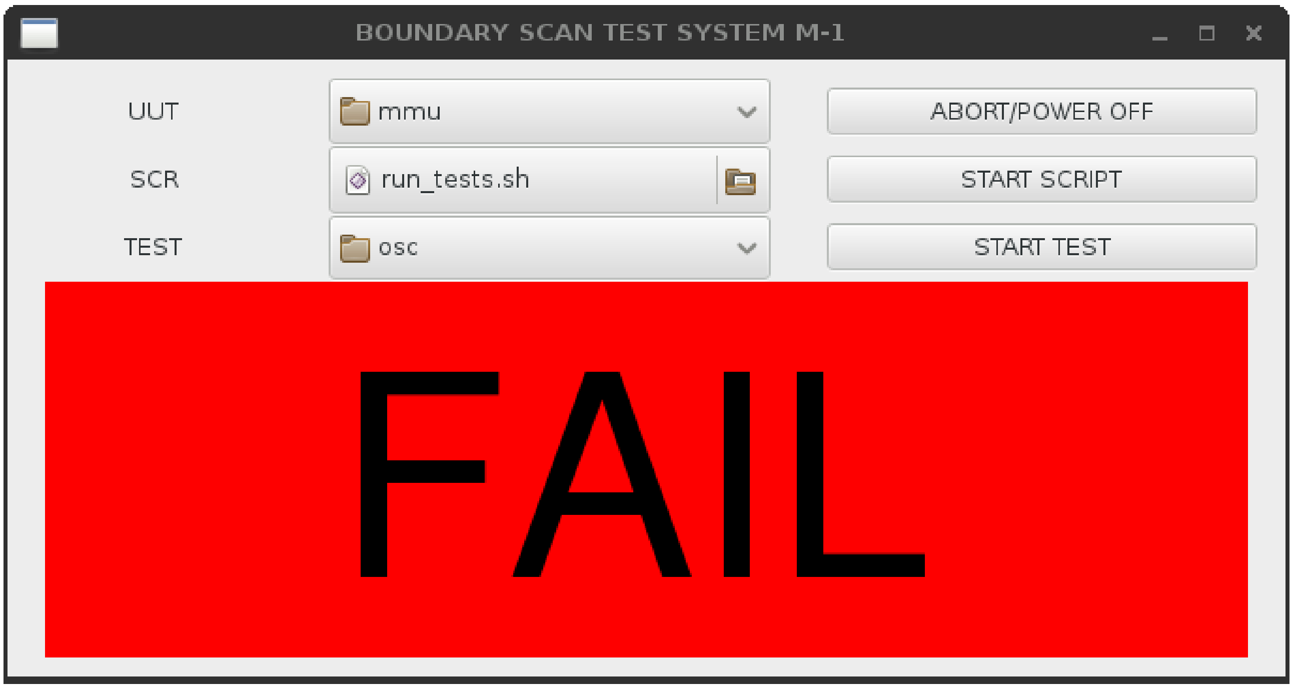
The most obvious result of a test or a script is the brigth green “PASS” status window.



Screenshot 3: Test/Script passed

14.4. Test / Script FAIL

A failed test or script is displayed by the alarming red “FAIL” readout. For production mode this display is fully sufficient to tell the bad from the good UUTs. Additionally, for repair and debug, the console outputs more detailed information on which net, device and pin had failed (see Text 5 page 45).



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: Fail Report

15. References

(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 at https://github.com/sysprv/demo-corporate-documentation-public/blob/master/Niklaus%20Wirth%20-%20A%20Plea%20for%20Lean%20Software.pdf

16. Disclaimer

This document is believed to be accurate and reliable. I do not assume responsibility for inaccuracies any errors which may appear in this document. I reserve the right to change it at any time without notice, and do not make any commitment to update the information contained herein.

I am neither liable for direct damages nor consequential damages resulting from the application of *System M-1*.

Mario Blunk

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