Sistemi Lente/Prism

A Manufacturing Test Framework

The (good) Problem…

- Startup/Small Company develops a product, builds a prototype…
  - Customer loves it, orders thousands, due in 3 months…
  - Time to build a test system…
- Can you afford/budget to outsource the development?
  - Did you write clean specs to transfer knowledge of your product to an outsource to get the job done (in time)?
  - Can your core developers support a 3rd party while they prepare for launch?
- Can you develop the test system yourself?
  - More software, another PCB design…
  - Most test systems are about as complicated as the product they test…
  - Databases, test look up, revision control, …

The Good Solution…

Sistemi Lente/Prism Test Platform

A Framework to Develop/Deploy Production Test Suites

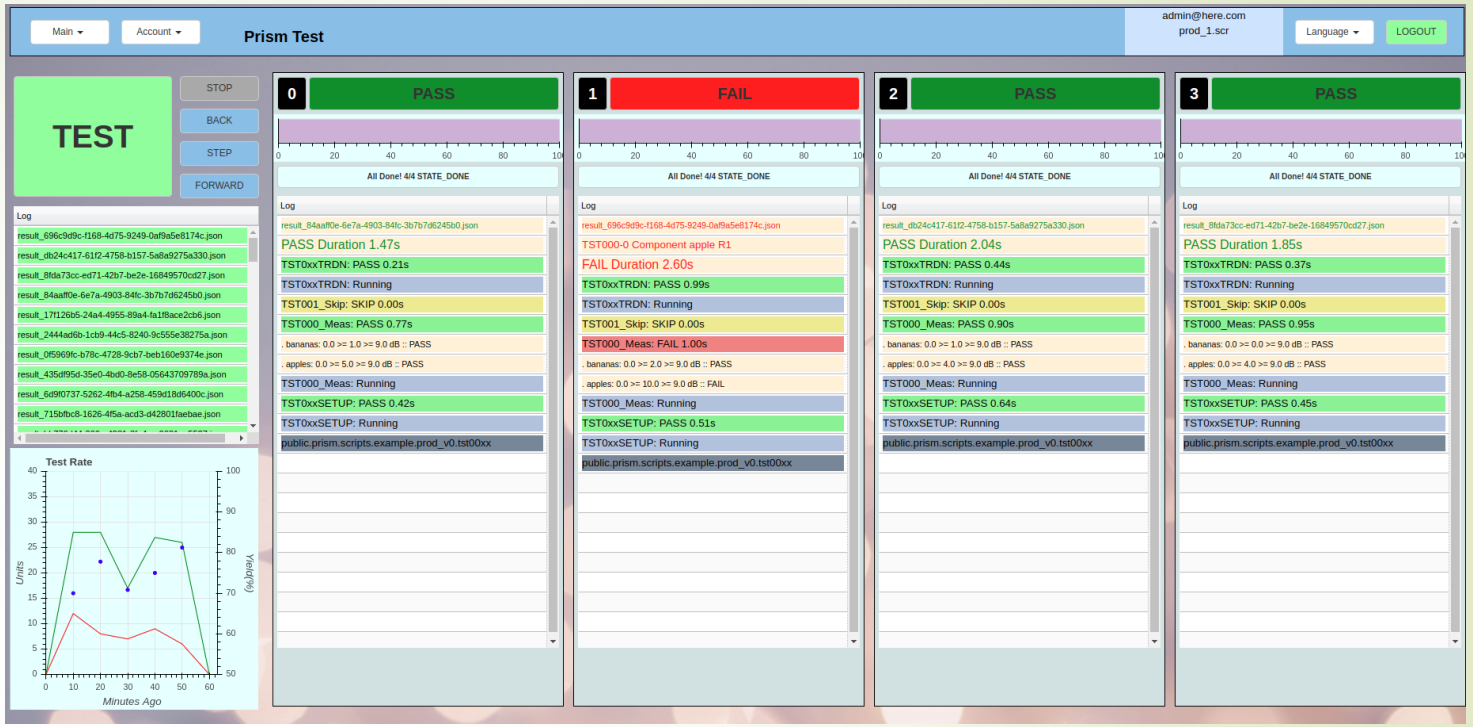
- Features
  - Graphical (web) User Interface
  - JSON style "Scripts" for Test Flow, Limits, etc
  - Tests programmed in Python
  - Production Monitoring Dashboard
  - Database and JSON Results
  - Scalability
  - Deployment Strategy and Version Control
  - Barcode Travellers for zero-effort Test Configuration
  - User Defined Production Tracking Variables
  - Security

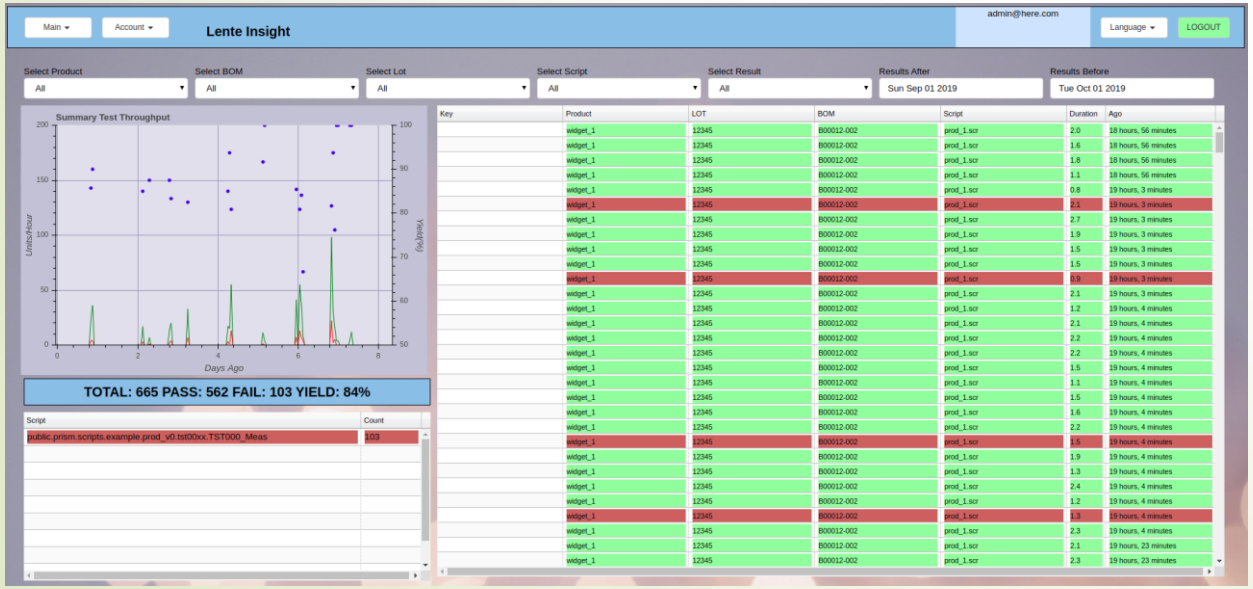
Estimated to **save 6-12 man-months of development** for a decent manufacturing test system that has comparable features

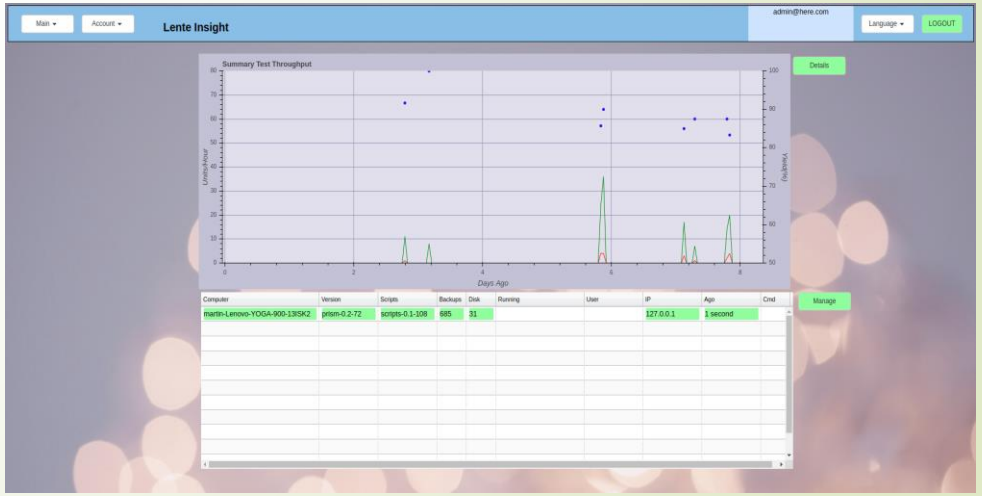
Sistemi Lente/Prism Test Platform

Graphical User Interface

- Color coded eye-catching views
- Get key metrics quickly
- Easy to train operators
- Supports Multiple Languages







Sistemi Lente/Prism Test Platform

JSON Style Test Scripts

- Drives the test bench
- Human readable
- Non-programmer can read this file and make changes
- Support for GUI driven variable substitution – see Appendix

```
{
 "info": {
    "product": "widget_1",
    "bom": "B00012-001",
    "lot": "201823",
    "location": "site-A"
  },
  "config": {
    "channel_hw_driver": ["tmi_scripts.prod_v0.drivers.tmi_fake"]
  },
  "tests": [
    {
      "module": "tmi_scripts.prod_v0.tst00xx",
      "options": {
        "fail_fast": false
      },
      "items": [
        {"id": "TST0xxSETUP",          "enable": true },
        {"id": "TST000_Meas",          "enable": true, "args": {"min": 0, "max": 10},
                                       "fail": [
                                         {"fid": "TST000-0", "msg": "Component apple R1"},
                                         {"fid": "TST000-1", "msg": "Component banana R1"}] },
        {"id": "TST0xxTRDN",           "enable": true }
      ]
    },
    {
      "module": "tmi_scripts.prod_v0.tst01xx",
      "options": {
        "fail_fast": false
      },
      "items": [
        {"id": "TST1xxSETUP", "enable": true },
        {"id": "TST100_Meas", "enable": true,  "args": {"min": 0, "max": 11},
                              "fail": [ {"fid": "TST100-0", "msg": "Component R1"} ] },
        {"id": "TST100_Meas", "enable": true,  "args": {"min": 0, "max": 12},
                              "fail": [ {"fid": "TST100-0", "msg": "Component R1"} ] },
        {"id": "TST1xxTRDN",  "enable": true }
      ]
    }
  ]
}
```

Sistemi Lente/Prism Test Platform

Tests programmed in Python

- Each test item from the JSON script (previous slide), is a python coded function
- APIs to make test driver code easy
  - Save any measurement
  - Get user input (buttons, text entry)
  - Set product keys (ex serial number)
  - Add logs
- Vast Python Module Ecosystem to draw upon
  - PyVISA - Test Instrument Control Library

```
def TST000_Meas(self):
    """ Measurement example, with multiple failure messages
    - example of taking multiple measurements, and sending as a list of results
    - if any test fails, this test item fails

        {"id": "TST000_Meas",     "enable": true, "args": {"min": 0, "max": 10},
                                  "fail": [ {"fid": "TST000-0", "msg": "Component apple R1"},
                                            {"fid": "TST000-1", "msg": "Component banana R1"}] },
    :return:
    """
    ctx = self.item_start()   # always first line of test

    time.sleep(self.DEMO_TIME_DELAY * random() * self.DEMO_TIME_RND_ENABLE)

    FAIL_APPLE   = 0  # indexes into the "fail" list, just for code readability
    FAIL_BANANNA = 1

    measurement_results = []  # list for all the coming measurements...

    # Apples measurement...
    _result, _bullet = ctx.record.measurement("apples",
                                              random(),
                                              ResultAPI.UNIT_DB,
                                              ctx.item.args.min,
                                              ctx.item.args.max)
    # if failed, there is a msg in script to attach to the record, for repair purposes
    if _result == ResultAPI.RECORD_RESULT_FAIL:
        msg = ctx.item.fail[FAIL_APPLE]
        ctx.record.fail_msg(msg)

    self.log_bullet(_bullet)
    measurement_results.append(_result)

    # Bananas measurement...
    _result, _bullet = ctx.record.measurement("bananas",
                                              randint(0, 10),
                                              ResultAP
                                              ctx.item
                                              ctx.item

    # if failed, there is a msg in script to attach to
    if _result == ResultAPI.RECORD_RESULT_FAIL:
        msg = ctx.item.fail[FAIL_BANANNA]
        ctx.record.fail_msg(msg)

    self.log_bullet(_bullet)
    measurement_results.append(_result)

    # Note that we can send a list of measurements
    self.item_end(item_result_state=measurement_result
```

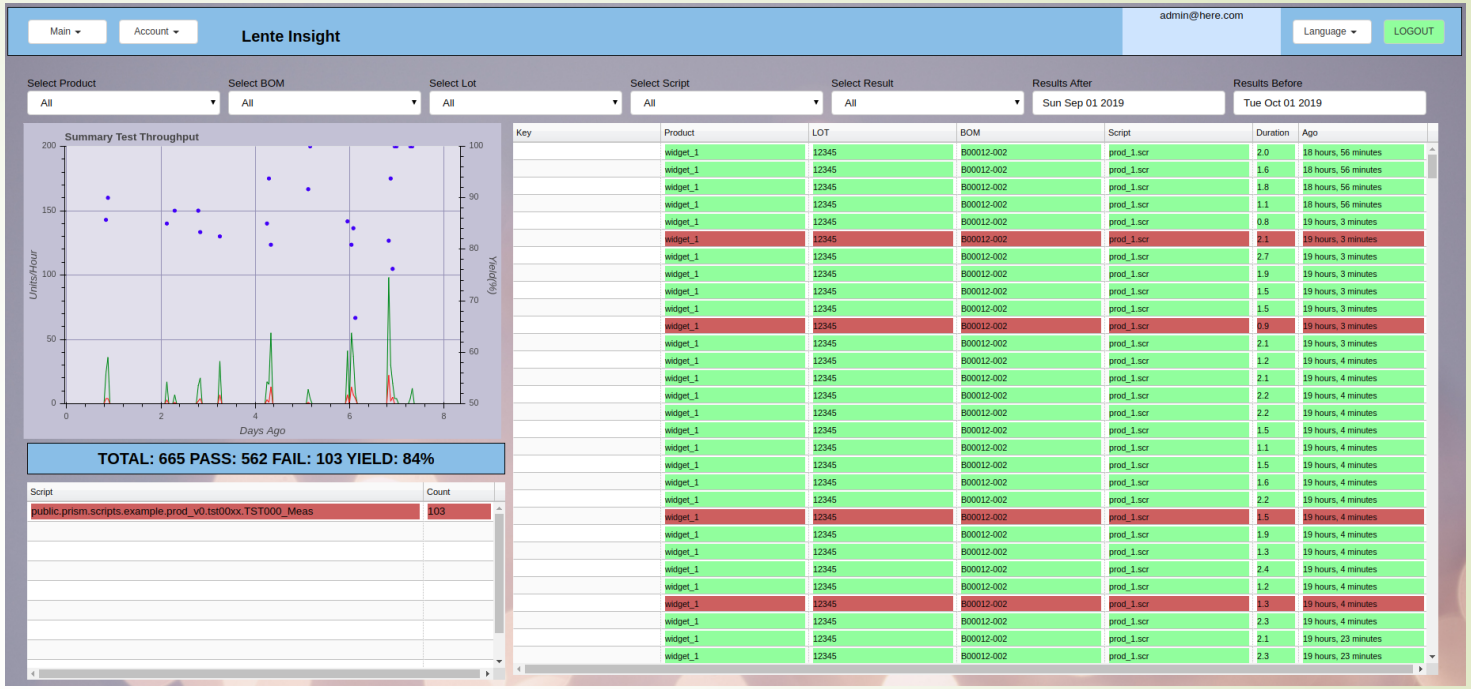
```
def TST008_TextInput(self):
    """ Text Input Box
    """
    ctx = self.item_start()   # always first line of test

    self.log_bullet("Please Enter Text!")

    user_text = self.input_textbox("Enter Some Text:", "change")
    if user_text["success"]:
        self.log_bullet("Text: {}".format(user_text["textbox"]))

        # qualify the text here,
        # make sure you don't timeout...

        _result = ResultAPI.RECORD_RESULT_PASS
    else:
        _result = ResultAPI.RECORD_RESULT_FAIL
        self.log_bullet(user_text.get("err", "UNKNOWN ERROR"))

    self.item_end(_result)  # always last line of test
```

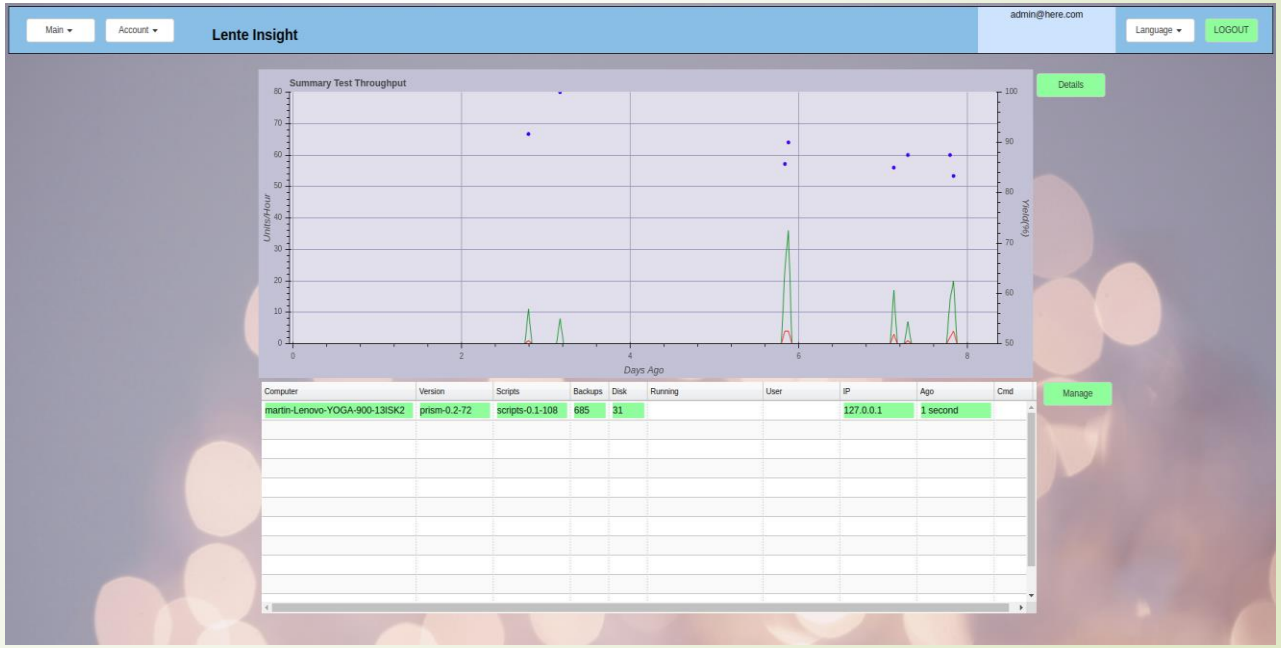
Sistemi Lente/Prism Test Platform

Production Monitoring Dashboard

- Lente
  - Realtime results
  - Can be on or off site
  - Transfers results into Postgres Database
  - Shows Prism Test Station(s) status
  - Manage Users and Scripts deployed

  - Select Filters to drill down to specific results





Sistemi Lente/Prism Test Platform

Database and JSON Results

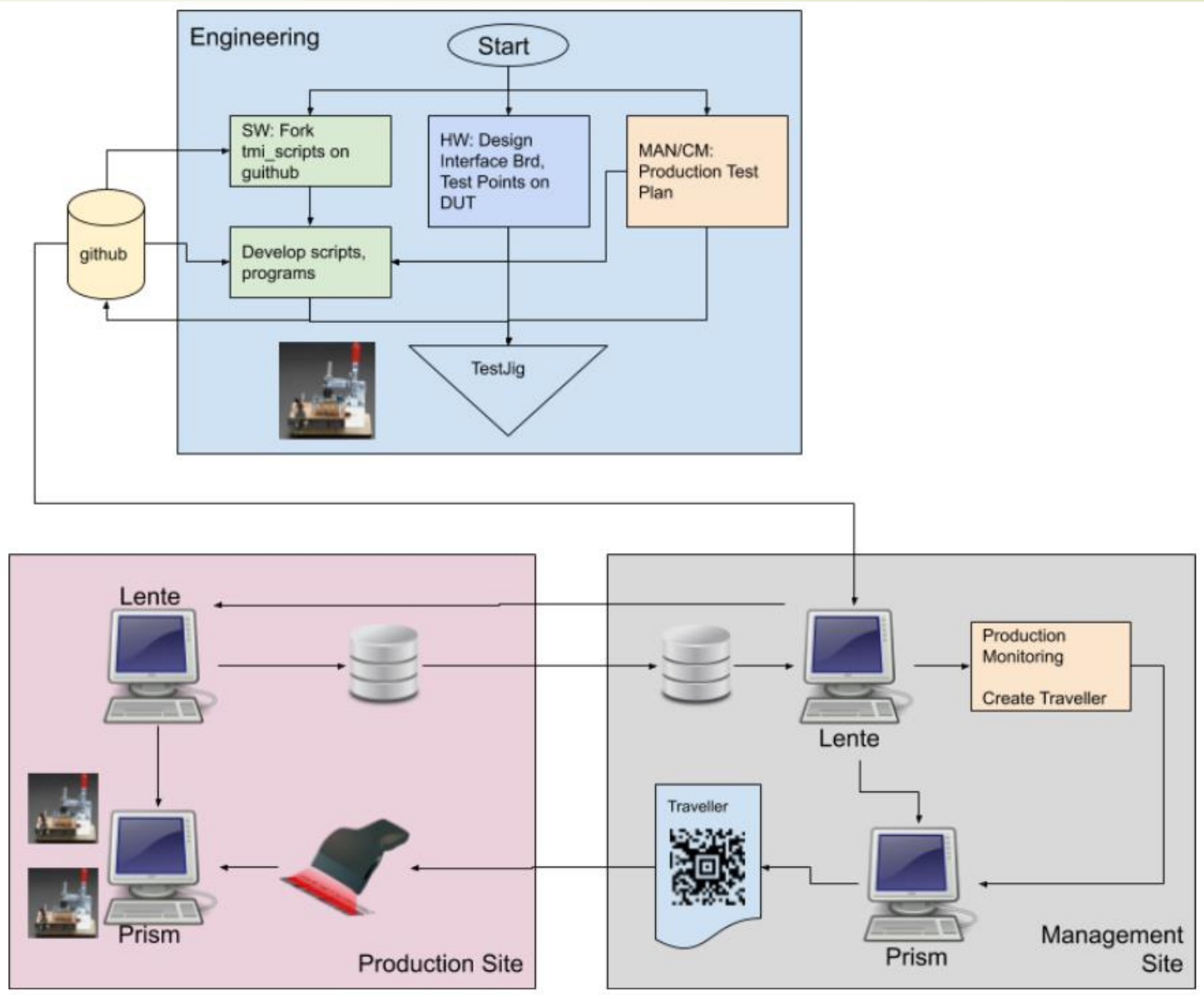
- Lente
  - Backend "normalized" SQL Database
    - All test results stored in a consistent way to make queries easier
  - Postgres
    - Secure, scalable, cloud options
    - JSON BLOB data


```
"result": {
  "meta": {
    "channel": 0,
    "result": "FAIL",
    "version": "TBD-framework version",
    "start": "2018-07-09T22:46:20.424386",
    "end": "2018-07-09T22:46:45.329920",
    "hostname": [
      "Windows",
      "DESKTOP-O6AMGKM",
      "10.0.17134",
      "AMD64",
      "Intel64 Family 6 Model 58 Stepping 9, GenuineIntel"
    ],
    "script": null
  },
  "keys": {
    "serial_num": 12345,
    "ruid": "0dc26c9a-909c-4df3-8c91-bfbe856d5ba2"
  },
  "info": {},
  "config": {},
  "tests": [
    {
      "name": "tests.example.example1.SETUP",
      "result": "PASS",        "timestamp_start": 1531176380.44,
      "timestamp_end": 1531176381.44,
      "measurements": []
    },
    {
      "name": "tests.example.example1.TST000",
      "result": "PASS",
      "timestamp_start": 1531176381.45,
      "timestamp_end": 1531176383.46,
      "measurements": [
        {
          "name": "tests.example.example1.TST000.apples",
          "min": 0,
          "max": 2,
          "value": 0.5,
          "unit": "dB",
          "pass": "PASS"
        },
        {
          "name": "tests.example.example1.TST000.banannas",
          "min": 0,
          "max": 2,
          "value": 1.5,
          "unit": "dB",
          "pass": "PASS"
        }
      ]
```

Sistemi Lente/Prism Test Platform

Deployment and Version Control

- A thought-out flow between Engineering, Production and Operations

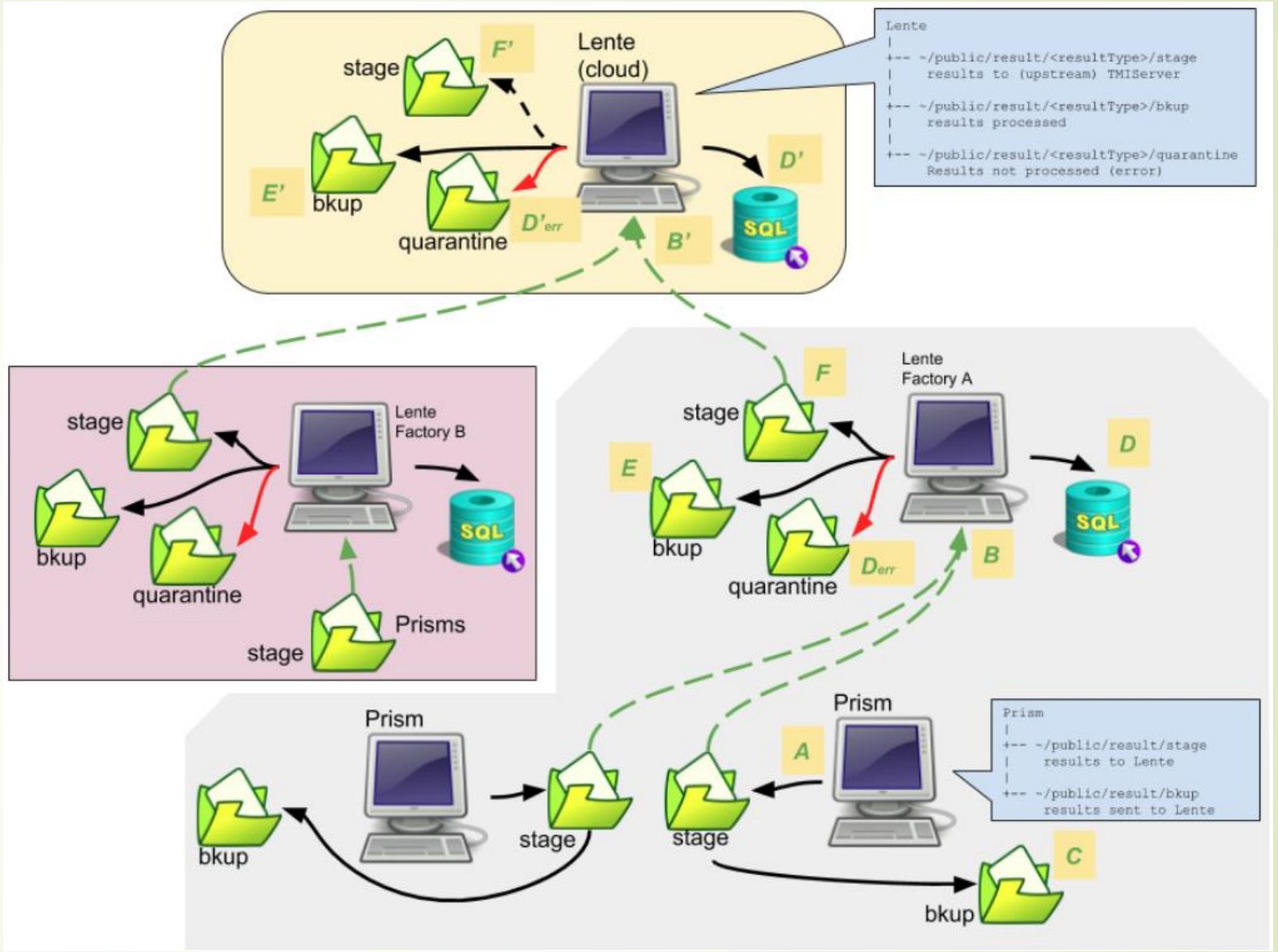


Sistemi Lente/Prism Test Platform

Deployment and Version Control

- Multiple Sites
- Pyramid structure
- Results backed up at every level

- Prism DOESN'T need Lente to operate. Results will be queued up until a Lente Server is available



Sistemi Lente/Prism Test Platform

Traveler

- Automates Test Configuration
  - No Manual entry
  - Scan and Go
- User Defined Production Tracking is encoded into the barcode
- Barcode is encrypted

Sistemi Lente/Prism Test Platform

Security

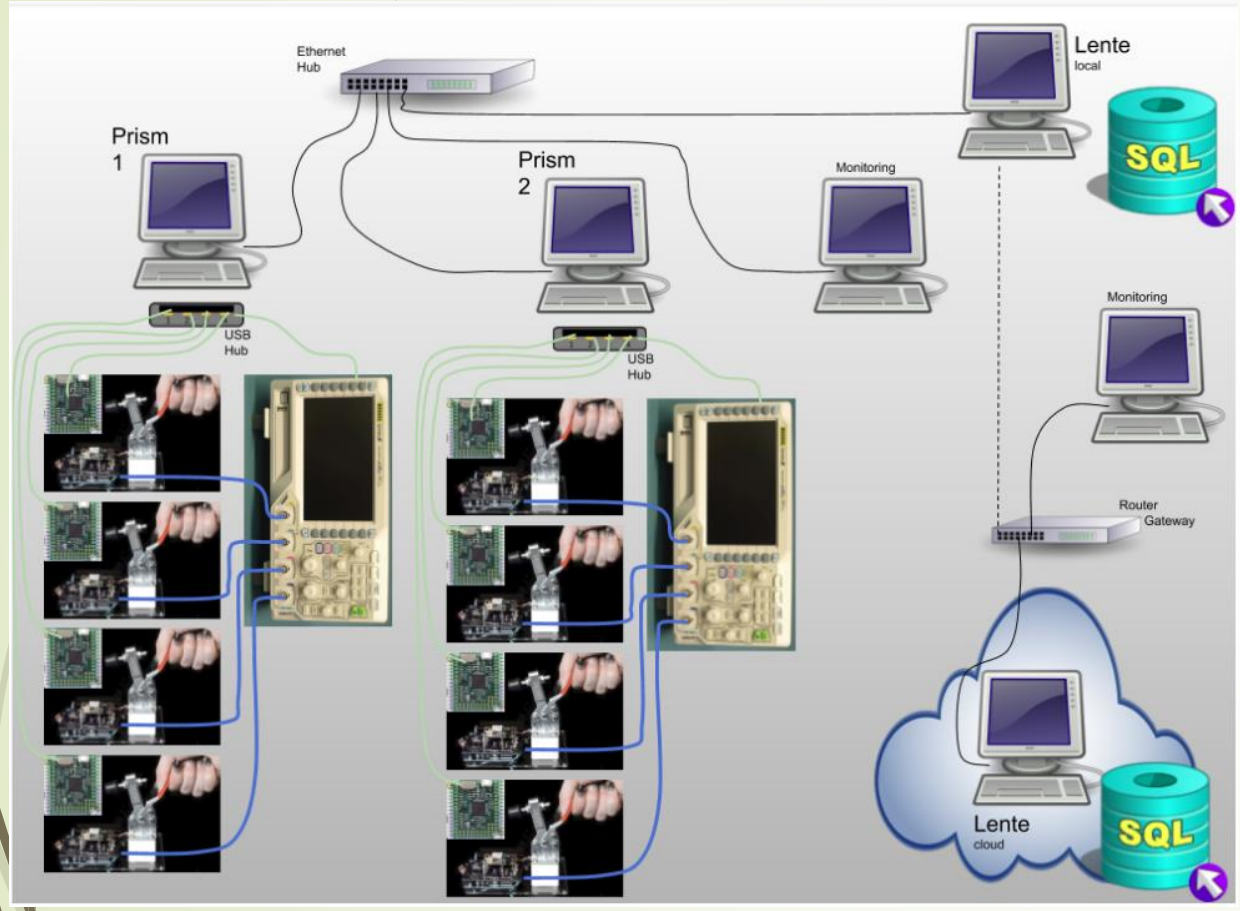
- Stations use Linux file/user security
  - Lente/Prism run as Docker containers, and run automatically when PC is booted
  - Lente/Prism are hosted in the Google Chrome browser
  - Scripts, Configuration Files, Results, etc, are not accessible by an operator (linux) login account
  - Scripts are also additionally protected by an encryption manifest (files can be read, but not changed)
- Results are optionally encrypted
- User Roles allow access to application functions

Sistemi Lente/Prism Test Platform

Appendix

Additional Notes

Sistemi Lente/Prism



Provides a framework to develop production test suites

Prism

- Runs Python (3.6) test code
- Easy API for collecting results, setting Pass/Fail
- Executes test code driven by JSON "script"
  - Human readable, a non-programmer can make changes

Lente

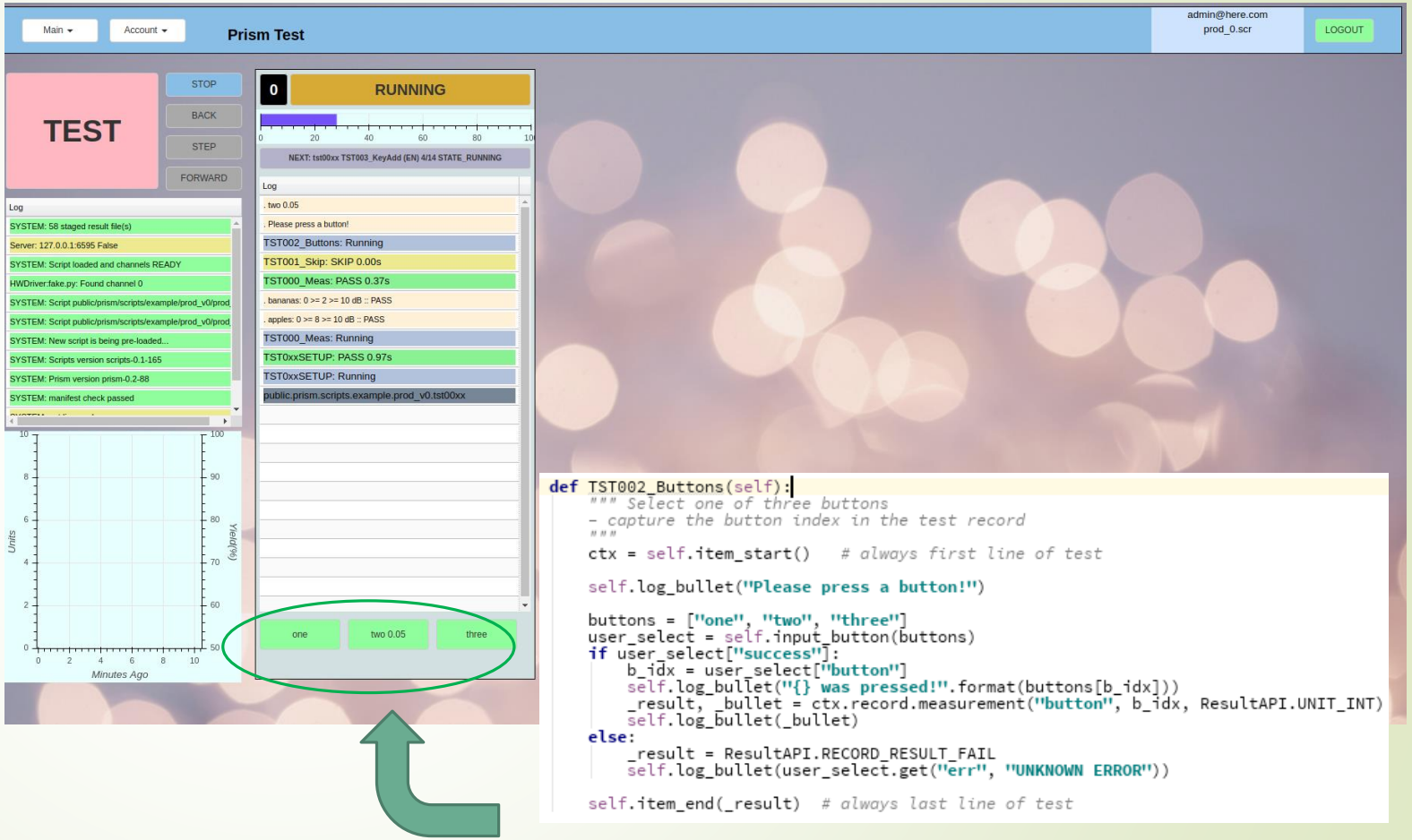
- Collects results from Prism stations
- Stores results in a SQL Database (extracts the JSON)
- Result traceability
- Dashboards

You worry about test code – Lente/Prism handles EVERYTHING else!

Sistemi Lente/Prism Test Platform

User Defined Buttons

- For the case when testing required operator to input a choice

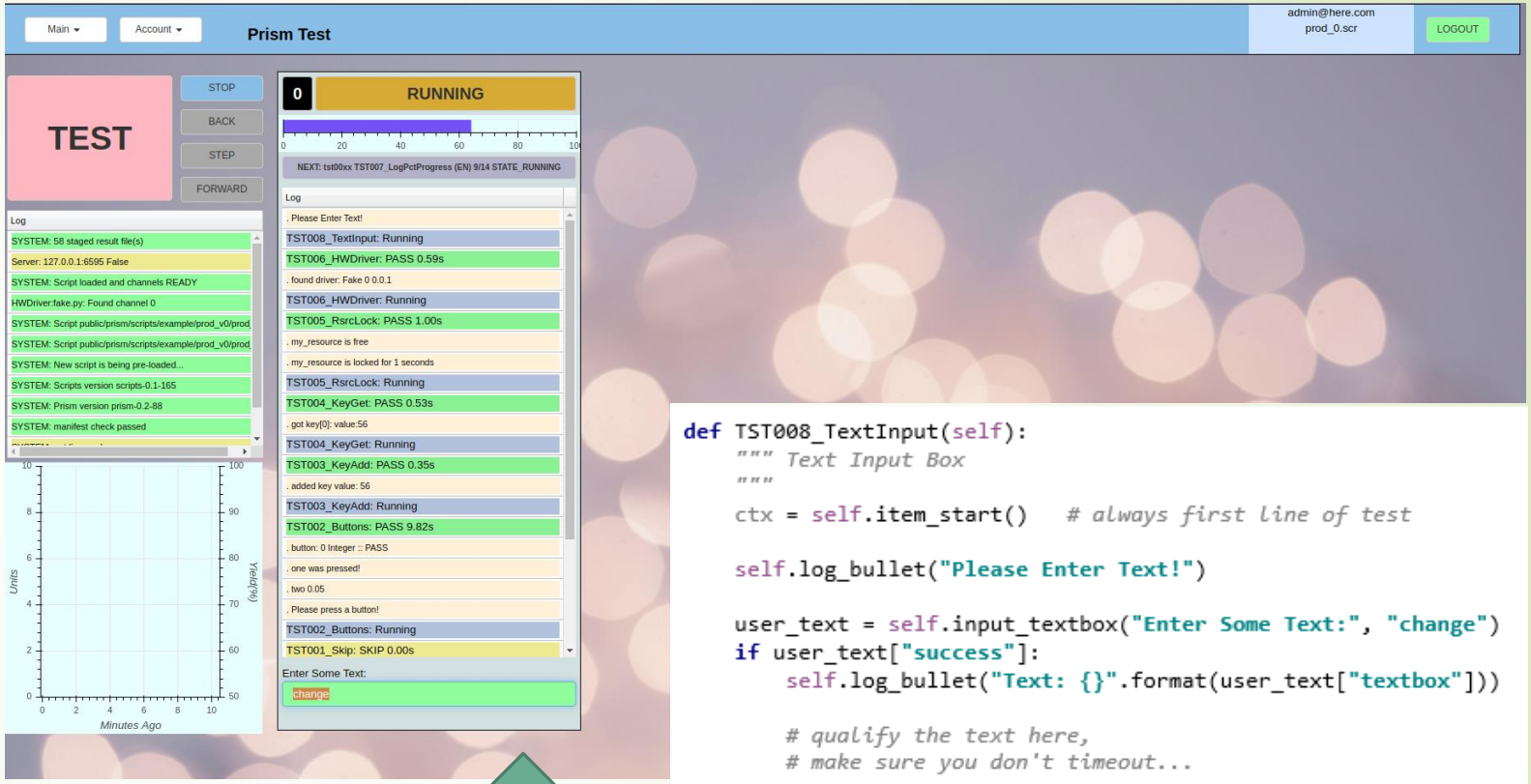



```
def TST002_Buttons(self):
    """ Select one of three buttons
    - capture the button index in the test record
    """
    ctx = self.item_start()   # always first line of test

    self.log_bullet("Please press a button!")

    buttons = ["one", "two", "three"]
    user_select = self.input_button(buttons)
    if user_select["success"]:
        b_idx = user_select["button"]
        self.log_bullet("{} was pressed!".format(buttons[b_idx]))
        _result, _bullet = ctx.record.measurement("button", b_idx, ResultAPI.UNIT_INT)
        self.log_bullet(_bullet)
    else:
        _result = ResultAPI.RECORD_RESULT_FAIL
        self.log_bullet(user_select.get("err", "UNKNOWN ERROR"))

    self.item_end(_result)  # always last line of test
```

Sistemi Lente/Prism Test Platform

Test View Text Entry

- NOTE: Text Entry not meant to be done by hand.
- Barcode Scanner input

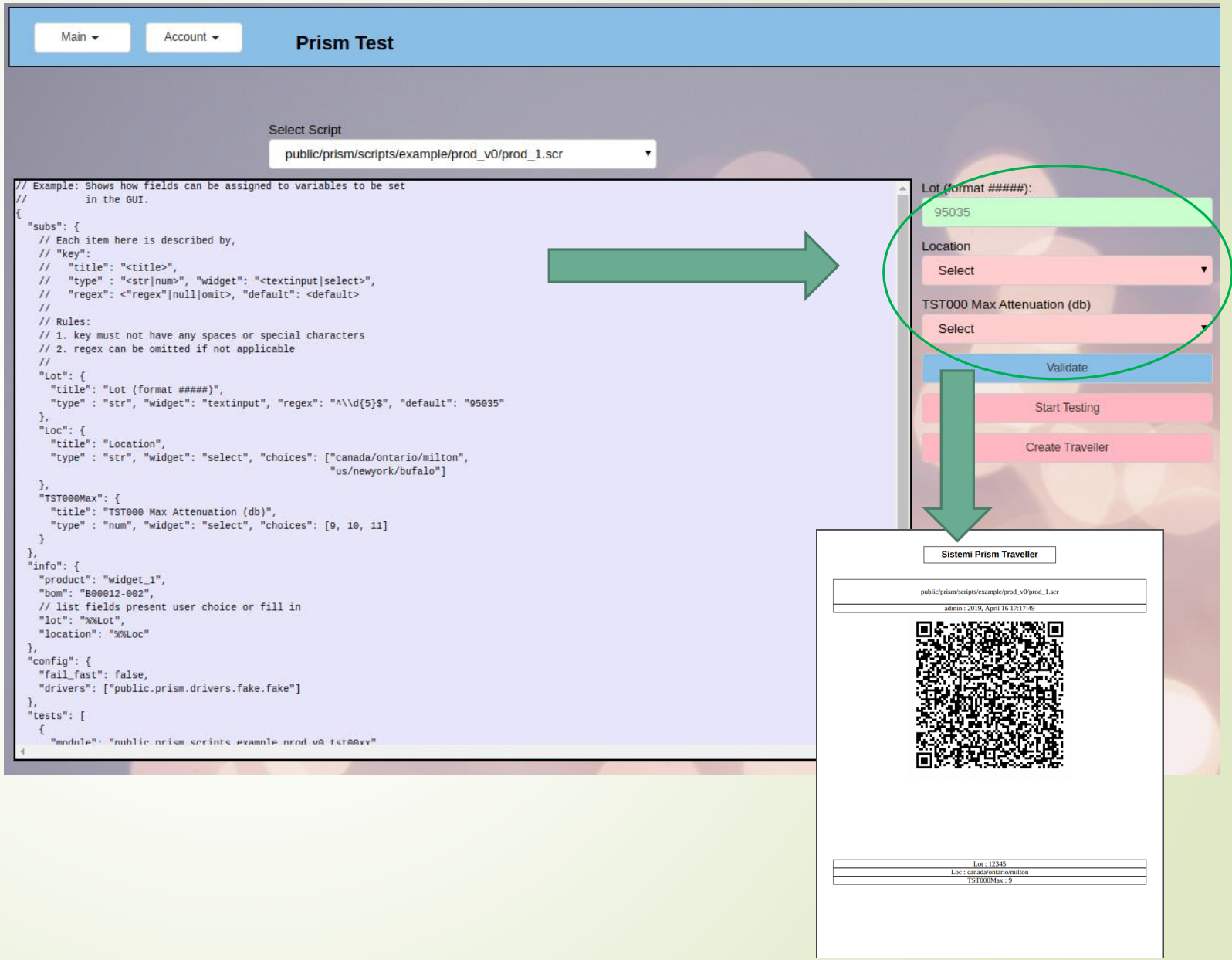



```
def TST008_TextInput(self):
    """ Text Input Box
    """
    ctx = self.item_start()   # always first line of test

    self.log_bullet("Please Enter Text!")

    user_text = self.input_textbox("Enter Some Text:", "change")
    if user_text["success"]:
        self.log_bullet("Text: {}".format(user_text["textbox"]))

        # qualify the text here,
        # make sure you don't timeout...

        _result = ResultAPI.RECORD_RESULT_PASS
    else:
        _result = ResultAPI.RECORD_RESULT_FAIL
        self.log_bullet(user_text.get("err", "UNKNOWN ERROR"))

    self.item_end(_result)  # always last line of test
```

Sistemi Lente/Prism Test Platform

JSON Style Test Scripts
(showing Variable Substitution)

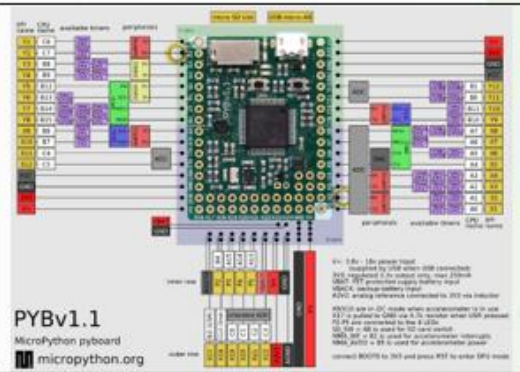
- JSON Script defines variable substitution
  - Drop Down Selection
  - Text Entry validated by Regex
- For example,
  - Lot Number
  - Location
  - Measurement limits
- Traveler can be created from User input(s) for hands free Production floor configuration



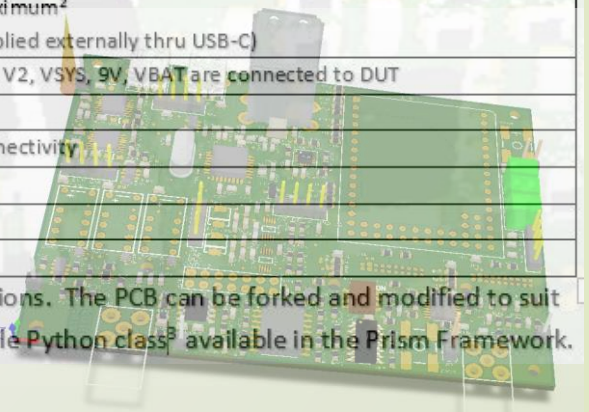
DUT Design for Testability

- Add test points for the bed of nails jig
- Understand the **IBA01** and/or MicroPython Board IO pin capabilities
  - Or create your own "interface board"
  - Create PCB to interface MicroPython Board to the DUT
  - Determine what external test equipment is required to test things that can't be tested **IBA01**
- Write (Python 3.6) Software within this framework…
  - Results will be normalized, stored in SQL DB
  - Logging
  - Results Dashboard

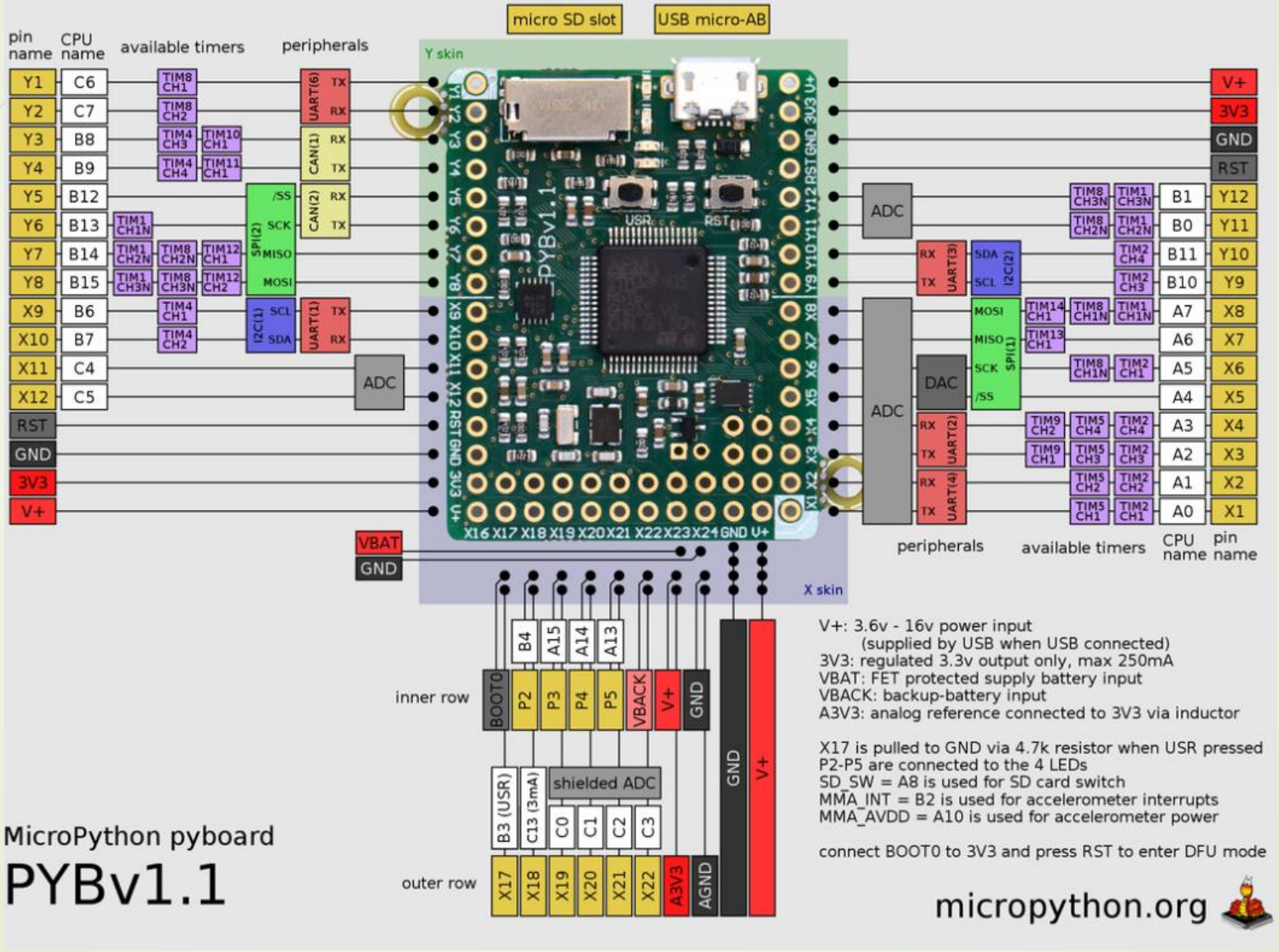
IBA01 Interface Board & Jig

Features[1]	
Embedded MicroPython Board • STM32F405 • (12bit)ADCs, DAC, GPIOs, UARTs PWMs, Timers, I2C, SPI • MicroSD Slot • Note some resources used by the IBA01, see schematic	
Two Programmable DC Supplies	V1 (TPS7A2501) • 1650-4500mV, 50mV Steps, 500mA Maximum[2] • Current measurement, ±100µA, 100mA Max V2 (TPS7A7200) • 500-3500mV, 50mV Steps, 500mA Maximum[2] • Current measurement, ±100µA, 100mA Max
Programable Battery Emulator	VBAT (LT1118) • Source and Sink Current to 800mA Maximum[2] • 1650-4500mV, 50mV Steps • Current measurement, ±1mA, 500mA Max
USB Embedded HUB	Two free USB (2.1) ports
USB Virtual Serial Port	Based FT2232
USB JTAG Programmer	Based FT2232
16Bit ADC	Two inputs, based on ADS1115
Two non-programmable Supplies	• 9V, 500mA Maximum[2] • 5V (VSYS) (Supplied externally thru USB-C)
DUT Supply Connect Relays	Relays control when V1, V2, VSYS, 9V, VBAT are connected to DUT
LoRa Module	RF Solutions RFM95W
Arduino Nano Slot	For WiFi/Bluetooth Connectivity
Digital Resistor	Based on TPL0102
Buffer Amplifier	Based on LTC6090
Level Translator	Based on TXS0104

The IBA01 PCB provides a prototype for all the above functions. The PCB can be forked and modified to suit specific DUT needs. All functions are available through simple Python class[3] available in the Prism Framework.

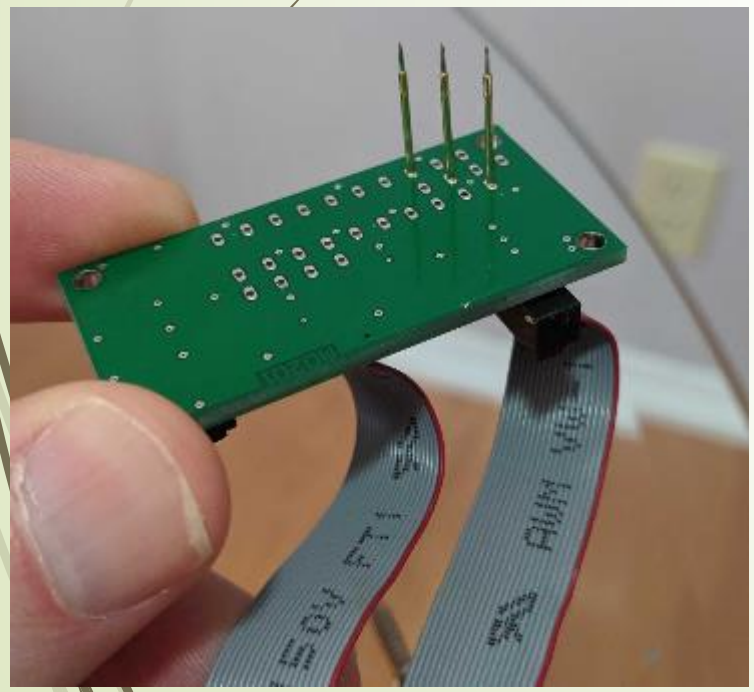


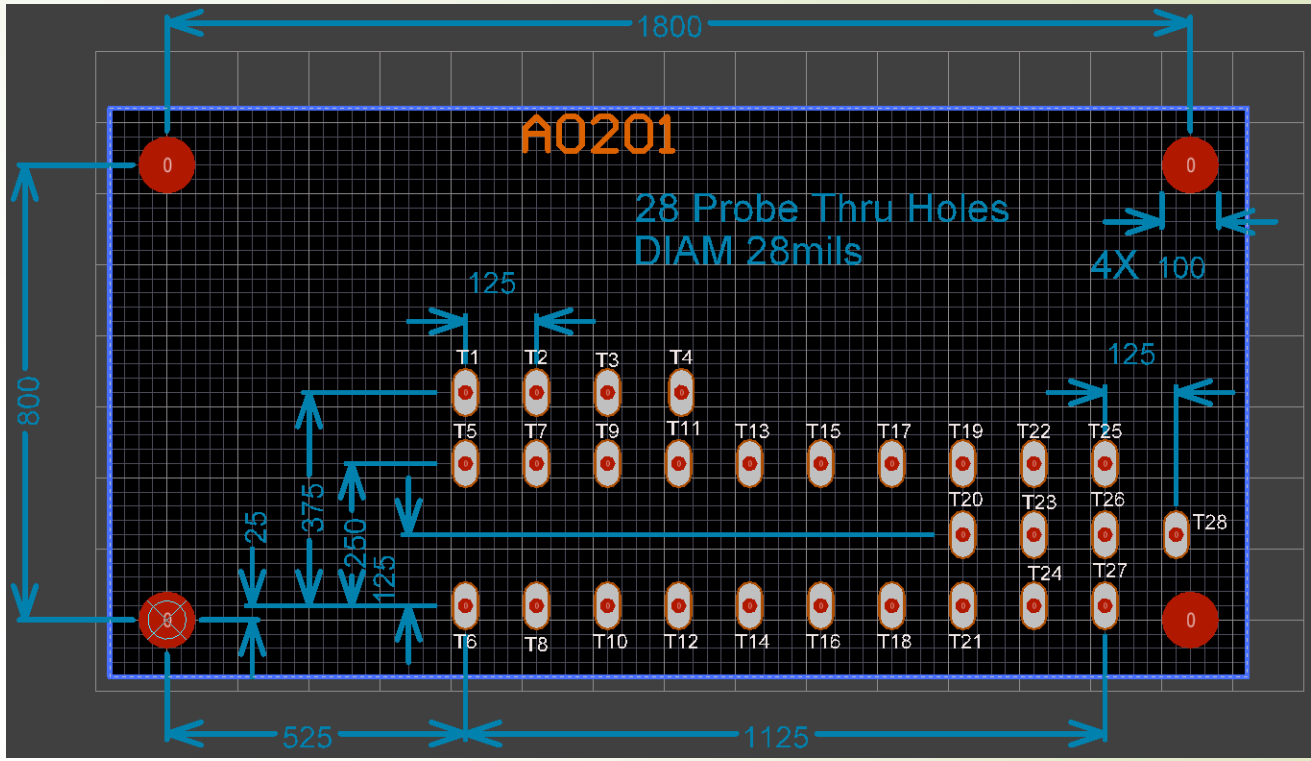
IBA01 Embedded PyBoard 1.1



IBA01 Probe Interface Board v1.x

- Regardless of your PCB design, if you place test pads to **use any subset** of the 28 pads that are in this arrangement – then you can use the already designed **IBA01**
- Any of the test points can be connected to the **IBA01** which can make ADC measurements, GPIO, JTAG, UART, I2C and other functions





Your exact test point requirements should be described and checked to see if they can be satisfied.

The IBA01 does have expansion bus where you can add a small PCB to add functionality

The IBA01 Interface Board is available on CircuitMaker as a project that can be forked, modified for your requirements