



# TMI System

A Manufacturing Test Framework

Ver09



# The 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?
  - Can your core developers support a 3rd party while they prepare for launch?
  - Will your customer tolerate you blaming a 3<sup>rd</sup> party test house if the order is late, or hasn't been completely tested?
- 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, ...



# TMI System for In-House Test Development

- In-House development
  - Core developers are present and engaged
    - Quicker answers to problems
    - Quicker changes to support production test
  - Technical details are know
    - No need to write detailed specs for external consumption
  - Transparent Scheduling

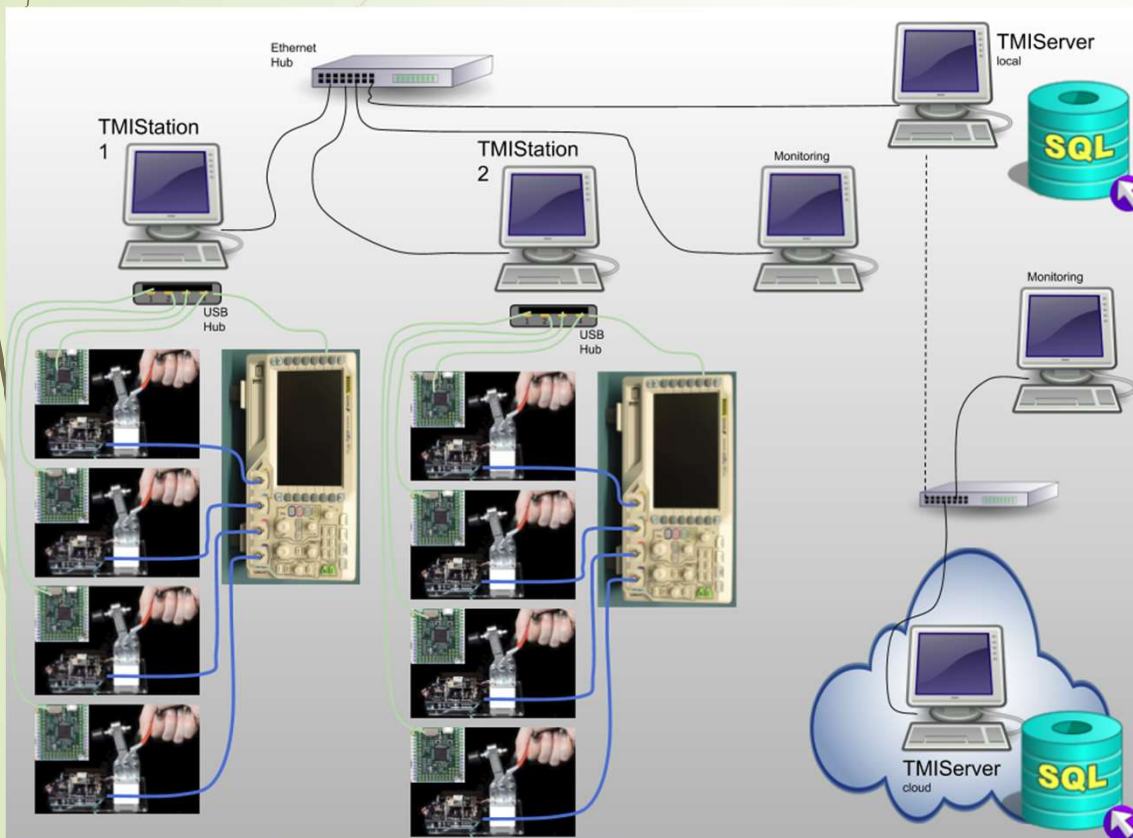


# TMI System for In-House Test Development

- Provides a framework for a HW/SW Test Engineer to start developing a test system WITHOUT, worrying about the “plumbing”...
  - User Interface
  - Programming Language
  - Production Monitoring
  - Database
  - Version Control
  - Scalable
  - Deployment
  - ...

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

# TMI System



- Provides a framework to develop production (or Engineering) test suites
- TMISStation
  - 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
- TMIServer
  - Collects results from TMISStations
  - Stores results in a SQL Database (extracts the JSON)
  - Result traceability

**You worry about test code – TMI handles EVERYTHING else!**

# TMI System Features

- ▶ JSON Test Scripts
  - ▶ Human readable, enable/disable tests, change limits
  - ▶ Non-programmer can make changes
- ▶ JSON Results
  - ▶ Human readable, easy to post process
- ▶ Result SQL Database
  - ▶ Dashboards and queries
- ▶ Python Codebase
  - ▶ Popular/easy programming
  - ▶ Multi-threaded for concurrency
- ▶ Traceability
  - ▶ Capture serial numbers, lot numbers, and any other identifier information from the DUT
  - ▶ All these identifiers go into SQL DB for query later
- ▶ Results stored in postgres SQL database which can be located anywhere (local, cloud, etc.)

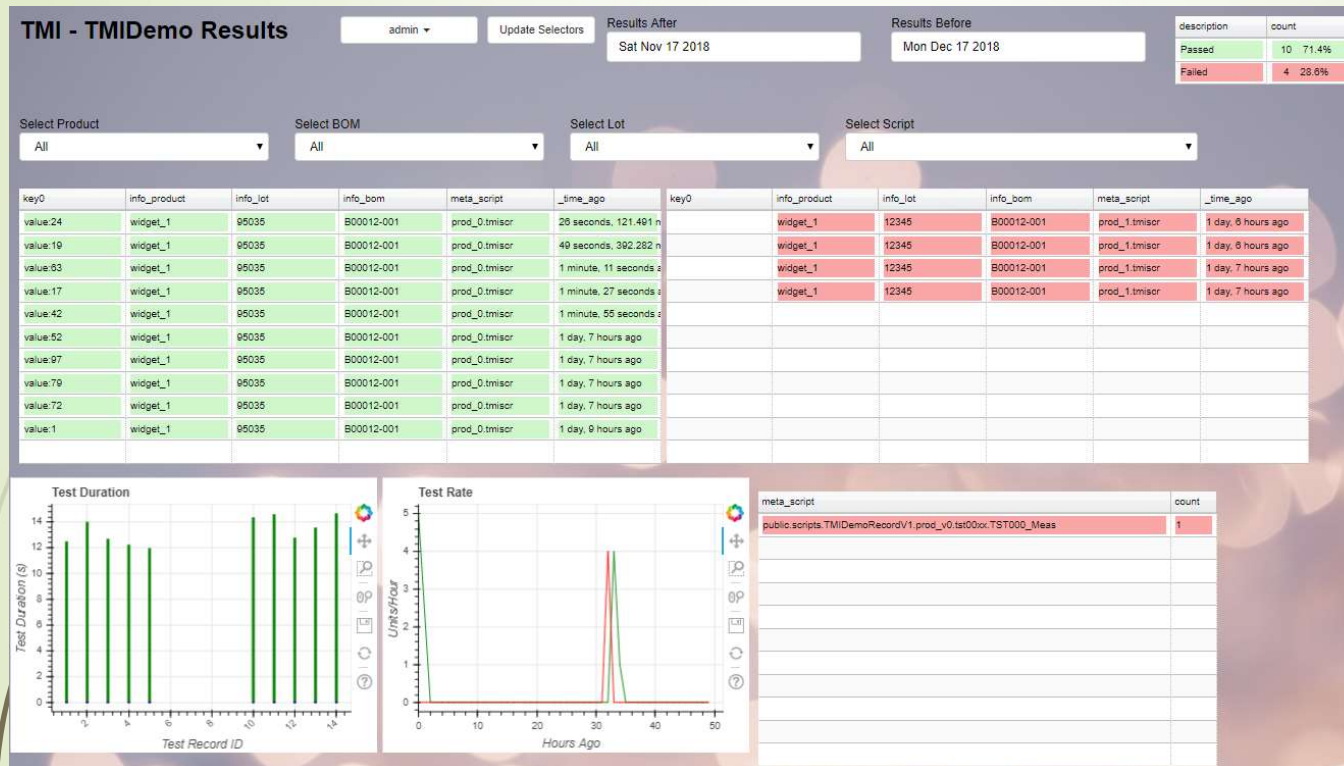
# GUI: Testing Panel



- 4 Channels are shown
- Each channel is an independent thread



# Result Server Dashboard



- Graphs to show trends
  - Test Duration and Pass/Fail Hourly Rate

- Top Row selectors to sort data
  - Can view data from a single LOT#
- Select row of Pass/Fail tables to bring up the test record details
- Summary Tables
  - Results reflect Selector settings
  - Pass/Fail Counts
  - Top failed tests and counts



# JSON Test Scripts

- Drives the test bench
- Human readable
- Each test item as an "id", which corresponds to python function that implements the test
- Non-programmer can read this file and make changes

```
{
  "info": {
    "product": "widget_1",
    "bom": "B00012-001",
    "lot": "201823",
    "location": "site-A"
  },
  "config": {
    "result_handler": "TMIDemoRecordV1",
    "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,
          "args": {"min": 0, "max": 10},
          "fail": [ {"fid": "TST000-0", "msg": "Component apple R1"}, {"fid": "TST000-1", "msg": "Component banana R1"} ]
        },
        {
          "id": "TST0xxMeas",
          "enable": true,
          "args": {"min": 0, "max": 12},
          "fail": [ {"fid": "TST100-0", "msg": "Component R1"} ]
        },
        {
          "id": "TST0xxTRDN",
          "enable": true
        }
      ]
    },
    {
      "module": "tmi_scripts.prod_v0.tst01xx",
      "options": {
        "fail_fast": false
      },
      "items": [
        {
          "id": "TST1xxSETUP",
          "enable": true,
          "args": {"min": 0, "max": 11},
          "fail": [ {"fid": "TST100-0", "msg": "Component R1"} ]
        },
        {
          "id": "TST1xxMeas",
          "enable": true,
          "args": {"min": 0, "max": 12},
          "fail": [ {"fid": "TST100-0", "msg": "Component R1"} ]
        },
        {
          "id": "TST1xxTRDN",
          "enable": true
        }
      ]
    }
  ]
}
```

# Python Test Code

- 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
- NOTE: Not shown in the code snippet is code related to controlling your hardware to make measurements.

```
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)

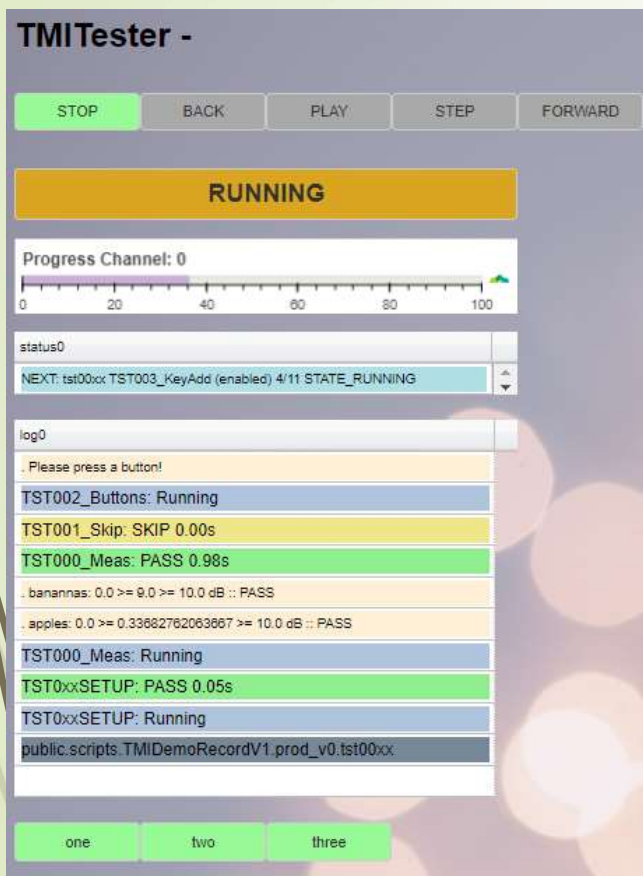
    # Bananas measurement...
    _result, _bullet = ctx.record.measurement("bananas",
                                              randint(0, 10),
                                              ResultAPI.UNIT_DB,
                                              ctx.item.args.min,
                                              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_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_results) # always last line of test
```

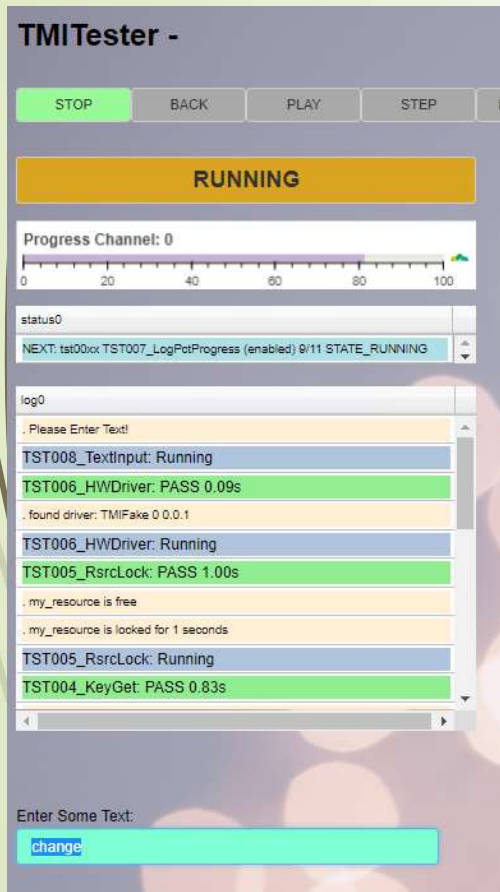
# Python Test Code – User Buttons



```
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
```

- When a test requires Operator input, one option is buttons
- Shows is the Test panel presenting three buttons to the Operator

# Python Test Code – Text Input



- For the case when text input is needed
  - Text input is NEVER a good thing for production environment, too slow and error prone
  - However this input is meant for **Barcode Scanners**, which can output text like a keyboard.
    - For example, scanning lot codes of parts used on a DUT

```
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
```

# JSON Results

- Human readable
  - Very useful in development; Can be encrypted
- Normalized, all results have the same structure, making it easier to process SQL in a standard way
- Each Result has unique RUID
- Measurement data “name” is a full path to the test
- Support for Postgres JSONB objects

```
"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-06AMGKM",
      "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"
        }
      ]
    }
  ]
}
```

# GUI: Test Configuration (optional)

**TMI Tester - Test Config** admin

Please select script....  
(selector includes all scripts found) public/scripts/TMIDemoRecordV1/prod\_v0/prod\_2.tmiscr

```
# Example: Shows substitution being used in more than one place.
{
  "subs": {
    "Lot": {
      "title": "Lot (format ####)",
      "type": "str", "widget": "textinput", "regex": "\\d{5}", "default": "95035"
    },
    "Loc": {
      "title": "Location",
      "type": "str", "widget": "select", "choices": ["canada/ontario/milton", "us/newyork/buffalo"]
    },
    "TST000Max": {
      "title": "TST000 Max Attenuation (db)",
      "type": "num", "widget": "select", "choices": [9, 10]
    },
    "TST100Max": {
      "title": "TST100 Max Attenuation (db)",
      "type": "num", "widget": "select", "choices": [11.1, 12.2, 13.3]
    }
  },
  "info": {
    "product": "widget_1",
    "bom": "B00012-001",
    # list fields present user choice or fill in
    "lot": "___TBD___",
    "location": "___TBD___"
  },
  "config": {
    "result_handler": "TMIDemoRecordV1",
    "fail_fast": false,
    "channel_hw_driver": ["public.scripts.TMIDemoRecordV1.prod_v0.drivers.tmi_fake"]
  },
  "tests": [
    {
      "module": "public.scripts.TMIDemoRecordV1.prod_v0.tst00xx",
      "options": {
        "fail_fast": false
      },
      "items": [
        {
          "id": "TST00xxSETUP",
          "enable": true,
          "args": {"min": 0, "max": "___TBD___"},
          "fail": [ {"fid": "TST000-0", "msg": "Component apple R1"}, {"fid": "TST000-1", "msg": "Component banana R1"} ],
          "enable": false
        },
        {
          "id": "TST00xxTRDN",
          "enable": true
        }
      ]
    }
  ]
}
```

Lot (format ####) FIXME

Location

TST000 Max Attenuation (db)

TST100 Max Attenuation (db)

log  
SYSTEM: Master (restarted)

- Scripts "subs" section can define items that are to be set at test time
- For example,
  - Lot Number
  - Location
  - Measurement limits
- Definition controls user options
- Regex patterns are also supported for text entry fields



## GUI: Test Configuration Create Barcode (optional)

**TMITester - Test Config** admin

Please select script...  
(selector includes all scripts found) public/scripts/TMIDemoRecordV1/prod\_v0/prod\_2.tmiscr

# Example: Shows substitution being used in more than one place.

```
{
  "subs": {
    "Lot": {
      "title": "Lot (format ####)",
      "type": "str", "widget": "textinput", "regex": "\\d{5}", "default": "95035"
    },
    "Loc": {
      "title": "Location",
      "type": "str", "widget": "select", "choices": ["canada/ontario/milton", "us/newyork/buffalo"]
    },
    "TST000Max": {
      "title": "TST000 Max Attenuation (db)",
      "type": "num", "widget": "select", "choices": [9, 10]
    },
    "TST100Max": {
      "title": "TST100 Max Attenuation (db)",
      "type": "num", "widget": "select", "choices": [11.1, 12.2, 13.3]
    }
  },
  "info": {
    "product": "widget_1",
    "bom": "B00012-001",
    "# list fields present user choice or fill in",
    "Lot": "12345",
    "location": "canada/ontario/milton"
  },
  "config": {
    "result_handler": "TMIDemoRecordV1",
    "fail_fast": false,
    "channel_hw_driver": ["public.scripts.TMIDemoRecordV1.prod_v0.drivers.tmi_fake"]
  },
  "tests": [
    {
      "module": "public.scripts.TMIDemoRecordV1.prod_v0.tst00xx",
      "options": {
        "fail_fast": false
      },
      "items": [
        {
          "id": "TST00xSETUP",
          "enable": true,
          "args": {
            "min": 0, "max": 9
          },
          "fail": [
            {
              "fid": "TST000-0",
              "msg": "Component apple R1"
            },
            {
              "fid": "TST000-1",
              "msg": "Component banana R1"
            }
          ]
        },
        {
          "id": "TST001_Skip",
          "enable": false
        },
        {
          "id": "TST00xTROW",
          "enable": true
        }
      ]
    }
  ]
}
```

Lot (format ####)  
12345

Location  
canada/ontario/milton

TST000 Max Attenuation (db)  
9

TST100 Max Attenuation (db)  
12.2

Apply

Traveller

Submit

Test

log

SYSTEM: Master (restarted)

**TMITester** **TMI Traveller**

public/scripts/TMIDemoRecordV1/prod\_v0/prod\_2.tmiscr

admin : 2018, December 17 19:44:25

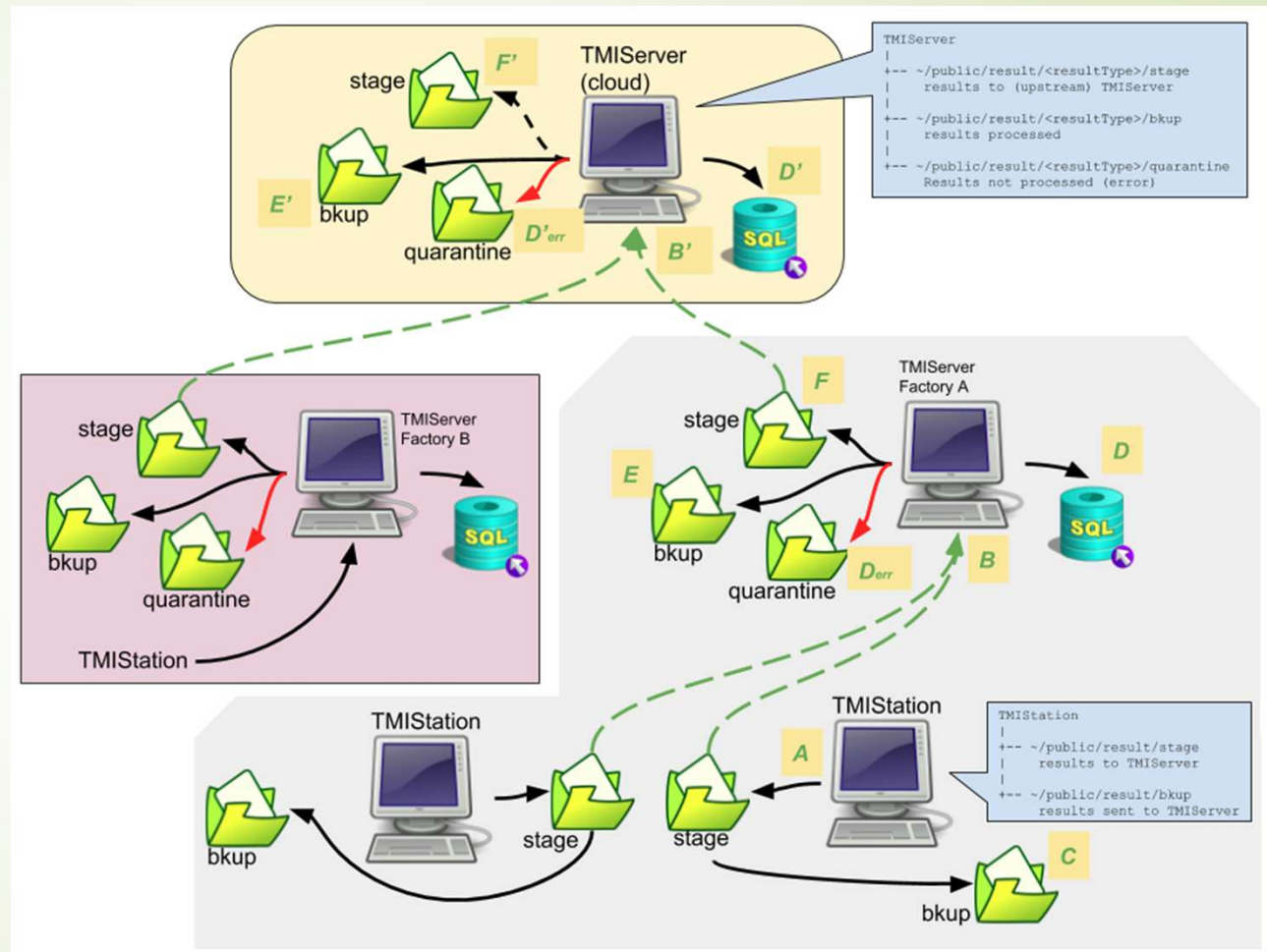


- Travellers can be made to capture all subs variables, and then scanned on the production floor
- No manual entry by test operators

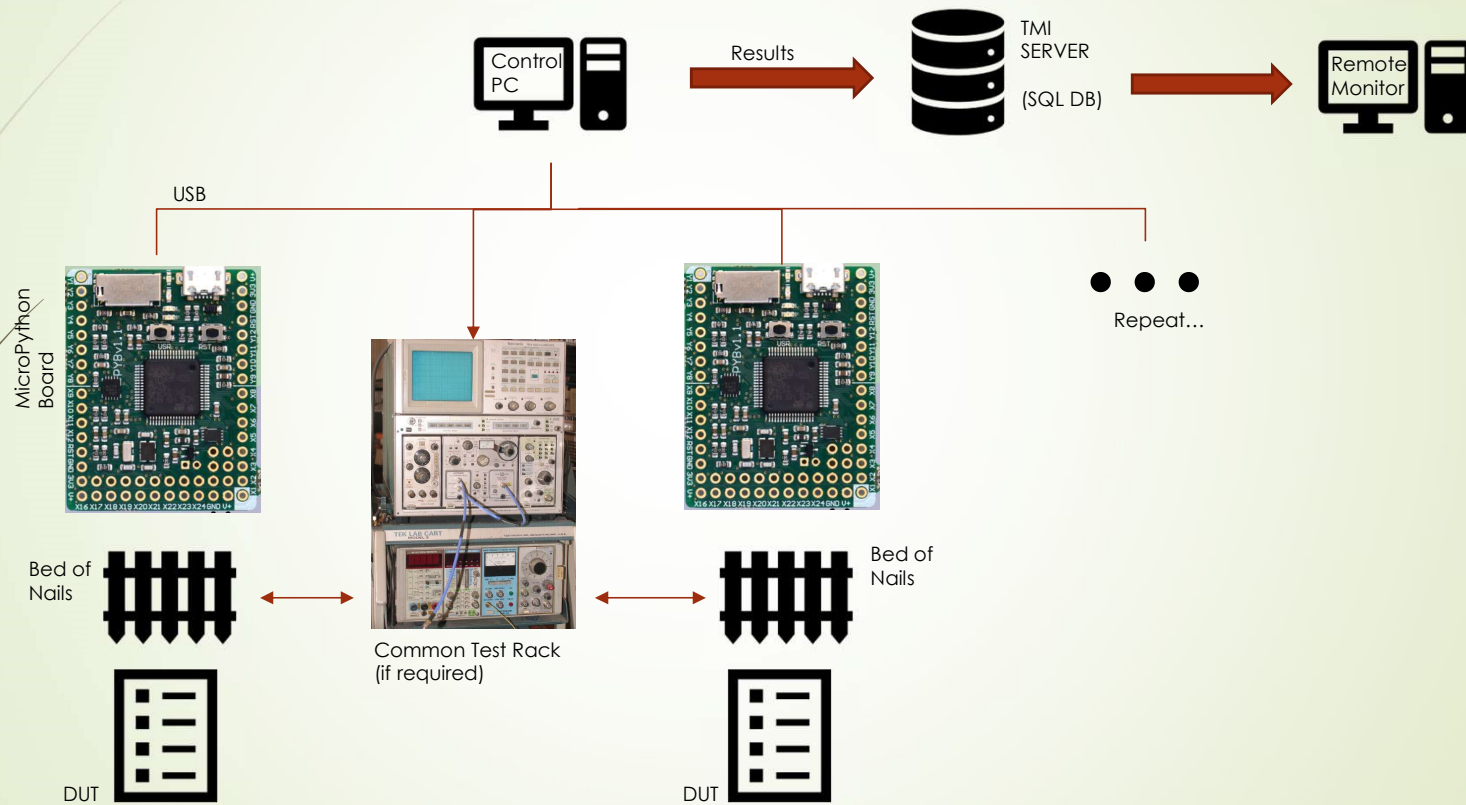


# Result Flow

- (JSON) Result files flow upwards
- Backups are made along the way
- Any TMIServer can present a dashboard based on the results within its SQL DB
- TMISStation also back backup data

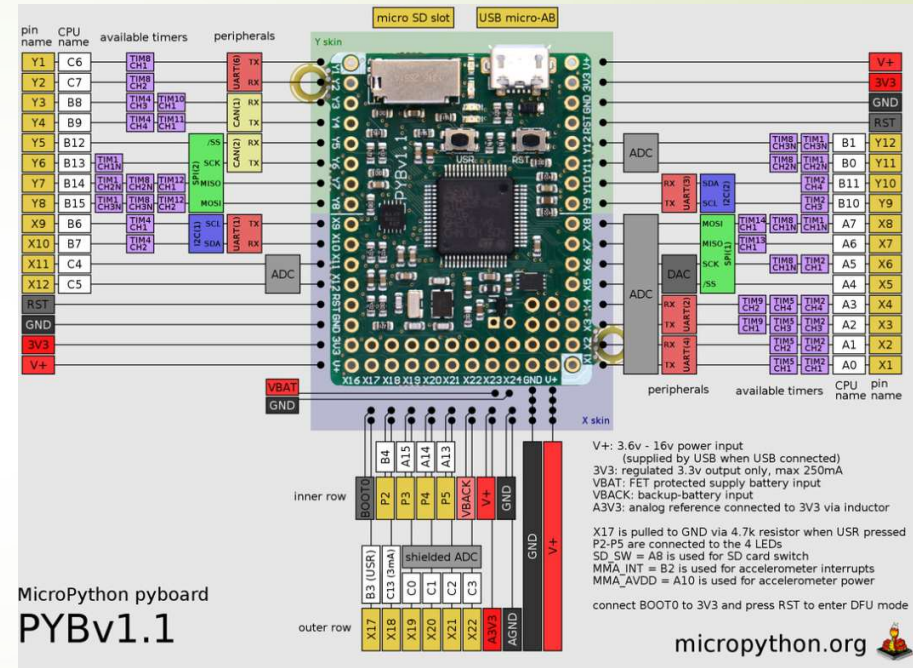


# System HW Block Diagram



## MicroPython Test Instrument

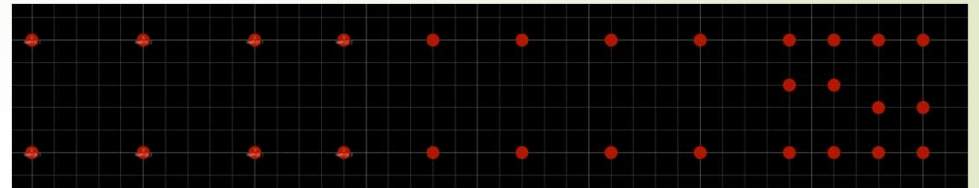
- Low Cost
  - Read/Write GPIOs
  - ADC
  - Proxy for Serial, I2C, SPI commands
- 
- NOTE: The MicroPython board may not be suitable for your application. This board is used to demonstrate and develop the features of the framework.



# MicroPython Interface Proto Board v1.x

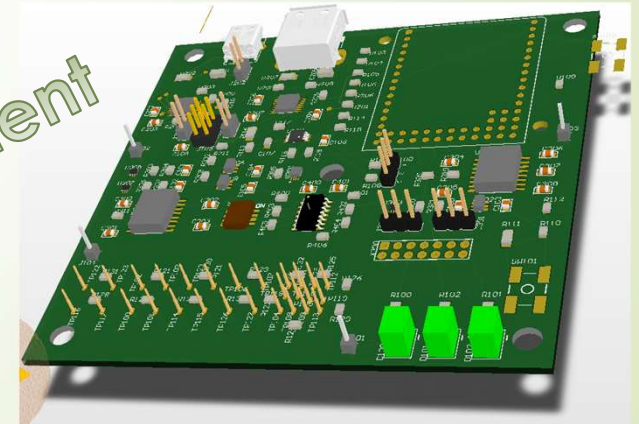
- Regardless of your PCB design, if you place test pads to **use any subset** of the 28 pads that are in this arrangement – then you can use this already designed Interface Board
- Any of the test points can be connected to the MicroPython board which can make ADC measurements, GPIO, UART, I2C and other functions
- This Interface board also supports 2 programmable Voltage rails and one current measurement sensor
- Your exact test point needs should be described and checked to see if they can be satisfied by the MicroPython Interface Board v1.x

Your PCB Design (grid is 50x50mils)



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

In  
Development

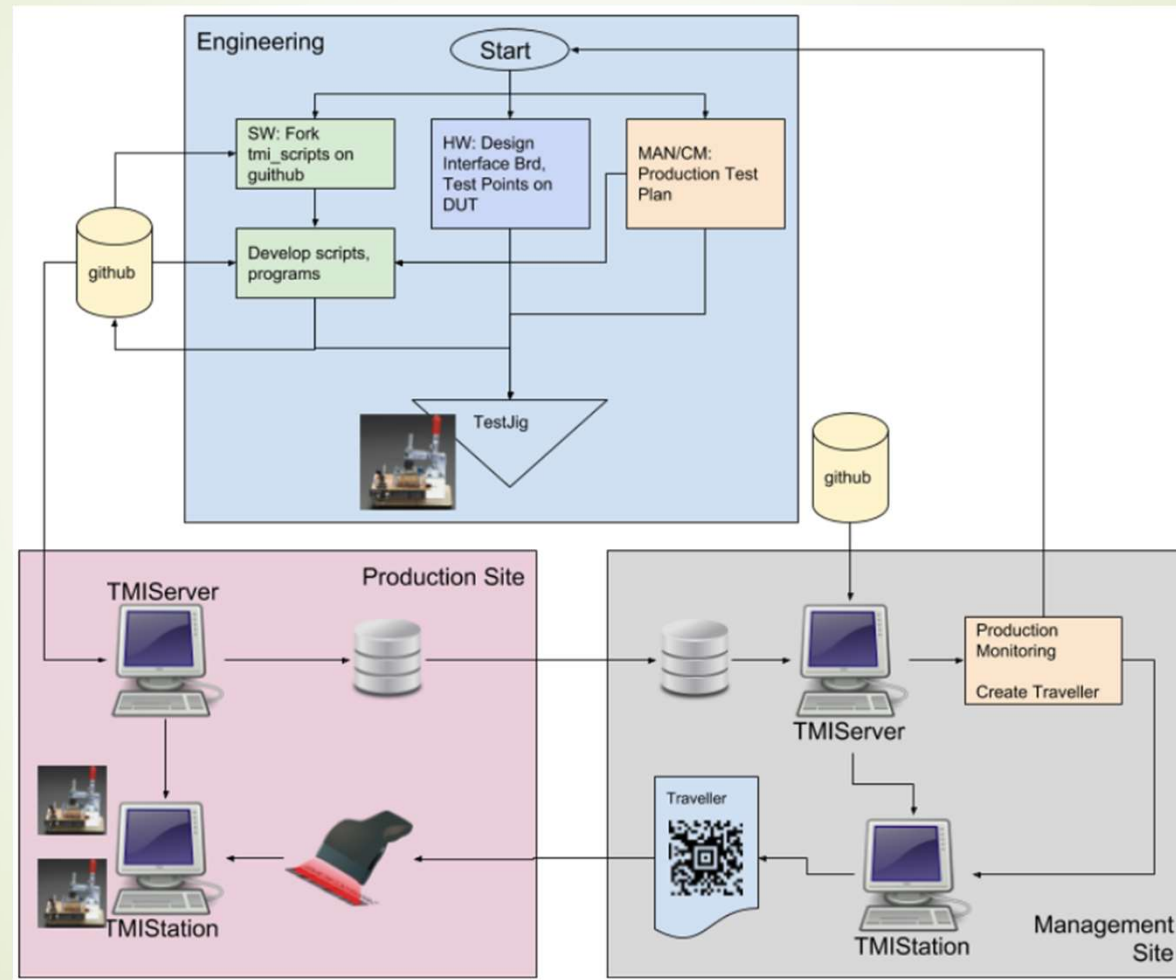




# DUT Design for Testability

- Add test points for the bed of nails jig
- Understand the MicroPython Board IO pin capabilities
  - Or create your own “interface board”
- Create PCB to interface MicroPython Board to the DUT
  - Or use the already designed MicroPython Board V1.x
- Determine what external test equipment is required to test things that can't be tested with MicroPython Board
- Write (Python 3.6) Software within this framework...
  - Results will be normalized, stored in SQL DB
  - Logging
  - Results Dashboard

# Development/Production Model







# Security

- ▀ Accounts
  - ▀ User assigned Roles for access to different system features
  - ▀ PWs stored encrypted
- ▀ Results
  - ▀ can be encrypted
- ▀ TMISStation
  - ▀ Code/Scripts can be “locked down”, tests will not start if any locked down file has been altered
  - ▀ Travellers to control
    - ▀ what Script is run,
    - ▀ what variable parameters are used – production operator “hands free”





# Project Current Status

- ▶ Ready to Demo
- ▶ Features in Development
  - ▶ More dashboard statistical graphs
  - ▶ More TMIServer – TMISStation management tasks

# Other

- DUT measurements
  - Considering the Diligent Analog Discover 2
    - 2CH (100Msample/sec) scope, GPIO, Power sources, Digital Bus Analyzers (SPI, I<sup>2</sup>C, UART, Parallel)
    - Python drivers
    - CON
      - \$400 each!
      - No (serial, i2c, etc.) ports

