Automated Test System

# Objectives

The Automated Test System (ATS) is a common methodology used within companies to test products as part of an automated factory. It seems that the user base of sigrok and smuview is quite small. It seems that there is not good examples of how to use this in a commercial environment. The aim of the ATS code is to give examples of how these may be used in a commercial environment for doing common ATS functions.

The ATS functions should aim to provide:

* a template for carrying out automated tests
* consistent way of collecting system information
  + Client (who is this being performed for)
  + Operator (who is doing the test)
  + Device under test (what is being tested – part number and revision – serial number)
  + Test rig (what test rig did the testing – test rigs typically have their own part number and revision – also there may be several test rigs, so need a test rig serial number)
  + Use consistent naming > for dut and testrig (product,model,version,serialnumber) > all physical objects should have this naming convention
* consistent way of specifying the test setup
  + what devices are needed to be connected by the test
  + the system must be able to attach/re-attach to devices that are already connected regardless of current state
* consistent way of collecting test data
* consistent presentation of information in cvs format for analysis in spreadsheet or other data collection and analysis systems. Each output line must contain a consistent output format for ease of analysis
  + time stamp from start of execution
  + title of what the information is
  + data presentation in a consistent csv format
* easy way of skipping tests with
  + must perform list,
  + must skip list, and
  + do all keyword
* a test procedure methology that has a consistent interface and naming – multiple results per test, results with UCL and LCL controls, result value reported along with final PASS/FAIL/ERROR/SKIP
  + Test procedure (test) – a list of commands that performs some functions – and gets test data into a variable
  + Test examination (examination) – a command within a test – that identifies a variable to be examined in a particular way
    - Examination input – variable – the variable name to be examined (this would be used in the test procedure – usually something that has been read from test equipment
    - Examination input – LCL – lower control limit for the examination
    - Examination input – UCL – upper control limit for the examination
    - Examination output – value – the value of the variable
    - Examination output – result – the analysis of value against LCL and UCL, valid results are PASS/FAIL/ERROR/SKIP
      * PASS = the test performed correctly and the examination variable was correctly within the LCL/UCL bounds
      * FAIL = the test performed correctly and the examination variable was not within the LCL/UCL bounds
      * ERROR = the test was not able to perform correctly, some error condition occurred
      * SKIP = the operator used the skip system to skip this test and so not performed

# Needs

The ATS will need to communicate to custom devices that sigrok does not know about. This is usually a DUT that is being designed/tested. The commands to send to it are known, but custom/proprietory and not wanted to be part of the sigrok system. It is okay to assume that this will be a serial port. An easy way to get/send command strings to DUT/custom devices is needed. Just use standard python serial port?

The ATS will need some gui elements:

* Start test user button - colours
  + red (X) - instrument error / not found – or required user information not present – or if system has DUT detection, DUT is not detected – safety mechanisms tripped
  + green (1) - ready to go
  + grey (.) - test in progress
* stop button – colour
  + grey (.) - when instrument not running
  + red (0) - running a test – this is the stop
* separate start / stop locations is a common interface – reasoning:
  + multiple presses on the start location does not have a start/stop action
  + multiple presses on the stop location does not have an accidental re-start action
* Information zone / window – need to put all the text information about the test in this zone / client / operation / DUT information
* Test progress zone / window – need to put all the text information about the test progress in this zone
* Test graphical meters (nice to have) – be able to place graphical meters of certain parameters in a controlled way. Smuview has many graphical meters, but a custom display window does not look possible. I may want this test to show a particular format of a few elements in my custom way. How do I do that?

The gui request could be broken into two different ways.

1. As smuview seem to be fixed window information:
   1. A start/stop button view
   2. A way to have 2 (or more) text windows that a script can write to
2. If smuview has ways of making custom views:
   1. How to make a custom view
   2. Add a button element
   3. Add a text output element
   4. Add a graphical display element

Not too sure which way smuview currently is or where it is headed, but to make an ATS – these would be great

# Example ATS

Example of how an ats should behave

ats.statemachine has states "initialise", "ready", “starting”, "running", “user stop”, “reporting”, "exiting"

* power up at “initialise”
  + all the required test equipment drivers are accessible
  + make sure all the hardware is intialised to the defined controlled init state
  + init the gui (grey start / grey stop / display info)
  + once done > transition to ”ready”
* in “ready” we can 1. exit or 2. start (3. Reset?)
  + exit > exit the ats script
  + start > transition to “starting”:
  + reset > transition to “initialise” to force a re-init of the hardware (its nice to have a reset button – sometimes hardware gets in a weird state and a quick access Reset sometimes saves the day)
* in “starting” we transistion to “running”
  + grey the start button
  + red the stop button
  + collect the required operator and sn info
  + begin to run the required test scripts
* in “running” we can be interrupted by a user stop (abort) or end due to test script completion (normal)
  + User stop
    - pressing stop transistions to “user stop” where the system attempts to stop the current test and reset the hardware to a contolled state.
    - Once control is achieved "operator stop error" being added to the test report
    - The system then transistions to “reporting”
  + Test script completion
    - Ensure system is left in safe state
    - Transistion to “reporting”
* In reporting
  + Report totals of all the test tallying values – pass/fail/error/skipped
  + Generate output files (csv dumps)
  + Transistion to “ready”

**------------------------- END OF ATS OBJECTIVES -------------------------**