About this BoF
aimed at

- individual kernel engineers, subsystem maintainers and other interested individuals
- a single device-under-test attached to a development box and able to triage results coming out of LKFT and other systems based on LAVA
- allowing re-running LKFT test jobs with locally patched kernel tree etc.
- not having to cope with dozens of devices
Are these assumptions correct?

- already able to run LAVA locally
- locally automate jobs working with already-supported/known-automatable devices
- locally automate jobs working with correct/same version of firmware
- replicating lab hardware is not always affordable/desirable/available
- 100’s of jobs reliability rather than 100K’s of jobs reliability target is ok
- world-wide sourceable control hardware is preferable
- accurately describable/reproducible control hardware is important
- HiKey 6220 is a great working example “the most problematic board due to various reasons” (LAVA Integration Stories)
Rolling your own LAVA automation?
Many have tried ...
Re-cap What LAVA Needs for (local) Target Control

- Automation-compatible serial connection
  - screen can work
  - ser2net is better
  - udev rules

- Power control
  - Putting the device in a known state. Power off for dev boards is usually unambiguous
  - Automated reset. Working assumption for most devices is that simply cycling power is sufficient for a full reset

- USB control (assuming Hikey example)
  - OTG port can be used for loading images via fastboot
  - We need a mechanism to switch off the OTG port after loading the image so that the Type A port starts working and brings up the connected USB ethernet adapter
Device Dictionary entries for Target Control

connection_command  (combination of ser2net entry and udev rules)
power_on_command
power_off_command
hard_reset_command  (power cycle)
pre_power_command    (power up the OTG connection)
pre_os_command      (power down the OTG connection)

E.g.
{% set power_off_command = 'python /usr/bin/relay-ctrl.py 3 LOW' %} \  
{% set power_on_command = 'python /usr/bin/relay-ctrl.py 3 HIGH' %} \  
{% set hard_reset_command = 'python /usr/bin/relay-ctrl.py 3 RESET' %} \  
{% set connection_command = 'telnet localhost 7000' %} \  
{% set soft_reset_command = 'fastboot reboot -s 547E6F8B000B0909' %} \  
{% set pre_power_command = 'python /usr/bin/relay-ctrl.py 4 HIGH' %} \  
{% set pre_os_command = 'python /usr/bin/relay-ctrl.py 4 LOW' %} \  
{% set fastboot_serial_number = '547E6F8B000B0909' %}"
Sourcing of DUT Control Hardware

- Expensive hardware that is the same as the hardware in the lab
  - Known to work
  - Expensive
  - World-wide availability?

- Other ‘pro’ level hardware
  - May be well documented & hopefully should work
  - Expensive - can’t try all available models

- Branded consumer hardware
  - Hopefully cheap
  - Poor documentation
  - May get lucky

- Unbranded consumer hardware (Ebay, Ali etc)
  - Very cheap
  - Things that look the same might not be the same

- Community hardware designs
  - Cheap
  - Often well documented
  - WW availability pretty good (may be copies)
## DUT Power Control Implementations (PDU/uPDU)

<table>
<thead>
<tr>
<th>Method</th>
<th>Detail</th>
<th>Pros/Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mains switching products</td>
<td>● Managed PDUs, controlled via telnet or SNMP</td>
<td>Replicate Lab setup</td>
</tr>
<tr>
<td></td>
<td>● Consumer controllable switches</td>
<td>Depends on budget, space and availability</td>
</tr>
<tr>
<td>Mains switching crafted solutions</td>
<td>● Inserting a (hopefully mains-rated) relay into the line</td>
<td>Danger of death (bad) and even possible equipment destruction (worse). Not recommended</td>
</tr>
<tr>
<td>Low voltage switching</td>
<td>● USB controlled relays</td>
<td>Some solutions require complex/conflicting USB driver setups (since e.g. they repurpose FTDI chips for PIO rather than serial)</td>
</tr>
<tr>
<td></td>
<td>● Ethernet controlled relays</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Microcontroller hosted via Shield/Cape/Hat relay boards</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Transistor switching via motor control boards</td>
<td></td>
</tr>
</tbody>
</table>
# USB Control Implementations

<table>
<thead>
<tr>
<th>Method</th>
<th>Detail</th>
<th>Pros/Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB Cable Hacking</td>
<td>● Modify USB cables to route 5V via a relay</td>
<td>Depends on craft skills?</td>
</tr>
<tr>
<td></td>
<td>● Hack hubs to add relays or switching transistors</td>
<td></td>
</tr>
<tr>
<td>Downstream hub control</td>
<td>● Pro (expensive) hubs like Cambrionix <a href="https://cambrionix.com">https://cambrionix.com</a></td>
<td>Depends on budget, or what have/can find locally</td>
</tr>
<tr>
<td></td>
<td>● Dev board (cheap) hubs like Yepkit <a href="https://www.yepkit.com/">https://www.yepkit.com/</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Very few consumer hubs actually support per-port power switching. There’s a list here <a href="https://github.com/mvp/uhubctl">https://github.com/mvp/uhubctl</a></td>
<td></td>
</tr>
<tr>
<td>Root hub control</td>
<td>● quite a few modern motherboards have built-in root hubs that do support this feature</td>
<td>● Depends on motherboard</td>
</tr>
<tr>
<td></td>
<td>● BBB and RPi host ports are controllable</td>
<td>● can use a BBB/RPi remotely via usbip</td>
</tr>
</tbody>
</table>
Discussion
Demo Friday

- 3 Hikeys
- automated in 3 different ways
- running standard LAVA jobs from the lab
Thank You

#HKG18
HKG18 keynotes and videos on: connect.linaro.org
For further information: www.linaro.org