Embedded Linux Conference

Board Farm APIs for Automated Testing of Embedded Linux – An Update

Harish Bansal

Technical Engineer
Timesys

Tim Bird

Principal Software Engineer
Sony Electronics

THE LINUX FOUNDATION

Abstract

This talk presents an update on work to create a standard API between automated tests and Board Farm hardware and software. Last year, we introduced the notion of a dual REST/command-line API that could be used for discovery, control and operation of hardware and network resources in a test lab. Since then, the scope of the work has increased, and there are now APIs for control of additional lab hardware.

Multiple implementations of the API (both server and client side) have been created. We will describe the new APIs we have added, and demonstrate new tests that work with the REST API system, including power measurement tests and hardware serial port tests. Also, we will discuss how we envision using the API architecture for additional hardware testing, such as CANbus, or A/V testing. Although different equipment is utilized in different test labs (or Board Farms), by using the REST API the same test can be run in the different labs to obtain test results and provide quality assurance for products.

It is hoped that this Board Farm API abstraction will pave the way for more sharing of automated tests and testing resources, to accelerate the use of automated testing for products based on embedded Linux.

Outline

- **Review of REST API concepts**
- **Status since last year**
  - Resource model
- **Demos**
  - Power measurement APIs and demo
  - Camera APIs and demo
  - Serial port APIs and demo
- **Proof point**
  - 3 Test Frameworks (Lava, Fuego, Robot Framework) running on top of APIs
- **Future directions**

Problem statement (review)

- **There are many tests but no standardized way of running tests on physical devices**
- **There are many different Test Frameworks**
- **There are a few Board Farm frameworks**
  - But no standardized way to use different Test Frameworks or run tests
- **Every farm implements test infrastructure differently**
  - Many labs use ad-hoc infrastructure
    - Cobble together available hardware, and write custom scripts for control and data collection
  - Tests written for one lab do not work in another lab
- **Nobody can share tests**

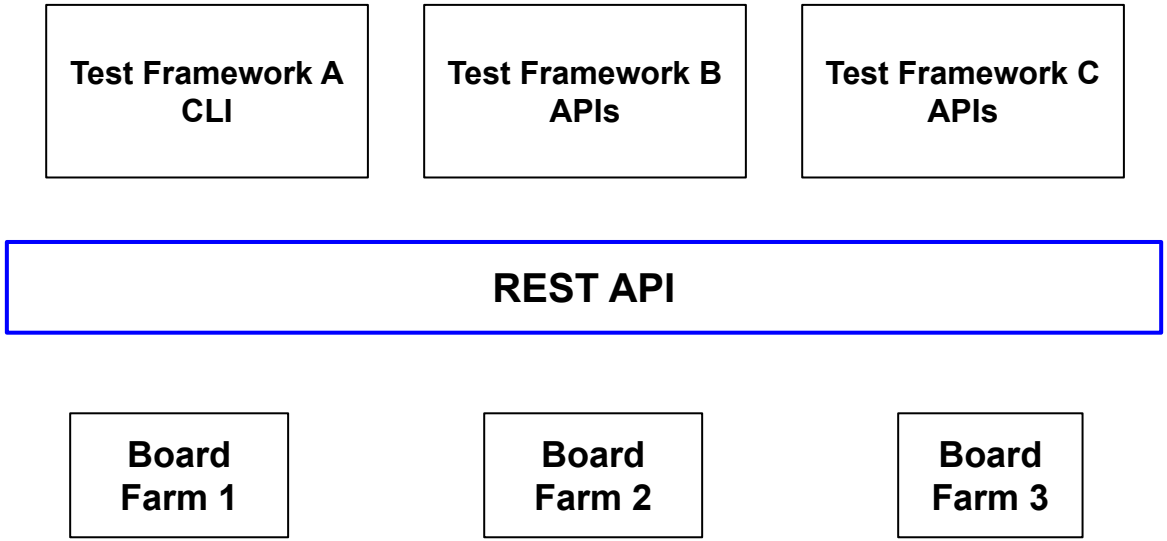
Solution:

- **Creating a standard method to access a Board Farm allows:**
  - Board Farm technologies can evolve separately from the interface to the farm
  - Tests can be written that work in more than one lab
  - Test Frameworks can work with more than one lab

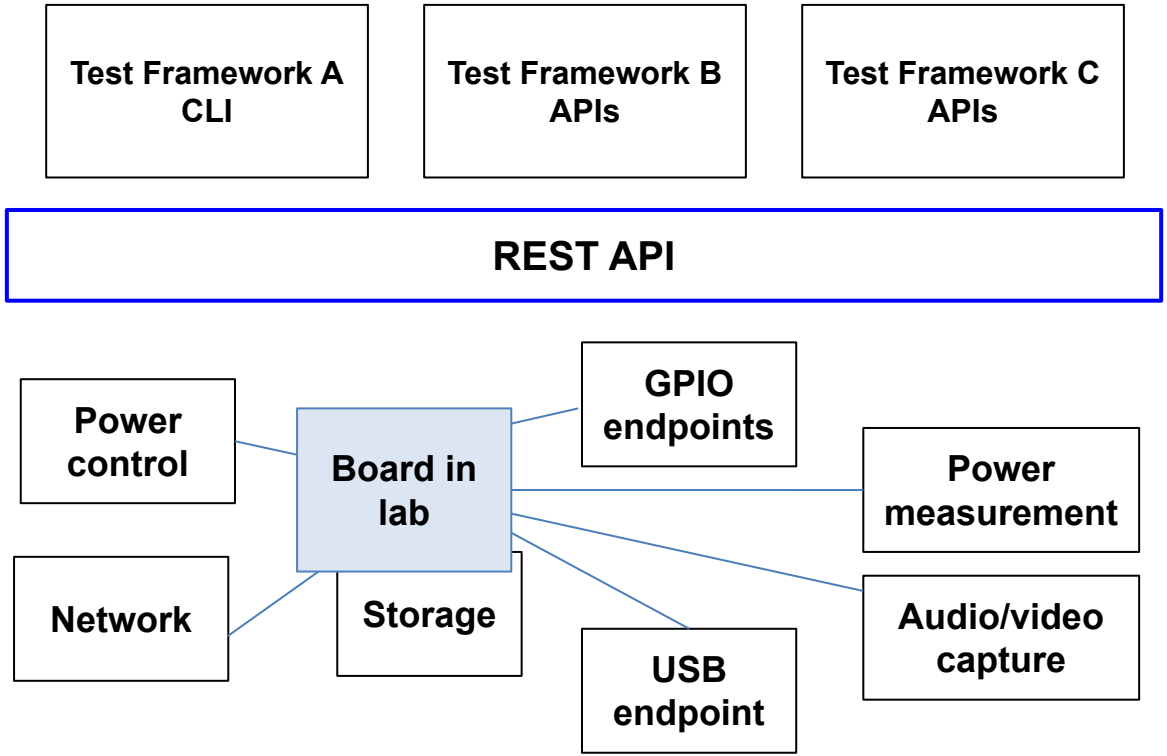
Examples of hardware/software integration tests

- **GPIO test, serial port test**
  - Need to control two endpoints
    - One on device under test (DUT) and one external endpoint
- **Audio, video playback test**
  - Need to control two endpoints
    - One on device under test (DUT) and a capture device
- **Power measurement (via external power monitor)**
  - Need to control two endpoints:
    - Application or workload profile on DUT
    - Capture of power measurement data on external power monitor
- **USB connect/disconnect (robustness) testing**
  - Need to control two endpoints:
    - Application or monitor on DUT
    - USB hardware external to board (drop/reconnect vbus)

High level concept 1 – API between framework and lab



High level concept 2 – API between test and lab



REST API elements

- **API proposal**
  - 2 parts
    - web-based REST API
    - Command line interface
- **REST API based on https and JSON**
  - Extension to LAVA/Django REST API
  - Only requires curl and jq
- **Command line tool**
  - Same operations as REST API
  - Suitable for automated use, as well as human interactive use

What has happened since last year?

Added since last year

- **More implementations**
- **Added the resource model**
  - new API: get-resource
- **Added the generic capture API model**
  - start-capture, stop-capture, get-data, delete
- **APIs for new resource types:**
  - power measurement
  - image and video capture
  - serial port receive and transmit
- **Direct support for API in Fuego**
- **Created test example in Robot Test Framework**

More implementations

- **Extended original implementation**
  - LAVA server is based on Django
  - Is in production use now, as part of Timesys Embedded Board Farm service
  - EBF client supports new APIs
    - Is a shell-based client using curl and jq
    - Git repo: https://github.com/TimesysGit/board-farm-rest-api
- **Created LabControl server and client implementation**
  - lcserver is a plain CGI script (no framework)
  - 'lc' is a python client, using the python requests module
  - source is available now
    - But should be considered alpha-level quality
    - Git Repo: https://github.com/tbird20d/labcontrol.git

Implementation issues

- **Found some incompatibilities between implementations**
- **Goal:**
  - Use both clients (ebf and lc) with both servers (EBF and labcontrol)
- **Use of python requests module showed some issues with API definitions**
  - curl and requests perform same operations with different form encoding
    - Have to make sure that wire protocol matches exactly
- **Both labs have APIs that the other lab does not support yet**
  - EBF: APIs for storage management
  - labcontrol: serial receive/transmit

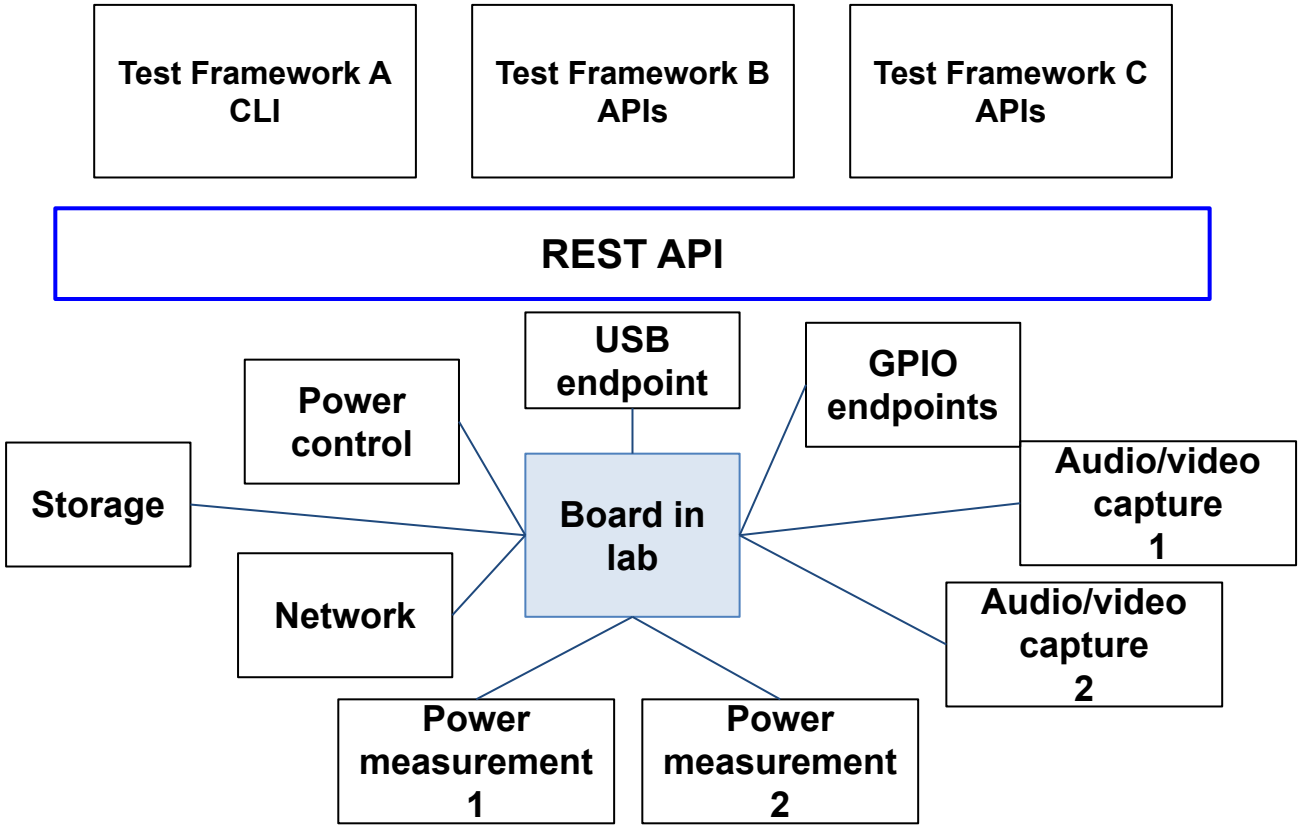
Added the resource model

- **Previously, all operations were relative to the device under test**
  - e.g.: api/v0.2/devices/{board}/power/on
- **Introduced new 'resource' model**
  - To perform an operation:
    - First, get the resource that is associated with the DUT, for this operation type
    - Perform operations with a resource, instead of board:
      - api/v0.2/resources/{resource}/camera/start-capture
- **Uses a new api to retrieve the resource assignment**
  - api/v0.2/devices/{board}/get-resource/camera
  - resource=$(ebf rpi4 get-resource camera)

Resource model benefits

- **Is more flexible**
  - More than one resource of a particular type can be assigned to a board
    - board-based API assumes 1:1 mapping between lab resource and DUT
    - e.g.: power-measurement at multiple probe points on the board
  - A resource can be associated with multiple boards
    - e.g.: power control – it's very common for a single power controller to control multiple boards
- **In the future:**
  - Can support dynamic multiplexing
    - Control the resource assignment at runtime
    - Reserve the resource for the duration of usage

DUT connections with resource model



Supported resource types

- **Currently supported resource types are:**
  - Power-measurement
  - Camera
  - Serial
- **Ones that are envisioned:**
  - Canbus
  - USB

Added the generic capture API model

- **Generic ‘capture’ API consist of 4 verbs:**
  - Start-capture
    - Begin capturing data
    - Returns a token for capture data manipulation
  - Stop-capture
    - Stop capturing data
  - Get-data
    - Retrieve the data from the server
  - Delete
    - Remove the data from the server

APIs for new resource types

- **Power measurement**
  - start-capture, stop-capture, get-data, delete
- **Image and video capture**
  - capture still image
  - start-capture, start-capture with duration, stop-capture, get-ref – for videos
- **Serial port transmit**
  - DUT as transmitter
  - lab resource as receiver
    - set-config, start-capture, stop-capture, get-data, delete
- **Serial port receive**
  - DUT as receiver
  - lab resource as transmitter
    - set-config, put-data

The gory details

	REST APIs	CLI Commands
Resource	**Get Resource** /api/v0.2/devices/{DeviceName}/get-resource/{Resource-Type}/ **Get Resource by feature** /api/v0.2/devices/{DeviceName}/get-resource/{Resource-Type}/{feature_name}	**Get Resource** $CLIENT {DeviceName} get-resource {ResourceType} **Get Resource by feature** $CLIENT {DeviceName} get-resource {ResourceType} {feature}
Power Measurement	**Start Capture** /api/v0.2/resources/{ResourceName}/power-measurement/start-capture/ **Stop Capture** /api/v0.2/resources/{ResourceName}/power-measurement/stop-capture/{token} **Get Data** /api/v0.2/resources/{ResourceName}/power-measurement/get-data/{token} **Delete Data** /api/v0.2/resources/{ResourceName}/power-measurement/delete/{token}	**Start Capture** $CLIENT {ResourceName} power-measurement start **Stop Capture** $CLIENT {ResourceName} power-measurement stop {token} **Get Data** $CLIENT {ResourceName} power-measurement get-data {token} **Delete Data** $CLIENT {ResourceName} power-measurement delete {token}
Camera	**Capture Still Image** /api/v0.2/resources/{ResourceName}/camera/capture/ **Start Capture** /api/v0.2/resources/{ResourceName}/camera/start-capture/ /api/v0.2/resources/{ResourceName}/camera/start-capture/{Duration} **Get Reference** /api/v0.2/resources/{ResourceName}/camera/get-ref/{token}/	**Capture Still Image** $CLIENT {ResourceName} camera capture $CLIENT {ResourceName} camera capture -o {Filename} **Start Capture** $CLIENT {ResourceName} camera start-capture $CLIENT {ResourceName} camera start-capture -d {Duration} **Get Reference** $CLIENT {ResourceName} camera get-ref {Video-Id} $CLIENT {ResourceName} camera get-ref {Video-Id} -o {Filename}

The gory details (prototype API – not yet confirmed)

	REST APIs	CLI Commands
Serial	**Start Capture** /api/v0.2/resources/{ResourceName}/serial/start-capture/ **Stop Capture** /api/v0.2/resources/{ResourceName}/serial/stop-capture/{token} **Get Data** /api/v0.2/resources/{ResourceName}/serial/get-data/{token} **Delete Data** /api/v0.2/resources/{ResourceName}/serial/delete/{token} **Set Config** /api/v0.2/resources/{ResourceName}/serial/set-config/ POST - { "baud_rate": "115200" } as data for post **Put Data** /api/v0.2/resources/{ResourceName}/serial/put-data/{token} POST - raw data	**Start Capture** $CLIENT {ResourceName} serial start **Stop Capture** $CLIENT {ResourceName} serial stop {token} **Get Data** $CLIENT {ResourceName} serial get-data {token} **Delete Data** $CLIENT {ResourceName} serial delete {token} **Set Config** echo "{ \"baud_rate\": \"115200\"" | $CLIENT {ResourceName} serial set-config **Put Data** $CLIENT {ResourceName} serial put-data <raw_data

Direct support for API in Fuego

- **Last year, Fuego used the API via a wrapper (‘ttc’) that it already supported**
- **Now, a Fuego user can specify a transport of either ‘ebf’ or ‘lc’ for a board, and have tests performed using the API directly**
  - This requires less configuration inside the Fuego docker container
  - No 'ttc' wrapper between Fuego and the board farm client

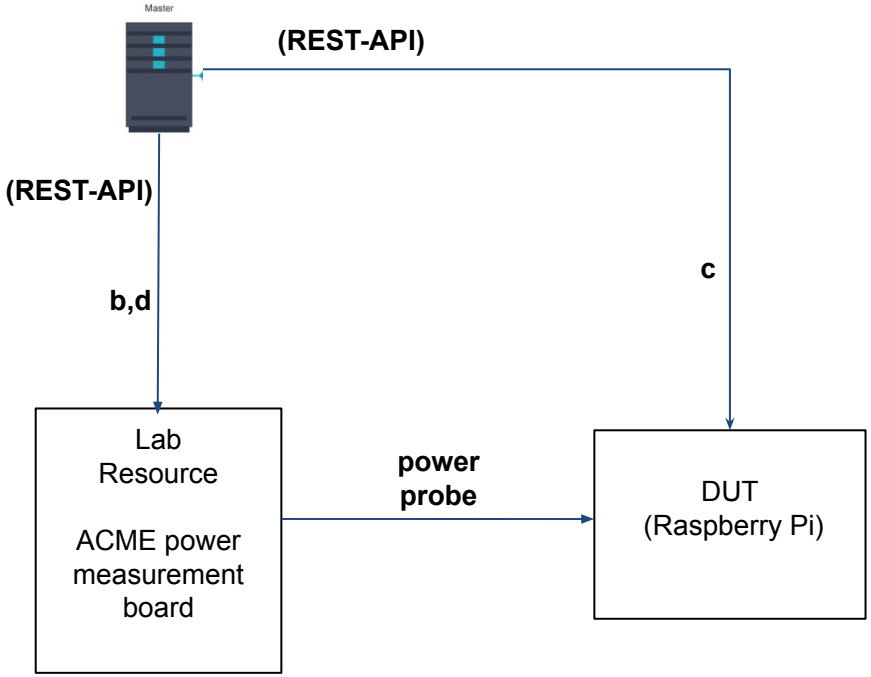
Some miscellaneous features

- **Support for recursive directory copy**
  - Can copy an entire directory to or from the DUT
- **Support for debugging commands**
  - ebf supports '—debug' argument, which generates a trace of the API request and response
    - Is very useful to see data structures on wire
- **Have added sample tests showing API usage**

Use case:

Lab-independent power measurement test

Power measurement test – REST API use overview



Assumption:

- **Lab knows the binding of DUT and power measurement device**
  - Resource ACME is assigned to the Raspberry Pi

Test Steps:

a. **Get PM resource associated with DUT**
b. **Start PM data capture**
c. **Run Workload**
d. **Stop capture, get data**
e. **Analyze and report results**

Video of actual PM test execution

(We did it!!)

What the API looks like in practice

Excerpt from power-measurement-test.sh:

```
echo "Getting power measurement resource for board"
RESOURCE=$($CLIENT $BOARD get-resource power-measurement)

echo "Starting power measurement"
token=$($CLIENT $RESOURCE power-measurement start)
if [ "$?" != "0" ] ; then
    error_out "Could not start power measurement with $CLIENT, with resource $RESOURCE"
fi

echo "Performing some workload (stress test)"
${CLIENT} ${BOARD} ssh run "${WORKLOAD_COMMAND}"

echo "Stopping power measurement"
$CLIENT $RESOURCE power-measurement stop $token || \
    error_out "Could not stop power measurement with $CLIENT"

echo "Getting data"
POWER_DATA=$($CLIENT $RESOURCE power-measurement get-data $token) || \
    error_out "Could not get power data with $CLIENT, using token $token"
echo $POWER_DATA

MAX_POWER_USED=`echo "$POWER_DATA" | xargs -n 1| tail -n+2| cut -d',' -f2,3 --output-delimiter=' '| awk '{printf "%.3f\\n", $1*$2/1000000}'| sort -r| head -1`
echo "MAX-POWER-USED=$MAX_POWER_USED"
echo "THRESHOLD-POWER=$THRESHOLD_POWER"
echo "Deleting the data on the server"
$CLIENT $RESOURCE power-measurement delete $token || \
    echo "Warning: Could not delete data for token $token on server"
```

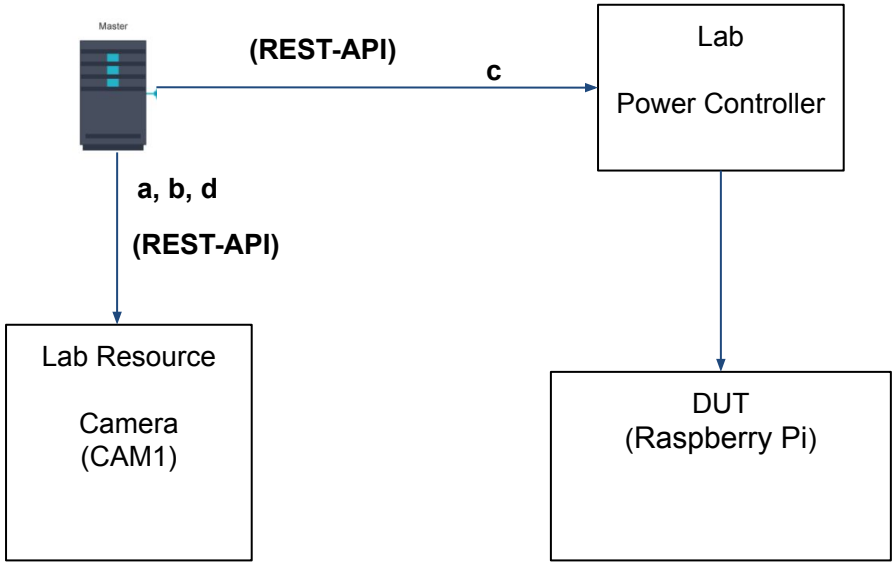
PM CLI tool commands

Operation	CLI Command	Example
Get Resource	{TestClient} {DeviceName} get-resource power-measurement	*RESOURCE=$(ebf baylibre_rpi2-1 get-resource power-measurement)*
Start Capture	{TestClient} {ResourceName} power-measurement start	*token=$(ebf $RESOURCE power-measurement start)*
Stop Capture	{TestClient} {ResourceName} power-measurement stop {token}	*ebf $RESOURCE power-measurement stop $token*
Get Data	{TestClient} {ResourceName} power-measurement get-data {token}	*ebf $RESOURCE power-measurement get-data $token*
Delete Data	{TestClient} {ResourceName} power-measurement delete {token}	*ebf $RESOURCE power-measurement delete $token*

PM REST API details

Operation	Route	Response *(Data Type - JSON)*
Get Resource	/api/v0.2/devices/{DeviceName}/get-resource/{ResourceType}/	Success *{"result": "success", "data": <Resource Id>}* Failure *{"result": "fail", message:<reason for failure>}*
Start Capture	/api/v0.2/resources/{ResourceName}/power-measurement/start-capture/	Success *{"result": "success", "data": <token>}* Failure *{"result": "fail", message:<reason for failure>}*
Stop Capture	/api/v0.2/resources/{ResourceName}/power-measurement/stop-capture/{token}	Success *{"result": "success"}* Failure *{"result": "fail", message:<reason for failure>}*
Get Data	/api/v0.2/resources/{ResourceName}/power-measurement/get-data/{token}	Success *{"result": "success", "data": <csv string with power measurement readings>}* Failure *{"result": "fail", message:<reason for failure>}*
Delete Data	/api/v0.2/resources/{ResourceName}/power-measurement/delete/{token}	Success *{"result": "success"}* Failure *{"result": "fail", message:<reason for failure>}*

Camera



Assumption:

- **Lab knows the binding of DUT and camera**
  - Resource CAM1 is assigned to the Raspberry Pi

Test Steps:

a. **Get Camera resource associated with DUT**
b. **Start Video Recording for a configured duration**
c. **Reboot DUT**
d. **Get Video recording**

What the API looks like in practice

Excerpt from camera-test.sh:

```
RESOURCE=$(ebf $DUT get-resource camera)
if [ $? -eq 0 ];then
  echo "Start Capturing Video"
  VIDEO_ID=$(ebf $RESOURCE camera start-capture -d $DURATION)
  if [ $? -eq 0 ];then
    echo "Rebooting the Board"
    ebf $DUT power reboot
    if [ $? -eq 0 ];then
      sleep "$TIME"s
      VIDEO_URL=$(ebf $RESOURCE camera get-ref $VIDEO_ID)
      echo "VIDEO_URL=$VIDEO_URL"
    else
      echo "Couldn't reboot the board"
      exit 1
    fi
  else
    echo "Couldn't start video capturing"
    exit 1
  fi
else
  echo "Couldn't get camera resource for video capturing"
  exit 1
fi
```

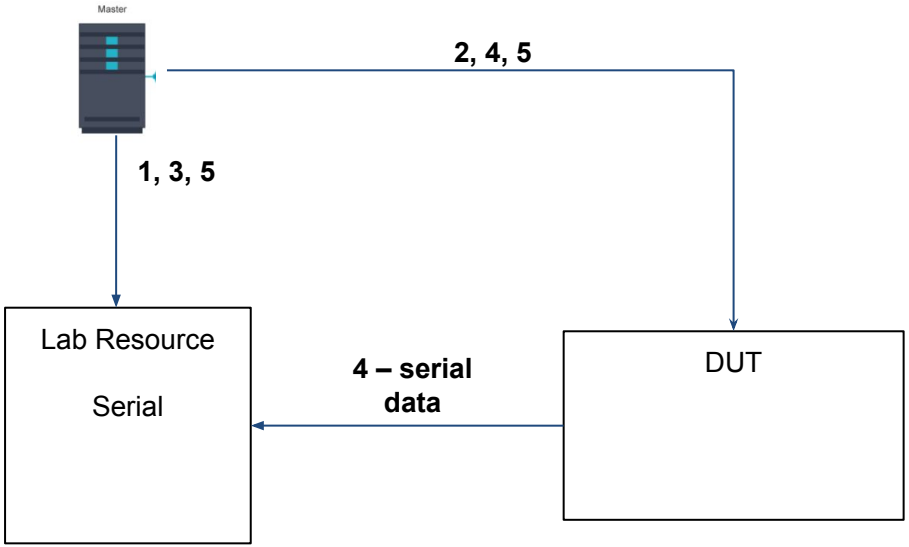
Camera CLI commands

Operation	CLI Command	Example
Get Resource	{TestClient} {DeviceName} get-resource camera	*RESOURCE=$(ebf raspbian get-resource camera)*
Capture	{TestClient} {ResourceName} camera capture {TestClient} {ResourceName} camera capture -o {Filename}	*ebf $RESOURCE camera capture* *ebf $RESOURCE camera capture -o screenshot.jpeg*
Start Capture	{TestClient} {ResourceName} camera start-capture {TestClient} {ResourceName} camera start-capture -d {Duration}	*VIDEO_ID=$(ebf $RESOURCE camera start-capture)* *VIDEO_ID=$(ebf $RESOURCE camera start-capture -d $DURATION)*
Get Reference	{TestClient} {ResourceName} camera get-ref {VIDEO_ID} {TestClient} {ResourceName} camera get-ref {VIDEO_ID} -o {Filename}	*ebf $RESOURCE camera get-ref $VIDEO_ID* *ebf $RESOURCE camera get-ref $VIDEO_ID -o recording.mp4*

Camera REST API details

Operation	Route	Response *(Data Type - JSON)*
Get Resource	/api/v0.2/devices/{DeviceName}/get-resource/{ResourceType}/	Success *{"result": "success", "data": <Resource Id>}* Failure *{"result": "fail", message:<reason for failure>}*
Capture	/api/v0.2/resources/{ResourceName}/camera/capture/	Success *{"result": "success", "data": <Image URL>}* Failure *{"result": "fail", message:<reason for failure>}*
Start Capture	/api/v0.2/resources/{ResourceName}/camera/start-capture/ /api/v0.2/resources/{ResourceName}/camera/start-capture/{Duration}	Success *{"result": "success", "data": {"token": <Video Id>}}* Failure *{"result": "fail", message:<reason for failure>}*
Get Reference	/api/v0.2/resources/{ResourceName}/camera/get-ref/{token}	Success *{"result": "success", "data": <Video URL>}* Failure *{"result": "fail", message:<reason for failure>}*

Serial



Test of Serial hardware

RS232

1. Use REST API to configure lab resource as Rx or Tx, and baud rate
2. Use local commands to set DUT serial RX or TX and baud rate
3. Initiate capture
4. Initiate serial data transfer
5. End capture, collect log
6. Compare transmission vs capture data

Video of Fuego serial test execution

Workspace
Build Now
Configure
Delete Project
Rename

Build History trend

find

#22 Aug 26, 2021 10:16 PM
testlog run.json

#21 Aug 26, 2021 10:13 PM
testlog run.json

#20 Aug 26, 2021 10:08 PM
testlog run.json

#19 Aug 26, 2021 10:06 PM
testlog run.json

#18 Aug 26, 2021 10:05 PM
testlog run.json

#17 Aug 26, 2021 10:04 PM

lcbbb-Functional.BF_serial_tx-default

board: lcbbb
test set: default
kernel: 4.4.155-ti-r155

test case	results					
	build_number					
	17	18	19	20	21	22
01_Check_transmission_at_baud-rate_150	PASS	PASS	PASS	PASS	PASS	PASS
02_Check_transmission_at_baud-rate_1200	PASS	PASS	PASS	PASS	PASS	PASS
03_Check_transmission_at_baud-rate_9600	PASS	PASS	PASS	PASS	PASS	PASS
04_Check_transmission_at_baud-rate_19200	PASS	PASS	PASS	PASS	PASS	PASS
05_Check_transmission_at_baud-rate_38400	PASS	PASS	PASS	PASS	PASS	PASS
06_Check_transmission_at_baud-rate_115200	PASS	PASS	PASS	PASS	PASS	PASS
07_Check_transmission_at_baud-rate_921600	PASS	PASS	PASS	PASS	PASS	PASS
Totals						
pass	7	7	7	7	7	7
fail	0	0	0	0	0	0
skip	0	0	0	0	0	0
error	0	0	0	0	0	0

Workspace
Recent Changes

What the API looks like in practice

Excerpt from serial-transmit-test.sh:

```
test_one_rate() {
    TESTCASE="Check transmission at baud-rate $BAUD_RATE"
    stty -F $DEVICE $BAUD_RATE raw -echo -echoe -echok
     echo '{ "baud_rate": "$BAUD_RATE" }' | \
        $CLIENT $RESOURCE set-config serial

    echo "Capturing data at lab resource $RESOURCE"
    TOKEN="$($CLIENT $RESOURCE serial start)"

    echo "Transmitting data from DUT"
    echo -n "$SEND_DATA" >$DEVICE

    $CLIENT $RESOURCE serial stop $TOKEN
    RECEIVED_DATA="$($CLIENT $RESOURCE serial get-data $TOKEN)"
    $CLIENT $RESOURCE serial delete $TOKEN || \
        echo "Warning: Could not delete data on server"

    # compare the data to get the testcase result
    if [ "$SEND_DATA" != "$RECEIVED_DATA" ] ; then
        fail "$TESTCASE" "Received data does not match sent data"
    else
        succeed "$TESTCASE"
    fi
```

Serial REST API – CLI tool commands

Operation	CLI Command	Example
Get Resource	{TestClient} {DeviceName} get-resource serial [{feature}]	*RESOURCE=$(lc bbb get-resource serial uart1)*
Set Config	{TestClient} {ResourceName} set-config {json config}	*echo '{ "baud_rate": "9600" }' | $(lc $RESOURCE serial set-config)*
Start Capture	{TestClient} {ResourceName} serial start	*token=$(lc $RESOURCE serial start)*
Stop Capture	{TestClient} {ResourceName} serial stop {token}	*lc $RESOURCE serial stop $token*
Get Data	{TestClient} {ResourceName} serial get-data {token}	*lc $RESOURCE serial get-data $token*
Delete Data	{TestClient} {ResourceName} serial delete {token}	*lc $RESOURCE serial delete $token*

Serial REST API details

Operation	Route	Response *(Data Type - JSON)*
Get Resource	/api/v0.2/devices/{DeviceName}/get-resource/serial/{feature}/	Success *{"result": "success", "data": <Resource Id>}* Failure *{"result": "fail", message:<reason for failure>}*
Start Capture	/api/v0.2/resources/{ResourceName}/serial/start-capture/	Success *{"result": "success", "data": <token>}* Failure *{"result": "fail", message:<reason for failure>}*
Stop Capture	/api/v0.2/resources/{ResourceName}/serial/stop-capture/{token}	Success *{"result": "success"}* Failure *{"result": "fail", message:<reason for failure>}*
Get Data	/api/v0.2/resources/{ResourceName}/seriat/get-data/{token}	Success *{"result": "success", "data": <raw serial data>}* Failure *{"result": "fail", message:<reason for failure>}*
Delete Data	/api/v0.2/resources/{ResourceName}/serial/delete/{token}	Success *{"result": "success"}* Failure *{"result": "fail", message:<reason for failure>}*

PM test Rosetta Stone

- **Test in 3 different frameworks:**
  - LAVA/Standalone – test executes on DUT itself
  - Robot Test Framework
  - Fuego

Robot Framework PM test

```
*** Settings ***
Library    Process
*** Variables ***
${DUT}                          baylibre_rpi2-1
${TEST_CLIENT}                  ebf
${COMMAND_GET_RESOURCE}         ${TEST_CLIENT} ${DUT} get-resource power-measurement
${WORKLOAD_COMMAND}             sudo stress --cpu 4 --io 3 --vm 2 --vm-bytes 256M --timeout 60s  2> /dev/null
${MAX_POWER_COMMAND}            xargs -n 1|tail -n+2| cut -d',' -f2,3 --output-delimiter=' '|awk '{printf "%.3f\\n", $1*$2/1000000}'|sort -r|head -1
${THRESHOLD_POWER}              2.5
*** Test Cases ***
Get Power-Measurement
   ${result}    Run Process    ${COMMAND_GET_RESOURCE}    shell=True
   Set Suite Variable    ${RESOURCE}    ${result.stdout}

   ${result}    Run Process    ${TEST_CLIENT} ${RESOURCE} power-measurement start    shell=True
   Set Suite Variable    ${TOKEN}    ${result.stdout}

   ${result}    Run Process    ${TEST_CLIENT} ${DUT} ssh run "${WORKLOAD_COMMAND}"    shell=True
   ${result}    Run Process    ${TEST_CLIENT} ${RESOURCE} power-measurement stop ${TOKEN}    shell=True
   Should Match    ${result.stdout}    success

   ${result}    Run Process    ${TEST_CLIENT} ${RESOURCE} power-measurement get-data ${TOKEN}    shell=True
   Set Suite Variable    ${POWER_DATA}    ${result.stdout}

   ${result}    Run Process    echo "${POWER_DATA}" | ${MAX_POWER_COMMAND}    shell=True
   Set Suite Variable    ${MAX_POWER_USED}    ${result.stdout}
   Should Be True    ${MAX_POWER_USED} <= ${THRESHOLD_POWER}

   ${result}    Run Process    ${TEST_CLIENT} ${RESOURCE} power-measurement delete ${TOKEN}    shell=True
```

Video of Robot Framework power measurement test

REPORT

Power-Measurement-Using-EBF-And-Robot Log

Generated
20210824 11:06:29 UTC+05:30
27 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	1	1	0	0	00:01:14	

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags						

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Power-Measurement-Using-EBF-And-Robot	1	1	0	0	00:01:14	

Test Execution Log

SUITE **Power-Measurement-Using-EBF-And-Robot** 00:01:13.676

Full Name: Power-Measurement-Using-EBF-And-Robot
Source: /opt/test/Power-Measurement-Using-EBF-And-Robot.robot
Start / End / Elapsed: 20210824 11:05:15.393 / 20210824 11:06:29.069 / 00:01:13.676
Status: 1 test total, 1 passed, 0 failed, 0 skipped

TEST **Get Power-Measurement** 00:01:13.637

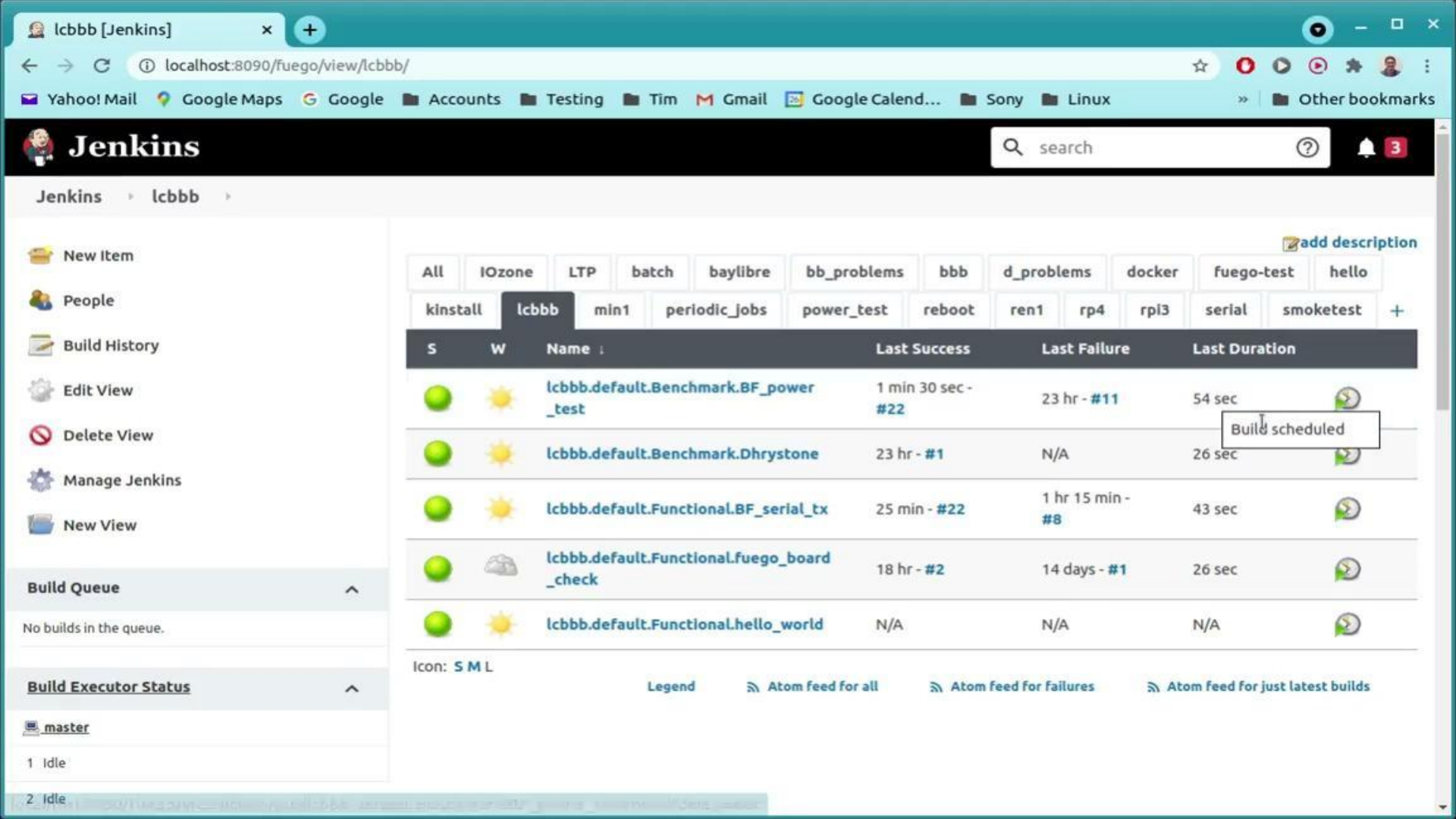
Full Name: Power-Measurement-Using-EBF-And-Robot.Get Power-Measurement
Start / End / Elapsed: 20210824 11:05:15.431 / 20210824 11:06:29.068 / 00:01:13.637
Status: PASS

KEYWORD ${result} = Process.**Run Process** ${COMMAND_GET_RESOURCE}, shell=True 00:00:01.603
KEYWORD Builtin.**Set Suite Variable** ${RESOURCE}, ${result.stdout} 00:00:00.003
KEYWORD ${result} = Process.**Run Process** ${TEST_CLIENT} ${RESOURCE} power-measurement start, shell=True 00:00:01.238
KEYWORD Builtin.**Set Suite Variable** ${ID}, ${result.stdout} 00:00:00.002
KEYWORD ${result} = Process.**Run Process** ${TEST_CLIENT} ${DUT} ssh run "${WORKLOAD_COMMAND}", shell=True 00:01:05.625
KEYWORD ${result} = Process.**Run Process** ${TEST_CLIENT} ${RESOURCE} power-measurement stop ${ID}, shell=True 00:00:01.585
KEYWORD Builtin.**Should Match** ${result.stdout}, success 00:00:00.002
KEYWORD ${result} = Process.**Run Process** ${TEST_CLIENT} ${RESOURCE} power-measurement get-data ${ID}, shell=True 00:00:01.981
KEYWORD Builtin.**Set Suite Variable** ${POWER_DATA}, ${result.stdout} 00:00:00.002
KEYWORD ${result} = Process.**Run Process** echo "${POWER_DATA}" | ${MAX_POWER_COMMAND}, shell=True 00:00:00.049

Documentation: Runs a process and waits for it to complete.
Start / End / Elapsed: 20210824 11:06:27.475 / 20210824 11:06:27.524 / 00:00:00.049

11:06:27.476 INFO
```
Starting process:
echo "timestamp,voltage,current 1575244403,5147.164,251.488 1575244404,5145.399,259.879 1575244405,5146.011,256.994 1575244405,5146.318,255.648 1575244406,5146.170,256.486 1575244407,5141.307,280.604 1575244408,5137.000,301.942 1575244409,5133.862,317.577
1575244410,5131.332,330.177 1575244411,5129.396,339.817 1575244413,5127.061,351.475 1575244414,5125.825,357.643 1575244415,5124.740,363.029 1575244416,5123.804,367.695 1575244416,5123.012,371.619 1575244417,5122.280,375.269 1575244418,5121.675,378.263
1575244419,5121.090,381.194 1575244420,5120.519,384.036 1575244421,5120.043,386.424 1575244422,5119.654,388.357 1575244424,5119.085,391.172 1575244425,5118.758,392.791 1575244426,5118.448,394.341 1575244427,5118.158,395.763 1575244427,5117.890,397.094
1575244428,5117.635,398.352 1575244429,5117.415,399.448 1575244430,5117.208,400.470 1575244431,5117.027,401.360 1575244432,5116.830,402.351 1575244434,5116.581,403.594 1575244435,5116.429,404.360 1575244436,5116.271,405.140 1575244437,5116.154,405.714
1575244438,5116.002,406.465 1575244439,5115.886,407.032 1575244439,5115.754,407.685 1575244440,5115.650,408.199 1575244441,5115.522,408.824 1575244442,5115.428,409.284 1575244444,5115.273,410.062 1575244445,5115.169,410.584 1575244446,5115.091,410.969
1575244447,5114.995,411.445 1575244448,5114.921,411.806 1575244449,5114.832,412.248 1575244450,5114.765,412.576 1575244450,5114.680,412.989 1575244451,5114.609,413.333 1575244453,5114.515,413.795 1575244454,5114.442,414.154 1575244455,5114.389,414.421
1575244456,5114.319,414.766 1575244457,5114.263,415.048 1575244458,5114.203,415.341 1575244459,5114.144,415.640 1575244460,5114.092,415.896 1575244461,5114.038,416.170 1575244461,5113.991,416.405 1575244463,5113.912,416.795 1575244464,5113.864,417.030
```

Video of Fuego power measurement test


lcbbb [Jenkins]
localhost:8090/fuego/view/lcbbb/
Yahoo! Mail
Google Maps
Google
Accounts
Testing
Tim
Gmail
Google Calend...
Sony
Linux
Other bookmarks
Jenkins
search
3
Jenkins
lcbbb
New Item
People
Build History
Edit View
Delete View
Manage Jenkins
New View
Build Queue
No builds in the queue.
Build Executor Status
master
1 Idle
2 Idle
add description
All
IOzone
LTP
batch
baylibre
bb_problems
bbb
d_problems
docker
fuego-test
hello
kinstall
lcbbb
min1
periodic_jobs
power_test
reboot
ren1
rp4
rpi3
serial
smoketest
S
W
Name
Last Success
Last Failure
Last Duration
lcbbb.default.Benchmark.BF_power_test
1 min 30 sec - #22
23 hr - #11
54 sec
Build scheduled
lcbbb.default.Benchmark.Dhrystone
23 hr - #1
N/A
26 sec
lcbbb.default.Functional.BF_serial_tx
25 min - #22
1 hr 15 min - #8
43 sec
lcbbb.default.Functional.fuego_board_check
18 hr - #2
14 days - #1
26 sec
lcbbb.default.Functional.hello_world
N/A
N/A
N/A
Icon: S M L
Legend
Atom feed for all
Atom feed for failures
Atom feed for just latest builds

Fuego PM test

```
function test_run {
    WORKLOAD_COMMAND="sudo stress --cpu 4 --io 3 --vm 2 … --timeout 20s ..."
    echo "Executing power test using '$TEST_CLIENT' on '$DUT' using resource '$RESOURCE'"
    token=$($TEST_CLIENT $RESOURCE power-measurement start) || true
   ...
    report "echo Running 'stress' workload on $DUT"
    report_append "${WORKLOAD_COMMAND}"
   …
    echo "Stopping power measurement capture"
    $TEST_CLIENT $RESOURCE power-measurement stop $token

    echo "Getting power measurement data"
    log_this "echo START_POWER_DATA"
    log_this "$TEST_CLIENT $RESOURCE power-measurement get-data $token"
    log_this "echo END_POWER_DATA"

    echo "Removing power measurement data from the server"
    $TEST_CLIENT $RESOURCE power-measurement delete $token
}
```

What's next?

- **Promote the use of the API and implementations**
- **Add APIs**
  - canbus is next on the list
    - expect to be able to use set-config/capture/put APIs
  - USB connect/disconnect
  - What API would you like to see?
- **Add more clients and client examples**
  - LTP serial port test
  - Upstream EBF changes to LAVA
- **Use for more production testing**
  - More real-world testing, especially for new APIs, to help refine them

What's next? (cont'd)

- **Would be good to establish an ecosystem of lab resource drivers**
  - Establish standards for dropping new resources into existing labs
    - Example – Easily add Fuego resource controller with hardware into Timesys EBF infrastructure

Supplemental Slides

(Not presented)

Embedded Linux Conference

Questions or comments?

Direct support for API in Fuego (a bit more)

- **Fuego can run tests in 'standalone' mode, or as more traditional host/target Fuego jobs**
  - Standalone mode requires device under test to be configured with lab client
  - Host/target mode does farm operations from test framework host